ATMOS-ORIC1

manuel de référence tome 2

TRAVAUX PRATIQUES

André CHENIÈRE

- UN ÉDITEUR DE LANGAGE MACHINE
- UN MONITEUR COMPACT ET RELOGEABLE
- DIVERSES MÉTHODES DE TRI
- DÉPLACEMENT ET PROTECTION D'UN PROGRAMME BASIC

ISOSOFT EDITION

ATMOS-ORIC 1

manuel de référence

par André Chénière

TOME 2

TRAVAUX PRATIQUES

ISOSOFT
édition

INTRODUCTION

Bien que chacune des quatre parties de ce livre soit consacrée à un sujet particulier, il est préférable de commencer par les deux premières.
Prenez le temps de vous équiper des programmes utilitaires proposés.
Quelques heures de frappe fastidieuse seront largement récompensées par l'acquisition d'outils que vous jugerez vite indispensables.
Ces programmes sont écrits en langage machine pour une efficacité optimum.
N'y voyez aucune exclusive à l'égard du Basic.
Selon l'application envisagée, l'un ou l'autre langage convient mieux que l'autre ou bien souvent on les mélange suivant un savant dosage.
Il n'y a pas d'incompatilité entre eux comme nous le verrons en de nombreuses occasions.

Tous les programmes ou sous-programmes en langage machine sont accompagnés d'un listing d'assemblage documenté.
Contentez-vous d'en saisir les grandes lignes dans un premier temps.
Vous y reviendrez pour glaner quelques idées; les index en fin de volume vous guideront dans des recherches particulières.
Ces programmes ne sont pas des modèles infaillibles mais vous reconnaîtrez du moins qu'ils sont construits sans complications.
Ils ont été composés à l'aide de l'As des AS, publié par Isosoft.
C'est un Editeur-Assembleur-Désassembleur simple et pratique que je vous recommande.

Le langage Basic reprend le premier rôle dans les deux dernières parties.
En plus de leur intérêt propre, les sujets évoqués fournissent un prétexte pour approfondir certains aspects du fonctionnement d'un programme.

Curieusement, le problème du tri est rarement traité dans les livres ou revues spécialisés ou alors c'est de manière théorique et plutôt rébarbative par d'éminents informaticiens.
Dans l'étude présentée, le sujet est volontairement limité à ses applications immédiates sur votre ordinateur.
Avantage supplémentaire, vous vous familiariserez avec la structure des tableaux de variables.
Un programme d'enregistrement et de lecture sur cassette (STORE et RECALL) est proposé aux utilisateurs d'Oric-1.

La mobilité d'un programme est une qualité que vous apprécierez sûrement en utilisant le moniteur.
Le propos de la dernière partie est de donner à un programme Basic la même liberté de mouvements.
Même si l'aspect pratique paraît ici moins évident, suivez pas à pas les diverses étapes.
Au terme de cet exercice de style, les pointeurs Basic ne devraient plus avoir de secrets pour vous.

En conclusion, l'objectif visé est unique.
Il est de vous faire progresser dans la connaissance de votre machine en vous dotant de moyens de contrôle pratiques et en vous suggérant des exercices profitables.
C'est un lien puissant entre les différentes parties de cet ouvrage dont le manque de cohésion n'est qu'apparent.

PREMIERE PARTIE

UN EDITEUR DE LANGAGE MACHINE

BUT ET DESCRIPTION

De nombreux programmes en langage machine sont publiés dans les magazines spécialisés.
Ils se présentent souvent sous forme de listing d'assemblage comportant les mnémoniques du code-source et le code-objet en hexadécimal.
Pour entrer un tel programme, il n'est pas nécessaire, ni même souhaitable, d'utiliser un assembleur.
Le moyen classique est un programme de chargement en Basic.
Le code machine, rangé en DATA, est lu puis implanté par les commandes READ et POKE incluses dans une boucle FOR-NEXT.
La méthode servira encore à charger le programme utilitaire dont vous allez vous doter.

L'éditeur de langage machine, objet de cette première partie, inscrit directement en mémoire le code entré en hexadécimal.
L'étape Basic intermédiaire est évitée.
La corvée de clavier subsiste mais vous l'effectuerez dans de meilleures conditions de confort et de sécurité.

Le programme est entièrement écrit en langage machine.
Vous trouverez plus loin son listing d'assemblage commenté.
Logé initialement en #6000, il occupe 1,5 K-octets dans une zone rarement utilisée. Néammoins, notez qu'il n'est pas auto-protégé contre une interférence possible avec le Basic.
Un aspect opérationnel intéressant. Il peut être aisément déplacé grâce au Moniteur présenté dans la deuxième partie de ce livre (page 56).
La cohabitation avec d'autres programmes quelconques est ainsi assurée dans tous les cas.

Le mode d'emploi est simple; nous en reparlerons plus tard.
Auparavant vous taperez le chargeur Basic des pages suivantes.
Les lignes DATA ont été fabriquées à l'aide du programme de conversion binaire-DATA de la page 42.
Ce programme prend du code en mémoire et le transforme en data hexadécimal qu'il place dans des lignes numérotées utilisables en Basic.
Une ancienne version figurait dans mon manuel de référence.
Elle a été remaniée dans le but d'harmoniser le listing Basic sur imprimante avec celui produit par l'éditeur de langage machine.

En somme, le chargeur Basic effectue l'opération inverse puisqu'il lit le code en DATA puis l'inscrit en mémoire.

Important: Le programme de chargement calcule une somme de contrôle globale mais sauvegardez-le quand même sur cassette avant de le lancer. On ne sait jamais, deux erreurs peuvent s'annuler mutuellement.

```
10 REM * Editeur langage machine ATMOS *
20 S=0:FOR A=#6000 TO #65E8:READ C$:C=VAL("#"+C$):POKE A,C
30 S=S+C:NEXT:IF S<>169882 THEN PRINT"erreur":PING:END
40 CALL#6000

100 DATA 20,B8,F8,A9,F8,8D,78,02,A9,D0,8D,7A,02,A9,C0,8D
105 DATA 7C,02,A9,03,8D,7D,02,A9,19,8D,7E,02,A2,28,BD,A5
110 DATA 65,9D,7F,BB,CA,D0,F7,20,E7,64,20,E8,C5,C9,31,90
115 DATA 09,C9,34,90,0B,D0,03,4C,B2,F8,20,9F,FA,4C,1C,60
120 DATA 85,78,A2,3A,86,77,20,24,64,F0,D1,A2,08,86,75,94
125 DATA 5F,CA,95,5F,CA,D0,F8,86,6E,86,6F,86,79,20,C2,64
130 DATA A5,78,C9,31,D0,09,20,CE,CC,20,10,64,4C,88,61,20
135 DATA F0,CB,A2,43,86,77,20,24,64,F0,A1,85,66,84,67,C5
140 DATA 60,98,E5,61,B0,06,20,9F,FA,4C,72,60,20,C5,64,A5
145 DATA 78,C9,32,D0,0C,20,CE,CC,20,24,61,20,9B,64,4C,88
150 DATA 61,A2,1A,BD,CE,65,9D,8D,BB,CA,D0,F7,A2,4C,20,E7
155 DATA 64,20,E8,C5,C9,38,F0,02,06,75,A2,66,20,E7,64,20
160 DATA E8,C5,C9,4F,D0,02,66,79,20,16,C8,A9,0D,20,D9,CC
165 DATA 20,40,61,08,24,79,10,20,20,D4,CC,20,D4,CC,20,D4
170 DATA CC,A6,6F,E4,75,F0,04,E6,6F,D0,ED,20,7F,61,20,89
175 DATA 63,A4,69,A6,68,20,09,65,20,F0,CB,28,90,20,20,CA
180 DATA 63,A5,75,C9,08,F0,03,20,CC,63,20,78,EB,30,0F,A5
185 DATA 6E,C9,30,D0,BB,20,E8,C5,A9,00,85,6E,F0,B2,20,2F
190 DATA C8,4C,1C,60,20,74,64,A9,0D,20,D9,CC,20,40,61,90
195 DATA 0E,20,CA,63,A5,6E,C9,19,F0,05,20,F0,CB,D0,ED,60
200 DATA 20,F3,64,86,6A,84,6B,A0,00,A5,66,C5,6A,A5,67,E5
205 DATA 6B,90,16,B1,6A,20,0E,65,20,D4,CC,E6,6A,D0,02,E6
210 DATA 6B,E6,6F,A5,6F,C5,75,D0,E0,60,20,74,64,A9,0D,20
215 DATA D9,CC,20,40,61,90,03,4C,9B,64,A9,11,20,D9,CC,A5
220 DATA 6A,A4,6B,85,64,84,65,60,A9,20,8D,8E,BB,20,F3,64
225 DATA AD,A1,BB,29,F0,D0,03,20,29,63,A9,00,85,68,85,69
230 DATA A9,02,85,70,20,E8,C5,A0,02,C4,70,D0,65,AC,A1,BB
235 DATA C0,0C,F0,4F,C9,1B,D0,03,4C,1C,60,C9,2B,D0,09,20
240 DATA 89,63,20,C8,64,4C,9A,61,C9,08,D0,06,20,4B,62,4C
245 DATA A4,61,C9,0B,D0,06,20,6F,62,4C,A4,61,48,20,20,63
250 DATA 68,90,2F,C9,09,D0,06,20,AF,62,4C,A4,61,C9,0A,D0
255 DATA 06,20,DB,62,4C,A4,61,C9,1A,D0,08,A9,0C,8D,A1,BB
260 DATA 4C,90,61,C9,7F,D0,0B,A9,2C,8D,A1,BB,20,59,63,4C
265 DATA 9A,61,20,4F,64,B0,06,20,9F,FA,4C,A4,61,20,D9,CC
270 DATA C6,70,F0,03,4C,A4,61,A0,00,A5,68,91,64,20,D4,CC
275 DATA E6,6F,20,20,63,B0,03,20,D8,63,20,B6,63,F0,03,4C
280 DATA 90,61,20,F0,CB,20,C4,62,4C,8D,61,A5,6F,D0,05,20
285 DATA 17,63,F0,18,C6,6F,A9,08,A2,03,A4,6F,10,02,A2,11
290 DATA 20,D9,CC,CA,D0,FA,20,EC,63,4C,87,62,4C,9F,FA,20
295 DATA 17,63,F0,F8,20,02,64,A9,0B,20,D9,CC,38,A5,64,E9
300 DATA 08,85,64,B0,02,C6,65,A5,6E,10,23,20,74,64,A9,BF
305 DATA 85,CA,85,C8,A9,90,85,C9,A9,B8,85,C7,A9,D0,85,CE
310 DATA A9,BB,85,CF,20,FB,C3,20,6D,61,A9,00,85,6E,60,E6
315 DATA 6F,A2,03,A4,6F,C0,08,D0,02,A2,11,20,D9,CC,CA,D0
320 DATA FA,20,B6,63,A5,6E,C9,19,D0,10,C6,6E,20,74,64,A9
```

```
325 DATA 0D,20,D9,CC,20,40,61,4C,9B,64,60,20,CA,63,38,A5
330 DATA 66,E5,62,AA,A5,67,E5,63,A8,B0,0C,E8,D0,03,C8,F0
335 DATA 06,20,02,64,4C,6A,61,A9,0A,20,D9,CC,18,A5,64,69
340 DATA 08,85,64,90,02,E6,65,A5,6E,C9,19,D0,02,C6,6E,20
345 DATA 20,63,B0,B8,4C,6A,61,A5,60,C5,62,A5,61,E5,63,60
350 DATA A5,66,C5,64,A5,67,E5,65,60,A5,64,A4,65,85,CE,84
355 DATA CF,20,D8,63,18,A5,66,85,C9,69,01,85,C7,A5,67,85
360 DATA CA,69,00,85,C8,20,FB,C3,20,24,61,20,9B,64,AC,69
365 DATA 02,A9,20,91,12,C8,91,12,60,A5,64,A4,65,85,6A,84
370 DATA 6B,A0,01,B1,6A,88,91,6A,E6,6A,D0,02,E6,6B,A5,6A
375 DATA C5,66,A5,6B,E5,67,90,E9,20,10,64,20,24,61,B0,06
380 DATA 20,D4,CC,20,D4,CC,4C,9B,64,A5,60,A4,61,85,6A,84
385 DATA 6B,A0,00,84,68,84,69,A5,6A,C5,64,A5,6B,E5,65,B0
390 DATA 14,A5,68,71,6A,85,68,A5,69,69,00,85,69,E6,6A,D0
395 DATA 02,E6,6B,D0,E2,60,E6,64,D0,02,E6,65,A5,6F,C9,08
400 DATA D0,07,20,CA,63,A9,00,85,6F,60,E6,6E,18,A5,62,69
405 DATA 08,85,62,90,02,E6,63,60,18,A5,66,69,01,85,66,85
410 DATA 68,A5,67,69,00,85,67,85,69,4C,C5,64,C6,64,A5,64
415 DATA C9,FF,D0,02,C6,65,A5,6F,10,07,20,02,64,A9,07,85
420 DATA 6F,60,38,A5,62,E9,08,85,62,B0,02,C6,63,C6,6E,60
425 DATA 38,A5,66,E9,01,85,66,85,68,A5,67,E9,00,85,67,85
430 DATA 69,4C,C5,64,A6,77,20,E7,64,20,80,CD,86,E9,84,EA
435 DATA A2,00,86,68,86,69,20,E2,00,F0,0D,20,4F,64,B0,06
440 DATA 20,9F,FA,4C,24,64,D0,EE,A5,68,A4,69,A6,35,60,85
445 DATA 76,49,30,C9,0A,90,06,69,88,C9,FA,90,14,A2,03,0A
450 DATA 0A,0A,0A,0A,26,68,26,69,B0,06,CA,10,F6,38,B0,01
455 DATA 18,A5,76,60,AD,69,02,85,71,AD,68,02,85,72,A5,12
460 DATA 85,73,A5,13,85,74,A5,6E,85,76,A5,6F,85,77,A5,62
465 DATA A4,63,85,6C,84,6D,A9,11,4C,D9,CC,A5,71,8D,69,02
470 DATA A5,72,8D,68,02,A5,73,85,12,A5,74,85,13,A5,76,85
475 DATA 6E,A5,77,85,6F,A5,6C,A4,6D,85,62,84,63,A9,11,4C
480 DATA D9,CC,A9,02,2C,A9,09,2C,A9,11,85,77,A2,FF,86,2F
485 DATA E8,A5,69,20,93,D9,A5,68,20,93,D9,A9,00,9D,00,01
490 DATA A0,01,A6,77,4C,65,F8,BD,24,65,F0,06,20,D9,CC,E8
495 DATA D0,F5,60,A6,62,A4,63,20,09,65,A9,3A,20,D9,CC,A9
500 DATA 00,85,6F,20,D4,CC,4C,D4,CC,98,20,0E,65,8A,48,4A
505 DATA 4A,4A,4A,20,19,65,68,29,0F,09,30,C9,3A,90,02,69
510 DATA 06,4C,D9,CC,0C,31,2E,4E,6F,75,76,65,6C,6C,65,20
515 DATA 65,6E,74,72,65,65,0D,0A,32,2E,45,64,69,74,69,6F
520 DATA 6E,0D,0A,33,2E,49,6D,70,72,65,73,73,69,6F,6E,0D
525 DATA 0A,34,2E,53,6F,72,74,69,65,0D,0A,0A,0A,00,0B,0D
530 DATA 0E,44,65,62,75,74,00,0B,0D,0E,46,69,6E,20,20,00
535 DATA 38,20,6F,75,20,31,36,20,63,6F,6C,3F,20,28,64,65
540 DATA 66,61,75,74,20,31,36,29,20,00,0D,0A,53,6F,6D,6D
545 DATA 65,20,64,65,20,63,6F,6E,74,72,6F,6C,65,3F,20,28
550 DATA 4F,2F,4E,29,20,00,41,23,20,20,20,20,2C,45,23,20
555 DATA 20,20,20,20,00,53,3D,20,20,20,20,20,20,20,45,53
560 DATA 43,2C,2B,2C,44,45,4C,2C,43,54,52,4C,2D,5A,20,20
565 DATA 34,38,20,6C,69,67,6E,65,73,20,20,53,50,43,3D,53
570 DATA 74,6F,70,2C,53,75,69,74,65
```

```
10 REM * Editeur langage machine ORIC-1 *
20 S=0:FOR A=#6000 TO #65BE:READ C$:C=VAL("#"+C$):POKE A,C
30 S=S+C:NEXT:IF S<>160426 THEN PRINT"erreur":PING:END
40 CALL#6000

100 DATA 20,88,F8,A9,A8,8D,6D,02,A9,19,8D,6F,02,A2,28,BD
105 DATA 7B,65,9D,7F,BB,CA,D0,F7,20,BD,64,20,F8,C5,C9,31
110 DATA 90,09,C9,34,90,0B,D0,03,4C,82,F8,20,85,FA,4C,0D
115 DATA 60,85,78,A2,3A,86,77,20,FA,63,F0,D1,A2,08,86,75
120 DATA 94,5F,CA,95,5F,CA,D0,F8,86,6E,86,6F,86,79,20,98
125 DATA 64,A5,78,C9,31,D0,09,20,0A,CC,20,E6,63,4C,7A,61
130 DATA 20,9F,CB,A2,43,86,77,20,FA,63,F0,A1,85,66,84,67
135 DATA C5,60,98,E5,61,B0,06,20,85,FA,4C,63,60,20,9B,64
140 DATA A5,78,C9,32,D0,0C,20,0A,CC,20,16,61,20,71,64,4C
145 DATA 7A,61,A2,1A,BD,A4,65,9D,8D,BB,CA,D0,F7,A2,4C,20
150 DATA BD,64,20,F8,C5,C9,38,F0,02,06,75,A2,66,20,BD,64
155 DATA 20,F8,C5,C9,4F,D0,02,66,79,38,6E,F1,02,A9,0D,20
160 DATA 12,CC,20,32,61,08,24,79,10,20,20,0D,CC,20,0D,CC
165 DATA 20,0D,CC,A6,6F,E4,75,F0,04,E6,6F,D0,ED,20,71,61
170 DATA 20,5F,63,A4,69,A6,68,20,DF,64,20,9F,CB,28,90,20
175 DATA 20,A0,63,A5,75,C9,08,F0,03,20,A2,63,20,05,E9,30
180 DATA 0F,A5,6E,C9,30,D0,BB,20,F8,C5,A9,00,85,6E,F0,B2
185 DATA 4E,F1,02,4C,0D,60,20,4A,64,A9,0D,20,12,CC,20,32
190 DATA 61,90,0E,20,A0,63,A5,6E,C9,19,F0,05,20,9F,CB,D0
195 DATA ED,60,20,C9,64,86,6A,84,6B,A0,00,A5,66,C5,6A,A5
200 DATA 67,E5,6B,90,16,B1,6A,20,E4,64,20,0D,CC,E6,6A,D0
205 DATA 02,E6,6B,E6,6F,A5,6F,C5,75,D0,E0,60,20,4A,64,A9
210 DATA 0D,20,12,CC,20,32,61,90,03,4C,71,64,A9,11,20,12
215 DATA CC,A5,6A,A4,6B,85,64,84,65,60,A9,20,8D,8E,BB,20
220 DATA C9,64,AD,A1,BB,29,F0,D0,03,20,FF,62,A9,00,85,68
225 DATA 85,69,A9,02,85,70,20,F8,C5,A0,02,C4,70,D0,65,AC
230 DATA A1,BB,C0,0C,F0,4F,C9,1B,D0,03,4C,0D,60,C9,2B,D0
235 DATA 09,20,5F,63,20,9E,64,4C,8C,61,C9,08,D0,06,20,3D
240 DATA 62,4C,96,61,C9,0B,D0,06,20,61,62,4C,96,61,48,20
245 DATA F6,62,68,90,2F,C9,09,D0,06,20,85,62,4C,96,61,C9
250 DATA 0A,D0,06,20,B1,62,4C,96,61,C9,1A,D0,08,A9,0C,8D
255 DATA A1,BB,4C,82,61,C9,7F,D0,0B,A9,2C,8D,A1,BB,20,2F
260 DATA 63,4C,8C,61,20,25,64,B0,06,20,85,FA,4C,96,61,20
265 DATA 12,CC,C6,70,F0,03,4C,96,61,A0,00,A5,68,91,64,20
270 DATA 0D,CC,E6,6F,20,F6,62,B0,03,20,AE,63,20,8C,63,F0
275 DATA 03,4C,82,61,20,9F,CB,20,9A,62,4C,7F,61,A5,6F,D0
280 DATA 05,20,ED,62,F0,18,C6,6F,A9,08,A2,03,A4,6F,10,02
285 DATA A2,13,20,12,CC,CA,D0,FA,20,C2,63,4C,79,62,4C,85
290 DATA FA,20,ED,62,F0,F8,20,D8,63,A9,0B,20,12,CC,38,A5
295 DATA 64,E9,08,85,64,B0,02,C6,65,A5,6E,10,07,20,5C,61
300 DATA A9,00,85,6E,60,E6,6F,A2,03,A4,6F,C0,08,D0,02,A2
305 DATA 13,20,12,CC,CA,D0,FA,20,8C,63,A5,6E,C9,19,D0,10
310 DATA C6,6E,20,4A,64,A9,0D,20,12,CC,20,32,61,4C,71,64
315 DATA 60,20,A0,63,38,A5,66,E5,62,AA,A5,67,E5,63,A8,B0
320 DATA 0C,E8,D0,03,C8,F0,06,20,D8,63,4C,5C,61,A9,0A,20
```

```
325 DATA 12,CC,18,A5,64,69,08,85,64,90,02,E6,65,A5,6E,C9
330 DATA 19,D0,02,C6,6E,20,F6,62,B0,B8,4C,5C,61,A5,60,C5
335 DATA 62,A5,61,E5,63,60,A5,66,C5,64,A5,67,E5,65,60,A5
340 DATA 64,A4,65,85,CE,84,CF,20,AE,63,18,A5,66,85,C9,69
345 DATA 01,85,C7,A5,67,85,CA,69,00,85,C8,20,FF,C3,20,16
350 DATA 61,20,71,64,AC,69,02,A9,20,91,12,C8,91,12,60,A5
355 DATA 64,A4,65,85,6A,84,6B,A0,01,B1,6A,88,91,6A,E6,6A
360 DATA D0,02,E6,6B,A5,6A,C5,66,A5,6B,E5,67,90,E9,20,E6
365 DATA 63,20,16,61,B0,06,20,0D,CC,20,0D,CC,4C,71,64,A5
370 DATA 60,A4,61,85,6A,84,6B,A0,00,84,68,84,69,A5,6A,C5
375 DATA 64,A5,6B,E5,65,B0,14,A5,68,71,6A,85,68,A5,69,69
380 DATA 00,85,69,E6,6A,D0,02,E6,6B,D0,E2,60,E6,64,D0,02
385 DATA E6,65,A5,6F,C9,08,D0,07,20,A0,63,A9,00,85,6F,60
390 DATA E6,6E,18,A5,62,69,08,85,62,90,02,E6,63,60,18,A5
395 DATA 66,69,01,85,66,85,68,A5,67,69,00,85,67,85,69,4C
400 DATA 9B,64,C6,64,A5,64,C9,FF,D0,02,C6,65,A5,6F,10,07
405 DATA 20,D8,63,A9,07,85,6F,60,38,A5,62,E9,08,85,62,B0
410 DATA 02,C6,63,C6,6E,60,38,A5,66,E9,01,85,66,85,68,A5
415 DATA 67,E9,00,85,67,85,69,4C,9B,64,A6,77,20,BD,64,20
420 DATA F4,CC,86,E9,84,EA,A2,00,86,68,86,69,20,E2,00,F0
425 DATA 0D,20,25,64,B0,06,20,85,FA,4C,FA,63,D0,EE,A5,68
430 DATA A4,69,A6,35,60,85,76,49,30,C9,0A,90,06,69,88,C9
435 DATA FA,90,14,A2,03,0A,0A,0A,0A,0A,26,68,26,69,B0,06
440 DATA CA,10,F6,38,B0,01,18,A5,76,60,AD,69,02,85,71,AD
445 DATA 68,02,85,72,A5,12,85,73,A5,13,85,74,A5,6E,85,76
450 DATA A5,6F,85,77,A5,62,A4,63,85,6C,84,6D,A9,11,4C,12
455 DATA CC,A5,71,8D,69,02,A5,72,8D,68,02,A5,73,85,12,A5
460 DATA 74,85,13,A5,76,85,6E,A5,77,85,6F,A5,6C,A4,6D,85
465 DATA 62,84,63,A9,11,4C,12,CC,A9,02,2C,A9,09,2C,A9,11
470 DATA 85,77,A2,FF,86,2F,E8,A5,69,20,F5,D8,A5,68,20,F5
475 DATA D8,A9,00,9D,00,01,A0,01,A6,77,4C,2F,F8,BD,FA,64
480 DATA F0,06,20,12,CC,E8,D0,F5,60,A6,62,A4,63,20,DF,64
485 DATA A9,3A,20,12,CC,A9,00,85,6F,20,0D,CC,4C,0D,CC,98
490 DATA 20,E4,64,8A,48,4A,4A,4A,4A,20,EF,64,68,29,0F,09
495 DATA 30,C9,3A,90,02,69,06,4C,12,CC,0C,31,2E,4E,6F,75
500 DATA 76,65,6C,6C,65,20,65,6E,74,72,65,65,0D,0A,32,2E
505 DATA 45,64,69,74,69,6F,6E,0D,0A,33,2E,49,6D,70,72,65
510 DATA 73,73,69,6F,6E,0D,0A,34,2E,53,6F,72,74,69,65,0D
515 DATA 0A,0A,0A,00,0B,0D,0E,44,65,62,75,74,00,0B,0D,0E
520 DATA 46,69,6E,20,20,00,38,20,6F,75,20,31,36,20,63,6F
525 DATA 6C,3F,20,28,64,65,66,61,75,74,20,31,36,29,20,00
530 DATA 0D,0A,53,6F,6D,6D,65,20,64,65,20,63,6F,6E,74,72
535 DATA 6F,6C,65,3F,20,28,4F,2F,4E,29,20,00,41,23,20,20
540 DATA 20,20,2C,45,23,20,20,20,20,20,00,53,3D,20,20,20
545 DATA 20,20,20,20,45,53,43,2C,2B,2C,44,45,4C,2C,43,54
550 DATA 52,4C,2D,5A,20,20,34,38,20,6C,69,67,6E,65,73,20
555 DATA 20,53,50,43,3D,53,74,6F,70,2C,53,75,69,74,65
```

MODE D'EMPLOI

Après lancement, l'inscription du code en mémoire (ligne 20) dure une quinzaine de secondes puis l'ELM est appelé (ligne 30).
Un menu apparaît, offrant quatre options qui seront sélectionnées par leur numéro.
L'option 4 fait sortir du programme, retourner au Basic.
Vous pouvez l'essayer dès maintenant et sauvegarder le programme par:

`CSAVE"ELM",A#6000,E#65FF` relancez-le ensuite avec `CALL#6000`

OPTION 1: NOUVELLE ENTREE

Elle permet l'introduction d'un nouveau bloc de code.
A la demande du programme, tapez l'adresse de début, en hexadécimal, sans symbole #, suivie de RETURN. Cette procédure est valable aussi pour l'adresse de fin demandée par les autres options.
L'adresse est affichée. Vous pouvez entrer le code à votre convenance.
Seuls les chiffres hexadécimaux sont acceptés. Le curseur passe automatiquement à la position suivante après la frappe de deux chiffres valides.
Les lignes sont de huit octets. Au passage à la ligne suivante, une nouvelle adresse de base s'affiche.
L'éditeur est du type pleine page. Le curseur se déplace dans les quatre directions à l'aide des flèches dans les limites du bloc de code affiché.
Après l'entrée d'un premier chiffre hexa, le programme en exige un second avant d'accepter toute commande.
Ainsi, si le premier est erroné, il faut taper un deuxième chiffre pour pouvoir revenir en arrière et corriger.
Les indications de la ligne supérieure ont la signification suivante:
- A et E sont les adresses de début et de fin.
- S est la somme de contrôle dont nous reparlerons.

Dans la partie droite, on trouve un résumé des commandes disponibles.
- ESC permet le retour au menu.
- la commande + affiche une somme de contrôle.
- DEL supprime le code sous le curseur. Ce code est écrasé par le décalage du reste du bloc vers le bas de la mémoire.
- CTRL-Z met en mode insertion. Un espace est créé à l'emplacement pointé par le curseur, le reste du bloc étant décalé vers le haut. DEL supprime l'espace et fait sortir de ce mode.

OPTION 2: EDITION

Elle permet d'intervenir directement dans une zone en mémoire dont les adresses de début et de fin sont spécifiées.
La seule différence avec l'option précédente est que le bloc de code est défini au départ au lieu d'être construit peu à peu.
L'image de la zone mémoire apparaît à l'écran avec défilement possible vers le haut ou vers le bas.
Toutes les facilités évoquées plus haut, surimpression, suppression, insertion, ajout, sont disponibles.
Grâce à cette option, un travail de saisie au clavier peut être interrompu, sauvegardé, puis repris plus tard.
Si le bloc à manipuler est important (plusieurs K-octets, par exemple), ne soyez pas surpris du retard à l'affichage en mode suppression ou insertion.
Le temps nécessaire à l'exécution de milliers de décalages n'est pas négligeable malgré la rapidité du langage machine.

OPTION 3: IMPRESSION

Vérifiez que votre imprimante est branchée et sous tension avant d'utiliser cette option.
Elle assure la reproduction sur papier du contenu d'une zone en mémoire.
Ce vidage (en anglais: dump) est imprimé en hexadécimal.
Vous choisirez 8 ou 16 colonnes (8 ou 16 octets par ligne) en fonction des caractéristiques de votre imprimante.
Une somme de contrôle peut être imprimée en fin de ligne.

Après 48 lignes, une pause permet de changer la feuille de papier.
L'action sur la barre d'espace (en fait, sur une touche quelconque) supprime cette pause.

Il y a retour au menu en fin de bloc. L'impression peut être interrompue avant la fin par la frappe d'une touche quelconque.

SOMME DE CONTROLE

La commande + actionne une routine qui calcule la somme, sur deux octets, de toutes les valeurs en mémoire depuis le début jusqu'à celle précédant le curseur (y comprise).
Si le résultat dépasse FFFF la retenue est ignorée.
Le nombre est affiché en hexadécimal.
Il suffit de le comparer à un nombre que l'on sait exact. La non-concordance indique une erreur.
La fiabilité est très grande mais pas absolue. (Il peut se produire une inversion ou deux erreurs peuvent s'annuler mutuellement).

En option impression, une somme de contrôle, toujours calculée depuis le début du bloc, peut être imprimée en fin de ligne de 8 ou 16 octets.

Vous utiliserez cette facilité pour entrer le Moniteur proposé en seconde partie. Après, c'est la désuétude! A moins que l'idée ne séduise les auteurs de programmes Oric?

PRECAUTIONS

Le code entré au clavier est inscrit directement à l'emplacement prévu, sans passer par une mémoire tampon intermédiaire. Le buffer d'entrée (#35-84) ne sert que pour le recueil des adresses.

Le programme peut intervenir sur toute l'étendue de la mémoire vive (00-BFFF).
Attention aux emplacements utilisés par le système (00-3FF et B400-BBDF).

Surveillez l'adresse de fin pour les interférences possibles avec d'autres valeurs à préserver. L'ancien contenu de cet emplacement est perdu lors d'un ajout bien sûr, mais aussi dès que le mode insertion est activé. Dans ce cas DEL ne rétablit pas la valeur initiale.

En terminant votre travail, notez les adresses de début et de fin. Elles vous serviront pour une sauvegarde éventuelle.

```
        ORG $6000
        OBJ $6000
```

```
ATMOS

CURBAS =$12
FLG    =$2F
BUF    =$35
DEPAD  =$60
BASAD  =$62
CRSAD  =$64
FINAD  =$66
TEMPA  =$68
TEMPB  =$6A
TEMPC  =$6C
LINCT  =$6E
COLCT  =$6F
CHRCT  =$70
SAV1   =$71
SAV2   =$72
SAV3   =$73
SAV4   =$74
NCOL   =$75
TMP1   =$76
TMP2   =$77
TMP3   =$78
TMP4   =$79
HIGHDS =$C7
HIGHTR =$C9
LOWTR  =$CE
CHRGET =$E2
TXTPTR =$E9
FBUF   =$100
CURROW =$268
CURCOL =$269
VDUL2  =$278
VDUL1  =$27A
NCHR   =$27C
NLIN   =$27E
MOVUP  =$C3FB
INLIN  =$C592
INCHR  =$C5E8
LPRTON =$C816
LPRTOF =$C82F
CRDO   =$CBF0
STROUT =$CCB0
CLS    =$CCCE
OUTSPC =$CCD4
OUTDO  =$CCD9
INPTRQ =$CD80
BINHEX =$D993
RDKEY  =$EB78
STOUT  =$F865
RESET  =$F8B2
RSET0  =$F8B8
PING   =$FA9F
```

```
ORIC-1

CURBAS =$12
FLG    =$2F
BUF    =$35
DEPAD  =$60
BASAD  =$62
CRSAD  =$64
FINAD  =$66
TEMPA  =$68
TEMPB  =$6A
TEMPC  =$6C
LINCT  =$6E
COLCT  =$6F
CHRCT  =$70
SAV1   =$71
SAV2   =$72
SAV3   =$73
SAV4   =$74
NCOL   =$75
TMP1   =$76
TMP2   =$77
TMP3   =$78
TMP4   =$79
HIGHDS =$C7
HIGHTR =$C9
LOWTR  =$CE
CHRGET =$E2
TXTPTR =$E9
FBUF   =$100
CURROW =$268
CURCOL =$269
SCRNAD =$26D
NLIN   =$26F
PRTFLG =$2F1

MOVUP  =$C3FF
INLIN  =$C5A2
INCHR  =$C5F8


CRDO   =$CB9F
STROUT =$CBED
CLS    =$CC0A
OUTSPC =$CC0D
OUTDO  =$CC12
INPTRQ =$CCF4
BINHEX =$D8F5
RDKEY  =$E905
STOUT  =$F82F
RESET  =$F882
RSET0  =$F888
PING   =$FA85
```

Dans la page ci-contre, sont définis les pointeurs, les variables et les routines du système utilisés par le programme.
Toutes les variables propres à l'ELM sont logées dans le buffer d'entrée:

- DEPAD adresse de début du bloc de code concerné.
- BASAD adresse de base affichée à chaque début de ligne.
- CRSAD adresse matérialisée par le curseur.
- FINAD adresse de fin du bloc de code.
- TEMPA B et C temporaires 2 octets.
- LINCT compte les lignes.
- COLCT compte les colonnes, nombre d'octets par ligne.
- CHRCT compte les chiffres hexadécimaux valides entrés au clavier.
- SAV1 2, 3 et 4 sauvegarde des paramètres position curseur.
- NCOL nombre de colonnes (octets par ligne), normalement 8.
- TMP1 2, 3 et 4 temporaires 1 octet.

Les autres étiquettes concernent le système. Les principales différences entre Atmos et Oric 1 sont relatives au mode de gestion de l'écran texte.

- CURBAS adresse 1er emplacement de la ligne où se trouve le curseur.
- FLG drapeau temporaire, utilisé par la routine de conversion binaire-hexadécimal BINHEX (D993 ou D8F5), provoque l'abandon des zéros hexa non significatifs; intervention importune à éliminer.
- BUF début du buffer d'entrée
- HIGHDS, HIGHTR et LOWTR pointeurs utilisés par MOVUP (BLTU).
- CHRGET lit un caractère, TXTPTR pointeur de texte.
- FBUF buffer FAC, ici recueille la chaîne ASCII d'un nombre hexa.
- CURROW, CURCOL ligne ou colonne en cours à l'écran.
- VDUL1, VDUL2 (Atmos) adresses des 1ère et 2ème lignes écran.
- SCRNAD (Oric 1) adresse du coin sup. gauche de l'écran (norm. BB80).
- NCHR capacité écran (normalement 1040 caractères).
- PRTFLG drapeau impression.
- NLIN nombre de lignes autorisées à l'écran (normalement 27).
- MOVUP routine BLTU, transfert de bloc vers le haut, amputée du début (test de la mémoire).
- INLIN recueille texte dans le buffer d'entrée.
- INCHR recueille caractère entré au clavier dans l'accumulateur.
- LPRTON, LPRTOF (Atmos) positionnement drapeau impression.
- CRDO met à la ligne.
- STROUT affiche, à la position d'écriture en cours, la chaîne de caractères pointée par A(-), Y(+) et se terminant par 00 ou ". Cette routine n'est pas utilisée par le programme. L'entrée d'une adresse en mode immédiat compliquerait la relogeabilité (voir p.56)
- CLS vide l'écran, OUTSPC sort un espace, OUTDO sort caractère en A.
- INPTRQ affiche un point d'interrogation puis appelle INLIN.
- BINHEX convertit une valeur binaire sur un octet en deux digits hexadécimaux ASCII rangés dans FBUF.
- RDKEY lit mém. clavier (2DF); si touche enfoncée, N=1 carac. en A.
- STOUT affiche un message sur ligne supérieure écran (status line), chaîne pointée par A(-), Y(+) et terminée par 00. Problème évoqué plus haut (STROUT) ne se pose pas car le programme utilise STOUT pour afficher une chaîne se trouvant dans FBUF (adresse fixe).
- RESET réinit. partielle, contenu mémoire inchangé, retour Basic.
- RSET0 idem mais retour au programme appelant.

```
                        ;Aménagement
                        ;écran texte

6000 20 B8 F8 SCRN   JSR RSET0
6003 A9 F8           LDA #$F8
6005 8D 78 02        STA VDUL2
6008 A9 D0           LDA #$D0
600A 8D 7A 02        STA VDUL1
600D A9 C0           LDA #$C0
600F 8D 7C 02        STA NCHR
6012 A9 03           LDA #$03
6014 8D 7D 02        STA NCHR+1
6017 A9 19           LDA #$19
6019 8D 7E 02        STA NLIN

6000 20 88 F8 SCRN   JSR RSET0
6003 A9 A8           LDA #$A8
6005 8D 6D 02        STA SCRNAD
6008 A9 19           LDA #$19
600A 8D 6F 02        STA NLIN

                        ;Affichage menu

601C A2 28     MENU  LDX #$28
601E BD A5 65 MEN1   LDA MSG2,X
6021 9D 7F BB        STA $BB7F,X
6024 CA              DEX
6025 D0 F7           BNE MEN1
6027 20 E7 64        JSR PRMSG
602A 20 E8 C5 OPT    JSR INCHR
602D C9 31           CMP #"1"
602F 90 09           BCC ERR
6031 C9 34           CMP #"4"
6033 90 0B           BCC GTDEP
6035 D0 03           BNE ERR
6037 4C B2 F8 QUIT   JMP RESET

603A 20 9F FA ERR    JSR PING
603D 4C 1C 60        JMP MENU

600D A2 28     MENU  LDX #$28
600F BD 7B 65 MEN1   LDA MSG2,X
6012 9D 7F BB        STA $BB7F,X
6015 CA              DEX
6016 D0 F7           BNE MEN1
6018 20 BD 64        JSR PRMSG
601B 20 F8 C5 OPT    JSR INCHR
601E C9 31           CMP #"1"
6020 90 09           BCC ERR
6022 C9 34           CMP #"4"
6024 90 0B           BCC GTDEP
6026 D0 03           BNE ERR
6028 4C 82 F8 QUIT   JMP RESET

602B 20 85 FA ERR    JSR PING
602E 4C 0D 60        JMP MENU

                        ;Adresse début

6040 85 78     GTDEP STA TMP3
6042 A2 3A           LDX #$3A
6044 86 77           STX TMP2
6046 20 24 64        JSR ADGET
6049 F0 D1           BEQ MENU
604B A2 08     INIPT LDX #$08
604D 86 75           STX NCOL
604F 94 5F     IPT0  STY DEPAD-1,X
6051 CA              DEX
6052 95 5F           STA DEPAD-1,X
6054 CA              DEX
6055 D0 F8           BNE IPT0
6057 86 6E           STX LINCT
6059 86 6F           STX COLCT
605B 86 79           STX TMP4
605D 20 C2 64 PRDEP  JSR STAT
6060 A5 78           LDA TMP3
6062 C9 31           CMP #"1"
6064 D0 09           BNE GTFIN

6031 85 78     GTDEP STA TMP3
6033 A2 3A           LDX #$3A
6035 86 77           STX TMP2
6037 20 FA 63        JSR ADGET
603A F0 D1           BEQ MENU
603C A2 08     INIPT LDX #$08
603E 86 75           STX NCOL
6040 94 5F     IPT0  STY DEPAD-1,X
6042 CA              DEX
6043 95 5F           STA DEPAD-1,X
6045 CA              DEX
6046 D0 F8           BNE IPT0
6048 86 6E           STX LINCT
604A 86 6F           STX COLCT
604C 86 79           STX TMP4
604E 20 98 64 PRDEP  JSR STAT
6051 A5 78           LDA TMP3
6053 C9 31           CMP #"1"
6055 D0 09           BNE GTFIN
```

Après RSET0 pour démarrer sur des bases solides, le programme entreprend la réduction de la fenêtre d'écran de 27 à 25 lignes. L'affichage sur la première et la dernière ligne est supprimé. Avantages: lisibilité améliorée, meilleur cadrage du code affiché. NLIN est fixé à 25 décimal sur les deux systèmes.
Pour Oric 1, il suffit de modifier aussi SCRNAD (BBA8 au lieu de BB80)
Pour Atmos, les adresses des deux premières lignes VDUL1 et VDUL2 sont augmentées de 40 décimal. Là encore, seuls les octets faibles sont à changer, les modifications intervenant dans la même page #BB.
NCHR, capacité écran, utilisée pour le défilement automatique, doit être ajustée (960 caractères au lieu de 1040).

Une boucle inscrit le message étiqueté MSG2 (40 carac.) sur toute la ligne supérieure. Examinez ce message en fin de programme (page 40); il commence après 00 qui marque la fin du message précédent.
Vers le milieu, on trouve un autre 00. Cette valeur, attribut encre noire, occulte la deuxième partie du message qui sera démasquée plus tard (00 sera remplacé par 20, code espace).
En sortie de boucle, X=0 indexe le début du menu MSG1 affiché par PRMSG. Ce message est farci de caractères de contrôle dont le premier, 0C (CTRL-L), efface l'écran.
Sous l'étiquette OPT, INCHR attend la frappe d'une touche, choix de l'option. Le code du caractère, recueilli dans A, est testé.
S'il est inférieur au code du chiffre 1, ou supérieur à celui de 4, erreur, Ping, retour menu.
S'il correspond au chiffre 4, retour Basic via RESET.
L'option choisie étant 1, 2 ou 3, il convient de demander l'entrée d'une adresse de début.

Le code de l'option retenue est sauvegardé dans TMP3.
TMP2 est chargé avec #3A. Cette valeur servira à l'indexation dans la table MSG1 pour l'affichage du mot 'Debut' assuré par la routine ADGET, recueil d'une adresse hexadécimale.
Au retour de ADGET, si Z=1 pas d'adresse enregistrée, retour menu.
Si Z=0 ...
la boucle INIPT inscrit l'adresse en A(-), Y(+)
dans les quatre pointeurs DEPAD, BASAD, CRSAD et FINAD.
Au passage, NCOL reçoit la valeur 08.

En sortie de boucle, X=0. LINCT, COLCT et TMP4 sont initialisés avec cette valeur. TMP4 servira de drapeau dans LLIST pour l'impression éventuelle d'une somme de contrôle.
L'adresse de début est affichée par STAT (status).
Le code de l'option choisie est récupéré dans TMP3.
S'il ne s'agissait pas de l'option 1, saut à GTFIN pour recueillir une adresse de fin. Si oui, Nouvelle entrée, NEWF, page suivante.

```
                        ;Nouvelle entrée

6066 20 CE CC NEWF   JSR CLS       6057 20 0A CC NEWF   JSR CLS
6069 20 10 64        JSR DECFA     605A 20 E6 63        JSR DECFA
606C 4C 88 61        JMP CHIN0     605D 4C 7A 61        JMP CHIN0


                        ;Adresse fin

606F 20 F0 CB GTFIN  JSR CRDO      6060 20 9F CB GTFIN  JSR CRDO
6072 A2 43    GTF0   LDX #$43      6063 A2 43    GTF0   LDX #$43
6074 86 77           STX TMP2      6065 86 77           STX TMP2
6076 20 24 64        JSR ADGET     6067 20 FA 63        JSR ADGET
6079 F0 A1           BEQ MENU      606A F0 A1           BEQ MENU
607B 85 66           STA FINAD     606C 85 66           STA FINAD
607D 84 67           STY FINAD+1   606E 84 67           STY FINAD+1
607F C5 60           CMP DEPAD     6070 C5 60           CMP DEPAD
6081 98              TYA           6072 98              TYA
6082 E5 61           SBC DEPAD+1   6073 E5 61           SBC DEPAD+1
6084 B0 06           BCS PRFIN     6075 B0 06           BCS PRFIN
6086 20 9F FA        JSR PING      6077 20 85 FA        JSR PING
6089 4C 72 60        JMP GTF0      607A 4C 63 60        JMP GTF0
608C 20 C5 64 PRFIN  JSR STAT+3    607D 20 9B 64 PRFIN  JSR STAT+3
608F A5 78           LDA TMP3      6080 A5 78           LDA TMP3
6091 C9 32           CMP #"2"      6082 C9 32           CMP #"2"
6093 D0 0C           BNE LPRTF     6084 D0 0C           BNE LPRTF


                        ;Edition

6095 20 CE CC EDITF  JSR CLS       6086 20 0A CC EDITF  JSR CLS
6098 20 24 61        JSR LIST      6089 20 16 61        JSR LIST
609B 20 9B 64        JSR RESCUR    608C 20 71 64        JSR RESCUR
609E 4C 88 61        JMP CHIN0     608F 4C 7A 61        JMP CHIN0


                        ;Impression

60A1 A2 1A    LPRTF  LDX #$1A      6092 A2 1A    LPRTF  LDX #$1A
60A3 BD CE 65 LPR0   LDA MSG3,X    6094 BD A4 65 LPR0   LDA MSG3,X
60A6 9D 8D BB        STA $BB8D,X   6097 9D 8D BB        STA $BB8D,X
60A9 CA              DEX           609A CA              DEX
60AA D0 F7           BNE LPR0      609B D0 F7           BNE LPR0


                        ;8 ou 16 col?

60AC A2 4C           LDX #$4C      609D A2 4C           LDX #$4C
60AE 20 E7 64        JSR PRMSG     609F 20 BD 64        JSR PRMSG
60B1 20 E8 C5        JSR INCHR     60A2 20 F8 C5        JSR INCHR
60B4 C9 38           CMP #"8"      60A5 C9 38           CMP #"8"
60B6 F0 02           BEQ LPR1      60A7 F0 02           BEQ LPR1
60B8 06 75           ASL NCOL      60A9 06 75           ASL NCOL


                        ;Somme de contrôle?

60BA A2 66    LPR1   LDX #$66      60AB A2 66    LPR1   LDX #$66
60BC 20 E7 64        JSR PRMSG     60AD 20 BD 64        JSR PRMSG
60BF 20 E8 C5        JSR INCHR     60B0 20 F8 C5        JSR INCHR
60C2 C9 4F           CMP #"O"      60B3 C9 4F           CMP #"O"
60C4 D0 02           BNE LLIST     60B5 D0 02           BNE LLIST
60C6 66 79           ROR TMP4      60B7 66 79           ROR TMP4
```

CLS vide l'écran. L'adresse de fin, FINAD, est ajustée; elle doit être égale à DEPAD - 1 (fichier vierge). Le JMP CHINO fait entrer dans la boucle de recueil des caractères tapés au clavier.

Recueil de l'adresse de fin. CRDO met à la ligne.
TMP2 reçoit l'index du mot 'Fin'.
Même procédure ADGET que pour l'adresse de début.
Retour menu si pas d'entrée.

L'adresse, recueillie en A(-), Y(+), est rangée dans FINAD.

Elle est comparée à l'adresse de début DEPAD.
Si elle lui est égale ou supérieure, branchement à PRTFIN et affichage

Sinon, Ping et retour à GTFO pour un nouvel essai.

Affichage de FINAD par STAT+3.
Le code de l'option retenue est récupéré.
S'il ne s'agissait pas de l'option 2, branchement à LPRTF, Impression
Si oui, mode édition, EDITF.

L'écran est effacé. LIST affiche le code dans les limites définies par DEPAD et FINAD. Si le bloc dépasse la capacité de l'écran, seule une première page de 25 lignes est affichée. Le curseur sauvegardé par LIST est restauré. C'est CHINO qui le mettra en place sur le premier octet.

Début de la section responsable de l'impression.
Les indications de la moitié droite de la ligne supérieure sont remplacées par MSG3, dernier message figurant en fin de programme.

La proposition du choix de l'impression sur 8 ou 16 col. est affichée par PRMSG.
INCHR recueille le code du caractère correspondant à ce choix.
Si le caractère tapé est le chiffre 8, NCOL conserve la valeur 08.
Toute autre touche le fait égal à 16 (valeur par défaut) par décalage arithmétique à gauche (multiplication par deux).

Le message suivant propose l'impression d'une somme de contrôle.
Si la réponse est différente de (O)ui, l'impression commence aussitôt.
Si c'est oui, l'égalité met à un l'indicateur de retenue C.
Cet indicateur devient le bit 7 de TMP4 par la rotation à droite ROR TMP4. Le drapeau TMP4 prend alors une valeur négative (#80).

```
                        ;Llist 48 lignes

60C8 20 16 C8 LLIST   JSR LPRTON      60B9 38       LLIST   SEC
60CB A9 0D            LDA #$0D        60BA 6E F1 02         ROR PRTFLG
60CD 20 D9 CC         JSR OUTDO       60BD A9 0D            LDA #$0D
60D0 20 40 61 LLST0   JSR LST2        60BF 20 12 CC         JSR OUTDO
60D3 08               PHP             60C2 20 32 61 LLST0   JSR LST2
60D4 24 79            BIT TMP4        60C5 08               PHP
60D6 10 20            BPL LLST3       60C6 24 79            BIT TMP4
60D8 20 D4 CC LLST1   JSR OUTSPC      60C8 10 20            BPL LLST3
60DB 20 D4 CC         JSR OUTSPC      60CA 20 0D CC LLST1   JSR OUTSPC
60DE 20 D4 CC         JSR OUTSPC      60CD 20 0D CC         JSR OUTSPC
60E1 A6 6F            LDX COLCT       60D0 20 0D CC         JSR OUTSPC
60E3 E4 75            CPX NCOL        60D3 A6 6F            LDX COLCT
60E5 F0 04            BEQ LLST2       60D5 E4 75            CPX NCOL
60E7 E6 6F            INC COLCT       60D7 F0 04            BEQ LLST2
60E9 D0 ED            BNE LLST1       60D9 E6 6F            INC COLCT
60EB 20 7F 61 LLST2   JSR LSL2        60DB D0 ED            BNE LLST1
60EE 20 89 63         JSR CHKSM       60DD 20 71 61 LLST2   JSR LSL2
60F1 A4 69            LDY TEMPA+1     60E0 20 5F 63         JSR CHKSM
60F3 A6 68            LDX TEMPA       60E3 A4 69            LDY TEMPA+1
60F5 20 09 65         JSR PRTYX       60E5 A6 68            LDX TEMPA
60F8 20 F0 CB LLST3   JSR CRDO        60E7 20 DF 64         JSR PRTYX
60FB 28               PLP             60EA 20 9F CB LLST3   JSR CRDO
60FC 90 20            BCC LLST5       60ED 28               PLP
60FE 20 CA 63         JSR INC2        60EE 90 20            BCC LLST5
6101 A5 75            LDA NCOL        60F0 20 A0 63         JSR INC2
6103 C9 08            CMP #$08        60F3 A5 75            LDA NCOL
6105 F0 03            BEQ LLST4       60F5 C9 08            CMP #$08
6107 20 CC 63         JSR INC3        60F7 F0 03            BEQ LLST4
610A 20 78 EB LLST4   JSR RDKEY       60F9 20 A2 63         JSR INC3
610D 30 0F            BMI LLST5       60FC 20 05 E9 LLST4   JSR RDKEY
610F A5 6E            LDA LINCT       60FF 30 0F            BMI LLST5
6111 C9 30            CMP #$30        6101 A5 6E            LDA LINCT
6113 D0 BB            BNE LLST0       6103 C9 30            CMP #$30
6115 20 E8 C5         JSR INCHR       6105 D0 BB            BNE LLST0
6118 A9 00            LDA #$00        6107 20 F8 C5         JSR INCHR
611A 85 6E            STA LINCT       610A A9 00            LDA #$00
611C F0 B2            BEQ LLST0       610C 85 6E            STA LINCT
611E 20 2F C8 LLST5   JSR LPRTOF      610E F0 B2            BEQ LLST0
6121 4C 1C 60         JMP MENU        6110 4E F1 02 LLST5   LSR PRTFLG
                                      6113 4C 0D 60         JMP MENU

                        ;List jusqu'à
                        ;25 lignes

6124 20 74 64 LIST    JSR SAVCUR      6116 20 4A 64 LIST    JSR SAVCUR
6127 A9 0D            LDA #$0D        6119 A9 0D            LDA #$0D
6129 20 D9 CC         JSR OUTDO       611B 20 12 CC         JSR OUTDO
612C 20 40 61 LST0    JSR LST2        611E 20 32 61 LST0    JSR LST2
612F 90 0E            BCC LST1        6121 90 0E            BCC LST1
6131 20 CA 63         JSR INC2        6123 20 A0 63         JSR INC2
6134 A5 6E            LDA LINCT       6126 A5 6E            LDA LINCT
6136 C9 19            CMP #$19        6128 C9 19            CMP #$19
6138 F0 05            BEQ LST1        612A F0 05            BEQ LST1
613A 20 F0 CB         JSR CRDO        612C 20 9F CB         JSR CRDO
613D D0 ED            BNE LST0        612F D0 ED            BNE LST0
613F 60         LST1  RTS             6131 60         LST1  RTS
```

Pour la sortie sur imprimante, le bit 7 du drapeau 2F1 est positionné à un par JSR LPRTON sur Atmos ou par SEC, ROR 2F1 sur Oric 1.
Après un retour en début de ligne
LLST0 marque le début d'une boucle dont l'élément principal est l'impression d'une ligne par LST2.
Suivant la valeur de NCOL une ligne de 8 ou 16 octets est imprimée ou éventuellement une dernière ligne incomplète.
Au retour de LST2, C=0 indique la fin du bloc de code.
Le registre d'état est empilé pour test ultérieur de cet indicateur.
Le drapeau TMP4 est testé par BIT TMP4, s'il est positif branchement en LLST3.
Si TMP4 est négatif, l'impression d'une somme de contrôle est exécutée après la sortie de trois espaces, ou plus s'il s'agit d'une dernière ligne incomplète (comparaison COLCT avec NCOL).

Par l'appel de LSL2, le contenu de TEMPB, adresse du dernier octet de la ligne, est transféré dans CRSAD pour calcul de la somme par CHKSM
Le résultat, chargé dans Y(+),X(-), est imprimé par PRTYX.

CRDO met à la ligne.
Le registre d'état est desempilé.
Si C=0, la ligne imprimée était la dernière. Sortie par LLST5.
Sinon, LINCT et BASAD sont ajustés par INC2.
S'agissant d'une impression sur 16 colonnes, BASAD est augmenté une nouvelle fois de 8 par INC3.

Le clavier est testé par RDKEY.
La frappe d'une touche interrompt l'impression, sortie par LLST5.
Le compteur de lignes est testé.
Si son contenu est différent de #30, (48 déci) retour début de boucle.
Si égal, 48 lignes imprimées, INCHR attend la frappe d'une touche.
Le compteur de lignes est remis à zéro.
Retour en début de boucle d'impression, branchement forcé.

Sortie de la routine d'impression. Le bit 7 de 2F1 est remis à zéro.
Retour au menu.

Cette routine est appelée de divers endroits. Elle affiche le code à partir des valeurs actuelles de CURBAS, BASAD et LINCT, c'est-à-dire à partir du début de n'importe quelle ligne. Elle n'efface pas l'écran.
Le code présent à l'écran est modifié par surimpression.
Après sauvegarde de la position du curseur et retour en début de ligne, une boucle est entamée en LST0 par appel de LST2, listage d'une ligne.
Au retour, C=0 fin du bloc → sortie. C=1 → incrémentation pointeurs
Si le compteur de lignes LINCT est égal à #19, 25 lignes sont affichées, l'écran est plein, sortie.
Sinon, CRDO fait passer la position d'écriture sur la ligne suivante puis branchement forcé en début de boucle.
Sortie. Retour à l'appelant.

```
                        ;List 1 ligne

6140 20 F3 64 LST2   JSR PRADR      6132 20 C9 64 LST2   JSR PRADR
6143 86 6A           STX TEMPB      6135 86 6A           STX TEMPB
6145 84 6B           STY TEMPB+1    6137 84 6B           STY TEMPB+1
6147 A0 00           LDY #$00       6139 A0 00           LDY #$00
6149 A5 66     LST3  LDA FINAD      613B A5 66     LST3  LDA FINAD
614B C5 6A           CMP TEMPB      613D C5 6A           CMP TEMPB
614D A5 67           LDA FINAD+1    613F A5 67           LDA FINAD+1
614F E5 6B           SBC TEMPB+1    6141 E5 6B           SBC TEMPB+1
6151 90 16           BCC LST5       6143 90 16           BCC LST5
6153 B1 6A           LDA (TEMPB),Y  6145 B1 6A           LDA (TEMPB),Y
6155 20 0E 65        JSR PRBYT      6147 20 E4 64        JSR PRBYT
6158 20 D4 CC        JSR OUTSPC     614A 20 0D CC        JSR OUTSPC
615B E6 6A           INC TEMPB      614D E6 6A           INC TEMPB
615D D0 02           BNE LST4       614F D0 02           BNE LST4
615F E6 6B           INC TEMPB+1    6151 E6 6B           INC TEMPB+1
6161 E6 6F     LST4  INC COLCT      6153 E6 6F     LST4  INC COLCT
6163 A5 6F           LDA COLCT      6155 A5 6F           LDA COLCT
6165 C5 75           CMP NCOL       6157 C5 75           CMP NCOL
6167 D0 E0           BNE LST3       6159 D0 E0           BNE LST3
6169 60        LST5  RTS            615B 60        LST5  RTS

                        ;List 1 ligne
                        ;après scrolling

616A 20 74 64 LSTLN  JSR SAVCUR     615C 20 4A 64 LSTLN  JSR SAVCUR
616D A9 0D     LSL0  LDA #$0D       615F A9 0D     LSL0  LDA #$0D
616F 20 D9 CC        JSR OUTDO      6161 20 12 CC        JSR OUTDO
6172 20 40 61        JSR LST2       6164 20 32 61        JSR LST2
6175 90 03           BCC LSL1       6167 90 03           BCC LSL1
6177 4C 9B 64        JMP RESCUR     6169 4C 71 64        JMP RESCUR
617A A9 11     LSL1  LDA #$11       616C A9 11     LSL1  LDA #$11
617C 20 D9 CC        JSR OUTDO      616E 20 12 CC        JSR OUTDO
617F A5 6A     LSL2  LDA TEMPB      6171 A5 6A     LSL2  LDA TEMPB
6181 A4 6B           LDY TEMPB+1    6173 A4 6B           LDY TEMPB+1
6183 85 64           STA CRSAD      6175 85 64           STA CRSAD
6185 84 65           STY CRSAD+1    6177 84 65           STY CRSAD+1
6187 60              RTS            6179 60              RTS


                        ;Entrée caractère

                        ;-démasque cmdes

6188 A9 20     CHIN0 LDA #$20       617A A9 20     CHIN0 LDA #$20
618A 8D 8E BB        STA $BB8E      617C 8D 8E BB        STA $BB8E

                        ;-affiche adr.base

618D 20 F3 64 CHIN1  JSR PRADR      617F 20 C9 64 CHIN1  JSR PRADR

                        ;-test insér

6190 AD A1 BB CHIN2  LDA $BBA1      6182 AD A1 BB CHIN2  LDA $BBA1
6193 29 F0           AND #$F0       6185 29 F0           AND #$F0
6195 D0 03           BNE CHIN3      6187 D0 03           BNE CHIN3
6197 20 29 63        JSR INSER      6189 20 FF 62        JSR INSER
```

Cette routine affiche ou imprime une ligne de codes hexadécimaux précédée de l'adresse du premier octet de la ligne.
A l'entrée, cette adresse de base est affichée par PRADR puis transférée dans TMPB, variable de contrôle de la boucle d'affichage.
Y=0 pour l'instruction d'adressage indirect indexé LDA (TMPB),Y.
La boucle d'affichage débute en LST3.
Les pointeurs FINAD, adresse de fin de bloc, et TMPB, qui pointe alors l'octet suivant celui qui vient d'être affiché, sont comparés.
Si FINAD < TMPB, C = 0, le dernier octet a été affiché, sortie.
Sinon l'octet pointé par TMPB, chargé dans A, est traité par PRTBYT, routine qui affiche en hexadécimal le contenu de l'accumulateur
La routine OUTSPC sort un espace.
Le pointeur TEMPB est incrémenté.
Il contient maintenant l'adresse de l'octet suivant.

Le compteur de colonnes COLCT est incrémenté.
Son contenu est chargé dans A ...
et comparé à NCOL (8 en mode affichage, 8 ou 16 en mode impression).
S'il n'y a pas égalité, retour en début de boucle LST3. Si oui ...
Sortie de la routine.

Après défilement de l'écran (scrolling), consécutif à un déplacement du curseur par les flèches haut ou bas, affichage de la ligne adéquate en haut ou en bas de l'écran. La position du curseur est sauvegardée et aprés un retour en début de ligne, la ligne est affichée par LST2.
Si la ligne est complète (dans tous les cas pour une ligne supérieure) sortie de la routine via RESCUR (restauration du curseur).
Si la ligne est incomplète (fin de bloc), l'ancienne position du curseur est ignorée. Il se place après le dernier code affiché.
L'emplacement en mémoire correspondant étant pointé par TMPB, le contenu de ce pointeur est transféré dans CRSAD avant sortie.

* * *

Cette section, ATMOS 6188-624A, ORIC 1 617A-623C, est la boucle d'entrée des caractères au clavier. C'est le coeur du programme dans sa fonction principale d'édition (options 1 et 2).

Le 00 inclus dans le message à la ligne supérieure est remplacé par #20, code espace, demasquant la partie droite, résumé des commandes.

PRADR affiche une adresse en début de ligne. A l'entrée dans la boucle en option édition, cette adresse (1ère ligne) est déjà affichée, PRADR positionne le curseur sur le premier code.

Le contenu de #BBA1, chargé dans l'accumulateur, est testé par AND F0
Normalement #2C, code de la virgule précédant CTRL-Z, ligne supérieure
En mode insertion, il est remplacé par #0C, attribut clignotement.
Dans ce cas, la routine insertion est exécutée avant l'entrée du code.

```
                        ;-compteur 2 carac

619A A9 00     CHIN3    LDA #$00       618C A9 00     CHIN3    LDA #$00
619C 85 68              STA TEMPA      618E 85 68              STA TEMPA
619E 85 69              STA TEMPA+1    6190 85 69              STA TEMPA+1
61A0 A9 02              LDA #$02       6192 A9 02              LDA #$02
61A2 85 70              STA CHRCT      6194 85 70              STA CHRCT

                        ;-lecture clavier
                        ;et test caractère

61A4 20 E8 C5  CHR0     JSR INCHR      6196 20 F8 C5  CHR0     JSR INCHR
61A7 A0 02              LDY #$02       6199 A0 02              LDY #$02
61A9 C4 70              CPY CHRCT      619B C4 70              CPY CHRCT
61AB D0 65              BNE CHEX       619D D0 65              BNE CHEX
61AD AC A1 BB           LDY $BBA1      619F AC A1 BB           LDY $BBA1
61B0 C0 0C              CPY #$0C       61A2 C0 0C              CPY #$0C
61B2 F0 4F              BEQ CHR7       61A4 F0 4F              BEQ CHR7
61B4 C9 1B              CMP #$1B       61A6 C9 1B              CMP #$1B
61B6 D0 03              BNE CHR1       61A8 D0 03              BNE CHR1
61B8 4C 1C 60           JMP MENU       61AA 4C 0D 60           JMP MENU
61BB C9 2B     CHR1     CMP #$2B       61AD C9 2B     CHR1     CMP #$2B
61BD D0 09              BNE CHR2       61AF D0 09              BNE CHR2
61BF 20 89 63           JSR CHKSM      61B1 20 5F 63           JSR CHKSM
61C2 20 C8 64           JSR STAT+6     61B4 20 9E 64           JSR STAT+6
61C5 4C 9A 61           JMP CHIN3      61B7 4C 8C 61           JMP CHIN3
61C8 C9 08     CHR2     CMP #$08       61BA C9 08     CHR2     CMP #$08
61CA D0 06              BNE CHR3       61BC D0 06              BNE CHR3
61CC 20 4B 62           JSR LEFT       61BE 20 3D 62           JSR LEFT
61CF 4C A4 61           JMP CHR0       61C1 4C 96 61           JMP CHR0
61D2 C9 0B     CHR3     CMP #$0B       61C4 C9 0B     CHR3     CMP #$0B
61D4 D0 06              BNE CHR4       61C6 D0 06              BNE CHR4
61D6 20 6F 62           JSR UP         61C8 20 61 62           JSR UP
61D9 4C A4 61           JMP CHR0       61CB 4C 96 61           JMP CHR0
61DC 48        CHR4     PHA            61CE 48        CHR4     PHA
61DD 20 20 63           JSR FCHK       61CF 20 F6 62           JSR FCHK
61E0 68                 PLA            61D2 68                 PLA
61E1 90 2F              BCC CHEX       61D3 90 2F              BCC CHEX
61E3 C9 09              CMP #$09       61D5 C9 09              CMP #$09
61E5 D0 06              BNE CHR5       61D7 D0 06              BNE CHR5
61E7 20 AF 62           JSR RIGHT      61D9 20 85 62           JSR RIGHT
61EA 4C A4 61           JMP CHR0       61DC 4C 96 61           JMP CHR0
61ED C9 0A     CHR5     CMP #$0A       61DF C9 0A     CHR5     CMP #$0A
61EF D0 06              BNE CHR6       61E1 D0 06              BNE CHR6
61F1 20 DB 62           JSR DOWN       61E3 20 B1 62           JSR DOWN
61F4 4C A4 61           JMP CHR0       61E6 4C 96 61           JMP CHR0
61F7 C9 1A     CHR6     CMP #$1A       61E9 C9 1A     CHR6     CMP #$1A
61F9 D0 08              BNE CHR7       61EB D0 08              BNE CHR7
61FB A9 0C              LDA #$0C       61ED A9 0C              LDA #$0C
61FD 8D A1 BB           STA $BBA1      61EF 8D A1 BB           STA $BBA1
6200 4C 90 61           JMP CHIN2      61F2 4C 82 61           JMP CHIN2
6203 C9 7F     CHR7     CMP #$7F       61F5 C9 7F     CHR7     CMP #$7F
6205 D0 0B              BNE CHEX       61F7 D0 0B              BNE CHEX
6207 A9 2C              LDA #$2C       61F9 A9 2C              LDA #$2C
6209 8D A1 BB           STA $BBA1      61FB 8D A1 BB           STA $BBA1
620C 20 59 63           JSR DELET      61FE 20 2F 63           JSR DELET
620F 4C 9A 61           JMP CHIN3      6201 4C 8C 61           JMP CHIN3
```

Le pointeur TMPA, deux octets, est initialisé à zéro. Il servira à la conversion des chiffres hexadécimaux en une valeur binaire sur un octet par la routine VALHX. Après l'entrée des deux digits, l'octet faible TMPA contient cette valeur qui sera inscrite en mémoire.
CHRCT qui compte les caractères hexa entrés est initialisé.

- CHR0 le caractère tapé est recueilli dans l'accumulateur par INCHR
  Si le contenu de CHRCT est différent de 2, un seul chiffre hexa enregistré, le caractère ne peut être qu'un deuxième chiffre hexadécimal, branchement en CHEX.
  Si CHRCT = 2, test #BBA1
  Si (#BBA1) = #0C, on se trouve en mode insertion, seule l'entrée de DEL ou d'un chiffre hexa est autorisée, branchement en CHR7.
- Code du caractère en A est-il #1B, ESC ?
  Non, essayer autre chose, → CHR1.
  Oui, retour menu.
- CHR1 - Est-ce #2B, code du signe + ?
  Non, essayer autre chose, → CHR2.
  Oui, calculer la somme de contrôle ...
  et l'afficher en haut de l'écran ...
  Retour en CHIN3 pour nouveau caractère, TMPA doit être réinitialisé
- CHR2 - Est-ce #08, code du recul curseur, flèche gauche ?
  Non, essayer autre chose, → CHR3.
  Oui, exécuter routine LEFT ...
  puis retour en CHR0 pour nouveau caractère.
- CHR3 - Est-ce #0B, remontée curseur, flèche haut ?
  Non, essayer autre chose, → CHR4.
  Oui, exécuter routine UP ...
  puis retour en CHR0 pour nouveau caractère.
- CHR4 - le code est sauvegardé en pile.
  Le test FCHK vérifie la position du curseur.
  Le code est désempilé.
  Doit être caractère hexa si curseur en fin de bloc, → CHEX.
  Sinon, Est-ce #09, avancée curseur, flèche droite ?
  Non, essayer autre chose, → CHR5.
  Oui, exécuter routine RIGHT ...
  puis retour en CHR0 pour nouveau caractère.
- CHR5 - Est-ce #0A, descente curseur, flèche bas ?
  Non, essayer autre chose, → CHR6.
  Oui, exécuter routine DOWN ...
  puis retour en CHR0 pour nouveau caractère.
- CHR6 - Est-ce #1A, CTRL-Z, mise en mode insertion ?
  Non, essayer autre chose, → CHR7.
  Oui, l'attribut #0C est placé ...
  en #BBA1 qui servira de drapeau puis ...
  retour en CHIN2 pour exécution INSER avant nouveau caractère.
- CHR7 - Est-ce #7F, code de DEL, suppression ?
  Non, doit être caractère hexadécimal, → CHEX.
  Oui, le drapeau insertion est baissé par ...
  remise en place du code de la virgule en #BBA1 puis ...
  la routine DEL est exécutée puis ...
  retour en CHIN3 (réinitialisation TMPA), nouveau caractère.

```
                        ;-doit être hexa

6212 20 4F 64 CHEX   JSR VALHX      6204 20 25 64 CHEX   JSR VALHX
6215 B0 06           BCS CHX1       6207 B0 06           BCS CHX1
6217 20 9F FA        JSR PING       6209 20 85 FA        JSR PING
621A 4C A4 61        JMP CHR0       620C 4C 96 61        JMP CHR0
621D 20 D9 CC CHX1   JSR OUTDO      620F 20 12 CC CHX1   JSR OUTDO
6220 C6 70           DEC CHRCT      6212 C6 70           DEC CHRCT
6222 F0 03           BEQ CHX2       6214 F0 03           BEQ CHX2
6224 4C A4 61        JMP CHR0       6216 4C 96 61        JMP CHR0

                        ;-2 car.hexa entrés

6227 A0 00     CHX2  LDY #$00       6219 A0 00     CHX2  LDY #$00
6229 A5 68           LDA TEMPA      621B A5 68           LDA TEMPA
622B 91 64           STA (CRSAD),Y  621D 91 64           STA (CRSAD),Y
622D 20 D4 CC        JSR OUTSPC     621F 20 0D CC        JSR OUTSPC
6230 E6 6F           INC COLCT      6222 E6 6F           INC COLCT
6232 20 20 63        JSR FCHK       6224 20 F6 62        JSR FCHK
6235 B0 03           BCS CHX3       6227 B0 03           BCS CHX3
6237 20 D8 63        JSR INCFA      6229 20 AE 63        JSR INCFA
623A 20 B6 63 CHX3   JSR INCPT      622C 20 8C 63 CHX3   JSR INCPT
623D F0 03           BEQ CHX4       622F F0 03           BEQ CHX4
623F 4C 90 61        JMP CHIN2      6231 4C 82 61        JMP CHIN2
6242 20 F0 CB CHX4   JSR CRDO       6234 20 9F CB CHX4   JSR CRDO
6245 20 C4 62        JSR RGT1       6237 20 9A 62        JSR RGT1
6248 4C 8D 61 CHX5   JMP CHIN1      623A 4C 7F 61 CHX5   JMP CHIN1


                        ;Routine
                        ;flèche gauche

624B A5 6F     LEFT  LDA COLCT      623D A5 6F     LEFT  LDA COLCT
624D D0 05           BNE LFT0       623F D0 05           BNE LFT0
624F 20 17 63        JSR DCHK       6241 20 ED 62        JSR DCHK
6252 F0 18           BEQ LFT2       6244 F0 18           BEQ LFT2
6254 C6 6F     LFT0  DEC COLCT      6246 C6 6F     LFT0  DEC COLCT
6256 A9 08           LDA #$08       6248 A9 08           LDA #$08
6258 A2 03           LDX #$03       624A A2 03           LDX #$03
625A A4 6F           LDY COLCT      624C A4 6F           LDY COLCT
625C 10 02           BPL LFT1       624E 10 02           BPL LFT1
625E A2 11           LDX #$11       6250 A2 13           LDX #$13
6260 20 D9 CC LFT1   JSR OUTDO      6252 20 12 CC LFT1   JSR OUTDO
6263 CA              DEX            6255 CA              DEX
6264 D0 FA           BNE LFT1       6256 D0 F-           BNE LFT1
6266 20 EC 63        JSR DECPT      6258 20 C2 63        JSR DECPT
6269 4C B7 62        JMP UP0        625B 4C 79 62        JMP UP0
626C 4C 9F FA LFT2   JMP PING       625E 4C 85 FA LFT2   JMP PING
```

Après avoir subi les tests précédents, le caractère entré au clavier doit correspondre à un chiffre hexadécimal de 0 à F. VALHX le vérifie.
Au retour de cette routine si C=0, caractère non hexa, Ping, puis ... retour en CHR0 pour nouveau caractère.
Si C=1, le caractère hexa, reconnu correct, est affiché par OUTDO puis le compteur de caractères hexa valides, CHRCT, est décrémenté; ... si CHRCT=0, deux chiffres entrés, branchement en CHX2 pour inscription en mémoire, sinon retour en CHR0 pour l'entrée du deuxième chiffre.

Deux chiffres hexa ayant été entrés, la valeur binaire correspondante, élaborée en TMPA par VALHX, est inscrite à l'adresse pointée par CRSAD emplacement matérialisé à l'écran par la position du curseur puis ...
OUTSPC affiche un espace ...
le compteur de colonnes est incrémenté ...
FCHK recherche la position du curseur par rapport à FINAD.
Si FINAD <= CRSAD, C=1, on se trouve à l'intérieur du bloc → CHX3.
Sinon, il s'agit d'un ajout, FINAD est incrémenté par INCFA.
CHX3, les pointeurs sont incrémentés par INCPT; au retour ...
- si Z=0 la fin de ligne n'est pas atteinte, retour en CHIN2 ... pour test INSER, réintialisation variables et nouveau caractère.
- si Z=1, fin de ligne, CRDO met à la ligne, appel de RGT2 pour test LINCT=25 (défilement vers le haut) et affichage éventuel d'une dernière ligne. Retour en CHIN1

* * *

Si le curseur n'est pas en première colonne ...
recul curseur autorisé → LFT0.
S'il y est, test début du bloc par DCHK ...
si début du bloc, → LFT2, sortie de la routine via Ping.
LFT0, le compteur de colonnes est décrémenté ...
le code du caractère recul curseur, 08, est rechargé dans A ...
puis 'affiché' 3 fois par OUTDO, ou 17 fois si COLCT < 0 ...
Il s'agit alors de revenir sur le dernier octet de la ligne précédente
Notez que sur Oric 1, dans ce dernier cas, il faut 'sauter' aussi les deux colonnes de gauche de l'écran, dites protégées (le curseur doit reculer de #13 positions au lieu de #11).

Après déplacement du curseur, les pointeurs sont décrémentés par DECPT sortie par UP0 (voir plus loin) pour un éventuel scrolling vers le bas.

```
                        ;Routine
                        ;flèche haut

626F 20 17 63 UP        JSR DCHK
6272 F0 F8              BEQ LFT2
6274 20 02 64           JSR DEC2
6277 A9 0B              LDA #$0B
6279 20 D9 CC           JSR OUTDO
627C 38                 SEC
627D A5 64              LDA CRSAD
627F E9 08              SBC #$08
6281 85 64              STA CRSAD
6283 B0 02              BCS UP0
6285 C6 65              DEC CRSAD+1
6287 A5 6E      UP0     LDA LINCT
6289 10 23              BPL UP1
628B 20 74 64 SCRLDN    JSR SAVCUR
628E A9 BF              LDA #$BF
6290 85 CA              STA HIGHTR+1
6292 85 C8              STA HIGHDS+1
6294 A9 90              LDA #$90
6296 85 C9              STA HIGHTR
6298 A9 BB              LDA #$B8
629A 85 C7              STA HIGHDS
629C A9 D0              LDA #$D0
629E 85 CE              STA LOWTR
62A0 A9 BB              LDA #$BB
62A2 85 CF              STA LOWTR+1
62A4 20 FB C3           JSR MOVUP
62A7 20 6D 61           JSR LSL0
62AA A9 00              LDA #$00
62AC 85 6E              STA LINCT
62AE 60         UP1     RTS
```

```
6261 20 ED 62 UP        JSR DCHK
6264 F0 F8              BEQ LFT2
6266 20 D8 63           JSR DEC2
6269 A9 0B              LDA #$0B
626B 20 12 CC           JSR OUTDO
626E 38                 SEC
626F A5 64              LDA CRSAD
6271 E9 08              SBC #$08
6273 85 64              STA CRSAD
6275 B0 02              BCS UP0
6277 C6 65              DEC CRSAD+1
6279 A5 6E      UP0     LDA LINCT
627B 10 07              BPL UP1
627D 20 5C 61           JSR LSTLN
6280 A9 00              LDA #$00
6282 85 6E              STA LINCT
6284 60         UP1     RTS
```

```
                        ;Routine
                        ;flèche droite

62AF E6 6F      RIGHT   INC COLCT
62B1 A2 03              LDX #$3
62B3 A4 6F              LDY COLCT
62B5 C0 08              CPY #$8
62B7 D0 02              BNE RGT0
62B9 A2 11              LDX #$11
62BB 20 D9 CC RGT0      JSR OUTDO
62BE CA                 DEX
62BF D0 FA              BNE RGT0
62C1 20 B6 63           JSR INCPT
62C4 A5 6E      RGT1    LDA LINCT
62C6 C9 19              CMP #$19
62C8 D0 10              BNE RGT3
62CA C6 6E              DEC LINCT
62CC 20 74 64 RGT2      JSR SAVCUR
62CF A9 0D              LDA #$0D
62D1 20 D9 CC           JSR OUTDO
62D4 20 40 61           JSR LST2
62D7 4C 9B 64           JMP RESCUR
62DA 60         RGT3    RTS
```

```
6285 E6 6F      RIGHT   INC COLCT
6287 A2 03              LDX #$3
6289 A4 6F              LDY COLCT
628B C0 08              CPY #$8
628D D0 02              BNE RGT0
628F A2 13              LDX #$13
6291 20 12 CC RGT0      JSR OUTDO
6294 CA                 DEX
6295 D0 FA              BNE RGT0
6297 20 8C 63           JSR INCPT
629A A5 6E      RGT1    LDA LINCT
629C C9 19              CMP #$19
629E D0 10              BNE RGT3
62A0 C6 6E              DEC LINCT
62A2 20 4A 64 RGT2      JSR SAVCUR
62A5 A9 0D              LDA #$0D
62A7 20 12 CC           JSR OUTDO
62AA 20 32 61           JSR LST2
62AD 4C 71 64           JMP RESCUR
62B0 60         RGT3    RTS
```

DCHK teste si le curseur se trouve sur la première ligne du bloc.
Si BASAD = DEPAD, c'est le cas, sortie de la routine via Ping en LFT2
Sinon, déplacement vers le haut possible, DEC2 ajuste BASAD et LINCT.
Le code #0B, remontée curseur, est rechargé dans A, puis ...
'affiché' par OUTDO.
Le contenu du pointeur CRSAD ...
qui détient l'adresse ...
matérialisée à l'écran par ...
la position du curseur ...
est diminuée de 8.

Si LINCT > 0, le curseur n'était pas sur la ligne supérieure, sortie.
SI LINCT < 0, le défilement vers le bas est nécessaire.
- Sur Oric 1, ce 'scrolling' est automatique. Il suffit d'appeler LSTLN et de remettre LINCT à zéro. La routine LSTLN sauvegarde le curseur, affiche une nouvelle ligne supérieure qui se substitue à la précédente, puis restaure le curseur.
- Sur Atmos, le scrolling vers le bas de l'écran est assuré par MOVUP
Voici les trois pointeurs utilisés et les paramètres correspondants:
. LOWTR, #CE,CF, plus basse adresse à déplacer, soit #BBD0.
. HIGHTR, #C9,CA, plus haute adresse à déplacer + 1, soit #BF90.
. HIGHDS, #C7,C8, destination + 1 de la plus haute adresse à déplacer soit #BFB8.

SAVCUR, appelée avant ce déplacement en mémoire d'écran, supprime l'image du curseur et évite son transfert intempestif sur la 2e ligne.
Une nouvelle ligne supérieure est substituée à la précédente (entrée dans LSTLN en LSL0), le curseur est réaffiché par RESCUR à la position préalablement sauvegardée.
LINCT est remis à zéro avant sortie.

L'avancée du curseur est autorisée car le test FCHK a déjà été effectué (61DD Atmos, 61CF Oric-1).
Le curseur passe sur l'octet suivant. COLCT est incrémenté.
Le code #09, avancée curseur, est 'affiché' normalement trois fois.
17 ou 19 fois (Oric 1) s'il doit passer à la ligne suivante
(voir routine flèche gauhe).

Incrémentation des pointeurs par INCPT.
Si LINCT <> 25, sortie normale.
Si LINCT = 25, le défilement vers le haut a eu lieu.
LINCT est réajusté, le curseur est sauvegardé pour l'affichage de la ligne suivante au bas de l'écran par LST2 puis le curseur est remis en place.

Remarque: Les routines de déplacement du curseur, LEFT, UP, RIGHT et DOWN, ne modifient pas le code objet en mémoire. Elles agissent seulement sur sa visualisation à l'écran.

```
                        ;Routine
                        ;flèche bas

62DB 20 CA 63 DOWN   JSR INC2        62B1 20 A0 63 DOWN   JSR INC2
62DE 38              SEC             62B4 38              SEC
62DF A5 66           LDA FINAD       62B5 A5 66           LDA FINAD
62E1 E5 62           SBC BASAD       62B7 E5 62           SBC BASAD
62E3 AA              TAX             62B9 AA              TAX
62E4 A5 67           LDA FINAD+1     62BA A5 67           LDA FINAD+1
62E6 E5 63           SBC BASAD+1     62BC E5 63           SBC BASAD+1
62E8 A8              TAY             62BE A8              TAY
62E9 B0 0C           BCS DWN1        62BF B0 0C           BCS DWN1
62EB E8              INX             62C1 E8              INX
62EC D0 03           BNE DWN0        62C2 D0 03           BNE DWN0
62EE C8              INY             62C4 C8              INY
62EF F0 06           BEQ DWN1        62C5 F0 06           BEQ DWN1
62F1 20 02 64 DWN0   JSR DEC2        62C7 20 D8 63 DWN0   JSR DEC2
62F4 4C 6A 61        JMP LSTLN       62CA 4C 5C 61        JMP LSTLN
62F7 A9 0A     DWN1  LDA #$0A        62CD A9 0A     DWN1  LDA #$0A
62F9 20 D9 CC        JSR OUTDO       62CF 20 12 CC        JSR OUTDO
62FC 18              CLC             62D2 18              CLC
62FD A5 64           LDA CRSAD       62D3 A5 64           LDA CRSAD
62FF 69 08           ADC #$08        62D5 69 08           ADC #$08
6301 85 64           STA CRSAD       62D7 85 64           STA CRSAD
6303 90 02           BCC DWN2        62D9 90 02           BCC DWN2
6305 E6 65           INC CRSAD+1     62DB E6 65           INC CRSAD+1
6307 A5 6E     DWN2  LDA LINCT       62DD A5 6E     DWN2  LDA LINCT
6309 C9 19           CMP #$19        62DF C9 19           CMP #$19
630B D0 02           BNE DWN3        62E1 D0 02           BNE DWN3
630D C6 6E           DEC LINCT       62E3 C6 6E           DEC LINCT
630F 20 20 63 DWN3   JSR FCHK        62E5 20 F6 62 DWN3   JSR FCHK
6312 B0 B8           BCS RGT2        62E8 B0 B8           BCS RGT2
6314 4C 6A 61        JMP LSTLN       62EA 4C 5C 61        JMP LSTLN

                        ;Test début

6317 A5 60     DCHK  LDA DEPAD       62ED A5 60     DCHK  LDA DEPAD
6319 C5 62           CMP BASAD       62EF C5 62           CMP BASAD
631B A5 61           LDA DEPAD+1     62F1 A5 61           LDA DEPAD+1
631D E5 63           SBC BASAD+1     62F3 E5 63           SBC BASAD+1
631F 60              RTS             62F5 60              RTS

                        ;Test fin

6320 A5 66     FCHK  LDA FINAD       62F6 A5 66     FCHK  LDA FINAD
6322 C5 64           CMP CRSAD       62F8 C5 64           CMP CRSAD
6324 A5 67           LDA FINAD+1     62FA A5 67           LDA FINAD+1
6326 E5 65           SBC CRSAD+1     62FC E5 65           SBC CRSAD+1
6328 60              RTS             62FE 60              RTS
```

LINCT est incrémenté et BASAD est augmenté de 8 par INC2.

La soustraction FINAD - nouveau BASAD est effectuée ...
au passage, les octets faible et fort du résultat sont transférés
respectivement dans X et Y.
Après l'opération
 - si C=1, FINAD >= BASAD, descente curseur possible → DWN1.

Le résultat de la soustraction précédente, en X(-),Y(+),
est augmenté de un.
S'il était égal à FFFF (BASAD=FINAD+1), la descente curseur est
autorisée.

Sinon, les pointeurs reprennent leur ancienne valeur et
sortie par LSTLN qui place le curseur en fin de bloc.
Le code #$0A, descente curseur est rechargé dans l'accumulateur
et 'affiché' par OUTDO.

Le pointeur d'adresse CRSAD ...
correspondant à la position du curseur ...
est mis à jour (augmenté de 8).

Le compteur de lignes, LINCT, est testé
et réajusté éventuellement.

Dans tous les cas, la ligne est réaffichée.

Selon sa position par rapport à la fin du bloc (test FCHK) ...
le curseur est remis en place (sortie par RGT2)
ou positionné à la fin du bloc par LSTLN.

Comparaison de DEPAD, début du bloc de code, avec BASAD, adresse du
premier octet de la ligne sur laquelle se trouve le curseur.
Utilisée par LEFT et UP, routines flèches gauche et haut.
Note: Au retour BEQ peut être employé car on a toujours BASAD>=DEPAD
      Evidemment, BCS conviendrait aussi dans UP, flèche haut.

Comparaison de FINAD, fin du bloc de code, avec CRSAD, adresse dans le
bloc de code matérialisée par la position du curseur.
A la sortie, si C=0 (FINAD<CRSAD), le curseur est en fin de bloc.

```
                        ;Routine insertion

6329 A5 64     INSER  LDA CRSAD       62FF A5 64     INSER  LDA CRSAD
632B A4 65            LDY CRSAD+1     6301 A4 65            LDY CRSAD+1
632D 85 CE            STA LOWTR       6303 85 CE            STA LOWTR
632F 84 CF            STY LOWTR+1     6305 84 CF            STY LOWTR+1
6331 20 D8 63         JSR INCFA       6307 20 AE 63         JSR INCFA
6334 18               CLC             630A 18               CLC
6335 A5 66            LDA FINAD       630B A5 66            LDA FINAD
6337 85 C9            STA HIGHTR      630D 85 C9            STA HIGHTR
6339 69 01            ADC #$01        630F 69 01            ADC #$01
633B 85 C7            STA HIGHDS      6311 85 C7            STA HIGHDS
633D A5 67            LDA FINAD+1     6313 A5 67            LDA FINAD+1
633F 85 CA            STA HIGHTR+1    6315 85 CA            STA HIGHTR+1
6341 69 00            ADC #$00        6317 69 00            ADC #$00
6343 85 C8            STA HIGHDS+1    6319 85 C8            STA HIGHDS+1
6345 20 FB C3         JSR MOVUP       631B 20 FF C3         JSR MOVUP
6348 20 24 61         JSR LIST        631E 20 16 61         JSR LIST
634B 20 9B 64         JSR RESCUR      6321 20 71 64         JSR RESCUR
634E AC 69 02         LDY CURCOL      6324 AC 69 02         LDY CURCOL
6351 A9 20            LDA #$20        6327 A9 20            LDA #$20
6353 91 12            STA (CURBAS),Y  6329 91 12            STA (CURBAS),Y
6355 C8               INY             632B C8               INY
6356 91 12            STA (CURBAS),Y  632C 91 12            STA (CURBAS),Y
6358 60               RTS             632E 60               RTS

                        ;Routine
                        ;suppression

6359 A5 64     DELET  LDA CRSAD       632F A5 64     DELET  LDA CRSAD
635B A4 65            LDY CRSAD+1     6331 A4 65            LDY CRSAD+1
635D 85 6A            STA TEMPB       6333 85 6A            STA TEMPB
635F 84 6B            STY TEMPB+1     6335 84 6B            STY TEMPB+1
6361 A0 01     DLT0   LDY #$01        6337 A0 01     DLT0   LDY #$01
6363 B1 6A            LDA (TEMPB),Y   6339 B1 6A            LDA (TEMPB),Y
6365 88               DEY             633B 88               DEY
6366 91 6A            STA (TEMPB),Y   633C 91 6A            STA (TEMPB),Y
6368 E6 6A            INC TEMPB       633E E6 6A            INC TEMPB
636A D0 02            BNE DLT1        6340 D0 02            BNE DLT1
636C E6 6B            INC TEMPB+1     6342 E6 6B            INC TEMPB+1
636E A5 6A     DLT1   LDA TEMPB       6344 A5 6A     DLT1   LDA TEMPB
6370 C5 66            CMP FINAD       6346 C5 66            CMP FINAD
6372 A5 6B            LDA TEMPB+1     6348 A5 6B            LDA TEMPB+1
6374 E5 67            SBC FINAD+1     634A E5 67            SBC FINAD+1
6376 90 E9            BCC DLT0        634C 90 E9            BCC DLT0
6378 20 10 64         JSR DECFA       634E 20 E6 63         JSR DECFA
637B 20 24 61         JSR LIST        6351 20 16 61         JSR LIST
637E B0 06            BCS DLT2        6354 B0 06            BCS DLT2
6380 20 D4 CC         JSR OUTSPC      6356 20 0D CC         JSR OUTSPC
6383 20 D4 CC         JSR OUTSPC      6359 20 0D CC         JSR OUTSPC
6386 4C 9B 64  DLT2   JMP RESCUR      635C 4C 71 64  DLT2   JMP RESCUR
```

Cette routine décale d'une position vers le haut de la mémoire le code compris entre les adresses pointées par CRSAD et FINAD.
Ce transfert est assuré par MOVUP.
L'adresse de fin, FINAD, est incrémentée puis affichée par DECFA.

Les pointeurs utilisés par MOVUP et les paramètres correspondants sont
- LOWTR, #CE,CF, plus basse adresse à déplacer, CRSAD.
- HIGHTR, #C9,CA, plus haute adresse à déplacer + 1 FINAD+1.
- HIGHDS, #C7,C8, destination + 1 de la plus haute adresse à daplacer FINAD+1+1.

LIST affiche à l'écran la partie du code modifiée.
Le curseur sauvegardé par LIST est remis en place par RESCUR.

puis, avant sortie de la routine ...

un caractère espace est inscrit à l'emplacement du curseur et ...
à l'emplacement suivant de la mémoire d'écran.

Cette routine décale d'une position vers le bas de la mémoire le code compris entre les adresses pointées par CRSAD+1 et FINAD.
Le contenu de CRSAD est transféré dans
le pointeur temporaire TMPB.
DLTO, début de la boucle de décalage.
Le code est décalé d'une position vers le bas en commençant par l'adresse la plus faible, le contenu de CRSAD+1 passe en CRSAD et ainsi de suite jusqu'à FINAD.

Après chaque décalage, TMPB est incrémenté ...

puis comparé à FINAD.

Retour en début de boucle, DLTO, tant que TMPB < FINAD.
Fin de boucle, l'adresse FINAD est décrémentée et affichée par DECFA.
LIST affiche la partie modifiée. Au retour de LIST ...
- si C=1, la fin du bloc n'a pas été atteinte, sortie via RESCUR.
- si C=0, deux JSR OUTSPC effacent le code hexadécimal excédentaire en fin de bloc.

Le curseur, sauvegardé par LIST est remis en place. Sortie.

```
                        ;Routine
                        ;somme de contrôle

6389 A5 60     CHKSM    LDA DEPAD
638B A4 61              LDY DEPAD+1
638D 85 6A              STA TEMPB
638F 84 6B              STY TEMPB+1
6391 A0 00              LDY #$00
6393 84 68              STY TEMPA
6395 84 69              STY TEMPA+1
6397 A5 6A     ADD      LDA TEMPB
6399 C5 64              CMP CRSAD
639B A5 6B              LDA TEMPB+1
639D E5 65              SBC CRSAD+1
639F B0 14              BCS SUM
63A1 A5 68              LDA TEMPA
63A3 71 6A              ADC (TEMPB),Y
63A5 85 68              STA TEMPA
63A7 A5 69              LDA TEMPA+1
63A9 69 00              ADC #$00
63AB 85 69              STA TEMPA+1
63AD E6 6A              INC TEMPB
63AF D0 02              BNE ADD0
63B1 E6 6B              INC TEMPB+1
63B3 D0 E2     ADD0     BNE ADD
63B5 60        SUM      RTS

                        ;Incrémentation
                        ;des pointeurs

                        ;-curseur

63B6 E6 64     INCPT    INC CRSAD
63B8 D0 02              BNE INC0
63BA E6 65              INC CRSAD+1
63BC A5 6F     INC0     LDA COLCT
63BE C9 08              CMP #$08
63C0 D0 07              BNE INC1
63C2 20 CA 63           JSR INC2
63C5 A9 00              LDA #$00
63C7 85 6F              STA COLCT
63C9 60        INC1     RTS

                        ;-adresse de base

63CA E6 6E     INC2     INC LINCT
63CC 18        INC3     CLC
63CD A5 62              LDA BASAD
63CF 69 08              ADC #$08
63D1 85 62              STA BASAD
63D3 90 02              BCC INC4
63D5 E6 63              INC BASAD+1
63D7 60        INC4     RTS
```

```
635F A5 60     CHKSM    LDA DEPAD
6361 A4 61              LDY DEPAD+1
6363 85 6A              STA TEMPB
6365 84 6B              STY TEMPB+1
6367 A0 00              LDY #$00
6369 84 68              STY TEMPA
636B 84 69              STY TEMPA+1
636D A5 6A     ADD      LDA TEMPB
636F C5 64              CMP CRSAD
6371 A5 6B              LDA TEMPB+1
6373 E5 65              SBC CRSAD+1
6375 B0 14              BCS SUM
6377 A5 68              LDA TEMPA
6379 71 6A              ADC (TEMPB),Y
637B 85 68              STA TEMPA
637D A5 69              LDA TEMPA+1
637F 69 00              ADC #$00
6381 85 69              STA TEMPA+1
6383 E6 6A              INC TEMPB
6385 D0 02              BNE ADD0
6387 E6 6B              INC TEMPB+1
6389 D0 E2     ADD0     BNE ADD
638B 60        SUM      RTS

                        ;Incrémentation
                        ;des pointeurs

                        ;-curseur

638C E6 64     INCPT    INC CRSAD
638E D0 02              BNE INC0
6390 E6 65              INC CRSAD+1
6392 A5 6F     INC0     LDA COLCT
6394 C9 08              CMP #$08
6396 D0 07              BNE INC1
6398 20 A0 63           JSR INC2
639B A9 00              LDA #$00
639D 85 6F              STA COLCT
639F 60        INC1     RTS

                        ;-adresse de base

63A0 E6 6E     INC2     INC LINCT
63A2 18        INC3     CLC
63A3 A5 62              LDA BASAD
63A5 69 08              ADC #$08
63A7 85 62              STA BASAD
63A9 90 02              BCC INC4
63AB E6 63              INC BASAD+1
63AD 60        INC4     RTS
```

Cette routine calcule la somme sur deux octets (la retenue éventuelle est ignorée) des valeurs en mémoire comprises entre les adresses DEPAD (début du bloc) et CRSAD-1 (octet précédant le curseur) incluses.
L'adresse de début du bloc, DEPAD, est transférée dans TMPB.

TMPA, qui recueillera la somme, est initialisé à zéro.

ADD, début de boucle d'addition.
Test de fin de boucle, ...
TMPB est comparé à CRSAD.

- Si C=1, TMPB=CRSAD, → SUM, sortie.
- Si C=0, TMPB<CRSAD, la valeur suivante ...
  est ajoutée ...
  à la somme.

Le pointeur TMPB ...
est incrémenté.

ADD0, retour en début de boucle, branchement forcé.
Sortie.

Incrémentation du pointeur CRSAD ...
qui contient l'adresse ...
matérialisée par le curseur.
Le contenu du compteur de colonnes, COLCT ...
est comparé à la valeur 08.
Si COLCT est différent de 08, fin de ligne non atteinte, retour.
Si COLCT=08, 8 octets ont été affichés. INC2 ajuste LINCT et BASAD ...
pour l'affichage de la ligne suivante.
COLCT est remis à zéro.
Sortie.

Le compteur de lignes, LINCT, est incrémenté.

L'adresse de base, BASAD, en-tête d'une ligne, ...
adresse du premier des 8 octets, ...
est augmentée de 8 ...
en vue de l'affichage de la ligne suivante.

Sortie.

```
                        ;Incrémentation
                        ;adresse fin
                        ;et affichage

63D8 18        INCFA    CLC
63D9 A5 66              LDA FINAD
63DB 69 01              ADC #$01
63DD 85 66              STA FINAD
63DF 85 68              STA TEMPA
63E1 A5 67              LDA FINAD+1
63E3 69 00              ADC #$00
63E5 85 67              STA FINAD+1
63E7 85 69              STA TEMPA+1
63E9 4C C5 64           JMP STAT+3

63AE 18        INCFA    CLC
63AF A5 66              LDA FINAD
63B1 69 01              ADC #$01
63B3 85 66              STA FINAD
63B5 85 68              STA TEMPA
63B7 A5 67              LDA FINAD+1
63B9 69 00              ADC #$00
63BB 85 67              STA FINAD+1
63BD 85 69              STA TEMPA+1
63BF 4C 98 64           JMP STAT+3

                        ;Décrémentation
                        ;des pointeurs

                        ;-curseur

63EC C6 64     DECPT    DEC CRSAD
63EE A5 64              LDA CRSAD
63F0 C9 FF              CMP #$FF
63F2 D0 02              BNE DEC0
63F4 C6 65              DEC CRSAD+1
63F6 A5 6F     DEC0     LDA COLCT
63F8 10 07              BPL DEC1
63FA 20 02 64           JSR DEC2
63FD A9 07              LDA #$07
63FF 85 6F              STA COLCT
6401 60        DEC1     RTS

63C2 C6 64     DECPT    DEC CRSAD
63C4 A5 64              LDA CRSAD
63C6 C9 FF              CMP #$FF
63C8 D0 02              BNE DEC0
63CA C6 65              DEC CRSAD+1
63CC A5 6F     DEC0     LDA COLCT
63CE 10 07              BPL DEC1
63D0 20 D8 63           JSR DEC2
63D3 A9 07              LDA #$07
63D5 85 6F              STA COLCT
63D7 60        DEC1     RTS

                        ;-adresse de base

6402 38        DEC2     SEC
6403 A5 62              LDA BASAD
6405 E9 08              SBC #$08
6407 85 62              STA BASAD
6409 B0 02              BCS DEC3
640B C6 63              DEC BASAD+1
640D C6 6E     DEC3     DEC LINCT
640F 60                 RTS

63D8 38        DEC2     SEC
63D9 A5 62              LDA BASAD
63DB E9 08              SBC #$08
63DD 85 62              STA BASAD
63DF B0 02              BCS DEC3
63E1 C6 63              DEC BASAD+1
63E3 C6 6E     DEC3     DEC LINCT
63E5 60                 RTS

                        ;Décrémentation
                        ;adresse fin
                        ;et affichage

6410 38        DECFA    SEC
6411 A5 66              LDA FINAD
6413 E9 01              SBC #$01
6415 85 66              STA FINAD
6417 85 68              STA TEMPA
6419 A5 67              LDA FINAD+1
641B E9 00              SBC #$00
641D 85 67              STA FINAD+1
641F 85 69              STA TEMPA+1
6421 4C C5 64           JMP STAT+3

63E6 38        DECFA    SEC
63E7 A5 66              LDA FINAD
63E9 E9 01              SBC #$01
63EB 85 66              STA FINAD
63ED 85 68              STA TEMPA
63EF A5 67              LDA FINAD+1
63F1 E9 00              SBC #$00
63F3 85 67              STA FINAD+1
63F5 85 69              STA TEMPA+1
63F7 4C 9B 64           JMP STAT+3
```

Le pointeur FINAD, qui détient l'adresse de fin du bloc de code, est incrémenté. La nouvelle adresse est affichée à la ligne supérieure par STAT+3.
L'addition est employée au lieu de la technique classique avec INC (voir encadré ci-dessous) car le résultat doit etre transféré dans le pointeur temporaire TMPA pour l'affichage à l'écran.

* * *

Le commentaire des routines qui suivent offrant peu d'intérêt, nous allons récapituler les techniques d'incrémentation, décrémentation, addition et soustraction de nombres sur deux octets. Comme il s'agit le plus souvent de pointeurs d'adresse, PTR sera le nombre à modifier, octet faible PTRL, octet fort PTRH.

Incrémentation, décrémentation d'un nombre sur deux octets.

L'octet faible est augmenté ou diminué de un par INC ou DEC.
Si PTRL=00 après INC, retenue, l'octet fort doit être incrémenté ou si PTRL=FF après DEC, emprunt, l'octet fort doit être décrémenté.
Sinon, INC ou DEC PTRH est évitée, branchement à suite.

```
INCPTR  INC PTRL          DECPTR  DEC PTRL
        BNE SUITE                 LDA PTRL
        INC PTRH                  CMP #$FF
SUITE   ...                       BNE SUITE
                                  DEC PTRH
                          SUITE   ...
```

Addition ou soustraction d'une valeur VAL sur un octet.

Notez la nécessité de positionner l'indicateur de retenue; C=0 par CLC avant l'addition, C=1 par SEC avant la soustraction.
Le résultat est replacé dans PTR, il pourrait être rangé ailleurs.

```
CLC         SEC         ou encore   CLC         SEC
LDA PTRL    LDA PTRL                LDA PTRL    LDA PTRL
ADC VAL     SBC VAL                 ADC VAL     SBC VAL
STA PTRL    STA PTRL                STA PTRL    STA PTRL
LDA PTRH    LDA PTRH                BCC SUITE   BCS SUITE
ADC #$00    SBC #$00                INC PTRH    DEC PTRH
STA PTR     STA PTRH        SUITE   ...         ...
```

Addition ou soustraction d'une valeur sur deux octets.

```
CLC         SEC         Remarque: Après l'opération,
LDA PTRL    LDA PTRL    la valeur de C indique une retenue
ADC VALL    SBC VALL    ou un emprunt éventuel.
STA PTRL    STA PTRL     - après l'addition, si C=1, retenue,
LDA PTRH    LDA PTRH       le résultat est plus grand que FFFF.
ADC VALH    SBC VALH     - après la soustraction, C=0, emprunt,
STA PTRH    STA PTRH       le résultat est plus petit que 0000.
```

```
                        ;Recueil d'une
                        ;adresse

6424 A6 77     ADGET    LDX TMP2
6426 20 E7 64           JSR PRMSB
6429 20 80 CD           JSR INPTRQ
642C 86 E9              STX TXTPTR
642E 84 EA              STY TXTPTR+1
6430 A2 00              LDX #$00
6432 86 68              STX TEMPA
6434 86 69              STX TEMPA+1
6436 20 E2 00  GTCHR    JSR CHRGET
6439 F0 0D              BEQ AGT0
643B 20 4F 64           JSR VALHX
643E B0 06              BCS CHROK
6440 20 9F FA           JSR PING
6443 4C 24 64           JMP ADGET
6446 D0 EE     CHROK    BNE GTCHR
6448 A5 68     AGT0     LDA TEMPA
644A A4 69              LDY TEMPA+1
644C A6 35              LDX BUF
644E 60                 RTS

63FA A6 77     ADGET    LDX TMP2
63FC 20 BD 64           JSR PRMSG
63FF 20 F4 CC           JSR INPTRQ
6402 86 E9              STX TXTPTR
6404 84 EA              STY TXTPTR+1
6406 A2 00              LDX #$00
6408 86 68              STX TEMPA
640A 86 69              STX TEMPA+1
640C 20 E2 00  GTCHR    JSR CHRGET
640F F0 0D              BEQ AGT0
6411 20 25 64           JSR VALHX
6414 B0 06              BCS CHROK
6416 20 85 FA           JSR PING
6419 4C FA 63           JMP ADGET
641C D0 EE     CHROK    BNE GTCHR
641E A5 68     AGT0     LDA TEMPA
6420 A4 69              LDY TEMPA+1
6422 A6 35              LDX BUF
6424 60                 RTS

                        ;Test validité
                        ;caractère hexa

644F 85 76     VALHX    STA TMP1
6451 49 30              EOR #$30
6453 C9 0A              CMP #$0A
6455 90 06              BCC HEX
6457 69 88              ADC #$88
6459 C9 FA              CMP #$FA
645B 90 14              BCC VHX1
645D A2 03     HEX      LDX #$03
645F 0A                 ASL A
6460 0A                 ASL A
6461 0A                 ASL A
6462 0A                 ASL A
6463 0A        NXBT     ASL A
6464 26 68              ROL TEMPA
6466 26 69              ROL TEMPA+1
6468 B0 06              BCS VHX0
646A CA                 DEX
646B 10 F6              BPL NXBT
646D 38                 SEC
646E B0 01              BCS VHX1
6470 18        VHX0     CLC
6471 A5 76     VHX1     LDA TMP1
6473 60                 RTS

6425 85 76     VALHX    STA TMP1
6427 49 30              EOR #$30
6429 C9 0A              CMP #$0A
642B 90 06              BCC HEX
642D 69 88              ADC #$88
642F C9 FA              CMP #$FA
6431 90 14              BCC VHX1
6433 A2 03     HEX      LDX #$03
6435 0A                 ASL A
6436 0A                 ASL A
6437 0A                 ASL A
6438 0A                 ASL A
6439 0A        NXBT     ASL A
643A 26 68              ROL TEMPA
643C 26 69              ROL TEMPA+1
643E B0 06              BCS VHX0
6440 CA                 DEX
6441 10 F(              BPL NXBT
6443 38                 SEC
6444 B0 01              BCS VHX1
6446 18        VHX0     CLC
6447 A5 76     VHX1     LDA TMP1
6449 60                 RTS
```

ADGET est appelée pour le recueil d'une adresse de début ou de fin.
A l'entrée, X est chargé avec la valeur en TMP2, index dans la table des messages. Le mot 'Debut' ou 'Fin' est affiché par PRMSG.
INPTRQ inscrit dans le buffer d'entrée les codes ASCII des caractères tapés au clavier.
Après RETURN, l'adresse #34 (BUF-1), en X(-) Y(+), est transférée dans TXTPTR à l'usage de CHRGET.
TMPA, qui recueillera l'adresse convertie en binaire, est initialisé à zéro. En GTCHR, l'appel de CHRGET marque le début d'une boucle d'analyse du buffer d'entrée. La sortie de boucle intervient si la valeur recueillie dans l'accumulateur par CHRGET est égale à zéro, fin d'entrée. Les caractères sont analysés et manipulés tour à tour par VALHX. Un caractère invalide (C=0) est rejeté avec Ping et ... retour en début de routine.
En CHROK, branchement forcé, caractère suivant.
En AGTO, sortie de la routine. A(-) et Y(+) sont chargés avec l'adresse élaborée par VALHX dans TMPA.
Le registre X reçoit le contenu du premier emplacement du buffer d'entrée. Cela peut être 00 (pas d'entrée).

Le caractère est rangé dans TMP1 avant manipulation.
Le premier test ...
laisse passer les chiffres ...
de 0 à 9.
Le deuxième test ...
accepte aussi les lettres de A à F.
Sortie, C=0, si caractère invalide.
En HEX commence une boucle de conversion.
Après les premières manipulations, ...
les valeurs acceptées sont traitées tour à tour ...
par une série de décalages ...
et forment un nombre binaire ...
compris entre 00 et FFFF ...
dans TMPA.

Si le nombre entré est > FFFF, sortie caractère invalide.
Les zéros non significatifs ne sont pas rejetés.

Avant sortie, l'indicateur C est manipulé de façon à ce que ...
C=1 indique un caractère hexa valide.
C=0 indique un caractère invalide.
Le caractère entré est récupéré dans l'accumulateur.
Sortie de la routine.

```
                        ;Sauvegarde
                        ;position curseur

6474 AD 69 02 SAVCUR LDA CURCOL
6477 85 71           STA SAV1
6479 AD 68 02        LDA CURROW
647C 85 72           STA SAV2
647E A5 12           LDA CURBAS
6480 85 73           STA SAV3
6482 A5 13           LDA CURBAS+1
6484 85 74           STA SAV4
6486 A5 6E           LDA LINCT
6488 85 76           STA TMP1
648A A5 6F           LDA COLCT
648C 85 77           STA TMP2
648E A5 62           LDA BASAD
6490 A4 63           LDY BASAD+1
6492 85 6C           STA TEMPC
6494 84 6D           STY TEMPC+1
6496 A9 11           LDA #$11
6498 4C D9 CC        JMP OUTDO

644A AD 69 02 SAVCUR LDA CURCOL
644D 85 71           STA SAV1
644F AD 68 02        LDA CURROW
6452 85 72           STA SAV2
6454 A5 12           LDA CURBAS
6456 85 73           STA SAV3
6458 A5 13           LDA CURBAS+1
645A 85 74           STA SAV4
645C A5 6E           LDA LINCT
645E 85 76           STA TMP1
6460 A5 6F           LDA COLCT
6462 85 77           STA TMP2
6464 A5 62           LDA BASAD
6466 A4 63           LDY BASAD+1
6468 85 6C           STA TEMPC
646A 84 6D           STY TEMPC+1
646C A9 11           LDA #$11
646E 4C 12 CC        JMP OUTDO

                        ;Restaure
                        ;position curseur

649B A5 71     RESCUR LDA SAV1
649D 8D 69 02        STA CURCOL
64A0 A5 72           LDA SAV2
64A2 8D 68 02        STA CURROW
64A5 A5 73           LDA SAV3
64A7 85 12           STA CURBAS
64A9 A5 74           LDA SAV4
64AB 85 13           STA CURBAS+1
64AD A5 76           LDA TMP1
64AF 85 6E           STA LINCT
64B1 A5 77           LDA TMP2
64B3 85 6F           STA COLCT
64B5 A5 6C           LDA TEMPC
64B7 A4 6D           LDY TEMPC+1
64B9 85 62           STA BASAD
64BB 84 63           STY BASAD+1
64BD A9 11           LDA #$11
64BF 4C D9 CC        JMP OUTDO

6471 A5 71     RESCUR LDA SAV1
6473 8D 69 02        STA CURCOL
6476 A5 72           LDA SAV2
6478 8D 68 02        STA CURROW
647B A5 73           LDA SAV3
647D 85 12           STA CURBAS
647F A5 74           LDA SAV4
6481 85 13           STA CURBAS+1
6483 A5 76           LDA TMP1
6485 85 6E           STA LINCT
6487 A5 77           LDA TMP2
6489 85 6F           STA COLCT
648B A5 6C           LDA TEMPC
648D A4 6D           LDY TEMPC+1
648F 85 62           STA BASAD
6491 84 63           STY BASAD+1
6493 A9 11           LDA #$11
6495 4C 12 CC        JMP OUTDO
```

Les routines d'affichage à l'écran, dont OUTDO, laissent le curseur sur la future position d'écriture. SAVCUR et RESCUR ont pour but de mémoriser la position du curseur avant un listage et de la restituer après l'opération. Les paramètres concernés sont sauvegardés par SAVCUR dans des emplacements temporaires. De manière strictement inverse, ils sont restaurés par RESCUR.
Les instructions LDA #$11 et JMP OUTDO sont l'équivalent d'un CTRL-Q, bascule curseur.
Sur Atmos, elles peuvent être remplacées par JMP #CCD1 (CURTGL).
Ces routines n'appellent pas d'autre commentaire.

* * *

Profitons de la pause pour continuer notre digression à propos des nombres sur deux octets. Il s'agira cette fois de comparaisons.

Vous savez que les instructions CMP, CPX et CPY réalisent la soustraction virtuelle: contenu du registre - valeur comparée. Cette valeur est, soit une constante, soit le contenu d'un emplacement en mémoire. L'opération ne modifie pas les valeurs comparées, seuls les indicateurs C, Z et N sont affectés.

- Si contenu du registre < valeur comparée	C=0	Z=0	N?
- Si contenu du registre = valeur comparée	C=1	Z=1	N=0
- Si contenu du registre > valeur comparée	C=1	Z=0	N?

Je saisis l'occasion pour rectifier une erreur du manuel de référence, page 66. L'indicateur de signe est positionné selon le signe du résultat, comme après une soustraction normale. Il prend la valeur 0 en cas d'égalité.
Un branchement conditionnel, BEQ, BNE, BCC ou BCS suit généralement la comparaison. Certains assembleurs admettent une appellation différente pour les deux derniers:
BLT, Branch Less Than, branchement si inférieur équivaut à BCC.
BGE, Branch Greater or Equal, plus grand ou égal, signifie BCS.

Comparaison de deux nombres sur deux octets.

Soit à comparer un nombre N1 à un autre N2. Les octets de poids faible sont désignés par le suffixe L, ceux de poids fort par H.

```
   TESTH  LDA N1H            SEC               LDA N1L
          CMP N2H            LDA N1L           CMP N2L
          BCC PLUS PETIT     SBC N2L           LDA N1H
          BEQ TESTL          LDA N1H           SBC N2H
          BCS PLUS GRAND     SBC N2H           BCC PLUS PETIT
   TESTL  LDA N1L            BCC PLUS PETIT    BEQ EGAL
          CMP N2L            BEQ EGAL          BCS PLUS GRAND
          BCC PLUS PETIT     BCS PLUS GRAND
          BEQ EGAL
          BCS PLUS GRAND
```

Remarque: Si on teste la condition d'égalité, BEQ ou BNE doivent être précédés de BCC. Sinon, une instruction BCC ou BCS seule suffit pour l'infériorité ou la supériorité.

```
                        ;Affiche une
                        ;adresse en haut
                        ;de l'écran

64C2 A9 02     STAT     LDA #$02
64C4 2C A9 09           BIT $09A9
64C7 2C A9 11           BIT $11A9
64CA 85 77              STA TMP2
64CC A2 FF              LDX #$FF
64CE 86 2F              STX FLG
64D0 E8                 INX
64D1 A5 69              LDA TEMPA+1
64D3 20 93 D9           JSR BINHEX
64D6 A5 68              LDA TEMPA
64D8 20 93 D9           JSR BINHEX
64DB A9 00              LDA #$00
64DD 9D 00 01           STA FBUF,X
64E0 A0 01              LDY #$01
64E2 A6 77              LDX TMP2
64E4 4C 65 F8           JMP STOUT

                        ;Affiche un message

64E7 BD 24 65  PRMSG    LDA MSG1,X
64EA F0 06              BEQ PMG0
64EC 20 D9 CC           JSR OUTDO
64EF E8                 INX
64F0 D0 F5              BNE PRMSG
64F2 60        PMG0     RTS

                        ;Affiche adresse de
                        ;base de la ligne

64F3 A6 62     PRADR    LDX BASAD
64F5 A4 63              LDY BASAD+1
64F7 20 09 65           JSR PRTYX
64FA A9 3A              LDA #":"
64FC 20 D9 CC           JSR OUTDO
64FF A9 00              LDA #$00
6501 85 6F              STA COLCT
6503 20 D4 CC           JSR OUTSPC
6506 4C D4 CC           JMP OUTSPC

                        ;Affiche en hexa
                        ;contenus Y et X

6509 98        PRTYX    TYA
650A 20 0E 65           JSR PRBYT
650D 8A                 TXA
```

```
64B8 A9 02     STAT     LDA #$02
64BA 2C A9 09           BIT $09A9
64BD 2C A9 11           BIT $11A9
64C0 85 77              STA TMP2
64C2 A2 FF              LDX #$FF
64C4 86 2F              STX FLG
64C6 E8                 INX
64C7 A5 69              LDA TEMPA+1
64C9 20 F5 D8           JSR BINHEX
64CC A5 68              LDA TEMPA
64CE 20 F5 D8           JSR BINHEX
64D1 A9 00              LDA #$00
64D3 9D 00 01           STA FBUF,X
64D6 A0 01              LDY #$01
64D8 A6 77              LDX TMP2
64DA 4C 2F F8           JMP STOUT

                        ;Affiche un message

64DD BD FA 64  PRMSG    LDA MSG1,X
64E0 F0 06              BEQ PMG0
64E2 20 12 CC           JSR OUTDO
64E5 E8                 INX
64E6 D0 F5              BNE PRMSG
64E8 60        PMG0     RTS

                        ;Affiche adresse de
                        ;base de la ligne

64E9 A6 62     PRADR    LDX BASAD
64EB A4 63              LDY BASAD+1
64ED 20 DF 64           JSR PRTYX
64F0 A9 3A              LDA #":"
64F2 20 12 CC           JSR OUTDO
64F5 A9 00              LDA #$00
64F7 85 6F              STA COLCT
64F9 20 0D CC           JSR OUTSPC
64FC 4C 0D CC           JMP OUTSPC

                        ;Affiche en hexa
                        ;contenus Y et X

64DF 98        PRTYX    TYA
64E0 20 E4 64           JSR PRBYT
64E3 8A                 TXA
```

L'instruction BIT est un truc qui permet d'entrer dans une routine avec des valeurs différentes d'un même registre. Ici nous aurons A=#02, #09 ou #11 selon que l'entrée est en STAT, STAT+3 ou STAT+6. Cette valeur est rangée en TMP2 pour emploi ultérieur par STOUT.
Je disais que le drapeau FLG était responsable de l'abandon des zéros non significatifs au cours de la conversion binaire-hexa par BINHEX. Pour éviter cette action inopportune, il doit être différent de zéro. X lui donne la valeur FF. Ce registre est initialisé à zéro par INX. L'adresse en TMPA est convertie par BINHEX, octet fort d'abord. Cette routine transforme une valeur 8 bits en deux caractères hexadécimaux qui sont rangés dans le buffer FAC (FBUF, #FF-110). Elle est utilisée par la fonction Basic HEX$.
X sert d'index dans FBUF pour BINHEX. La chaîne des quatre caractères hexa de l'adresse doit être suivie d'un 00 pour l'affichage par STOUT. Elle commence en #100. A(-) et Y (+) sont chargés avec cette adresse. X=TMP2 détermine la position d'affichage par rapport au coin supérieur gauche de l'écran #BB80.

Boucle d'affichage d'un message du menu. X sert d'index dans la table MSG1. Il est chargé avec la valeur adéquate, préalablement à l'appel de la routine. Le caractère recueilli dans l'accumulateur est affiché par OUTDO. Chaque message est suivi d'un 00. BEQ en détecte ainsi la fin. BNE est un branchement forcé en début de boucle.

L'octet faible de l'adresse est transféré dans X, ...
l'octet fort est transféré dans Y ...
pour affichage en hexadécimal par PRTYX.
Le code du caractère ':', deux-points, est chargé dans l'accumulateur pour affichage par OUTDO.
Le compteur de colonnes, COLCT, est mis à zéro ...
le curseur étant placé sur le premier octet de la ligne ...
après l'affichage de deux espaces ...
par OUTSPC.

Les contenus des registres Y et X sont successivement transférés dans l'accumulateur pour affichage en hexadécimal par PRBYT.

```
                        ;Affiche en hexa
                        ;contenu Acc.

650E 48        PRBYT    PHA
650F 4A                 LSR A
6510 4A                 LSR A
6511 4A                 LSR A
6512 4A                 LSR A
6513 20 19 65           JSR PRHEX
6516 68                 PLA
6517 29 0F              AND #$0F
6519 09 30     PRHEX    ORA #$30
651B C9 3A              CMP #$3A
651D 90 02              BCC PRINT
651F 69 06              ADC #$06
6521 4C D9 CC  PRINT    JMP OUTDO

64E4 48        PRBYT    PHA
64E5 4A                 LSR A
64E6 4A                 LSR A
64E7 4A                 LSR A
64E8 4A                 LSR A
64E9 20 EF 64           JSR PRHEX
64EC 68                 PLA
64ED 29 0F              AND #$0F
64EF 09 30     PRHEX    ORA #$30
64F1 C9 3A              CMP #$3A
64F3 90 02              BCC PRINT
64F5 69 06              ADC #$06
64F7 4C 12 CC  PRINT    JMP OUTDO


                        ;Messages menu

                MSG1    $ 0C
                        "1.Nouvelle entree
                        $ 0D 0A
                        "2.Edition
                        $ 0D 0A
                        "3.Impression
                        $ 0D 0A
                        "4.Sortie
                        $ 0D 0A 0A 0A 00

                        $ 0B 0D 0E
                        "Debut
                        $ 00

                        $ 0B 0D 0E
                        "Fin
                        $ 20 20 00

                        "8 ou 16 col?
                        " (defaut 16)
                        $ 20 00

                        $ 0D 0A
                        "Somme de controle?
                        " (O/N)
                        $ 20


                        ;Messages
                        ;haut de l'écran

                MSG2    $ 00 41 23 20 20 20
                        $ 20 2C 45 23 20 20
                        $ 20 20 20 00 53 3D
                        $ 20 20 20 20 20 20
                        " ESC,+,DEL,CTRL-Z

                MSG3    "  48 lignes  SPC
                        "=Stop,Suite
```

Routine d'affichage en hexadécimal du contenu de l'accumulateur.
Vous l'incluerez telle quelle dans vos programmes.
Le contenu de l'accumulateur est empilé pour traitement ultérieur.
Quatre décalages à droite permettent de ne conserver que le quartet (demi-octet) gauche qui est converti en caractère hexadécimal ...
puis affiché par PRHEX.
La valeur initiale est désempilée.
L'instruction AND #$0F isole le quartet de droite.
En PRHEX, ORA #$30 transforme en code caractère affichable, la valeur binaire du quartet. Si ce code est < #3A, il s'agit d'un chiffre de 0 à 9 qui sera affiché directement.
Sinon, il faut ajouter 6 pour obtenir le code d'une lettre de A à F avant affichage par OUTDO.

Les messages du menu renferment des caractères de contrôle.
La signification des codes correspondants est rappelée ci-dessous.

0A, descente curseur une ligne (flèche bas, CTRL-J).
0B, remontée surseur une ligne (flèche haut, CTRL-K).
0C, effacement de l'écran, (CLS, CTRL-L).
0D, retour en début de ligne.
0E, efface la ligne sur laquelle se trouve le curseur (CTRL-N)

Ces caractères de contrôle sont interprétés comme tels par OUTDO, utilisée par la routine d'affichage PRMSG.
Le résultat est analogue au PRINT CHR$ du Basic.
La commande Basic PRINT utilise d'ailleurs OUTDO.

Les messages qui figurent sous MSG2 et MSG3, en bas de la page ci-contre, sont inscrits directement en mémoire d'écran.
Ce procédé s'apparente au POKE du Basic.
Les codes compris entre #00 et #17 deviennent des attributs.
Ce serait le cas avec STOUT qui sert à afficher les adresses sur la ligne supérieure, sauf pour 00 qui sert de marqueur de fin.

Il est intéressant de comparer STOUT avec STROUT, évoquée page 11.
Les deux routines ont des modes d'emploi presque identiques mais
La première agit comme POKE alors que la seconde, qui utilise OUTDO, agit comme PRINT.

CONVERSION BINAIRE-DATA

Ce programme construit des lignes Basic DATA numérotées de 5 en 5, à partir d'un bloc de code machine. Ce dernier sera présent en mémoire et éventuellement déplacé, sans relogement, à l'aide du moniteur (voir page 55), avant chargement du programme de conversion.
L'opération dure environ 30 secondes par page de 256 octets.
Deux sous-programmes en langage machine distincts sont utilisés.
L'un assure le rangement en mémoire des lignes DATA.
L'autre supprime le programme Basic devenu inutile.

- La ligne 1 inscrit en mémoire le code des sous-programmes. L'adresse d'implantation, #7000, n'est pas impérative; le code est relogeable. Si vous la changez, modifiez aussi les lignes 4, 12, 15 et 16.
- La ligne 2 demande un numéro pour la première ligne à construire. Il doit être égal ou supérieur à 30 car, dans un premier temps, les lignes DATA viennent s'ajouter au programme utilitaire.
- La ligne 3 demande l'adresse A, en hexadécimal (précédée de #) ou en décimal, du début du bloc à traiter.
- La ligne 4 demande l'adresse de fin F. Cette adresse est sauvegardée en #7004,7005, voir pourquoi ligne 15.

Alors commence une boucle qui garnit le buffer d'entrée (début #35) avec une ligne DATA exactement comme si elle était entrée au clavier.
- ligne 5, mise au format du numéro de ligne (suppression du signe +), appel du sous-programme de la ligne 17 pour inscription du numéro.
- ligne 6, GOSUB 17 pour l'inscription du mot DATA.
- ligne 7, inscription d'un espace, incrémentation index BUF.

Après initialisation du compteur J à zéro ...
la boucle REPEAT-UNTIL des lignes 8 à 15 traite 16 octets successifs.

. ligne 8, lecture du code en hexadécimal.

. lignes 9 à 11, mise au format (2 chiffres hexa sans #), inscription.

. ligne 12, sortie si fin du bloc, 00 fin de ligne, CALL#7006 pour le rangement de la ligne en mémoire puis CALL#705D pour la suppression de la partie utilitaire dont les numéros de lignes limites, 0 et 29, sont passés par l'intermédiaire des emplacements #80,81 et #82,83.

. ligne 13, incrémentation de A pour l'octet suivant et de J compteur de boucle REPEAT-UNTIL. Si J=16, 16 éléments dans la ligne DATA, l'inscription d'une virgule dans le buffer d'entrée est évitée.

. ligne 14, inscription du code d'une virgule.

. ligne 15, fin de boucle, 16 valeurs hexa ont été enregistrées dans le buffer d'entrée. Un 00 de fin de ligne est inscrit à leur suite. L'adresse de l'octet suivant et le numéro de ligne en cours sont sauvegardés en #7000,7001 et #7002,7003 car les variables Basic sont détruites par le sous-programme de mise en place de la ligne en mémoire appelé par CALL#7006.

- ligne 16, les variables Basic sont récupérées. Le numéro de ligne est augmenté de 5. GOTO 5 pour la construction d'une nouvelle ligne.
- ligne 17, sous-programme qui inscrit dans le buffer d'entrée le numéro de ligne, le mot 'DATA' et les codes hexadécimaux formattés.
- lignes 18 à 29, code machine responsable du rangement en mémoire d'une ligne DATA complète et de l'effacement du programme utilitaire lorsque tout est terminé.

```
 0 REM * Conversion binaire-DATA *
 1 FOR I=0 TO 188:READ C$:C=VAL("#"+C$):POKE #7000+I,C:NEXT
 2 TEXT:CLS:INPUT"No 1ere ligne DATA (>29)";L:IF L<30 THEN 2
 3 INPUT"Adresse debut code machine";A
 4 INPUT"Adresse fin code machine";F:DOKE#7004,F:IF F<=A THEN 4
 5 A$=MID$(STR$(L),2):BUF=#35:GOSUB 17
 6 A$="DATA":GOSUB 17
 7 POKE BUF,32:BUF=BUF+1
 8 J=0:REPEAT:A$=HEX$(PEEK(A))
 9 IF MID$(A$,2)=""THEN A$="#00"
 10 IF MID$(A$,3)=""THEN A$="#"+"0"+MID$(A$,2)
 11 A$=MID$(A$,2):GOSUB 17
 12 IF A=F THEN PULL:POKE BUF,0:CALL#7006:DOKE#80,0:DOKE#82,29:
CALL#705D
 13 A=A+1:J=J+1:IF J=16 THEN 15
 14 POKE BUF,44:BUF=BUF+1
 15 UNTIL J=16:POKE BUF,0:DOKE#7000,A:DOKE#7002,L:CALL#7006
 16 A=DEEK(#7000):L=DEEK(#7002)+5:F=DEEK(#7004):GOTO 5
 17 FOR I=1 TO LEN(A$):POKE BUF,ASC(MID$(A$,I,1)):BUF=BUF+1:
NEXT:RETURN
```

ATMOS

```
 18 DATA FF,FF,FF,FF,FF,FF,A5,E9,A4,EA,85,00,84,01,A2,34
 19 DATA A0,00,86,E9,84,EA,20,E2,00,20,E2,CA,20,FA,C5,84
 20 DATA 26,20,B3,C6,18,A5,9C,85,C9,65,26,85,C7,A4,9D,84
 21 DATA CA,90,01,C8,84,C8,20,F4,C3,A5,A0,A4,A1,85,9C,84
 22 DATA 9D,A4,26,88,B9,31,00,91,CE,88,10,F8,20,08,C7,20
 23 DATA 5F,C5,A5,00,A4,01,85,E9,84,EA,4C,C1,C8,A5,80,85
 24 DATA 33,A5,81,85,34,20,B3,C6,A5,CE,85,91,A5,CF,85,92
 25 DATA A5,82,85,33,A5,83,85,34,E6,33,D0,02,E6,34,20,B3
 26 DATA C6,A5,CE,C5,91,A5,CF,E5,92,B0,01,60,A0,00,B1,CE
 27 DATA 91,91,E6,CE,D0,02,E6,CF,E6,91,D0,02,E6,92,A5,9C
 28 DATA C5,CE,A5,9D,E5,CF,B0,E6,A6,92,A4,91,D0,01,CA,88
 29 DATA 86,9D,84,9C,20,08,C7,20,5F,C5,4C,A8,C4
```

ORIC-1

```
 18 DATA FF,FF,FF,FF,FF,FF,A5,E9,A4,EA,85,00,84,01,A2,34
 19 DATA A0,00,86,E9,84,EA,20,E2,00,20,98,CA,20,0A,C6,84
 20 DATA 26,20,DE,C6,18,A5,9C,85,C9,65,26,85,C7,A4,9D,84
 21 DATA CA,90,01,C8,84,C8,20,F8,C3,A5,A0,A4,A1,85,9C,84
 22 DATA 9D,A4,26,88,B9,31,00,91,CE,88,10,F8,20,33,C7,20
 23 DATA 6F,C5,A5,00,A4,01,85,E9,84,EA,4C,AD,C8,A5,80,85
 24 DATA 33,A5,81,85,34,20,DE,C6,A5,CE,85,91,A5,CF,85,92
 25 DATA A5,82,85,33,A5,83,85,34,E6,33,D0,02,E6,34,20,DE
 26 DATA C6,A5,CE,C5,91,A5,CF,E5,92,B0,01,60,A0,00,B1,CE
 27 DATA 91,91,E6,CE,D0,02,E6,CF,E6,91,D0,02,E6,92,A5,9C
 28 DATA C5,CE,A5,9D,E5,CF,B0,E6,A6,92,A4,91,D0,01,CA,88
 29 DATA 86,9D,84,9C,20,33,C7,20,6F,C5,4C,B5,C4
```

DEUXIEME PARTIE

UN MONITEUR COMPACT ET RELOGEABLE

BUT

Pourquoi un nouveau Moniteur, alors qu'il en existe plusieurs sur le marché?
Parce qu'à l'usage, ces programmes révèlent un grave défaut; ils sont lourds et encombrants.

De 6 à 8 K. pour l'apport de quelques fonctions supplémentaires. C'est excessif. On peut faire mieux.
Plus embarrassante encore est l'implantation fixe en haut de mémoire.
L'amputation d'une 'bonne' partie de l'espace utilisateur est rédhibitoire dans de nombreux cas.

C'est le bulldozer dans le jardinet de banlieue, l'armoire normande dans la chambre de bonne. Bref, ce n'est pas opérationnel.

Le Moniteur est un outil indispensable. Il doit être léger, simple et maniable. Les programmes du commerce ne présentant pas ces qualités, j'ai décidé de construire le mien.

Tout en offrant les fonctions essentielles, il occupe 2 K.octets, c'est raisonnable. Il est relogeable, c'est incomparablement plus pratique.

Des fonctions secondaires que je considère comme des gadgets (ne le répétez pas), ont été abandonnées.
A tort, peut-être, pour certaines comme STEP ou TRACE qui sont, paraît-il, le fin du fin pour la mise au point des programmes.
Avec le recul, il me semble avoir accordé trop de place (380 octets) à l'entrée du code hexadécimal et des caractères ASCII.

La perfection n'existe pas. Néanmoins, le programme devrait gagner votre adhésion grâce à un atout majeur, sa mobilité.
Le moniteur comporte une section chargée du relogement. Il s'en sert pour lui-même; c'est en quelque sorte un moniteur automobile.

Vous trouverez une description sommaire à la page suivante.
Le listing d'assemblage est en page 59. Il concerne l'Atmos.
Celui de l'Oric-1 est pratiquement identique, à quelques adresses près (adresses du système).

La réalisation n'est pas d'une grande originalité. Vous reconnaîtrez bon nombre de techniques classiques et éprouvées.
J'ai 'volé' des idées çà et là, selon le principe réaliste qu'il est idiot de vouloir réinventer la roue (il faut des siècles).
Pour la partie Assembleur-Désassembleur, je me suis inspiré des routines de l'Apple 2. A mon avis, elles sont des modèles du genre mais aussi, hélas, d'une effrayante complexité.

Le mode d'emploi du programme sera étudié en détail après l'entrée en mémoire.

DESCRIPTION

Le programme est initialement implanté à partir de l'adresse #9000 et il occupe 2 K.octets (#9000,97EC).
Il s'auto-protège contre une interférence possible avec un programme Basic par modification de HIMEM.

Les fonctions suivantes sont disponibles:
- Assemblage, mini-assembleur à une passe.
- Désassemblage, listage en langage symbolique 6502.
- Vidage mémoire (DUMP) en hexadécimal et visualisation des caractères ASCII correspondants.
- Entrée de codes hexadécimaux.
- Entrée de texte. (Ces deux fonctions peuvent être appelées directement ou à partir de l'assembleur).
- Exécution d'un programme (Go) avec insertion possible d'un BRK simulé.
- Déplacement d'octets vers le haut ou vers le bas de la mémoire.
- Relogement du code déplacé par la fonction précédente. (modification des adresses absolues internes).
- Auto-déplacement du Moniteur.

Afin de réduire la place occupée, le programme fait appel à de nombreuses adresses et routines du système avec les inconvénients (mineurs) que cela peut présenter.
La compatibilité est totale avec un autre programme (Basic ou langage machine) car, à chaque rentrée dans le Moniteur, ses pointeurs sont réinitialisés.
Evidemment, une situation adéquate lui aura été choisie au préalable.
(voir le mode d'emploi pour ces deux dernières remarques).

* * *

Vous allez maintenant entrer le Moniteur en mémoire à l'aide de l'Editeur de langage machine.
L'utilitaire étant chargé, appelez-le par CALL#6000.
Choisissez l'option 1. Nouvelle entrée; début: 9000 (sans #)

Entrez le code des pages suivantes.
Attention, il n'est pas le même pour l'Atmos que pour l'Oric-1.

Inutile de demander une somme de contrôle après l'entrée de chaque groupe de 16 octets. Vous le ferez à la fin.
Si une erreur est détectée, vous remonterez de proche en proche pour la localiser.

Le travail terminé, sauvegardez-le immédiatement:

```
CSAVE"",A#9000,E#97FF
```

```
9000:  20 BB F8 A9 00 85 00 85 A6 8D F5 02 A9 A4 8D 4B   07D2
9010:  02 20 F2 00 BA BD 00 01 85 01 85 A7 8D F6 02 8D   0E22
9020:  4C 02 A9 4C 8D 4A 02 20 0F C7 20 95 96 A9 FF 8D   14B4
9030:  0C 02 20 5A F7 20 F0 CB A9 2A 20 D9 CC 20 92 C5   1C1D
9040:  B6 E9 84 EA 20 E2 00 D0 08 A9 40 8D 4A 02 4C B2   2394
9050:  F8 C9 49 D0 0A AD B2 BB 49 69 8D B2 BB D0 CE A2   2D1E
9060:  09 CA 30 C9 DD C7 97 D0 F8 86 80 20 4C E9 84 02   35CE
9070:  85 03 8A F0 88 A6 80 20 85 90 AD 14 91 F0 03 20   3D48
9080:  28 91 4C 03 90 A5 01 48 8D D0 97 48 4C EB 00 4C   43BA
9090:  42 91 4C 6C 91 4C E7 91 4C 6E 92 4C BA 92 4C 18   4AE2
90A0:  93 4C FB 94 85 16 68 48 29 10 F0 0A 58 A5 16 28   5109
90B0:  20 1A 91 38 B0 0A A5 16 40 20 1A 91 20 28 91 38   559D
90C0:  68 E9 02 85 02 68 E9 00 85 03 20 F0 CB 20 44 96   5C25
90D0:  A2 00 BD D9 97 20 D9 CC A9 3D 20 D9 CC B5 50 20   6489
90E0:  0F 96 20 D4 CC E8 E0 04 D0 E8 20 D4 CC A2 07 BD   6D98
90F0:  DD 97 20 D9 CC CA 10 F7 20 F0 CB A2 1D 20 27 96   7619
9100:  A2 08 06 54 A9 30 69 00 20 D9 CC CA D0 F4 4C 7A   7D78
9110:  90 20 89 90 00 FF FF FF FF FF 08 85 50 86 51 84   86A4
9120:  52 BA 86 53 68 85 54 60 A9 00 8D 14 91 AD 15 91   8D58
9130:  85 0A AD 16 91 85 0B A0 02 B9 17 91 91 0A 88 10   9301
9140:  F8 60 F0 25 C9 2D D0 F9 20 4C E9 84 04 85 05 8A   9B1E
9150:  F0 EF A0 00 B9 03 00 99 14 91 B1 04 99 17 91 B9   A246
9160:  11 91 91 04 C8 C0 03 D0 EB 6C 02 00 D0 56 2C 82   A905
9170:  BB 50 04 38 6E F1 02 A5 35 C9 44 D0 4B A9 18 85   AFF5
9180:  80 20 3D 96 20 D4 CC A0 00 B1 02 20 0F 96 20 D4   B634
9190:  CC C8 C0 08 D0 F3 A0 00 B1 02 C9 7E B0 04 C9 20   BE8A
91A0:  B0 02 A9 7E 20 D9 CC C8 C0 08 D0 EC A5 02 69 07   C688
91B0:  85 02 90 02 E6 03 20 F0 CB C6 80 D0 C4 20 E8 C5   CF0F
91C0:  C9 20 F0 B9 4E F1 02 60 A9 18 48 20 9A 95 20 2E   D5E8
91D0:  96 85 02 84 03 20 F0 CB 68 38 E9 01 D0 EC 20 E8   DDB5
91E0:  C5 C9 20 F0 E3 D0 DD C9 2D D0 5D 20 4C E9 84 04   E6E3
91F0:  85 05 20 E8 00 C9 3E D0 4F 20 4C E9 84 06 85 07   ED06
9200:  8A F0 45 38 A5 04 E5 02 85 08 A5 05 E5 03 85 09   F33A
9210:  90 36 A2 05 B5 02 95 72 CA 10 F9 A5 07 C5 03 90   FA3C
9220:  0A D0 28 A5 06 C5 02 F0 1E B0 20 A0 00 B1 02 91   0072
9230:  06 E6 06 D0 02 E6 07 A5 02 C5 04 A5 03 E5 05 E6   070B
9240:  02 D0 02 E6 03 90 E6 60 4C 9F FA A5 06 65 08 85   0E20
9250:  C7 A5 07 65 09 85 C8 A5 04 69 01 85 C9 A5 05 69   14C2
9260:  00 85 CA A5 02 85 CE A5 03 85 CF 4C FB C3 D0 DB   1DB9
9270:  A5 02 D0 D4 85 06 85 0A A5 03 C9 04 90 CA C9 AD   2563
9280:  B0 C6 85 07 85 0B 38 E5 01 B0 04 49 FF 69 01 C9   2C42
9290:  08 90 B5 18 A5 01 85 03 69 07 85 05 A9 FF 85 04   3200
92A0:  20 03 92 18 A5 0A 85 02 69 DD 85 04 A5 0B 85 03   370A
92B0:  69 06 85 05 20 CC 92 6C 0A 00 C9 2D D0 B0 20 4C   3CD9
92C0:  E9 84 04 85 05 4C CC 92 85 02 84 03 A0 02 B1 02   42E1
92D0:  99 79 00 88 10 F8 20 51 96 A6 61 E0 02 D0 2C A5   4A14
92E0:  75 C5 7B 90 26 D0 06 A5 74 C5 7A 90 1E A5 7A E5   525F
92F0:  72 85 70 A5 7B E5 73 85 71 90 10 1B A0 01 A5 76   59A8
```

L'ELM pourra être appelé par G9200 <R>
Sa sortie est prévue vers le Basic.
Pour obtenir un retour au moniteur, vous changerez l'instruction
en 9237 (Atmos) ou 9228 (Oric-1) à l'aide du mini-assembleur.
Entrez la commande A9237 <R> pour Atmos, A9228 <R> pour Oric-1;
à l'adresse affichée, instrivez l'instruction JMP $F8B8 (ou F888).
Comme ceci:

```
ATMOS:    A9237 <R>            ORIC-1:   A9228 <R>
          9237:JMPF8B8 <R>               9228:JMPF888 <R>
```

Nouveau RETURN pour quitter le mini-assembleur.

L'ensemble sera sauvegardé par `CSAVE"",A#0A00,E#97FF`

COMMANDE (H)EXA, ENTREE DU CODE HEXADECIMAL

syntaxe: **H adresse** <RETURN>
ou H seulement à partir du mini-assembleur.

L'adresse spécifiée est affichée suivie de 'hex:'.
Seuls les chiffres hexadécimaux sont acceptés.
Le curseur passe automatiquement à la position suivante après l'entrée de deux chiffres.
La routine exige un deuxième chiffre avant d'autoriser les commandes d'édition.
Les flèches droite et gauche déplacent le curseur dans les limites du code affiché.
Les flèches haut et bas assurent l'insertion ou la suppression avec le décalage correspondant.
La flèche haut insére un NOP qui peut être remplacé par le code approprié.
La flèche bas supprime le code sous le curseur
Les valeurs binaires sont rangées temporairement dans le buffer d'entrée (80 octets maximum par passe).
Elles sont inscrites en mémoire par l'action sur RETURN ou sur ESC
- RETURN, introduction d'une nouvelle série dont l'adresse de début est affichée.
- ESC, sortie de la routine, retour au moniteur ou à l'assembleur.

COMMANDE (K)AR, ENTREE DU CARACTERE ASCII

syntaxe: **K adresse** <RETURN>
ou Kseulement à partir du mini-assembleur.

L'adresse spécifiée est affichée suivie de 'txt:'.
Le mode d'emploi est analogue à celui de la commande précédente.
Les flèches droite et gauche déplacent le curseur.
La flèche haut insére un espace.
La flèche bas supprime le caractère sous le curseur
RETURN valide le texte et propose une suite.
ESC fait sortir de la routine.

```
9600:  90 F2 20 31 96 AA E8 D0 01 C8 98 20 0F 96 8A 48   CBF5
9610:  4A 4A 4A 4A 20 1A 96 68 29 0F 09 30 C9 3A 90 02   D05B
9620:  69 06 4C D9 CC A2 03 20 D4 CC CA D0 FA 60 38 A5   D8F1
9630:  61 A4 03 AA 10 01 88 65 02 90 01 C8 60 A4 03 A6   DEA9
9640:  02 4C 0A 96 20 3D 96 A9 2D 20 D9 CC 20 25 96 A1   E4A1
9650:  02 A8 4A 90 09 6A F0 15 C9 A2 F0 11 29 87 4A AA   EB6D
9660:  BD E9 96 90 04 4A 4A 4A 4A 29 0F D0 04 A0 80 A9   F23A
9670:  00 AA BD 2D 97 85 60 29 03 85 61 98 29 8F AA 98   F8EE
9680:  A0 03 E0 8A F0 0B 4A 90 08 4A 4A 09 20 88 D0 FA   FFE7
9690:  C8 88 D0 F2 60 A2 0B BD DD 96 9D 81 BB CA D0 F7   0AA0
96A0:  86 08 A5 01 85 09 A9 0D 85 80 A9 20 85 FF 20 C2   114C
96B0:  96 C6 08 18 A5 09 69 07 85 09 A2 12 86 80 A9 2D   1704
96C0:  85 FF A2 FF 86 2F E8 A5 09 20 93 D9 A5 08 20 93   1F60
96D0:  D9 A9 00 9D 00 01 A8 A9 FF A6 80 4C 65 F8 20 20   26DF
96E0:  20 4D 6F 6E 69 74 65 75 72 04 20 54 30 0D 80 04   2B8B
96F0:  90 03 22 54 33 0D 80 04 90 04 20 54 33 0D 80 04   2F24
9700:  90 04 20 54 3B 0D 80 04 90 00 22 44 33 0D C8 44   333A
9710:  00 11 22 44 33 0D C8 44 A9 01 22 44 33 0D 80 04   36D1
9720:  90 01 22 44 33 0D 80 04 90 26 31 87 9A 00 21 81   3B36
9730:  82 00 00 59 4D 91 92 86 4A 85 9D 2C 29 2C 23 28   403F
9740:  24 59 00 58 24 24 00 1C 8A 1C 23 5D 8B 1B A1 9D   4482
9750:  8A 1D 23 9D 8B 1D A1 00 29 19 AE 69 A8 19 23 24   4993
9760:  53 1B 23 24 53 19 A1 00 1A 5B 5B A5 69 24 24 AE   4E29
9770:  AE A8 AD 29 00 7C 00 15 9C 6D 9C A5 69 29 53 84   5499
9780:  13 34 11 A5 69 23 A0 D8 62 5A 48 26 62 94 88 54   5A96
9790:  44 C8 54 68 44 E8 94 00 B4 08 84 74 B4 28 6E 74   6190
97A0:  F4 CC 4A 72 F2 A4 8A 00 AA A2 A2 74 74 74 72 44   6A2C
97B0:  68 B2 32 B2 00 22 00 1A 1A 26 26 72 72 88 C8 C4   6FC4
97C0:  CA 26 48 44 44 A2 C8 47 44 4C 4D 58 52 48 4B 41   7590
97D0:  8E 91 91 94 97 9A 9D 9D A0 41 58 59 53 43 5A 49   7D0A
97E0:  44 42 2F 56 4E 3A 78 65 68 3A 74 78 74            817C
```

MONITEUR ORIC-1

```
9000: 20 88 F8 A9 00 85 00 85 A6 8D F5 02 A9 A4 8D 31   0788
9010: 02 20 F2 00 BA BD 00 01 85 01 85 A7 BD F6 02 8D   0DD8
9020: 32 02 A9 4C 8D 30 02 20 3A C7 20 95 96 A9 FF 8D   1461
9030: 0C 02 20 29 F7 20 9F CB A9 2A 20 12 CC 20 A2 C5   1A91
9040: 86 E9 84 EA 20 E2 00 D0 08 A9 40 8D 30 02 4C 82   218E
9050: F8 C9 49 D0 0A AD 82 BB 49 69 8D 82 BB D0 CE A2   2B48
9060: 09 CA 30 C9 DD C7 97 D0 F8 86 80 20 13 E8 84 02   33BE
9070: 85 03 8A F0 B8 A6 80 20 85 90 AD 14 91 F0 03 20   3B38
9080: 28 91 4C 03 90 A5 01 48 BD D0 97 48 4C E8 00 4C   41AA
9090: 42 91 4C 6C 91 4C E7 91 4C 6E 92 4C BA 92 4C 18   48D2
90A0: 93 4C FB 94 B5 16 68 48 29 10 F0 0A 58 A5 16 28   4EF9
90B0: 20 1A 91 38 B0 0A A5 16 40 20 1A 91 20 28 91 38   538D
90C0: 68 E9 02 85 02 68 E9 00 85 03 20 9F CB 20 44 96   59C4
90D0: A2 00 BD D9 97 20 12 CC A9 3D 20 12 CC B5 50 20   609A
90E0: 0F 96 20 0D CC E8 E0 04 D0 E8 20 0D CC A2 07 BD   681B
90F0: DD 97 20 12 CC CA 10 F7 20 9F CB A2 1D 20 27 96   6F84
9100: A2 08 06 54 A9 30 69 00 20 12 CC CA D0 F4 4C 7A   761C
9110: 90 20 B9 90 00 FF FF FF FF FF 08 85 50 86 51 84   7F48
9120: 52 BA 86 53 68 85 54 60 A9 00 8D 14 91 AD 15 91   85FC
9130: 85 0A AD 16 91 85 0B A0 02 B9 17 91 91 0A B8 10   8BA5
9140: F8 60 F0 25 C9 2D D0 F9 20 13 E8 84 04 85 05 8A   9388
9150: F0 EF A0 00 B9 03 00 99 14 91 B1 04 99 17 91 B9   9AB0
9160: 11 91 91 04 C8 C0 03 D0 EB 6C 02 00 D0 56 2C 82   A16F
9170: BB 50 04 38 6E F1 02 A5 35 C9 44 D0 4B A9 18 85   A85F
9180: 80 20 3D 96 20 0D CC A0 00 B1 02 20 0F 96 20 0D   AD10
9190: CC C8 C0 08 D0 F3 A0 00 B1 02 C9 7E B0 04 C9 20   B566
91A0: B0 02 A9 7E 20 12 CC C8 C0 08 D0 EC A5 02 69 07   BCA0
91B0: 85 02 90 02 E6 03 20 9F CB C6 80 D0 C4 20 F8 C5   C4E3
91C0: C9 20 F0 B9 4E F1 02 60 A9 18 48 20 9A 95 20 2E   CBBC
91D0: 96 85 02 84 03 20 9F CB 68 38 E9 01 D0 EC 20 F8   D348
91E0: C5 C9 20 F0 E3 D0 DD C9 2D D0 5D 20 13 E8 84 04   DC3C
91F0: 85 05 20 E8 00 C9 3E D0 4F 20 13 E8 84 06 85 07   E225
9200: 8A F0 45 38 A5 04 E5 02 85 08 A5 05 E5 03 85 09   E859
9210: 90 36 A2 05 B5 02 95 72 CA 10 F9 A5 07 C5 03 90   EF5B
9220: 0A D0 28 A5 06 C5 02 F0 1E B0 20 A0 00 B1 02 91   F591
9230: 06 E6 06 D0 02 E6 07 A5 02 C5 04 A5 03 E5 05 E6   FC2A
9240: 02 D0 02 E6 03 90 E6 60 4C 85 FA A5 06 65 08 85   0325
9250: C7 A5 07 65 09 85 C8 A5 04 69 01 85 C9 A5 05 69   09C7
9260: 00 85 CA A5 02 85 CE A5 03 85 CF 4C FF C3 D0 D8   12C2
9270: A5 02 D0 D4 85 06 85 0A A5 03 C9 04 90 CA C9 AD   1A6C
9280: B0 C6 85 07 85 0B 38 E5 01 B0 04 49 FF 69 01 C9   214B
9290: 08 90 B5 18 A5 01 85 03 69 07 85 05 A9 FF 85 04   2709
92A0: 20 03 92 18 A5 0A 85 02 69 DD 85 04 A5 0B 85 03   2C13
92B0: 69 06 85 05 20 CC 92 6C 0A 00 C9 2D D0 B0 20 13   31A9
92C0: E8 B4 04 85 05 4C CC 92 85 02 84 03 A0 02 B1 02   37B0
92D0: 99 79 00 88 10 F8 20 51 96 A6 61 E0 02 D0 2C A5   3EE3
92E0: 75 C5 7B 90 26 D0 06 A5 74 C5 7A 90 1E A5 7A E5   472E
92F0: 72 85 70 A5 7B E5 73 85 71 90 10 18 A0 01 A5 76   4E77
```

```
9300:  65 70 91 02 C8 A5 77 65 71 91 02 20 2E 96 48 C5   551D
9310:  04 98 E5 05 68 90 B1 60 D0 FD A5 35 C9 48 D0 01   5D35
9320:  0A 85 08 20 3D 96 A2 04 24 08 10 05 BD E4 97 D0   62AE
9330:  03 BD E8 97 20 12 CC CA D0 EE 86 04 86 09 20 D1   6A7D
9340:  F8 A9 02 85 05 A9 00 85 07 20 F8 C5 24 08 10 09   7001
9350:  A4 05 C0 02 F0 0C 4C 30 94 C9 14 D0 05 20 12 CC   7628
9360:  D0 E7 C9 08 D0 08 E0 00 F0 43 20 77 94 CA 4C 49   7E2B
9370:  93 C9 1B D0 05 38 66 09 30 04 C9 0D D0 2B 86 06   83AC
9380:  A0 00 A2 FF E8 E4 06 90 05 A9 09 20 77 94 E4 04   8B19
9390:  F0 0C B5 35 91 02 E6 02 D0 02 E6 03 D0 E6 20 9F   92AA
93A0:  CB 24 09 30 03 4C 23 93 60 E0 50 D0 0A 86 06 20   97ED
93B0:  85 FA A6 06 4C 49 93 E4 04 F0 64 C9 09 D0 07 20   9F45
93C0:  77 94 E8 4C 49 93 C9 0A D0 25 CE 6A 02 86 06 C6   A6B4
93D0:  04 E4 04 F0 09 B5 36 95 35 20 67 94 D0 F3 20 75   ADC1
93E0:  94 E8 A9 08 20 77 94 CA E4 06 D0 F8 4C 3E 93 C9   B67B
93F0:  0B D0 2C 86 06 A5 04 C9 50 F0 B4 AA CE 6A 02 CA   BE22
9400:  B5 35 95 36 E4 06 D0 F7 A9 20 24 0B 10 02 A9 EA   C522
9410:  95 35 E6 04 B5 35 20 67 94 E4 04 D0 F7 F0 BF C9   CE02
9420:  20 90 8A C9 7E B0 86 24 08 30 05 20 12 CC D0 2B   D413
9430:  85 06 49 30 C9 0A 90 06 69 88 C9 FA 90 E3 A0 03   DB4A
9440:  0A 0A 0A 0A 0A 26 07 88 10 FA A5 06 20 12 CC C6   DFAA
9450:  05 F0 03 4C 49 93 20 0D CC A5 07 95 35 E4 04 D0   E5F1
9460:  02 E6 04 E8 4C 41 93 24 08 10 05 20 0F 96 A9 20   EAB4
9470:  20 12 CC E8 60 A9 20 A0 01 24 08 10 02 A0 03 48   EF8D
9480:  20 12 CC C9 20 F0 08 AD 69 02 29 FE D0 01 C8 68   F6AC
9490:  88 D0 EC 60 E9 81 4A D0 5F A4 68 A6 67 D0 01 88   FFA5
94A0:  CA 8A 18 E5 02 85 67 10 01 C8 98 E5 03 D0 49 A4   06FA
94B0:  61 B9 66 00 91 02 88 10 F8 A9 0B 20 12 CC 20 9A   0D09
94C0:  95 20 9F CB 20 2E 96 85 02 84 03 4C FB 94 A5 66   1400
94D0:  20 51 96 AA BD 07 97 C5 63 D0 13 BD 47 97 C5 62   1C59
94E0:  D0 0C A5 64 A4 60 C0 9D F0 AA C5 60 F0 C1 C6 66   263B
94F0:  D0 DC E6 64 C6 65 F0 D6 20 85 FA 20 3D 96 A9 3A   2F97
9500:  20 12 CC 20 A2 C5 86 E9 84 EA A5 35 D0 06 60 20   3729
9510:  18 93 30 E7 C9 48 F0 F7 C9 4B F0 F3 A2 03 20 E2   4081
9520:  00 0A E9 BD C9 C3 90 D0 0A 0A A0 04 0A 26 63 26   468E
9530:  62 88 10 F8 CA F0 F5 10 E5 A2 05 20 E2 00 DD 3B   4EE5
9540:  97 D0 15 20 E2 00 F0 11 DD 41 97 F0 0E BD 41 97   56AC
9550:  F0 07 C9 24 F0 03 C6 E9 18 C6 E9 26 64 E0 03 D0   5F36
9560:  17 20 13 E8 84 67 85 68 8A F0 07 A5 68 F0 01 E8   66A7
9570:  E8 E8 86 65 A2 03 C6 E9 86 66 CA 10 BE A5 64 0A   6F4D
9580:  0A 05 65 C9 20 B0 06 A6 65 F0 02 09 80 85 64 20   74EF
9590:  E2 00 D0 03 4C CE 94 4C F8 94 20 44 96 48 B1 02   7C1F
95A0:  20 0F 96 A2 01 20 27 96 C4 61 C8 90 F1 A2 03 C0   8337
95B0:  04 90 F2 68 A8 B9 47 97 85 62 B9 87 97 85 63 A9   8BB3
95C0:  00 A0 05 06 63 26 62 2A 88 D0 F8 69 3F 20 12 CC   9169
95D0:  CA D0 EC 20 0D CC A4 61 A2 06 E0 03 F0 1C 06 60   98EA
95E0:  90 0E BD 3A 97 20 12 CC BD 40 97 F0 03 20 12 CC   9F99
95F0:  CA D0 E7 60 88 30 E7 20 0F 96 A5 60 C9 E8 B1 02   A847
```

```
9600: 90 F2 20 31 96 AA E8 D0 01 C8 98 20 0F 96 BA 48   B00A
9610: 4A 4A 4A 4A 20 1A 96 68 29 0F 09 30 C9 3A 90 02   B470
9620: 69 06 4C 12 CC A2 03 20 0D CC CA D0 FA 60 38 A5   BB78
9630: 61 A4 03 AA 10 01 88 65 02 90 01 C8 60 A4 03 A6   C130
9640: 02 4C 0A 96 20 3D 96 A9 2D 20 12 CC 20 25 96 A1   C661
9650: 02 A8 4A 90 09 6A B0 15 C9 A2 F0 11 29 87 4A AA   CD2D
9660: BD E9 96 90 04 4A 4A 4A 4A 29 0F D0 04 A0 80 A9   D3FA
9670: 00 AA BD 2D 97 85 60 29 03 85 61 98 29 8F AA 98   DAAE
9680: A0 03 E0 8A F0 0B 4A 90 08 4A 4A 09 20 88 D0 FA   E1A7
9690: C8 88 D0 F2 60 A2 0B BD DD 96 9D 81 BB CA D0 F7   EC60
96A0: 86 08 A5 01 85 09 A9 0D 85 80 A9 20 85 FF 20 C2   F30C
96B0: 96 C6 08 18 A5 09 69 07 85 09 A2 12 B6 80 A9 2D   F8C4
96C0: 85 FF A2 FF 86 2F E8 A5 09 20 F5 D8 A5 08 20 F5   01E3
96D0: D8 A9 00 9D 00 01 A8 A9 FF A6 80 4C 2F F8 20 20   092B
96E0: 20 4D 6F 6E 69 74 65 75 72 04 20 54 30 0D 80 04   0DD7
96F0: 90 03 22 54 33 0D 80 04 90 04 20 54 33 0D 80 04   1170
9700: 90 04 20 54 3B 0D 80 04 90 00 22 44 33 0D C8 44   1586
9710: 00 11 22 44 33 0D C8 44 A9 01 22 44 33 0D 80 04   191D
9720: 90 01 22 44 33 0D 80 04 90 26 31 87 9A 00 21 81   1D82
9730: 82 00 00 59 4D 91 92 86 4A 85 9D 2C 29 2C 23 28   228B
9740: 24 59 00 58 24 24 00 1C 8A 1C 23 5D 8B 1B A1 9D   26CE
9750: 8A 1D 23 9D 8B 1D A1 00 29 19 AE 69 A8 19 23 24   2BDF
9760: 53 1B 23 24 53 19 A1 00 1A 5B 5B A5 69 24 24 AE   3075
9770: AE A8 AD 29 00 7C 00 15 9C 6D 9C A5 69 29 53 84   36E5
9780: 13 34 11 A5 69 23 A0 D8 62 5A 48 26 62 94 88 54   3CE2
9790: 44 C8 54 68 44 E8 94 00 B4 08 84 74 B4 28 6E 74   43DC
97A0: F4 CC 4A 72 F2 A4 8A 00 AA A2 A2 74 74 74 72 44   4C78
97B0: 68 B2 32 B2 00 22 00 1A 1A 26 26 72 72 88 C8 C4   5210
97C0: CA 26 48 44 44 A2 C8 47 44 4C 4D 58 52 48 4B 41   57DC
97D0: 8E 91 91 94 97 9A 9D 9D A0 41 58 59 53 43 5A 49   5F56
97E0: 44 42 2F 56 4E 3A 78 65 68 3A 74 78 74            63C8
```

MODE D'EMPLOI

Appelez le Moniteur; CALL#9000.
Par la suite, le vecteur #2F5,2F6 étant renseigné, vous pourrez utiliser la commande Basic !
La zone occupée par le moniteur est affichée sur la ligne supérieure.
En réalité, il se termine en #97EC mais, pour simplifier, toute la dernière page est prise en compte (voir en #929C, p.68 et en #96B1, p.83).

Le symbole * indique qu'il attend l'entrée d'une commande.
A défaut de l'entrée d'une commande valide, il y a retour au Basic.

A ce propos, ouvrons une parenthèse.
En examinant le listing, en #9027 page 60, on trouve un JSR CLEARC (Atmos #C70F, Oric-1 #C73A).
Cette instruction, exécutée après l'initialisation des pointeurs du programme, a deux buts:
- Réinitialisation de la pile
- Ajustement du pointeur Basic FRETOP (#A2,A3) associé à HIMEM.

Si la deuxième opération n'est pas effectuée avant retour au Basic, une chaine de caractères, définie en commande directe, telle que PRINT HEX$(DEEK(...)) par exemple, sera inscrite à l'ancien FRETOP, c'est-à-dire généralement dans le code que l'on croyait protégé par le nouveau HIMEM.

L'instruction JSR CLEARC supprime cet inconvénient mais elle en présente un autre, celui d'effacer les variables Basic. Ce n'est pas toujours souhaitable.
A l'aide du mini-assembleur, vous pouvez la remplacer par JSR STKINI (Atmos JSR C726, Oric-1 JSR C751), qui se contente de réinitialiser la pile.

Autre remarque. Le RTI est remis en place en #24A ou #230 avant retour au Basic. La gestion du BRK n'est donc plus assurée pour une routine en langage machine actionnée à partir du Basic par CALL.
Fermons la parenthèse.

Appuyez sur RETURN, puis sur le point d'exclamation ! suivi de RETURN.

La syntaxe générale d'une commande est son initiale suivie d'une ou plusieurs adresses en hexadécimal, sans le symbole #.

La commande complète est inscrite par INLIN dans le buffer d'entrée.
Elle est lue à l'aide de CHRGET, la ou les adresses étant converti(s) par HEXGET.
C'est également vrai pour le recueil et l'analyse par l'assembleur d'une instruction en langage d'assemblage.

Voyons les inconvénients légers que présente l'utilisation de ces routines Basic:
- INLIN: les caractères de contrôle restent actifs. L'entrée est limitée à 78 caractères. Ce ne sont pas vraiment des inconvénients.
- HEXGET: il y a retour au Basic avec message d'erreur si l'adresse entrée (associée à une commande ou à l'opérande d'une instruction) est erronée (supérieure à FFFF, OVERFLOW).
C'est plus gênant, bien qu'il suffise d'actionner la commande Basic ! pour revenir au moniteur.

BASCULE D'IMPRESSION

Elle est obtenue en appuyant sur la touche I puis sur RETURN.
La même opération affiche ou efface la lettre I sur la ligne supérieure.
Quand cette lettre-témoin est présente, la produit de (L)ist ou (D)ump, ci-après, est sorti sur imprimante.
Cette dernière doit être branchée et sous tension.

COMMANDE (L)IST, DESASSEMBLAGE

syntaxe: L adresse <RETURN>

Cette commande affiche 24 lignes du listing de désassemblage commençant à l'adresse spécifiée.
L'action sur la barre d'espacement fait apparaître les 24 lignes suivantes.
RETURN provoque le retour au moniteur.
La procédure est identique en mode impression.

COMMANDE (D)DUMP, VIDAGE HEXA ET ASCII

syntaxe: D adresse <RETURN>

Le contenu de la mémoire à partir de l'adresse spécifiée est affiché en hexadécimal sur 24 lignes.
Les caractères ASCII correspondants apparaissent sur la même ligne.
Les caractères de contrôle ainsi que les caractères en ASCII négatif sont ignorés et remplacés par un carré gris.
L'action sur la barre d'espacement fait apparaître les 24 lignes suivantes.
RETURN provoque le retour au moniteur.
La procédure est identique en mode impression.

COMMANDE (G)O, EXECUTION PROGRAMME MACHINE

syntaxe: G adresse <RETURN>

Lancement d'un programme en langage machine.
Un RTS en fin de programme assurera le retour au moniteur.

Cette exécution 'sous contrôle' permet l'utilisation de l'instruction BRK (00) qui n'est normalement pas reconnue par le système.
Lorsqu'elle est rencontrée, le programme est interrompu avec retour immédiat au moniteur.
L'adresse du BRK est affichée ainsi que le contenu des registres au moment de l'interruption.
Les 7 indicateurs du registre d'état P apparaissent en clair; présentation plus explicite qu'une valeur hexadécimale.

Une facilité associée à la commande (G)o permet l'introduction d'un point d'arrêt (BRK simulé) à un endroit choisi du programme.
Utile pour la mise au point, elle évite d'avoir à placer un BRK qu'il faut ensuite retrouver et supprimer.
Le résultat est exactement le même.

syntaxe: G adresse du programme - adresse du point d'arrêt <RETURN>

A l'adresse d'arrêt choisie, obligatoirement en mémoire vive, le code est remplacé par une instruction d'aiguillage sur une routine spéciale.

S'il passe par ce point au cours de l'exécution, le programme est remis dans son état initial puis branché sur la routine d'interruption par BRK.
La non-utilisation d'un point d'arrêt est détectée avant le retour au moniteur et, dans ce cas aussi, le programme est remis en état (voir p.64).

Qu'il s'agisse de l'instruction BRK ou du BRK simulé, les mêmes précautions concernant le choix de l'adresse d'interruption sont à observer.
Si le BRK est mal placé, au milieu d'une instruction JSR ou JMP par exemple, le pire peut arriver.

COMMANDE (M)OVE, DEPLACEMENT D'OCTETS

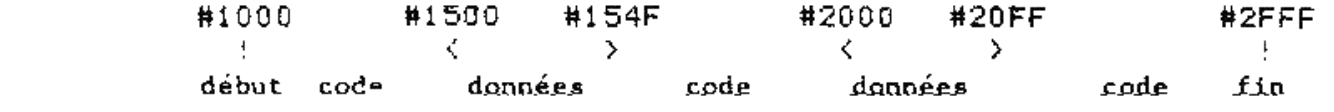
syntaxe: **M adresse de début - adresse de fin > destination** <RETURN>

Cette commande permet de déplacer un bloc de code vers le haut ou vers le bas dans toute la mémoire vive.
Il n'y a pas de limitation particulière. La seule condition est que le bloc source existe, c'est-à-dire que l'adresse de fin soit supérieure ou égale à l'adresse de début.
Evidemment, la nouvelle zone occupée ne devra pas empiéter sur la mémoire morte.
L'utilisateur doit s'assurer que le code à déplacer ne risque pas d'écraser des valeurs à préserver (adresses du système ou programme existant).

COMMANDE (R)ELOC, RELOGEMENT DU CODE DEPLACE

syntaxe: **R adresse de début du code à reloger - adresse de fin** <RETURN>

Cette commande n'est pas utilisable isolément. Elle doit suivre immédiatement une commande (M)ove dont elle est un complément éventuel.
Les adresses de début et de fin du bloc source ainsi que l'adresse de destination sont copiés par (M)ove, à son usage, dans des pointeurs situés dans le buffer d'entrée (à partir de £72).
C'est une zone vulnérable, aussi il convient de ne pas taper n'importe quoi au clavier entre les deux commandes et de s'en tenir à la syntaxe stricte, sans espaces inutiles, pour (R)eloc.
Prenons un exemple concret:
Soit un programme situé en £1000-2FFF à transférer en £4000 et à reloger.
Des blocs de données (tables ou messages) sont insérés dans le code actif selon le schéma suivant:



Les données ne doivent pas être traitées par la routine de relogement.
La séquence des commandes sera:
- **M1000-2FFF>4000**
- **R4000-44FF**
- R4550-4FFF
- R5100-5FFF

LIMITATIONS

Le début du relogement doit correspondre au premier octet d'une instruction (code opératoire).
Seul l'adressage absolu est traité.
Les adresses spécifiées en mode immédiat, à passer à une routine par exemple, ne sont pas modifiées (voir p.11 STROUT).
Il est nécessaire de connaître la structure du programme à reloger (instructions et données). Ce n'est pas toujours facile.

COMMANDE (X)MON, DEPLACEMENT DU MONITEUR

syntaxe: **X adresse <RETURN>**

Le moniteur s'implante à l'adresse spécifiée.
Il utilise pour se déplacer et se reloger, les routines des commandes précédentes.
Il est immédiatement opérationnel à son nouvel emplacement et protégé par l'ajustement automatique de HIMEM.

Limitations:
L'adresse choisie doit être celle d'un début de page,
exemples: X500, X2000, X8A00.
Le déplacement doit être supérieur ou égal à la longueur du moniteur (2 K.octets)
La zone que ce dernier peut occuper s'étend des adresses #400 à #B3FF.
Présence de disque, programme Basic, utilisation de la haute résolution etc, seront des facteurs à considérer lors du choix de l'adresse d'implantation.

APPLICATION PRATIQUE

Utilisation conjointe du Moniteur et de l'Editeur de Langage Machine.

Le moniteur dispose d'une commande permettant l'entrée en mémoire du code hexadécimal mais il est intéressant d'avoir sous la main un moyen plus performant.
On placera l'éditeur de langage machine à la suite du moniteur.

Pour éviter d'oblitérer la mémoire d'écran haute résolution, l'ensemble sera installé entre les adresses #8A00 et #97FF.
D'abord, il faut mettre en place le moniteur. Cela ne peut pas être fait directement; il faut passer par un emplacement intermédiaire.

Voici la procédure complète:
- Entrez successivement les commandes: X8000 puis X8A00 <R>
- RETURN à nouveau pour revenir au Basic.
- Chargez l'éditeur de langage machine (normalement en £6000)
- Revenez au moniteur avec la commande !.
- Vérifiez la présence de l'ELM: L6000 <R>
- Retour au moniteur par RETURN.
- Entrez les commandes de transfert et de relogement:
  M6000-65FF>9200 <R>
  R9200-9723 <R> (R9200-96F9 pour l'Oric-1)

```
9300: 65 70 91 02 C8 A5 77 65 71 91 02 20 2E 96 48 C5   604E
9310: 04 98 E5 05 68 90 B1 60 D0 FD A5 35 C9 48 D0 01   6866
9320: 0A 85 08 20 3D 96 A2 04 24 08 10 05 BD E4 97 D0   6DDF
9330: 03 BD E8 97 20 D9 CC CA D0 EE 86 04 86 09 20 0E   75B2
9340: F9 A9 02 85 05 A9 00 85 07 20 E8 C5 24 08 10 09   7B27
9350: A4 05 C0 02 F0 0C 4C 30 94 C9 14 D0 05 20 D9 CC   8215
9360: D0 E7 C9 08 D0 08 E0 00 F0 43 20 77 94 CA 4C 49   8A15
9370: 93 C9 18 D0 05 38 66 09 30 04 C9 0D D0 28 86 06   8F99
9380: A0 00 A2 FF E8 E4 06 90 05 A9 09 20 77 94 E4 04   9706
9390: F0 0C B5 35 91 02 E6 02 D0 02 E6 03 D0 E6 20 F0   9EE8
93A0: CB 24 09 30 03 4C 23 93 60 E0 50 D0 0A 86 06 20   A42B
93B0: 9F FA A6 06 4C 49 93 E4 04 F0 64 C9 09 D0 07 20   AB9D
93C0: 77 94 E8 4C 49 93 C9 0A D0 25 CE 6A 02 86 06 C6   B30C
93D0: 04 E4 04 F0 09 B5 36 95 35 20 67 94 D0 F3 20 75   BA19
93E0: 94 E8 A9 08 20 77 94 CA E4 06 D0 F8 4C 3E 93 C9   C2D3
93F0: 0B D0 2C 86 06 A5 04 C9 50 F0 B4 AA CE 6A 02 CA   CA7A
9400: B5 35 95 36 E4 06 D0 F7 A9 20 24 08 10 02 A9 EA   D17A
9410: 95 35 E6 04 B5 35 20 67 94 E4 04 D0 F7 F0 BF C9   DA5A
9420: 20 90 8A C9 7E B0 86 24 08 30 05 20 D9 CC D0 2B   E132
9430: 85 06 49 30 C9 0A 90 06 69 88 C9 FA 90 E3 A0 03   E869
9440: 0A 0A 0A 0A 0A 26 07 88 10 FA A5 06 20 D9 CC C6   ED90
9450: 05 F0 03 4C 49 93 20 D4 CC A5 07 95 35 E4 04 D0   F49E
9460: 02 E6 04 E8 4C 41 93 24 08 10 05 20 0F 96 A9 20   F961
9470: 20 D9 CC E8 60 A9 20 A0 01 24 08 10 02 A0 03 20   FED9
9480: D9 CC 88 D0 FA 60 EA EA EA EA EA EA EA EA EA EA   0C54
9490: EA EA EA EA E9 81 4A D0 5F A4 68 A6 67 D0 01 88   1651
94A0: CA 8A 18 E5 02 85 67 10 01 C8 98 E5 03 D0 49 A4   1DA6
94B0: 61 B9 66 00 91 02 88 10 FB A9 0B 20 D9 CC 20 9A   247C
94C0: 95 20 F0 CB 20 2E 96 85 02 84 03 4C FB 94 A5 66   2BC4
94D0: 20 51 96 AA BD 87 97 C5 63 D0 13 BD 47 97 C5 62   341D
94E0: D0 0C A5 64 A4 60 C0 9D F0 AA C5 60 F0 C1 C6 66   3DFF
94F0: D0 DC E6 64 C6 65 F0 D6 20 9F FA 20 3D 96 A9 3A   4775
9500: 20 D9 CC 20 92 C5 B6 E9 84 EA A5 35 D0 06 60 20   4FBE
9510: 18 93 30 E7 C9 48 F0 F7 C9 4B F0 F3 A2 03 20 E2   5916
9520: 00 0A E9 BD C9 C3 90 D0 0A 0A A0 04 0A 26 63 26   5F23
9530: 62 88 10 F8 CA F0 F5 10 E5 A2 05 20 E2 00 DD 3B   677A
9540: 97 D0 15 20 E2 00 F0 11 DD 41 97 F0 0E BD 41 97   6F41
9550: F0 07 C9 24 F0 03 C6 E9 18 C6 E9 26 64 E0 03 D0   77CB
9560: 17 20 4C E9 84 67 85 68 8A F0 07 A5 68 F0 01 E8   7F76
9570: E8 E8 86 65 A2 03 C6 E9 86 66 CA 10 BE A5 64 0A   881C
9580: 0A 05 65 C9 20 B0 06 A6 65 F0 02 09 80 85 64 20   8DBE
9590: E2 00 D0 03 4C CE 94 4C F8 94 20 44 96 48 B1 02   94EE
95A0: 20 0F 96 A2 01 20 27 96 C4 61 C8 90 F1 A2 03 C0   9C06
95B0: 04 90 F2 68 A8 B9 47 97 85 62 B9 87 97 85 63 A9   A482
95C0: 00 A0 05 06 63 26 62 2A 88 D0 F8 69 3F 20 D9 CC   AAFF
95D0: CA D0 EC 20 D4 CC A4 61 A2 06 E0 03 F0 1C 06 60   B347
95E0: 90 0E BD 3A 97 20 D9 CC BD 40 97 F0 03 20 D9 CC   B884
95F0: CA D0 E7 60 88 30 E7 20 0F 96 A5 60 C9 E8 B1 02   C432
```

COMMANDE (A)SSEMBLAGE, MINI ASSEMBLEUR

syntaxe: **A adresse de début d'assemblage** <RETURN>

L'adresse spécifiée est affichée suivie de deux-points.
Une instruction en langage d'assemblage peut être tapée suivie de RETURN.
L'assemblage en mémoire est immédiat si elle est reconnue valide.
Elle est réaffichée sur la même ligne avec son code objet tandis qu'apparaît l'adresse de l'instruction suivante.

L'action sur RETURN seul assure le retour au moniteur.

L'assembleur ne reconnaît que l'hexadécimal pour l'opérande.
Il est accommodant en matière de syntaxe. Les espaces ne sont pas obligatoires mais ne sont pas interdits, de même que le symbole $ ou les zéros non significatifs.

Les valeurs 'immédiates' doivent être précédees du signe #.
Le nom de l'accumulateur, 'A', doit être omis dans les instructions de décalage ou de rotation intéressant ce registre.

Une instruction incorrecte est signalée par Ping; l'adresse est reproduite à la ligne suivante.

Les lettres 'H' ou 'K' permettent la saisie du code hexadécimal ou du texte à partir de l'adresse affichée.
La procédure est alors la même que pour les commandes correspondantes.
Seule différence, ESC ramène à l'assembleur.

Le mini-assembleur se sert de INLIN et de HEXGET pour le recueil et l'analyse du code source.
Reportez-vous à la page 53 pour les particularités liées à l'emploi de ces routines.
J'ajoute qu'en cas de retour intempestif au Basic (OVERFLOW), il est recommandé de revenir au moniteur dans tous les cas afin d'assurer une sortie ultérieure plus orthodoxe (remise en place du RTI).

```
        ;Variables pge zéro

        ;$00,01 adr.monit.
        ;$02,03 adr.1
        ;$04,05 adr.2
        ;$06,07 adr.3
        ;$08,09 ptr.tmp.1
        ;$0A,0B ptr.tmp.2
        ;$04-09 tmp. HEXTXT
        ;$16 sauv.Acc. XRTI
        ;$50-54 reg. SAVREG
        ;$60-68 Ass.Désass.
        ;$60 format instr.
        ;$61 longueur-1
        ;$62 mnémon. gauche
        ;$63 mnémon. droit
        ;$64-68 temp.ASSMB
        ;$70-7B MOVE RELOC
        ;$80 temp. divers
```

Variables spécifiques du Moniteur.

Elles figurent telles quelles dans le listing. Des étiquettes n'auraient pas amélioré la lisibilité. Les premières sont des pointeurs d'adresse, dont #00,01 qui contient l'adresse du moniteur. Les routines des commandes (excepté HEXTXT) ont leurs variables dans le buffer d'entrée
Tous ces emplacements peuvent être utilisés par un programme étranger; à l'exception de #16, si le programme est lancé à partir du moniteur. Cet emplacement sert à la sauvegarde de l'accumulateur lors des interruptions. Sa position isolée devrait le garder à l'abri d'une modification intempestive.

```
        ;ATMOS          ;ORIC-1
FLG     =$2F            ;=$2F
BUF     =$35            ;=$35
MEMSIZ  =$A6            ;=$A6
HIGHDS  =$C7            ;=$C7
HIGHTR  =$C9            ;=$C9
LOWTR   =$CE            ;=$CE
CHRGET  =$E2            ;=$E2
CHRGOT  =$E8            ;=$E8
TXTPTR  =$E9            ;=$E9
XRTS    =$F2            ;=$F2
STACK   =$100           ;=$100
CAPLCK  =$20C           ;=$20C
RTI1    =$24A           ;=$230
RTI2    =$24B           ;=$231
RTI3    =$24C           ;=$232
MODE0   =$26A           ;=$26A
PRTFLG  =$2F1           ;=$2F1
EXCLV   =$2F5           ;=$2F5
BLTU0   =$C3FB          ;=$C3FF
INLIN   =$C592          ;=$C5A2
INCHR   =$C5E8          ;=$C5F8
CLEARC  =$C70F          ;=$C73A
STKINI  =$C726          ;=$C751
CRDO    =$CBF0          ;=$CB9F
OUTSPC  =$CCD4          ;=$CC0D
OUTDO   =$CCD9          ;=$CC12
BINHEX  =$D993          ;=$D8F5
HEXGET  =$E94C          ;=$E813
PRCAPS  =$F75A          ;=$F729
STOUT   =$F865          ;=$F82F
RESET   =$F8B2          ;=$F882
RSET0   =$F8B8          ;=$F888
CURSON  =$F90E          ;=$F8D1
PING    =$FA9F          ;=$FA85
```

Adresses du système utilisées par le moniteur.
La plupart sont décrites page 11.

- MEMSIZ, la plus haute adresse utilisable en Basic est MEMSIZ-1.
- CHRGOT, même effet que CHRGET sans incrémentation préalable de TXTPTR.
- XTRS, RTS de CHRGET utilisé par le moniteur pour se situer.
- STACK, pile.
- CAPLCK, drapeau, FF=majuscules, 7F=minuscules.
- RTI1, RTI2 et RTI3 reçoivent un JMP sur la routine de dérivation des interruptions. RTI1 (24A ou 230) contient normalement #40 (RTI).
- MODE0, fonctions diverses, ici visualisation curseur.
- EXCLV, vecteur commande !.
- BLTU0, voir MOVUP, page 11.
- CLEARC, même effet que la commande Basic CLEAR.
- STKINI, réinitialisation de la pile.
- HEXGET, lit un nombre hexa à l'aide de CHRGET et le convertit en un entier non signé dans Y(-),A(+).
- PRCAPS, affiche ou efface CAPS selon la valeur de CAPLCK.
- CURSON, papier blanc, encre noire, vidéo active, curseur visible.

```
                        ;Entrée moniteur
                        ;auto-localisation,
                        ;vecteur commande !
                        ;dérivation RTI

9000 20 B8 F8 MONIT   JSR RSET0     Cette section inscrit l'adresse du
9003 A9 00    MONIT0  LDA #$00      moniteur dans MEMSIZ, EXCLV et #00,01
9005 85 00            STA $00       L'implantation commençant toujours à
9007 85 A6            STA MEMSIZ    un debut de page, l'octet faible est
9009 8D F5 02         STA EXCLV     00. Pour se localiser lui-même, le
900C A9 A4            LDA #-XRTI    programme doit trouver l'octet fort.
900E 8D 4B 02         STA RTI2      Le JSR XRTS empile une adresse de
9011 20 F2 00 AUTOL   JSR XRTS      retour dont l'octet fort a la valeur
9014 BA               TSX           cherchée. Cette valeur, quoique péri-
9015 BD 00 01         LDA STACK,X   mée, n'est pas effacée par le désem-
9018 85 01            STA $01       pilage immédiat (#F2 est le RTS de
901A 85 A7            STA MEMSIZ+1  CHRGET). Sa position est pointée par
901C 8D F6 02         STA EXCLV+1   S, pointeur de pile. Il suffit des
901F 8D 4C 02         STA RTI3      instructions TSX et LDA $100,X pour
9022 A9 4C            LDA #$4C      la récupérer.
9024 8D 4A 02         STA RTI1      Le RTI (retour d'interruption) en
9027 20 0F C7         JSR CLEARC    #24A (Atmos) #230 (Oric-1) est rem-
902A 20 95 96         JSR PRMLOC    placé par JMP XRTI, détection du BRK.
902D A9 FF    MONIT1  LDA #$FF      CLEARC règle les pointeurs Basic
902F 8D 0C 02         STA CAPLCK    suivant HIMEM et initialise la pile.
9032 20 5A F7         JSR PRCAPS    CAPLCK = FF, mode majuscules. PRCAPS
9035 20 F0 CB         JSR CRDO      affiche CAPS et CRDO met à la ligne.

                        ;Affiche symbole *
                        ;et appelle INLIN
                        ;entrée commande

9038 A9 2A            LDA #"*"
903A 20 D9 CC         JSR OUTDO
903D 20 92 C5         JSR INLIN

                        ;TXTPTR=BUF-1
                        ;CHRGET recueille
                        ;le 1er caractère
                        ;ret.Basic si pas
                        ;d'entrée

9040 86 E9            STX TXTPTR    Au retour de INLIN, l'adresse BUF-1
9042 84 EA            STY TXTPTR+1  (#34) détenue par X(-) et Y(+) est
9044 20 E2 00         JSR CHRGET    transférée dans TXTPTR pour CHRGET.
9047 D0 08            BNE LPRNT?    (A) <> 0 entrée détectée, -> LPRNT?
9049 A9 40    BASIC   LDA #$40      (A) = 0, pas d'entrée. Le RTI est
904B 8D 4A 02         STA RTI1      remis en place en #24A ou #230.
904E 4C B2 FB         JMP RESET     Retour Basic par RESET.
```

```
                        ;Témoin impression
                        ;bascule, ret.monit

9051 C9 49     LPRNT?   CMP #"I"       Le caractère en A est-il "I" ?
9053 D0 0A              BNE FNDCMD     non, ce doit être une commande
9055 AD 82 BB           LDA $BB82      oui, si espace (#20) en $BB82,
9058 49 69              EOR #$69       EOR #69 le change en "I" (#49)
905A 8D 82 BB           STA $BB82      et réciproquement, bascule.
905D D0 CE              BNE MONIT1     branchement forcé.

                        ;Rech.cmde ds table
                        ;recueil 1ère adr.
                        ;pas trouvé ou pas
                        ;d'adr.-> ret.monit
                        ;X sera index table
                        ;oct.faibl.adr.cmde

905F A2 09     FNDCMD   LDX #$09       Boucle de recherche dans table des
9061 CA        NXCMD    DEX            commandes: G, D, L, M, R, H, K, A
9062 30 C9              BMI MONIT1     pas trouvé, retour entrée monit.
9064 DD C7 97           CMP CMDTBL,X   comparaison
9067 D0 F8              BNE NXCMD      pas bon, essayer commande suivante
9069 86 80              STX $80        trouvée, sauvegarde index table.
906B 20 4C E9           JSR HEXGET     lecture adresse qui suit commande.
906E 84 02              STY $02        recueillie dans Y(-), A(+).
9070 85 03              STA $03        transférée dans pointeur #02,03.
9072 8A                 TXA            si X=0 ...
9073 F0 B8              BEQ MONIT1     pas d'adresse, retour entrée monit
9075 A6 80              LDX $80        sinon, index table est récupéré.

                        ;Appel s/r cmde
                        ;au retour,si BPSAV
                        ;<> 0, suppr.point
                        ;arrêt non utilisé.

9077 20 85 90  EXECMD   JSR TOSUB      Appel routine cmde, après exécution
907A AD 14 91  RETMON   LDA BPSAV      retour direct entrée moniteur si
907D F0 03              BEQ RETM0      (BPSAV) = 0. Sinon, Pt.arrêt non
907F 20 2B 91           JSR XBRKPT     utilisé est supprimé par XBRKPT.
9082 4C 03 90  RETM0    JMP MONIT0     Entrée monit. Réinit. pointeurs.

                        ;Empil.adr.JMPCMD-1
                        ;cette adr.+1 sera
                        ;adr.retour CHRGOT
                        ;qui lit le carac.
                        ;après la 1ere adr.

9085 A5 01     TOSUB    LDA $01        Octet fort JMPCMD (page suivante)
9087 48                 PHA            le même que (#01) est empilé, puis
9088 BD D0 97           LDA CMDADL,X   l'octet faible - 1, trouvé dans la
908B 48                 PHA            table, index X, l'est à son tour.
908C 4C E8 00           JMP CHRGOT     sera adresse retour CHRGOT.
```

```
                        ;Sauts vers
                        ;routines commandes

908F 4C 42 91 JMPCMD JMP GO          Exécution d'un programme machine
9092 4C 6C 91        JMP DMPLST      Vidage hexa, ASCII ou Désassemblage
9095 4C E7 91        JMP MOVE        Déplacement d'octets
9098 4C 6E 92        JMP XMON        Déplacement du moniteur
909B 4C BA 92        JMP RELOC       Relogement du code déplacé
909E 4C 18 93        JMP HEXTXT      Entrée Hexa ou texte ASCII
90A1 4C FB 94        JMP ASMB        Mini-assembleur

                        ;Dérivation RTI
                        ;détecte interrupt.
                        ;logicielle (BRK).

90A4 85 16     XRTI  STA $16         sauvegarde contenu accumulateur.
90A6 68              PLA             contenu registre P empilé au moment
90A7 48              PHA             interruption, transféré dans A pour
90A8 29 10           AND #$10        test AND #10. Si A=0, indicateur de
90AA F0 0A           BEQ NOBRK       break B=0, interrupt système -> RTI
90AC 58              CLI             Si A<>0 (B=1) interruption par BRK.
90AD A5 16           LDA $16         CLI autorise à nouveau les interrupt
90AF 28              PLP             val.init. A et P sont récupérées.
90B0 20 1A 91        JSR SAVREG      sauvegarde des registres (#50-54)
90B3 38              SEC             puis ...
90B4 B0 0A           BCS BPT1        branchement forcé en BPT1.
90B6 A5 16     NOBRK LDA $16         Récup.valeur initiale accumulateur,
90B8 40              RTI             puis retour d'interrupt.matérielle.

                        ;Routine
                        ;point d'arrêt
                        ;Entrée BRK normal
                        ;a BPT1

90B9 20 1A 91 BPT0   JSR SAVREG      sauvegarde des registres
90BC 20 28 91        JSR XBRKPT      3 oct.initiaux remis en place à P.A.
90BF 38              SEC             L'adresse du BRK réel ou simulé + 2,
90C0 68        BPT1  PLA             (contenu du compteur ordinal au mo-
90C1 E9 02           SBC #$02        ment de l'interruption ou adresse de
90C3 85 02           STA $02         retour routine point d'arrêt + 1) est
90C5 68              PLA             désempilée et ajustée (-2). Elle est
90C6 E9 00           SBC #$00        transférée dans le pointeur #02,03 et
90C8 85 03           STA $03         sera affichée par PRBKAD, ci-dessous.
90CA 20 F0 CB        JSR CRDO        mise à la ligne.

                        ;Affiche adr.BRK

90CD 20 44 96 PRBKAD JSR INSDS1
```

```
                        ;Affiche contenu
                        ;des registres

90D0 A2 00     REGDSP LDX #$00       boucle d'affichage, X = compteur
90D2 BD D9 97  NXRG   LDA REGNM,X    le nom du registre
90D5 20 D9 CC         JSR OUTDO      est affiché.
90D8 A9 3D            LDA #"="       le signe =
90DA 20 D9 CC         JSR OUTDO      est affiché
90DD B5 50            LDA $50,X      le contenu du registre
90DF 20 0F 96         JSR PRBYT      est affiché en hexa par PRBYT.
90E2 20 D4 CC         JSR OUTSPC     affiche un espace
90E5 E8               INX            le compteur est incrémenté
90E6 E0 04            CPX #$04       les 4 registres affichés?
90E8 D0 E8            BNE NXRG       non, suivant
90EA 20 D4 CC         JSR OUTSPC     oui, affiche un espace.

                        ;détail registre P

90ED A2 07     PRPREG LDX #$07       boucle affichage 8 indicateurs
90EF BD DD 97  NXFLG  LDA FLGNM,X    le nom de l'indicateur
90F2 20 D9 CC         JSR OUTDO      est affiché.
90F5 CA               DEX            le compteur est décrémenté
90F6 10 F7            BPL NXFLG      indicateur suivant si compteur >=0.
90F8 20 F0 CB         JSR CRDO       tous indicateurs affichés, CRDO
90FB A2 1D            LDX #$1D       fait passer au début ligne suivante
90FD 20 27 96         JSR PRSPC1     sort 29 espaces pour aligner
9100 A2 08            LDX #$08       l'affichage des huit bits.
9102 06 54     NXBIT  ASL $54        décalage à gauche contenu P.
9104 A9 30            LDA #"0"       après addition, (A)=code'0'ou'1'se-
9106 69 00            ADC #$00       lon valeur bit passé en C après ASL
9108 20 D9 CC         JSR OUTDO      affichage chiffre '0' ou '1'
910B CA               DEX            décrémentation compteur
910C D0 F4            BNE NXBIT      bit suivant si compteur > 0
910E 4C 7A 90         JMP RETMON     Réinit. pointeurs.

                        ;Instr.à recopier
                        ;au point d'arrêt

9111 20 B9 90  BRKPT  JSR BPT0

                        ;Réservés BRKPT

9114           BPSAV  $ 00 FF FF FF FF FF

                        ;Sauv.registres

911A 08        SAVREG PHP            contenu registre P mis de côté
911B 85 50            STA $50        sauvegarde contenu accumulateur
911D 86 51            STX $51        sauvegarde contenu registre X
911F 84 52            STY $52        sauvegarde contenu registre Y
9121 BA               TSX            contenu pointeur de pile transféré
9122 86 53            STX $53        dans X puis sauvegardé.
9124 68               PLA            contenu registre P désempilé
9125 85 54            STA $54        dans A puis sauvegardé.
9127 60               RTS            sortie.
```

```
                        ;Supprime
                        ;point d'arrêt

9128 A9 00     XBRKPT LDA #$00       La valeur 00, inscrite en BPSAV,
912A 8D 14 91         STA BPSAV      servira de témoin, (voir en #907A)
912D AD 15 91         LDA BPSAV+1    adresse point d'arrêt, rangée en
9130 85 0A            STA $0A        BPSAV+1 et BPSAV+2 par GOBRK
9132 AD 16 91         LDA BPSAV+2    (voir ci-dessous), est transférée
9135 85 0B            STA $0B        dans pointeur #0A,0B.
9137 A0 02            LDY #$02       les contenus des 3 octets, qui
9139 B9 17 91 NXB0    LDA BPSAV+3,Y  avaient été stockés ...
913C 91 0A            STA ($0A),Y    en BPSAV+3, +4, et +5 par GOBRK
913E 88               DEY            (voir ci-dessous) ...
913F 10 F8            BPL NXB0       sont remis en place.
9141 60        RET0   RTS            sortie.



                        ;Commande (G)o
                        ;Exécution
                        ;code machine

9142 F0 25     GO     BEQ EXEC       rien après adresse Go -> exécution
9144 C9 2D            CMP #"-"       est-ce '-'?
9146 D0 F9            BNE RET0       non, retour moniteur.

                        ;avec pt.d'arrêt

9148 20 4C E9 GOBRK   JSR HEXGET     trait d'union doit être suivi d'une
914B 84 04            STY $04        adresse, elle est recueillie par
914D 85 05            STA $05        HEXGET puis transférée dans #04,05.
914F 8A               TXA            test, si X=0, HEXGET n'a pas trouvé
9150 F0 EF            BEQ RET0       d'adresse, retour moniteur.
9152 A0 00            LDY #$00       Boucle, Y=compteur 3 octets.
9154 B9 03 00 NXB1    LDA $03,Y      contenu de #03 (octet fort adr. GO)
9157 99 14 91         STA BPSAV,Y    est rangé en BPSAV,
915A B1 04            LDA ($04),Y    (voir remarque ci-dessous).
915C 99 17 91         STA BPSAV+3,Y  l'adresse du point d'arrêt est
915F B9 11 91         LDA BRKPT,Y    rangée en BPSAV+1 et BPSAV+2.
9162 91 04            STA ($04),Y    Contenus des 3 octets à Pt d'arrêt
9164 C8               INY            sont rangés en BPSAV+3 et la suite;
9165 C0 03            CPY #$03       ils sont remplacés par l'instruction
9167 D0 EB            BNE NXB1       JSR BPT0 (voir BRKPT page précédente)
9169 6C 02 00 EXEC    JMP ($02)      GO, saut à l'adresse en #02,03.
```

Remarque: Si l'adresse du GO est en page zéro et si un point d'arrêt est utilisé, il faut être certain que ce BRK simulé soit exécuté.
En effet, dans ce cas, le contenu de #03, transféré dans BPSAV par GOBRK, est égal à zéro et ne joue plus son rôle de valeur-témoin au retour dans le moniteur.
A ce propos, n'oubliez pas le RTS en fin de routine à exécuter.

```
                        ;Commande (D)ump
                        ;ou (L)ist

916C D0 56     DMPLST BNE RET1      sortie si carac. après adresse.
916E 2C B2 BB         BIT $BBB2     caractère en #BBB2 est-il un espace?
9171 50 04            BVC DLST0     oui, (bit 6 à zéro), sortie écran.
9173 38               SEC           non, c'est 'I', sortie imprimante ..
9174 6E F1 02         ROR PRTFLG    bit 7 du drapeau PRTFLG est mis à un
9177 A5 35     DLST0  LDA BUF       la commande est-elle ...
9179 C9 44            CMP #"D"      'D' pour DUMP, vidage?
917B D0 4B            BNE LIST      non, alors c'est 'L' pour LIST.

                        ;Dump
                        ;-affichage hexa

917D A9 18     DUMP   LDA #$18      Boucle 1, compteur 24 lignes ...
917F 85 80            STA $80       sera le contenu de #80.
9181 20 3D 96  DMP0   JSR PRTA1     affiche adresse en début de ligne.
9184 20 D4 CC         JSR OUTSPC    affiche un espace.
9187 A0 00            LDY #$00      Boucle 2, Y=compteur 8 octets.
9189 B1 02     DMP1   LDA ($02),Y   contenu (#02),Y est affiché ...
918B 20 0F 96         JSR PRBYT     en hexadécimal par PRBYT.
918E 20 D4 CC         JSR OUTSPC    affiche un espace.
9191 C8               INY           incrémentation compteur
9192 C0 08            CPY #$08      8 octets affichés?
9194 D0 F3            BNE DMP1      non, suivant. oui, sortie boucle 2

                        ;-affichage  ASCII

9196 A0 00     ASCII  LDY #$00      Boucle 3, Y=compteur 8 car.ASCII.
9198 B1 02     ASC0   LDA ($02),Y   code caracactère chargé en A ...
919A C9 7E            CMP #$7E      est-il >= #7E ?
919C B0 04            BCS NOPRT     oui, remplacer par carré noir.
919E C9 20            CMP #" "      est-ce un caractère de contrôle?
91A0 B0 02            BCS PRT       non, bon pour l'affichage. Oui,
91A2 A9 7E     NOPRT  LDA #$7E      un carré noir est envoyé à l' ...
91A4 20 D9 CC  PRT    JSR OUTDO     affichage par OUTDO.
91A7 C8               INY           incrémentation compteur.
91A8 C0 08            CPY #$08      8 caractères affichés?
91AA D0 EC            BNE ASC0      non, suivant. Oui, sortie boucle 3.

                        ;-ajuste adr.base

91AC A5 02     NXADR  LDA $02       En sortie de boucle 3, C=1 est pris
91AE 69 07            ADC #$07      en compte dans l'addition.
91B0 85 02            STA $02       ADR+7+1 = nouvelle adresse, celle du
91B2 90 02            BCC NXLN      premier octet de la ligne suivante.
91B4 E6 03            INC $03       inc.oct.fort si C=1.
91B6 20 F0 CB  NXLN   JSR CRDO      mise à la ligne.
91B9 C6 80            DEC $80       déc. compteur de lignes, si > 0 ...
91BB D0 C4            BNE DMP0      suivante, si = 0 sortie boucle 1 ..
91BD 20 E8 C5         JSR INCHR     INCHR attend la frappe d'une touche
91C0 C9 20            CMP #" "      est-ce la barre d'espacement?
91C2 F0 B9            BEQ DUMP      oui, c'est reparti pour 24 lignes.
91C4 4E F1 02  RET1   LSR PRTFLG    non, bit 7 PRTFLG remis à zéro.
91C7 60               RTS           sortie.
```

```
                        ;List
                        ;désassemblage

91C8 A9 18     LIST     LDA #$18        compteur 24 lignes
91CA 48        LST0     PHA             compteur est sauvegardé en pile.
91CB 20 9A 95           JSR INSDSP      désassmb.et affiche une instruction.
91CE 20 2E 96           JSR ADADJ       recherche l' adresse ...
91D1 85 02              STA $02         de l'instruction suivante qui est ..
91D3 84 03              STY $03         transférée dans pointeur #02,03.
91D5 20 F0 CB           JSR CRDO        mise à la ligne.
91D8 68                 PLA             compteur de lignes est désempilé
91D9 38                 SEC             puis...
91DA E9 01              SBC #$01        diminué de un. S'il est > 0 ...
91DC D0 EC              BNE LST0        ligne suivante. S'il est = 0 ...
91DE 20 E8 C5           JSR INCHR       INCHR attend la frappe d'une touche
91E1 C9 20              CMP #" "        est-ce la barre d'espacement?
91E3 F0 E3              BEQ LIST        oui, c'est reparti pour 24 lignes.
91E5 D0 DD              BNE RET1        non, bit 7 PRTFLG =0, retour monit.


                        ;Commande (M)ove
                        ;déplacement octet0
                        ;-lect.adr.et test0

91E7 C9 2D     MOVE     CMP #"-"        Si caractère après première adresse
91E9 D0 5D              BNE ERR         n'est pas '-', erreur, Ping, monit.
91EB 20 4C E9           JSR HEXGET      si oui, HEXGET recueille la 2ème adr
91EE 84 04              STY $04         qui est inscrite dans le ptr. #04,05
91F0 85 05              STA $05         (voir rem.concernant HEXGET page 53)
91F2 20 E8 00           JSR CHRGOT      CHRGOT lit le caractère qui suit.
91F5 C9 3E              CMP #">"        est-ce '>'?
91F7 D0 4F              BNE ERR         non, erreur, Ping, retour moniteur
91F9 20 4C E9           JSR HEXGET      oui, HEXGET recueille cette ...
91FC 84 06              STY $06         3ème adresse qui est transférée ...
91FE 85 07              STA $07         dans le pointeur #06,07.
9200 8A                 TXA             si cette 3ème adresse est absente
9201 F0 45              BEQ ERR         erreur, Ping, retour moniteur.

                        ;-nombre d'octets
                        ;à deplacer

9203 38        MOV0     SEC             Le résultat ...
9204 A5 04              LDA $04         de la soustraction adr2 - adr1 ...
9206 E5 02              SBC $02         est inscrit en #08,09.
9208 85 08              STA $08         En fait, il s'agit ...
920A A5 05              LDA $05         du nombre d'octets à déplacer - 1.
920C E5 03              SBC $03         Il n'est utilisé qu'en cas de dépla-
920E 85 09              STA $09         cement vers le haut (voir MOVUP).
9210 90 36              BCC ERR         si adr2<adr1, err., Ping, ret.monit.

                        ;-rens.ptrs.temp.
                        ;pour RELOC évent.

9212 A2 05              LDX #$05        Les contenus des 3 pointeurs #02,03
9214 B5 02     MOV1     LDA $02,X       #04,05 et #06,07 sont recopiés dans
9216 95 72              STA $72,X       #72,73  #74,75 et #76,77 pour ...
9218 CA                 DEX             utilisation éventuelle par RELOC ..
9219 10 F9              BPL MOV1        commande (R)elogement.
```

```
                        ;-détermine sens
                        ;du deplacement

921B A5 07              LDA $07         L'adresse de destination est comparée
921D C5 03              CMP $03         à l'adresse de début.
921F 90 0A              BCC MOVDN       Comparaison classique entre deux
9221 D0 28              BNE MOVUP       nombres sur 2 octets (voir page 37).
9223 A5 06              LDA $06         Vous remarquerez que C=1 dans tous
9225 C5 02              CMP $02         les cas où le déplacement vers le
9227 F0 1E              BEQ RET2        haut (MOVUP) est prévu.
9229 B0 20              BCS MOVUP       si adr1=adr3, simple retour monit.

                        ;-déplacement
                        ;vers le bas

922B A0 00    MOVDN     LDY #$00        Boucle de déplacement, Y restera à 0
922D B1 02    MVD0      LDA ($02),Y     contenu octet pointé par #02,03 est
922F 91 06              STA ($06),Y     inscrit dans octet pointé par #06,07
9231 E6 06              INC $06         le pointeur destination ...
9233 D0 02              BNE MVD1        est ...
9235 E6 07              INC $07         incrémenté.
9237 A5 02    MVD1      LDA $02         Le pointeur origine ...
9239 C5 04              CMP $04         est comparé au ...
923B A5 03              LDA $03         pointeur de fin.
923D E5 05              SBC $05         puis ...
923F E6 02              INC $02         incrémenté.
9241 D0 02              BNE MVD2        Si la comparaison précédente indique
9243 E6 03              INC $03         que origine était < fin, alors le
9245 90 E6    MVD2      BCC MVD0        transfert est continué jusqu'à fin.
9247 60       RET2      RTS             sinon, retour moniteur

                        ;Erreur, ret.monit.

9248 4C 9F FA ERR       JMP PING        Erreur, retour moniteur via Ping.

                        ;-déplacement
                        ;vers le haut
                        ;utilise BLTU

924B A5 06    MOVUP     LDA $06         adresse destination + contenu #08,09
924D 65 08              ADC $08         + C (voir ci-dessus et MOV0, page
924F 85 C7              STA HIGHDS      précédente) représentent la destina-
9251 A5 07              LDA $07         tion + 1 de la plus haute adresse à
9253 65 09              ADC $09         déplacer. Elle est inscrite dans ...
9255 85 C8              STA HIGHDS+1    HIGHDS pour BLTU0.
9257 A5 04              LDA $04         A l'usage ...
9259 69 01              ADC #$01        de cette même routine, ...
925B 85 C9              STA HIGHTR      la plus haute adresse ...
925D A5 05              LDA $05         à déplacer + 1 ...
925F 69 00              ADC #$00        est inscrite ...
9261 85 CA              STA HIGHTR+1    dans HIGHTR et ...
9263 A5 02              LDA $02         la plus
9265 85 CE              STA LOWTR       basse adresse ...
9267 A5 03              LDA $03         à déplacer est ...
9269 85 CF              STA LOWTR+1     inscrite dans LOWTR.
926B 4C FB C3           JMP BLTU0       transfert par BLTU0 et retour monit.
```

```
                        ;Commande (X)mon
                        ;déplacement monit.

                        ;-tests validité
                        ;destination
                                        adr.destination est en #02,03.
926E D0 D8      XMON    BNE ERR         Erreur si caractère après adresse.
9270 A5 02              LDA $02         octet faible adr.destination testé.
9272 D0 D4              BNE ERR         erreur si <>0. (adr.pas début page)
9274 85 06              STA $06         octet faible (00) rangé en #06
9276 85 0A              STA $0A         et en #0A.
9278 A5 03              LDA $03         octet fort adr.destination comparé à
927A C9 04              CMP #$04         - #04. Erreur si inférieur ...
927C 90 CA              BCC ERR         adr. destination doit être > #400
927E C9 AD              CMP #$AD         - #AD. Erreur si supérieur ou égal
9280 B0 C6              BCS ERR         adr. destination doit être < #AD00.
9282 85 07              STA $07         octet fort rangé en #07
9284 85 0B              STA $0B         et en #08.
9286 38                 SEC             La différence, en valeur absolue,
9287 E5 01              SBC $01         entre octet fort adr.destination et
9289 B0 04              BCS XM00        octet fort adr.actuelle du moniteur
928B 49 FF              EOR #$FF        doit être supérieure ou égale à #08
928D 69 01              ADC #$01        sinon, il se mordra la queue ou
928F C9 08      XM00    CMP #$08        s'écrasera la tête au cours du
9291 90 B5              BCC ERR         déplacement.

                        ;-déplacement

9293 18                 CLC             les pointeurs sont renseignés
9294 A5 01              LDA $01         pour la routine de transfert.
9296 85 03              STA $03
9298 69 07              ADC #$07        début du bloc en #02,03
929A 85 05              STA $05         fin du bloc   en #04,05
929C A9 FF              LDA #$FF        destination   en #06,07 (déja fait)
929E 85 04              STA $04
92A0 20 03 92           JSR MOV0        transfert par MOVE, entrée en MOV0

                        ;-Reloc. jusqu'à
                        ;début des données

92A3 18                 CLC             Les pointeurs sont renseignés
92A4 A5 0A              LDA $0A         pour la routine de relogement.
92A6 85 02              STA $02
92A8 69 DD              ADC #<MSG1-1    début du bloc en #02,03
92AA 85 04              STA $04         fin du bloc   en #04,05
92AC A5 0B              LDA $0B         (adresse début des tables - 1)
92AE 85 03              STA $03
92B0 69 06              ADC #$06
92B2 85 05              STA $05         Relogement par
92B4 20 CC 92           JSR RLC1        RELOC, entrée en RLC1.

                        ;-saut à nouvelle
                        ;adresse

92B7 6C 0A 00           JMP ($0A)       Le moniteur rejoint son emplacement.
```

```
                        ;Commande (R)eloc

                        ;-recueil adr.fin

92BA C9 2D     RELOC    CMP #"-"        caractère après 1ère adresse doit
92BC D0 B0              BNE XMON        être '-'. Sinon, err., Ping, monit.
92BE 20 4C E9           JSR HEXGET      HEXGET recueille adresse de fin du
92C1 84 04              STY $04         bloc à reloger. Elle est transférée
92C3 85 05              STA $05         dans le pointeur #04,05.
92C5 4C CC 92           JMP RLC1        saut en RLC1 avec val.init.#02,03

92C8 85 02     RLC0     STA $02         retour ici après ajustement adresse
92CA 84 03              STY $03         instruction suivante.

                        ;-sauvegarde 3 oct

92CC A0 02     RLC1     LDY #$02        Les contenus des 3 octets situés à
92CE B1 02     RLC2     LDA ($02),Y     l'emplacement pointé par #02,03 et
92D0 99 79 00           STA $79,Y       aux deux suivants sont inscrits en
92D3 88                 DEY             #79,7A et 7B. En sortie de boucle,
92D4 10 F8              BPL RLC2        A contient le code opératoire.

                        ;-instructions de
                        ;moins de 3 octets.
                        ;sont rejetées.

92D6 20 51 96           JSR INSDS2      Avec A=codop, INSDS2 détermine en
92D9 A6 61              LDX $61         #61, la longueur de l'instruction-1
92DB E0 02              CPX #$02        si contenu de #61 < 02, pas d'adres-
92DD D0 2C              BNE RLC4        sage absolu, instruction suivante.

                        ;-détecte adressage
                        ;absolu interne

92DF A5 75              LDA $75         Les pointeurs de début, #72,73, et
92E1 C5 7B              CMP $7B         de fin du bloc, #74,75, renseignés
92E3 90 26              BCC RLC4        par MOVE avant transfert, sont com-
92E5 D0 06              BNE RLC3        parés à l'opérande de l'instruction
92E7 A5 74              LDA $74         adresse absolue en #7A,7B.
92E9 C5 7A              CMP $7A         Si #74,75 < #7A,7B, l'adressage
92EB 90 1E              BCC RLC4        est en dehors du bloc, branchement
92ED A5 7A     RLC3     LDA $7A         en RLC4 pour l'instruction suivante.
92EF E5 72              SBC $72         En RLC3, C=1 après les comparaisons
92F1 85 70              STA $70         le résultat de la soustraction
92F3 A5 7B              LDA $7B         #7A,7B - #72,73 est le déplacement à
92F5 E5 73              SBC $73         ajouter à l'adresse de destination.
92F7 85 71              STA $71         si #7A,7B < #72,73, adresse hors du
92F9 90 10              BCC RLC4        bloc, RLC4 pour instruction suivante
```

```
                        ;-modification
                        ;adresse absolue

92FB 18                 CLC             Le déplacement précédemment calculé
92FC A0 01              LDY #$01        est ajouté à l'adresse de destination
92FE A5 76              LDA $76         Le résultat est la nouvelle adresse
9300 65 70              ADC $70         absolue interne, opérande de l'ins-
9302 91 02              STA ($02),Y     truction dont l'adresse est pointée
9304 C8                 INY             par #02,03. Cet opérande est modifié
9305 A5 77              LDA $77         par l'instruction STA (#02),Y
9307 65 71              ADC $71         (adressage indirect indexé) où Y a
9309 91 02              STA ($02),Y     successivement les valeurs 1 et 2.

                        ;-ajuste adresse
                        ;instr.suivante
                        ;compare à fin.

930B 20 2E 96 RLC4      JSR ADADJ       L'adresse de l'instruction suivante
930E 48                 PHA             est calculée par ADADJ d'après la
930F C5 04              CMP $04         valeur en #61.
9311 98                 TYA             Cette adresse en A(-),Y(+) est com-
9312 E5 05              SBC $05         parée à l'adresse de fin de reloge-
9314 68                 PLA             ment en #04,05.
9315 90 B1              BCC RLC0        Si elle est inférieure, on continue
9317 60         RET3    RTS             sinon, sortie de la routine.


                        ;Commande (H)exa ou
                        ;(K)ar. Entrée code
                        ;hexa ou texte.
                        ;variables:
                        ;$04 index fin
                        ;$05 compt car.hexa
                        ;$06 temp.car.ou X
                        ;$07 temp.chfr.hexa
                        ;$08 >=$80 si hexa
                        ;$09 >=$80 si ESC.

                        ;-affich.hex ou txt

9318 D0 FD      HEXTXT  BNE RET3        Sortie si syntaxe incorrecte.
931A A5 35              LDA BUF         A reçoit le code de la commande
931C C9 48              CMP #"H"        'H' ou 'K', il servira de drapeau
931E D0 01              BNE HXT0        positif pour 'K'
9320 0A                 ASL A           négatif pour 'H'
9321 85 08      HXT0    STA $08         rangé en #08.
9323 20 3D 96 HXT1      JSR PRTA1       affiche l'adresse en cours.
9326 A2 04              LDX #$04        boucle d'affichage, X = 4 caractères
9328 24 08      HXT2    BIT $08
932A 10 05              BPL HXT3        affiche 'hex' ...
932C BD E4 97           LDA MSG2-1,X    si commande (H)exa.
932F D0 03              BNE HXT4
9331 BD E8 97 HXT3      LDA MSG3-1,X    ou affiche 'txt' ...
9334 20 D9 CC HXT4      JSR OUTDO       si commande (K)ar
9337 CA                 DEX
9338 D0 EE              BNE HXT2        en sortie de boucle, X = 0 est rangé
933A 86 04              STX $04         dans #04, index fin dans buffer et
933C 86 09              STX $09         dans #09, drapeau sortie par ESC.
```

```
                        ;-vidéo+curseur

933E 20 0E F9 CHIN0   JSR CURSON

                        ;-init. var. hexa

9341 A9 02      CHIN1   LDA #$02        Initialisation à 2 du compteur de
9343 85 05              STA $05         caractères hexa valides, #05
9345 A9 00              LDA #$00        et à zéro l'emplacement #07 dans le-
9347 85 07              STA $07         quel sera élaboree la valeur binaire

                        ;-entrée carac.
                        ;X sert d'index

9349 20 E8 C5 CHIN2   JSR INCHR       recueille code caractère dans A.

                        ;-autorise CTRL-T
                        ;si texte

934C 24 08      TEST0   BIT $08         si entrée texte, autorise CTRL-T
934E 10 09              BPL CTLT        et saute le test suivant.

                        ;-hexa. Exige
                        ;entrée 2ème carac.

9350 A4 05      TEST1   LDY $05         si entrée hexa, test compteur carac.
9352 C0 02              CPY #$02        si compteur=2, touches de contrôle
9354 F0 0C              BEQ LEFT        autorisées. Sinon, caractère  en A
9356 4C 30 94           JMP CHEX        doit être un 2ème chiffre hexa.

9359 C9 14      CTLT    CMP #$14        entrée texte, est-ce CTRL-T?
935B D0 05              BNE LEFT        non, essayer flèche gauche
935D 20 D9 CC           JSR OUTDO       oui, bascule maj/min et retour
9360 D0 E7              BNE CHIN2       entrée, branchement forcé

9362 C9 08      LEFT    CMP #$08        est-ce flèche gauche?
9364 D0 0B              BNE ESC         non, essayer ESC
9366 E0 00              CPX #$00        oui, mais si index buffer X = 0 ...
9368 F0 43              BEQ ERR1        retour entrée CHIN2 via Ping.
936A 20 77 94           JSR PRCHR2      si X <> 0, recul curseur par PRCHR2
936D CA                 DEX             décrémentation index buffer.
936E 4C 49 93           JMP CHIN2       retour entrée.

                        ;-si ESC ou RETURN
                        ;contenu buf.entrée
                        ;rangé en mémoire.

9371 C9 1B      ESC     CMP #$1B        est-ce ESC?
9373 D0 05              BNE RETURN      non, essayer RETURN
9375 38                 SEC             oui, bit 7 drapeau #09
9376 66 09              ROR $09         mis à un.
9378 30 04              BMI STORE       branchement forcé.
```

```
937A C9 0D     RETURN CMP #$0D      est-ce RETURN?
937C D0 2B            BNE TEST3     non, test fin.

937E 86 06     STORE  STX $06       c'est ESC ou RETURN, rangement en
9380 A0 00            LDY #$00      mémoire, X sauvé dans #06 sera index
9382 A2 FF            LDX #$FF      buffer, Y=0 pour adr.indirect indexé.
9384 E8       STOR0  INX           boucle, avex X initialisé à zéro.
9385 E4 06            CPX $06       X comparé à index initial curs/buffer
9387 90 05            BCC STOR1     si inférieur,  STOR1 pour test fin.
9389 A9 09            LDA #$09      si supérieur ou egal, le curseur est
938B 20 77 94         JSR PRCHR2    déplacé à l'écran par PRCHR2.
938E E4 04    STOR1  CPX $04       test fin code ou chaine de caractères
9390 F0 0C            BEQ STOR3     si fin, rangement terminé.
9392 B5 35            LDA BUF,X     code machine ou ASCII, chargé dans A
9394 91 02            STA ($02),Y   rangé à l'adresse pointée par #02,03.
9396 E6 02            INC $02       incrémentation ...
9398 D0 02            BNE STOR2     pointeur ...
939A E6 03            INC $03       d'adresse.
939C D0 E6    STOR2  BNE STOR0     code suivant, branchement forcé.
939E 20 F0 CB STOR3  JSR CRDO      mise à la ligne.
93A1 24 09            BIT $09       test drapeau #09.
93A3 30 03            BMI RET4      si négatif, ESC, sortie, retour monit
93A5 4C 23 93         JMP HXT1      ou assembleur. Sinon, nouvelle série.
93A8 60       RET4   RTS           sortie, retour à l'appelant.

                      ;-80 carac.maxi

93A9 E0 50    TEST3  CPX #$50      test buffer plein, 80 carac. entrés?
93AB D0 0A            BNE TEST4     non, test suivant, position curseur.

93AD 86 06    ERR1   STX $06       oui, index curs/buf sauvé dans #06
93AF 20 9F FA ERR2   JSR PING      car Ping détruit contenu registres.
93B2 A6 06            LDX $06       index est récupéré.
93B4 4C 49 93         JMP CHIN2     retour en CHIN2 pour nouvelle entrée.

                      ;-carac.si curseur
                      ;à la fin.

93B7 E4 04    TEST4  CPX $04       test position du curseur sur position
93B9 F0 64            BEQ ASC1      vierge? Oui, doit être ASCII ou hexa.

93BB C9 09    RIGHT  CMP #$09      est-ce flèche droite?
93BD D0 07            BNE DOWN      non, essayer flèche bas.
93BF 20 77 94         JSR PRCHR2    oui, déplacement curseur et
93C2 E8               INX           incrémentation index.
93C3 4C 49 93         JMP CHIN2     retour en CHIN2 pour nouvelle entrée.

93C6 C9 0A    DOWN   CMP #$0A      est-ce flèche bas (suppression)?
93C8 D0 25            BNE UP        non, essayer flèche haut.
```

```
                        ;-suppression

93CA CE 6A 02 DELET   DEC MODE0     #26A = 02. Suppression curseur.
93CD 86 06            STX $06       index curseur/buffer sauvé en #06.
93CF C6 04            DEC $04       index fin (nb.octets) décrémenté.
93D1 E4 04      DLT0  CPX $04       boucle, X index dans buffer ...
93D3 F0 09            BEQ FIN       est comparé à #04, si égal -> fin.
93D5 B5 36            LDA BUF+1,X   Code dans buffer décalé d'une posi-
93D7 95 35            STA BUF,X     tion vers le bas, et affiché en hexa
93D9 20 67 94         JSR PRCHR0    ou ASCII par PRCHR0 qui incrémente X
93DC D0 F3            BNE DLT0      branchement forcé (X <> 0).

                        ;-sortie commune
                        ;inser. et delet.

93DE 20 75 94 FIN     JSR PRCHR1    caractères excédentaire(s) effacé(s).
93E1 E8               INX           index curs/buf est incrémenté.
93E2 A9 08      RETCUR LDA #$08     puis le curseur est ramené à
93E4 20 77 94 RTC0    JSR PRCHR2    sa position initiale par PRCHR2
93E7 CA               DEX           qui 'affiche' le code #08, recul
93E8 E4 06            CPX $06       curseur. Sortie de boucle si X=(#06).
93EA D0 F8            BNE RTC0      retour CHIN0 où CURSON affich.curseur
93EC 4C 3E 93         JMP CHIN0

93EF C9 0B      UP    CMP #$0B      est-ce flèche haut (insertion)?
93F1 D0 2C            BNE ASC1      non, voir si ASCII ou hexa.
93F3 86 06            STX $06       index sauvedardé.
93F5 A5 04            LDA $04       si 80 octets
93F7 C9 50            CMP #$50      dans buffer
93F9 F0 B4            BEQ ERR2      insertion interdite.
93FB AA               TAX           X index décal.commençant par la fin.

                        ;-insertion

93FC CE 6A 02 INSER   DEC MODE0     #26A = 02, suppression curseur.
93FF CA         INS0  DEX           boucle, index initial dernier octet
9400 B5 35            LDA BUF,X     décalage d'une position ...
9402 95 36            STA BUF+1,X   vers le haut du buffer.
9404 E4 06            CPX $06       X comparé à position initiale
9406 D0 F7            BNE INS0      si non égal, octet suivant.
9408 A9 20      INS1  LDA #$20      le code espace (#20), en mode
940A 24 08            BIT $08       entrée ASCII, ou le code NOP (#EA),
940C 10 02            BPL INS2      en mode entrée hexa, est inscrit
940E A9 EA            LDA #$EA      dans le buffer ....
9410 95 35      INS2  STA BUF,X     à l'emplacement indexé par X.
9412 E6 04            INC $04       L'index de fin est incrémenté.
9414 B5 35      INS3  LDA BUF,X     En INS3, une boucle affiche le
9416 20 67 94         JSR PRCHR0    nouveau contenu du buffer jusqu'à
9419 E4 04            CPX $04       ce qu'X (incrémenté par PRCHR0) soit
941B D0 F7            BNE INS3      égal à l'index de fin #04.
941D F0 BF            BEQ FIN       sortie par FIN, retour CHIN0 (CURSON)
```

```
                        ;test caractère
                        ;mis en buf. si
                        ;ASCII valide.

941F C9 20     ASC1     CMP #$20       Après les tests précédents, les deux
9421 90 8A     ERR3     BCC ERR1       comparaisons qui suivent ne laissent
9423 C9 7E              CMP #$7E       passer que les caractères affichables
9425 B0 86              BCS ERR1       (DEL exclus).
9427 24 08              BIT $08        Si le mode choisi est l'entrée hexa,
9429 30 05              BMI CHEX       branchement CHEX pour test ultérieur.
942B 20 D9 CC           JSR OUTDO      sinon, OUTDO affiche le carac. ASCII.
942E D0 2B              BNE PUTBUF     qui est stocké, branchement forcé.

                        ;-vérifie validité
                        ;caractère hexa
                        ;si 2 chiffr.entrés
                        ;code en $07 est
                        ;stocké dans buffer

9430 B5 06     CHEX     STA $06        Le code du caractère, rangé en #06,
9432 49 30              EOR #$30       est manipulé afin de déterminer s'il
9434 C9 0A              CMP #$0A       s'agit d'un chiffre hexa valide.
9436 90 06              BCC HEXOK      La première comparaison sélectionne
9438 69 88              ADC #$88       les chiffres de 0 à 9.
943A C9 FA              CMP #$FA       La seconde, les lettres de A à F.
943C 90 E3              BCC ERR3       retour entrée caractère si erreur.
943E A0 03     HEXOK    LDY #$03       La boucle en HEXOK, ...
9440 0A                 ASL A          par une série de décalages, ...
9441 0A                 ASL A          range en #07 ...
9442 0A                 ASL A          le quartet correspondant ...
9443 0A                 ASL A          au chiffre hexadécimal entré.
9444 0A        HEX0     ASL A
9445 26 07              ROL $07        Après l'entrée du deuxième chiffre,
9447 88                 DEY            l'emplacement #07 contiendra la
9448 10 FA              BPL HEX0       valeur binaire complète.
944A A5 06              LDA $06        Le code du caractère hexadécimal est
944C 20 D9 CC           JSR OUTDO      récupéré puis affiché par OUTDO.
944F C6 05              DEC $05        Le compteur carac. hexa enregistrés
9451 F0 03              BEQ TWOHEX     est décrémenté, si 0 -> TWOHEX.
9453 4C 49 93           JMP CHIN2      Sinon, CHIN2, nouvelle entrée (hexa!)

9456 20 D4 CC TWOHEX    JSR OUTSPC     2 chiffres hexa entrés. OUTSPC affiche
9459 A5 07              LDA $07        un espace. Code machine chargé en A

945B 95 35     PUTBUF   STA BUF,X      Rangement du code dans le buffer.
945D E4 04              CPX $04        Index buffer est comparé à index fin,
945F D0 02              BNE INCX       s'il est égal, les deux index
9461 E6 04              INC $04        sont incrémentés.
9463 E8        INCX     INX            Sinon seul X est incrémenté.
9464 4C 41 93           JMP CHIN1      Retour entrée CHIN1.(Réinit.var.hexa)
```

```
                        ;Routines
                        ;d'affichage

9467 24 08     PRCHR0 BIT $08      Le contenu de A est affiché:
9469 10 05            BPL PRC0     En mode texte, directement par OUTDO.
946B 20 0F 96         JSR PRBYT    En mode hexa, par PRBYT après conver-
946E A9 20            LDA #$20     sion; les 2 chiffres hexa sont suivis
9470 20 D9 CC PRC0    JSR OUTDO    d'un espace.
9473 E8               INX          L'index curseur/buffer est incrémenté
9474 60               RTS          avant sortie.

9475 A9 20     PRCHR1 LDA #$20   PRCHR1 LDA #$20   PRCHR1 et PRCHR2
9477 A0 01     PRCHR2 LDY #$01   PRCHR2 LDY #$01   servent uniquement
9479 24 08            BIT $08            BIT $08   à l'affichage d'un
947B 10 02            BPL PRC1           BPL PRC1  ou trois espace ou
947D A0 03            LDY #$03           LDY #$03  d'un ou trois ca-
947F 20 D9 CC PRC1    JSR OUTDO  PRC1    PHA       ractères de con-
9482 88               DEY                JSR OUTDO trôle, avancée ou
9483 D0 FA            BNE PRC1           CMP #$20  recul curseur, se-
9485 60               RTS                BEQ PRC2  lon le mode d'en-
                                         LDA $269  trée, texte ou
                        $ EA EA          AND #$FE  hexa, en vigueur.
                        $ EA EA          BNE PRC2
                        $ EA EA          INY
                        $ EA EA    PRC2  PLA       Voir remarque
                        $ EA EA          DEY       ci-dessous.
                        $ EA EA          BNE PRC1
                        $ EA EA          RTS
```

On trouve ici la seule différence notable entre les deux versions du programme. Sur Atmos, en mode 38 colonnes, quand la routine OUTDO 'affiche' l'un des codes de contrôle #08 ou #09, recul ou avancée curseur, elle 'saute' automatiquement les deux colonnes de gauche de l'écran, dites protégées.
Ce n'est pas le cas sur Oric-1. Il faut donner un coup de pouce au curseur lorsqu'il se trouve sur l'une de ces deux colonnes, c'est-à-dire lorsque la variable système CURCOL (#269) vaut 00 ou 01. Cette situation est détectée à l'aide de l'instruction AND #FE. Bien entendu, cette procédure ne s'applique pas aux caractères ASCII dont l'espace (code #20).

* * *

La dernière partie du listing, pages suivantes, concerne l'assembleur-désassembleur dont les routines ont été adaptées de l'Apple 2.
Tâche compliquée par le fait que le moniteur de cette machine utilise l'ASCII négatif (bit 7 à un) pour coder les caractères.
Le fonctionnement est basé sur une manipulation de valeurs binaires extrêmement complexe. Décalages, rotations, instructions logiques sont des opérations très difficiles à suivre.
Le commentaire vous paraîtra léger. Il aurait fallu des pages pour restituer le mécanisme dans le détail. J'en serais d'ailleurs incapable aujourd'hui.

```
                        ;Mini-assembleur
                        ;entrée à ASMB

                        ;-vérifie validité
                        ;branch. relatif
                        ;déplacement en $67

9494 E9 81      ASM0    SBC #$81
9496 4A                 LSR A           test longueur instruction
9497 D0 5F              BNE ERR4        incorrecte, retour ASMB via Ping
9499 A4 68              LDY $68
949B A6 67              LDX $67         résultat soustraction
949D D0 01              BNE ASM1
949F 88                 DEY             #67,68 - #02,03
94A0 CA         ASM1    DEX
94A1 8A                 TXA             = déplacement
94A2 18                 CLC
94A3 E5 02              SBC $02         rangé en #67
94A5 85 67              STA $67
94A7 10 01              BPL ASM2        doit être compris
94A9 C8                 INY
94AA 98         ASM2    TYA             entre -128 et +127
94AB E5 03              SBC $03         sinon
94AD D0 49              BNE ERR4        erreur, retour ASMB via Ping.

                        ;-assemblage

94AF A4 61      ASM3    LDY $61         boucle, Y = longueur-1
94B1 B9 66 00   ASM4    LDA $66,Y       code rangé
94B4 91 02              STA ($02),Y     à partir #02,03.
94B6 88                 DEY
94B7 10 F8              BPL ASM4

                        ;remonte le curseur
                        ;affich.instruction
                        ;retour assembleur

94B9 A9 0B              LDA #$0B
94BB 20 D9 CC           JSR OUTDO       remonte curseur sur même ligne
94BE 20 9A 95           JSR INSDSP      affiche instruction complète
94C1 20 F0 CB           JSR CRDO        met à la ligne
94C4 20 2E 96           JSR ADADJ       ajuste prochaine adresse
94C7 85 02              STA $02         d'assemblage
94C9 84 03              STY $03         rangée e  #02,03
94CB 4C FB 94           JMP ASMB        retour assembleur.
```

```
                        ;test validité
                        ;instruction

94CE A5 66     ASM5     LDA $66        code opératoire en #66, compteur
94D0 20 51 96           JSR INSDS2     A l'aide de INSDSP2, ...
94D3 AA                 TAX            une boucle recherche la concordance
94D4 BD 87 97           LDA MRGT,X     entre les deux octets du mnémonique
94D7 C5 63              CMP $63        #62 et 63, et ceux des tables MLFT
94D9 D0 13              BNE ASM6       et MRGT.
94DB BD 47 97           LDA MLFT,X     si la concordance est trouvée, ...
94DE C5 62              CMP $62        sortie de boucle
94E0 D0 0C              BNE ASM6       code opératoire en #66
94E2 A5 64              LDA $64        longueur-1 en #61
94E4 A4 60              LDY $60        format prévu en #60
94E6 C0 9D              CPY #$9D        -la valeur de ce dernier étant #9D
94E8 F0 AA              BEQ ASM0       indique l'adressage relatif, sortie
94EA C5 60              CMP $60        vers ASM0 pour test ultérieur.
94EC F0 C1              BEQ ASM3        -si le format élaboré en #64 pendant
94EE C6 66     ASM6     DEC $66        la lecture de l'opérande est correct
94F0 D0 DC              BNE ASM5       branchement en ASM3 pour assemblage.
94F2 E6 64              INC $64        L'instruction INC $64 autorise les
94F4 C6 65              DEC $65        adresses page 0 sur 2 ou 4 chiffres
94F6 F0 D6              BEQ ASM5       une fois.
94F8 20 9F FA  ERR4     JSR PING       fin de boucle, erreur, rentrée ASMB.


                        ;Commande (A)ssembl
                        ;entrée
                        ;mini-assembleur
                        ;-affichage adresse

94FB 20 3D 96  ASMB     JSR PRTA1      affiche l'adr. d'assemblage en #02,03
94FE A9 3A              LDA #":"
9500 20 D9 CC           JSR OUTDO      affiche deux-points.

                        ;-l'instruction est
                        ;rangée dans buffer
                        ;d'entrée par INLIN
                        ;Ret.monit. si pas
                        ;d'entrée (BUF=0)

9503 20 92 C5           JSR INLIN      INLIN pour l'entrée de l'instruction
9506 86 E9              STX TXTPTR     en langage d'assemblage. Au retour,
9508 84 EA              STY TXTPTR+1   réglage TXTPTR pour lecture par CHRGET
950A A5 35              LDA BUF        l'accumulateur est chargé avec le code
950C D0 06              BNE 1STCHR     du premier caractère entré.
950E 60                 RTS            retour moniteur si 00, pas d'entrée.

                        ;Test 1er caractère
                        ;si H ou K, appel
                        ;s/routine HEXTXT

950F 20 18 93  SUBHT    JSR HEXTXT     entrée hexa ou texte,
9512 30 E7              BMI ASMB       retour forcé assembleur.
9514 C9 48     1STCHR   CMP #"H"       test 1er caractère.
9516 F0 F7              BEQ SUBHT      si 'H' ...
9518 C9 4B              CMP #"K"       ou "K" ...
951A F0 F3              BEQ SUBHT      exécution SUBHT, sinon lecture instr.
```

```
                        ;-lecture mnémoniqu
                        ;les 3 caractères
                        ;sont "compactés"
                        ;dans $62 et $63

951C A2 03     MNEM     LDX #$03      boucle 3 caractères
951E 20 E2 00  NXCHR    JSR CHRGET    lecture par CHGET.
9521 0A                 ASL A
9522 E9 BD              SBC #$BD
9524 C9 C3              CMP #$C3      caractères dont code est inférieur
9526 90 D0              BCC ERR4      à celui de 'A' sont rejetés.
9528 0A                 ASL A
9529 0A                 ASL A         Une série de décalages
952A A0 04              LDY #$04      'compacte' les trois caractères
952C 0A        MNM0     ASL A         l'un après l'autre,
952D 26 63              ROL $63       dans #62 et #63.
952F 26 62              ROL $62
9531 88                 DEY
9532 10 F8              BPL MNM0
9534 CA                 DEX
9535 F0 F5              BEQ MNM0
9537 10 E5              BPL NXCHR     caractère suivant

                        ;-lecture opérande


                        ;-symboles
                        ;tables 6 carac.
9539 A2 05     OPRND    LDX #$05      Boucle de lecture
953B 20 E2 00  OPR0     JSR CHRGET    de l'opérande
953E DD 3B 97           CMP CHAR1,X
9541 D0 15              BNE OPR1      sortie après OPR6
9543 20 E2 00           JSR CHRGET    page suivante.
9546 F0 11              BEQ OPR2
9548 DD 41 97           CMP CHAR2,X
954B F0 0E              BEQ OPR3
954D BD 41 97           LDA CHAR2,X
9550 F0 07              BEQ OPR2      Les symboles entrés sont comparés
9552 C9 24              CMP #"$"      à ceux des tables CHAR1 et CHAR2.
9554 F0 03              BEQ OPR2
9556 C6 E9              DEC TXTPTR
9558 18        OPR1     CLC
9559 C6 E9     OPR2     DEC TXTPTR    Pas de vérification
955B 26 64     OPR3     ROL $64       de la syntaxe
955D E0 03              CPX #$03
955F D0 17              BNE OPR6
```

```
                        ;-valeur hexa.

9561 20 4C E9           JSR HEXGET   La valeur hexa
9564 84 67              STY $67      recueillie par HEXGET
9566 85 68              STA $68      est rangée en #67,68.
9568 8A                 TXA
9569 F0 07              BEQ OPR5
956B A5 68              LDA $68      X=0 si pas de valeur hexa
956D F0 01              BEQ OPR4     X=1 si octet fort = 00.
956F E8                 INX          X=2 si valeur > FF.
9570 E8           OPR4  INX
9571 E8                 INX
9572 86 65        OPR5  STX $65      rangé en #65.
9574 A2 03              LDX #$03
9576 C6 E9              DEC TXTPTR
9578 86 66        OPR6  STX $66
957A CA                 DEX
957B 10 BE              BPL OPR0     test fin de boucle lecture opérande
957D A5 64              LDA $64
957F 0A                 ASL A        La valeur élaborée en #64
9580 0A                 ASL A        est manipulée
9581 05 65              ORA $65      puis réinscrite en #64.
9583 C9 20              CMP #$20     sera format de l'instruction,
9585 B0 06              BCS OPR7     à comparer à (#60), trouvé par INSDS2
9587 A6 65              LDX $65
9589 F0 02              BEQ OPR7
958B 09 80              ORA #$80
958D 85 64        OPR7  STA $64
958F 20 E2 00           JSR CHRGET   Pas d'autre caractère après opérande
9592 D0 03              BNE OPR8     oui, erreur
9594 4C CE 94           JMP ASM5     non, test validité instruction entrée
9597 4C FB 94 OPR8      JMP ERR4     Ping, rentrée mini-assembleur.


                        ;Désassembleur
                        ;désassemble
                        ;une instruction

                        ;-affiche l'adresse
                        ;-index tables mném
                        ;mis en pile

959A 20 44 96 INSDSP    JSR INSDS1   affiche adresse, détermine format,
959D 48                 PHA          longueur-1, index mnémon. empilé

                        ;-affiche le code

959E B1 02      PRNTOP  LDA ($02),Y  boucle d'affichage du code
95A0 20 0F 96           JSR PRBYT    hexadécimal (Y=0 en sortie INSDS1)
95A3 A2 01              LDX #$01
95A5 20 27 96 PRNTSP    JSR PRSPC1   affiche un espace.
95A8 C4 61              CPY $61      terminé?
95AA C8                 INY
95AB 90 F1              BCC PRNTOP   non, octet suivant
```

```
                        ;-affich.mnémonique
                        ;X compte 3 carac.
                        ;recueil 2 oct.dans
                        ;tables, index en Y

95AD A2 03     PRMNEM LDX #$03     compteur 3 caractères
95AF C0 04            CPY #$04
95B1 90 F2            BCC PRNTSP   aligne les mnémoniques
95B3 68               PLA          récupère index tables mnémoniques
95B4 A8               TAY          le transfère dans Y
95B5 B9 47 97         LDA MLFT,Y   les 'parties' droite
95B8 85 62            STA $62      et gauche du mnémonique
95BA B9 87 97         LDA MRGT,Y   sont inscrites
95BD 85 63            STA $63      dans #62 et #63.

                        ;-"décompacte' et
                        ;affiche les 3 car.

95BF A9 00     PRMN1  LDA #$00     initialise accumulateur.
95C1 A0 05            LDY #$05     compteur 5 bits.
95C3 06 63     PRMN2  ASL $63
95C5 26 62            ROL $62      5 bits sont introduits
95C7 2A               ROL A        dans l'accumulateur.
95C8 88               DEY
95C9 D0 F8            BNE PRMN2    bit suivant
95CB 69 3F            ADC #$3F     ajuste code caractère
95CD 20 D9 CC         JSR OUTDO    et l'affiche.
95D0 CA               DEX          décrémentation compteur caractères
95D1 D0 EC            BNE PRMN1    caractère suivant.
95D3 20 D4 CC         JSR OUTSPC   affiche un espace

                        ;-afiche opérande

95D6 A4 61     PRADR  LDY $61      longueur-1 dans Y
95D8 A2 06            LDX #$06     compte 6 bits (7 à 2) de format (#60)
95DA E0 03     PRAD1  CPX #$03     si X=3
95DC F0 1C            BEQ PRAD5    instruction contient une adresse
95DE 06 60     PRAD2  ASL $60      décalage
95E0 90 0E            BCC PRAD3    C=0, pas d'affichage
95E2 BD 3A 97         LDA CHAR1-1  recueille symbole dans table CHAR1
95E5 20 D9 CC         JSR OUTDO    et l'affiche.
95E8 BD 40 97         LDA CHAR2-1  2ème symbole à afficher?
95EB F0 03            BEQ PRAD3    non, CHAR2 correspondant = 00
95ED 20 D9 CC         JSR OUTDO    oui, affichage.
95F0 CA        PRAD3  DEX
95F1 D0 E7            BNE PRAD1    bit suivant
95F3 60               RTS
95F4 88        PRAD4  DEY
95F5 30 E7            BMI PRAD2
95F7 20 0F 96         JSR PRBYT
95FA A5 60     PRAD5  LDA $60
95FC C9 E8            CMP #$E8     adressage relatif?
95FE B1 02            LDA ($02),Y
9600 90 F2            BCC PRAD4    non
```

```
                        ;-adressage relatif
                        ;élabore adresse
                        ;destination

9602 20 31 96 RELAD   JSR ADJ0    déplacement en A, élabore adresse
9605 AA               TAX         de destination - 1 en A(-),Y(+)
9606 E8               INX         transférée dans X(-),Y(+) ...
9607 D0 01            BNE PRTYX   puis incrémentée.
9609 C8               INY

                        ;-affiche en hexa
                        ;adr.en Y(+),X(-)

960A 98         PRTYX TYA
960B 20 0F 96         JSR PRBYT   voir pages 38 et 39
960E BA               TXA

                        ;-affiche en hexa
                        ;contenu accumul.

960F 48         PRBYT PHA
9610 4A               LSR A
9611 4A               LSR A
9612 4A               LSR A
9613 4A               LSR A
9614 20 1A 96         JSR PRHEX   voir pages 40 et 41
9617 68               PLA
9618 29 0F            AND #$0F

961A 09 30      PRHEX ORA #$30
961C C9 3A            CMP #$3A
961E 90 02            BCC PRINT
9620 69 06            ADC #$06
9622 4C D9 CC PRINT   JMP OUTDO

                        ;-afiche 3 espaces

9625 A2 03      PRTSPC LDX #$03
9627 20 D4 CC PRSPC1  JSR OUTSPC
962A CA               DEX
962B D0 FA            BNE PRSPC1
962D 60               RTS

                        ;Mise à jour adr.
                        ;instr.suivante

962E 38         ADADJ SEC         C=1 pour ajuster long. addition ADJ1.
962F A5 61            LDA $61     longueur-1 dans l'accumulateur
9631 A4 03      ADJ0  LDY $03     Y = octet fort adresse actuelle
9633 AA               TAX         test signe déplacement...
9634 10 01            BPL ADJ1    branchement relatif*
9636 88               DEY         décrémentation octet fort si négatif
9637 65 02      ADJ1  ADC $02     A = oct.faible + long. (ou déplacemt)
9639 90 01            BCC RET5
963B C8               INY         nouvelle adresse dans A(-),Y(+)
963C 60         RET5  RTS
```

```
                        ;Aff.adr.en $02,03

963D A4 03     PRTA1    LDY $03
963F A6 02              LDX $02
9641 4C 0A 96           JMP PRTYX


                        ;Sous-routines
                        ;du désassembleur.

                        ;Affiche adresse,
                        ;tiret et espaces

9644 20 3D 96 INSDS1    JSR PRTA1     affiche adresse de l'instruction
9647 A9 2D              LDA #$2D
9649 20 D9 CC           JSR OUTDO     affiche tiret
964C 20 25 96           JSR PRTSPC    affiche 3 espaces, au retour X=0
964F A1 02              LDA ($02,X)   code opératoire dans l'accumulateur

                        ;avec A=codop instr
                        ;élabor.format $60
                        ;longueur-1 en $61
                        ;index mnémon en A

9651 A8        INSDS2   TAY           codop transféré dans Y
9652 4A                 LSR A         test parité (bit 0)
9653 90 09              BCC IEVEN     si pair, OK
9655 6A                 ROR A         test bit 1
9656 B0 15              BCS ERR5      codop de la forme xxxxxx11 invalides
9658 C9 A2              CMP #$A2
965A F0 11              BEQ ERR5      code #89 n'existe pas.
965C 29 87              AND #$87      masque
965E 4A        IEVEN    LSR A         bit 0 dans C pour choix quartet
965F AA                 TAX
9660 BD E9 96           LDA FMT1,X    recueille valeur dans table FMT1
9663 90 04              BCC MASK      sélection quartet faible
9665 4A                 LSR A         ou fort.
9666 4A                 LSR A
9667 4A                 LSR A         sera index ...
9668 4A                 LSR A         dans FMT2.
9669 29 0F     MASK     AND #$0F
966B D0 04              BNE GTFMT
966D A0 80     ERR5     LDY #$80      si codop invalide, substitue #80
966F A9 00              LDA #$00      et fait index format = 0
9671 AA        GTFMT    TAX           X = index
9672 BD 2D 97           LDA FMT2,X    recueille format (mode d'adressage)
9675 85 60              STA $60       rangé dans #60.
9677 29 03              AND #$03      longueur instruction - 1
9679 85 61              STA $61       rangée dans #61.
967B 98                 TYA           récupère code opératoire
967C 29 8F              AND #$8F      masque pour test 1xxx1010
967E AA                 TAX           sauvegarde dans X
967F 98                 TYA           récupère codop à nouveau.
9680 A0 03              LDY #$03      calcul index dans tables
9682 E0 8A              CPX #$8A      des mnémoniques
9684 F0 08              BEQ MNDX3                        . . .
```

```
9686 4A        MNDX1  LSR A
9687 90 08            BCC MNDX3     calcul index dans tables
9689 4A               LSR A         des mnémoniques.
968A 4A        MNDX2  LSR A
968B 09 20            ORA #$20                    suite
968D 88               DEY
968E D0 FA            BNE MNDX2
9690 C8               INY           A la sortie,
9691 88        MNDX3  DEY           index dans l'accumulateur.
9692 D0 F2            BNE MNDX1
9694 60               RTS

                      ;Affiche position
                      ;moniteur

9695 A2 0B     PRMLOC LDX #$0B      affiche "   Moniteur"
9697 BD DD 96  PML0   LDA MSG1-1,X
969A 9D 81 BB         STA $BB81,X
969D CA               DEX
969E D0 F7            BNE PML0
96A0 86 08            STX $08       X=0 rangé dans #08
96A2 A5 01            LDA $01       octet fort adresse début moniteur
96A4 85 09            STA $09       rangé dans #09.
96A6 A9 0D            LDA #$0D      contenu #80 = #0D sera index X
96A8 85 80            STA $80       pour STOUT.
96AA A9 20            LDA #" "      code espace
96AC 85 FF            STA STACK-1   dans STACK-1 (#FF, FBUF)
96AE 20 C2 96         JSR PRADST    affichage adresse début
96B1 C6 08            DEC $0B       octet faible adresse fin = #FF
96B3 18               CLC           (en fait, c'est #EC)
96B4 A5 09            LDA $09       octet fort adresse fin
96B6 69 07            ADC #$07      = octet fort adresse début + #07
96B8 85 09            STA $09
96BA A2 12            LDX #$12      contenu #80 = #12
96BC 86 80            STX $80       sera index X pour STOUT
96BE A9 2D            LDA #"-"      code du tiret
96C0 85 FF            STA STACK-1   dans STACK-1 (#FF, FBUF)
96C2 A2 FF     PRADST LDX #$FF      affiche une adresse haut de l'écran
96C4 86 2F            STX FLG       (voir pages 38, 39)
96C6 E8               INX
96C7 A5 09            LDA $09
96C9 20 93 D9         JSR BINHEX    conversion octet fort
96CC A5 08            LDA $08
96CE 20 93 D9         JSR BINHEX    conversion octet faible
96D1 A9 00            LDA #$00
96D3 9D 00 01         STA STACK,X   00 en fin de chaîne hexa dans FBUF
96D6 A8               TAY           adresse chaîne #00FF
96D7 A9 FF            LDA #$FF      dans A(-),Y(+)
96D9 A6 80            LDX $80       X = index colonne début affichage
96DB 4C 65 F8         JMP STOUT     routine d'affichage ligne supérieure.
```

```
        ;Messages et tables

MSG1    "   Moniteur
FMT1    $ 04 20 54 30 0D 80     Tables du désassembleur
        $ 04 90 03 22 54 33     voir page ci-contre
        $ 0D 80 04 90 04 20
        $ 54 33 0D 80 04 90
        $ 04 20 54 3B 0D 80     Index dans FRMT2 (4 bits)
        $ 04 90 00 22 44 33
        $ 0D C8 44 00 11 22
        $ 44 33 0D C8 44 A9
        $ 01 22 44 33 0D 80
        $ 04 90 01 22 44 33
        $ 0D 80 04 90 26 31
        $ 87 9A
FMT2    $ 00 21 81 82 00 00     Format
        $ 59 4D 91 92 86 4A
        $ 85 9D
CHAR1   $ 2C 29 2C 23 28 24     ASCII:   ,   )   ,   #   (   $
CHAR2   $ 59 00 58 24 24 00     ASCII:   Y  nul  X   $   $  nul
MLFT    $ 1C 8A 1C 23 5D 8B     Mnémoniques sur deux octests
        $ 1B A1 9D 8A 1D 23     voir page ci-contre
        $ 9D 8B 1D A1 00 29
        $ 19 AE 69 A8 19 23
        $ 24 53 1B 23 24 53
        $ 19 A1 00 1A 5B 5B
        $ A5 69 24 24 AE AE     gauche
        $ A8 AD 29 00 7C 00
        $ 15 9C 6D 9C A5 69
        $ 29 53 84 13 34 11
        $ A5 69 23 A0
MRGT    $ D8 62 5A 48 26 62
        $ 94 88 54 44 C8 54
        $ 68 44 E8 94 00 B4
        $ 08 84 74 B4 28 6E
        $ 74 F4 CC 4A 72 F2
        $ A4 8A 00 AA A2 A2     droit
        $ 74 74 74 72 44 68
        $ B2 32 B2 00 22 00
        $ 1A 1A 26 26 72 72
        $ 88 C8 C4 CA 26 48
        $ 44 44 A2 C8
CMDTBL  "GDLMXRHKA              Table des commandes
CMDADL  $ 8E 91 91 94 97 9A     Octet faible - 1 de l'adr. du saut
        $ 9D 9D A0                  vers l'entrée d'une commande.
REGNM   "AXYS                   Registres
FLGNM   "CZIDB/VN               Indicateurs
MSG2    ":xeh
MSG3    ":txt
```

Les tables FMT1 et FMT2 sont utilisées par INSDS2 (9651), pour déterminer, d'après le code opératoire, la longueur de l'instruction, le mode d'adressage et l'index dans les tables de mnémoniques.

- La table FMT1 contient 68 octets.
Les 64 premières valeurs représentent 128 index de 4 bits dans la table FMT2 pour les codes opératoires de la forme xxxxxxxY0.
Les 4 dernières, 8 index pour les codes opératoires de la forme xxxxxY01.
Le quartet faible est utilisé pour Y=0, le quartet fort pour Y=1.

- La table FMT2 contient 14 octets.
Ces valeurs représentent les renseignements relatifs à la longueur (bits 0..1, longueur-1) et au mode d'adressage (bits 7..2).

00 code non valide	00 accumulateur	B6 absolu,Y
21 immédiat	59 (indirect,X)	4A (indirect)
81 page zéro	4D (indirect),Y	85 page zéro,Y
82 absolu	91 page zéro,X	9D relatif
00 implicite	92 absolu,X	

Les tables MLFT et MRGT, mnémonique gauche et droit, contiennent les deux octets obtenus après 'compactage' des trois caractères du mnémonique.

G	D		G	D		G	D		G	D	
1C	DB	BRK	00	00	???	00	00	???	15	1A	ASL
8A	62	PHP	29	B4	DEY	1A	AA	BIT	9C	1A	ROL
1C	5A	BPL	19	08	BCC	58	A2	JMP	6D	26	LSR
23	48	CLC	AE	84	TYA	5B	A2	JMP	9C	26	ROR
5D	26	JSR	69	74	LDY	A5	74	STY	A5	72	STX
BB	62	PLP	A8	B4	TAY	69	74	LDY	69	72	LDX
1B	94	BMI	19	28	BCS	24	74	CPY	29	88	DEC
A1	88	SEC	23	6E	CLV	24	72	CPX	53	C8	INC
9D	54	RTI	24	74	CPY	AE	44	TXA	84	C4	ORA
8A	44	PHA	53	F4	INY	AE	68	TXS	13	CA	AND
1D	C8	BVC	1B	CC	BNE	A8	B2	TAX	34	26	EOR
23	54	CLI	23	4A	CLD	AD	32	TSX	11	48	ADC
9D	68	RTS	24	72	CPX	29	B2	DEX	A5	44	STA
8B	44	PLA	53	F2	INX	00	00	???	69	44	LDA
1D	E8	BVS	19	A4	BEQ	7C	22	NOP	23	A2	CMP
A1	94	SEI	A1	8A	SED	00	00	???	A0	C8	SBC

TROISIEME PARTIE

METHODES DE TRI

Le tri est l'une des tâches fastidieuses pour lesquelles l'ordinateur se révèle un outil précieux.
C'est une fonction indispensable de certains logiciels, gestion, traitement de fichiers, analyse financière etc, généralement écrits en langage évolué.

Dans un but d'efficacité optimum, l'adaptation du sous-programme de tri doit être menée avec soin.
Les méthodes de base et leurs variantes sont nombreuses.
Le choix de l'une ou de l'autre résulte toujours d'un compromis entre des avantages et des inconvénients.

Vaste sujet qu'on ne peut pas prétendre épuiser en quelques pages.
Nous nous bornerons à étudier le mécanisme des méthodes les plus couramment employées en informatique individuelle pour trier des tableaux en mémoire centrale.
L'objectif visé est donc limité mais délibérément pratique.
Il est intéressant pour plusieurs raisons:

- il nous donne l'opportunité de revenir au Basic, injustement délaissé jusqu'ici, et à cette occasion, de revoir comment les tableaux de variables sont rangés en mémoire.

- il devrait vous aider à mieux saisir le fonctionnement des algorithmes de tri qui, en général, ne sont pas faciles à comprendre.

- enfin, dans ces procédures où l'efficacité est synonyme de rapidité de traitement, vous verrez ce que peut apporter le langage machine en association avec le Basic.

Avant de passer au plat de résistance, voici le dessert.
Ce petit programme illustre le tri à bulles, de mauvaise réputation, souvent qualifié de 'tarte à la crème' des études sur le tri, mais qu'on ne peut pas passer sous silence.
Les rangées et les colonnes représentent autant de petits tableaux de dix éléments qui sont triés et retriés tour à tour.

```
10 CLS:A=5:B=14
20 FOR Y=A TO B:FOR X=A TO B:PLOT X,Y,INT(RND(1)*10)+48:NEXT X,Y
100 E=0  'drapeau echanges
110 FOR X=A TO B:FOR Y=B TO A-1 STEP-1:I=SCRN(X,Y):J=SCRN(X,Y-1)
120 IF J<=I THEN 130 ELSE PLOT X,Y,J:PLOT X,Y-1,I:E=E+1
130 NEXT Y,X
140 FOR Y=A TO B:FOR X=B TO A-1 STEP-1:I=SCRN(X,Y):J=SCRN(X-1,Y)
150 IF I>=J THEN 160 ELSE PLOT X,Y,J:PLOT X-1,Y,I:E=E+1
160 NEXT X,Y:IF E<>0 THEN 100 ELSE PING:END
```

IMPLANTATION DES TABLEAUX EN MEMOIRE

Les tableaux de variables sont rangés après les variables simples dans une zone dont l'adresse de début est détenue par le pointeur ARYTAB (#9E,9F).
Chaque tableau comporte un en-tête d'identification suivi des valeurs des éléments rangées dans l'ordre des indices.
Pour un tableau à deux dimensions, par exemple, on trouvera normalement:
A(0,0), A(1,0), A(2,0) .... puis A(0,1), A(1,1), A(2,1) ... etc.
La fin de la zone des tableaux, début de l'espace libre, est indiquée par le pointeur STREND (STorage END) #A0,A1.

Une instruction DIM construit un tableau dont tous les éléments sont initialisés à zéro. Elle est nécessaire si un indice supérieur à 10 est utilisé dans une quelconque de ces dimensions.
Par exemple, après DIM A(N1,N2...), où N1 et N2... représentent les indices maximums autorisés, le tableau A pourra contenir (N1+1)*(N2+1)*... éléments (en comptant des indices 0).

Une valeur peut être affectée à une variable indicée sans dimensionnement préalable (indice maximum 10). Dans ce cas, un tableau de "largeur" 11 pour chaque dimension spécifiée est construit automatiquement.
Exemple, A(5,2)=123.456. La valeur réelle 123.456 est rangée à sa place dans le tableau A à deux dimensions, 11 colonnes de 11 éléments ; DIM A(10,10).

La contexture de l'en-tête d'un tableau est la suivante:
- les deux premiers octets contiennent le nom en ASCII positif (ASCII normal) ou négatif (ASCII + #80). Comme pour les variables simples, seuls les deux premiers caractères du nom déclaré sont significatifs.
Si le nom ne comporte qu'une seule lettre, le deuxième caractère est remplacé par une valeur nulle (#00 ou #80).
Les deux caractères sont en ASCII positif pour un tableau de nombres réels (exemple, #41 et #00 pour le tableau A).
Ils sont en ASCII négatif pour un tableau de nombres entiers (exemple, #C2 et #C3 pour BC%).
Le premier caractère est en ASCII positif et le second négatif pour un tableau de chaînes de caractères (exemple, #44 et #80 pour D$).
- les deux octets suivants contiennent un déplacement (octet faible d'abord) à ajouter à l'adresse du tableau pour obtenir celle du tableau suivant.
- le cinquième octet indique le nombre de dimensions.
- selon le nombre 'd' de dimensions, on trouve ensuite 'd' paires d'octets indiquant le nombre d'éléments dans chaque dimension en commençant par la dernière. L'octet fort est avant l'octet faible.

Après l'en-tête, les valeurs des éléments sont rangées sur 5, 2 ou 3 octets suivant la nature des variables indicées.
- 5 octets pour les nombres réels (valeur en virgule flottante).
- 2 octets pour les entiers signés (octet fort d'abord).
- 3 octets pour les descripteurs de chaîne (longueur et adresse).

QUATRE METHODES DE TRI

Le programme qui suit tente de visualiser le déroulement d'un tri selon l'une des quatre méthodes les plus courantes.
Le tableau, alphanumérique, est composé de 36 lettres de l'alphabet rangées de façon aléatoire. L'emplacement d'indice 0 n'est pas utilisé.

Le processus apparaît clairement pour le tri à bulles et le tri par insertion, moins nettement pour Shell ou Quicksort.
Ce programme donne une idée de l'efficacité relative des différentes méthodes mais ce n'est pas un essai comparatif. Les résultats sont faussés par la cohabitation des boucles de tri et par l'affichage intermédiaire. Ce dernier aggrave notamment le handicap naturel du tri à bulles.
Nous verrons plus en détail les quatre algorithmes.

```
10 N=36:DIM A$(N):GOSUB 1000:GET R$
20 R=VAL(R$):AC=#BBAB+40*R:POKE AC,45:GOSUB 50
30 ON R GOTO 100,200,300,400:CALL#247 'Oric-1 #F882

50 FOR X=1 TO N:PRINT A$(X);:NEXT:PRINT CHR$(13);:RETURN

99 REM Bulles
100 E=0:FOR I=2 TO N:J=I-1:IF A$(I)=>A$(J)THEN 120
110 T$=A$(I):A$(I)=A$(J):A$(J)=T$:E=E+1:GOSUB 50
120 NEXT:IF E<>0 THEN 100
130 PING:POKE AC,32:GET R$:GOSUB 1010:GOTO 20

199 REM Insertion
200 FOR I=2 TO N:J=I-1
210 K=J+1:IF A$(K)=>A$(J) THEN 230
220 T$=A$(J):A$(J)=A$(K):A$(K)=T$:J=J-1:IF J>0 THEN 210
230 GOSUB 50:NEXT:GOTO 130

299 REM Shell
300 P=N
310 P=INT(P/2):IF P=0 THEN 130
320 FOR I=1+P TO N:J=I-P
330 K=J+P:IF A$(K)=>A$(J) THEN 350
340 T$=A$(J):A$(J)=A$(K):A$(K)=T$:GOSUB 50:J=J-P:IF J>0 THEN 330
350 NEXT:GOTO 310

399 REM Quicksort
400 S=0:PA=1:PZ=N
410 M$=A$(INT((PA+PZ)/2)):I=PA:J=PZ
420 IF A$(I)<M$ THEN I=I+1:GOTO 420
430 IF A$(J)>M$ THEN J=J-1:GOTO 430
440 IF I>J THEN 480
450 IF I=J THEN 470
460 T$=A$(I):A$(I)=A$(J):A$(J)=T$:GOSUB 50
470 I=I+1:J=J-1:IF I=<J THEN 420
480 IF I=>PZ THEN 500
490 TA(S)=I:TB(S)=PZ:S=S+1
500 PZ=J:IF PA<PZ THEN 410
510 IF S=0 THEN 130
520 S=S-1:PA=TA(S):PZ=TB(S):GOTO 410

1000 REM

1010 CLS:POKE#26A,2:PRINT:PRINT"1 Bulles":PRINT"2 Insertion"
1020 PRINT"3 Shell":PRINT"4 Quicksort":PRINT
1030 FOR I=1 TO N:A$(I)=CHR$(INT(RND(1)*26)+65)
1040 PRINT A$(I);:NEXT:PRINT:PRINT:RETURN
```

Toutes les méthodes sont basées sur des comparaisons entre deux éléments suivies ou non d'un échange.
Remarquez la nécessité de sauvegarder A$(I) avant l'échange dans une variable temporaire T$ car cette valeur est 'écrasée' par A$(I)=A$(J).

Pour Bulles, Insertion ou Shell, il s'agit d'explorer l'ensemble du tableau dans l'ordre.
La construction de la boucle d'examen est secondaire. Le même modèle est employé pour ne pas compliquer les choses.
Dans cette même optique, l'indice 0 n'est pas utilisé.

TRI A BULLES

Un élément est comparé à son voisin immédiat, ici celui qui le précède.
L'échange a lieu s'ils ne sont pas dans l'ordre.
La variable E, initialisée à zéro, sert de drapeau.
En fin de boucle:
- si E=0, le parcours s'est fait sans échange, le tri est terminé.
- si E<>0, la boucle est recommencée avec E=0.

Prenons un exemple concret.
Soit à trier le tableau A$ composé de cinq lettres.

```
Rangement initial:   I=    1  2  3  4  5
                           S  E  B  A  C
```

La boucle FOR I=2 TO 5 est parcourue autant de fois que nécessaire;
l'élément I est comparé à l'élément J (I-1).
Pas d'échange si A$(I)=>A$(J).

```
 I=    1  2  3  4  5

       S  E  B  A  C     première fois, échange pour
       E  S  B  A  C     I=2
       E  B  S  A  C     I=3
       E  B  A  S  C     I=4
       E  B  A  C  S     I=5, en fin de boucle E<>0

       E  B  A  C  S     deuxième fois, échange pour
       B  E  A  C  S     I=2
       B  A  E  C  S     I=3
       B  A  C  E  S     I=4, en fin de boucle E<>0

       B  A  C  E  S     troisième fois, échange pour
       A  B  C  E  S     I=2, en fin de boucle E<>0

       A  B  C  E  S     quatrième fois, pas d'échange
                         E=0, tri terminé.
```

Nous dirons quelques mots des performances comparées.
S'agissant d'un tableau aléatoirement distribué, vous pouvez déjà constater le manque d'efficacité du tri à bulles dû à la multiplicité des échanges.

Autant dire tout de suite que cette méthode offre peu d'intérêt.

TRI PAR INSERTION

La boucle principale n'est parcourue qu'une fois.
Une boucle interne (lignes 210-220) compare chaque élément, à son tour, avec ceux qui le précèdent.
Il est mis directement en place devant les éléments plus forts que lui.
L'opération est analogue à celle qu'effectue le joueur de cartes en rangeant son jeu.
L'insertion suppose le décalage d'une position des éléments plus forts.
Il est obtenu par échanges successifs de l'élément à insérer avec ceux qui le précèdent et qui lui sont supérieurs.
C'est toujours le même élément initial, d'indice I, qui remonte de proche en proche vers le haut du tableau.
Cette étape intermédiaire est esquivée dans le programme de démonstration.
Pour la faire apparaître, supprimez le GOSUB 50 de la ligne 230 et replacez-le à la ligne 220, après l'échange, avant l'instruction J=J-1.

Soit à trier par insertion le tableau A$ de cinq lettres disposées initialement comme suit:

```
          I=   1  2  3  4  5
               S  O  L  A  R
```

La boucle FOR I=2 TO 5 est exécutée une seule fois.
La boucle interne assure les échanges (insertion).

```
        1  2  3  4  5

I=2     S  O  L  A  R   échange A$(2) et A$(1)

I=3     O  S  L  A  R   échanges A$(3) et A$(2), A$(2) et A$(1)

I=4     L  O  S  A  R   échanges A$(4) et A$(3), A$(3) et A$(2)
                                A$(2) et A$(1)

I=5     A  L  O  S  R   échange A$(5) et A$(4)

        A  L  O  R  S   n'est-ce pas simple?
```

Le tri par insertion a beaucoup de qualités.
Outre sa simplicité, il est le plus rapide pour les petits tableaux.
Malheureusement, ces performances se dégradent rapidement à mesure que la longueur du tableau augmente par suite des nombreux échanges nécessaires.
A partir d'une centaine d'éléments, il ne supporte plus la comparaison avec les deux champions que nous allons voir maintenant.

TRI SHELL

Dans le tri par insertion, un élément est comparé et échangé éventuellement avec son voisin immédiat.
D.L. SHELL, inventeur de la méthode qui porte son nom, a imaginé de faire effectuer des sauts plus grands dans le tableau.
Les éléments comparés sont séparés par une distance P appelée pas.
Le tableau est parcouru successivement avec un pas de plus en plus petit, le dernier étant P=1.
Un tri par insertion est appliqué aux sous-ensembles ainsi définis.
Lorsque P=1, on revient au cas de l'insertion simple mais le processus est très rapide car le tableau est déjà pratiquement ordonné.

Pour le pas initial, on prend généralement INT(N/2) et ensuite les parties entières des divisions successives par 2.
Des théoriciens ont démontré qu'il existe des séries plus efficaces.
Ainsi, par exemple, vous pouvez remplacer les lignes 300 et 310 par les suivantes:

```
300 P=1
305 P=3*P+1:IF P<N THEN 305
310 P=(P-1)/3:IF P<1 THEN 130
```

La méthode SHELL a les faveurs de l'informatique individuelle.
C'est un heureux compromis entre la simplicité et l'efficacité.

Vous rencontrerez souvent cet algorithme (parfois appelé SHELL-METZNER) écrit en Basic de multiples façons.
Quelquefois, vous aurez du mal à le reconnaitre.
Il est présenté ici sous sa forme la plus simple.

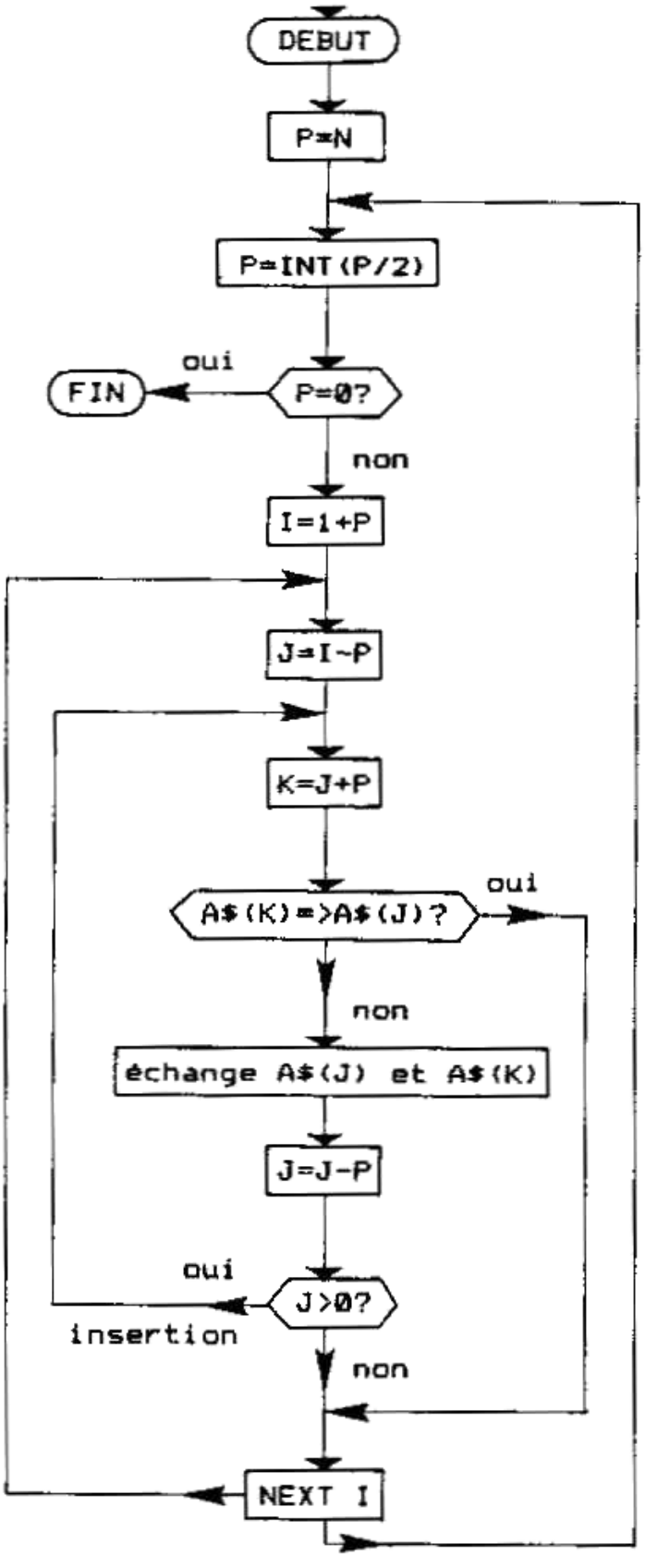
Vous remarquerez qu'il suffit de faire P=2, au lieu de P=N, à la ligne 300 et de supprimer le GOTO 310 à la ligne 350 pour retrouver le tri par insertion classique.

L'organigramme est reproduit à la page suivante.
Il servira aussi de guide pour la version en langage machine calquée le plus fidèlement possible sur la version Basic.

STABILITE DU TRI

Un tri est dit stable lorsqu'il respecte l'ordre initial des éléments égaux. C'est naturellement le cas lorsque les éléments comparés et éventuellement échangés sont contigus comme dans le tri à bulles ou le tri par insertion.
Cà ne l'est plus, par contre, quand ces mêmes éléments sont séparés; le tri est dit instable.
Ce facteur, sans importance si le tableau se résume à une simple liste, doit être pris en considération, le cas échéant, si le tableau comporte des informations associées que l'on veut conserver dans un certain ordre.

Le tri SHELL et le tri QUICKSORT, que nous allons étudier maintenant, sont instables.




```
300 P=N
310 P=INT(P/2):
    IF P=0 THEN 130
320 FOR I=1+P TO N:
    J=I-P
330 K=J+P:
    IF A$(K)=>A$(J)THEN 350
340 T$=A$(J):A$(J)=A$(K):
    A$(K)=T$: (gosub 50)
    J=J-P:
    IF J>0 THEN 330
350 NEXT:
    GOTO 310
```

tri SHELL

TRI QUICK SORT

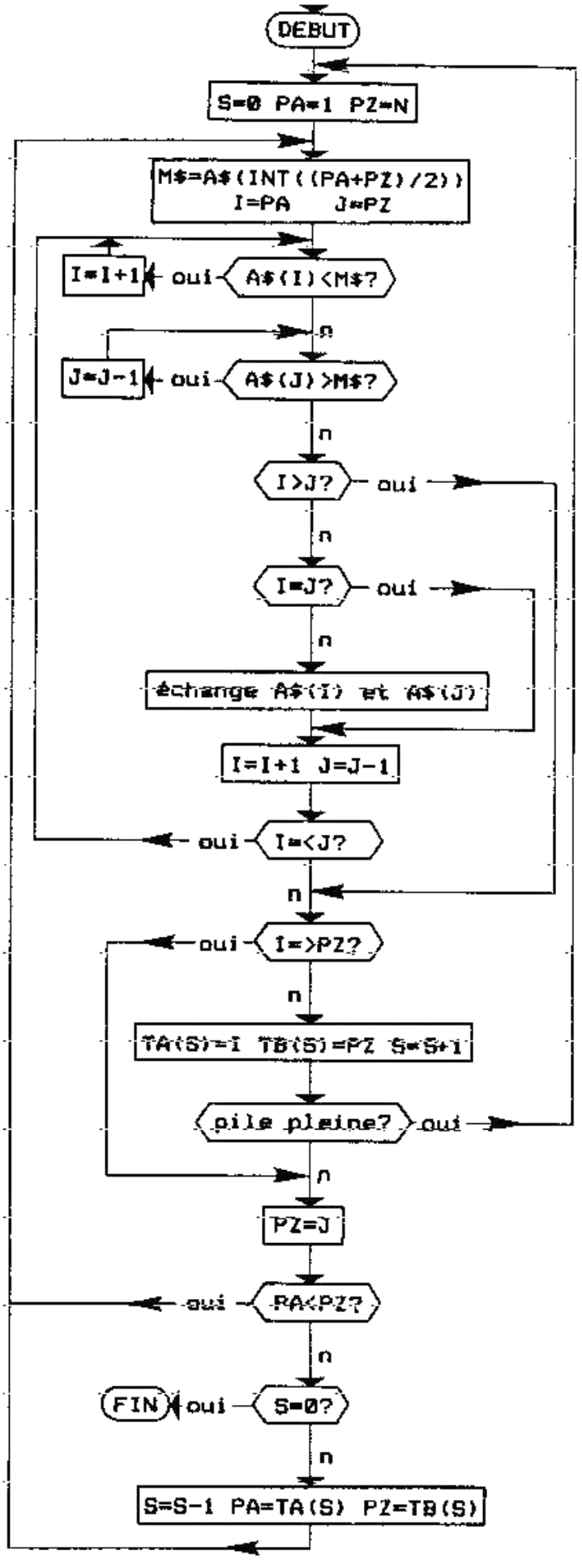
Quicksort (tri rapide) est une méthode originale qui, comme son nom l'indique, permet de trier un tableau en un temps étonnamment court.
Elle procède par partitions et échanges et de façon récursive.
Au départ de chaque itération, deux pointeurs PA et PZ sont initialisés avec le début et la fin du bloc et un élément appelé pivot est choisi.
Il peut être quelconque mais l'élément milieu du bloc donne généralement les meilleurs résultats (M$, ligne 410).
Les pointeurs PA et PZ sont transférés respectivement dans les variables de travail I et J.
I est incrémenté jusqu'à ce que A$(I) => M$ (ligne 420)
J est décrémenté jusqu'à ce que A$(J) =< M$ (ligne 430)
A$(I) et A$(J) sont échangés; I est incrémenté; J est décrémenté; si I=<J le processus est répété.
Dès que I>J, le tableau initial est partagé en deux sous-tableaux de part et d'autre de la valeur pivot.
Le premier, de PA à J, contient les éléments inférieurs ou égaux à M$
le second, de I à PZ, contient les éléments égaux ou supérieurs à M$.
Les deux sous-tableaux seront traités chacun à leur tour de la même façon et ainsi de suite par partitions successives.
Pendant le tri d'un sous-tableau, les pointeurs de début et de fin de l'autre sont empilés respectivement dans les tableaux auxiliaires TA et TB avec S formant pointeur (ligne 490).
Lorsqu'une partition est terminée (PZ=<PA, ligne 500), S est testé avant désempilage (510-520). Si S=0 le tri est terminé.

```
     PA                                Pvt                                PZ
     82  18   6   9  55  73  17  33  46   5  95  92  66   4  55  21
Ech.  1              2   3                  3                  2   1

     PA          Pvt          PZ
     21  18   6   9   4   5  17  33  46  73  95  92  66  55  55  82
Ech   1   2          2   1          ↑ ptrs début et fin empilés  ↑

     PA  Pvt PZ
      5   4   6   9  18  21  17
Ech.  1   1          ↑empilage↑
```

A ce point PZ=J devient supérieur à PA, désempilage et traitements successifs des sous-tableaux jusqu'à épuisement de la pile.

La durée du tri ne dépend pas seulement du nombre d'éléments à trier. Ainsi, le choix arbitraire de l'élément pivot peut conduire à des partitions déséquilibrées. Certains arrangements des éléments du tableau provoquent aussi un mauvais taux d'échanges. Dans ces deux cas, il y a gonflement anormal de la pile des paramètres et l'efficacité s'en ressent.
La méthode peut être optimisée par des procédures particulières de choix du pivot et de gestion de la pile.
Pour un système individuel, où la mémoire disponible limite la dimension des tableaux à quelques milliers d'éléments, la présente version simplifiée offre des performances largement suffisantes.

La taille des tableaux auxiliaires qui constituent la pile des paramètres est fonction du nombre d'éléments à trier et des facteurs évoqués ci-dessus.
Il semble que la valeur N/10 donne une bonne sécurité.
Dans le cas d'un grand tableau, il convient de les dimensionner avant le début du tri, ce qui n'est pas fait dans l'exemple présenté.
L'instruction serait de la forme DIM TA(N/10),TB(N/10) à la ligne 400.




```
400 S=0:PA=1:PZ=N

410 M$=A$(INT((PA+PZ)/2)):
    I=PA:J=PZ

420 IF A$(I)<M$ THEN I=I+1:
    GOTO 420

430 IF A$>M$ THEN J=J-1:
    GOTO 430

440 IF I>J THEN 480

450 IF I=J THEN 470

460 T$=A$(I):A$(I)=A$(J):
    A$(J)=T$: (gosub 50)

470 I=I+1:J=J-1

    IF I=<J THEN 420

480 IF I=>PZ THEN 500

490 TA(S)=I:TB(S)=PZ:S=S+1

    (langage machine)

500 PZ=J:

    IF PA<PZ THEN 410

510 IF S=0 THEN 130

520 S=S-1:PA=TA(S):PZ=TB(S):
    GOTO 410
```

TRI DES TABLEAUX ALPHANUMERIQUES

Les tableaux de chaînes de caractères sont souvent utilisés dans les applications courantes: comptabilité, gestion, palmarès, etc.
Le tri de ces tableaux présente une particularité liée au mode de traitement des chaînes par l'interpréteur Basic.
Contrairement aux valeurs numériques, la chaîne de caractère elle-même ne figure pas dans la table des variables. On n'y trouve que l'information permettant de la localiser; descripteur sur trois octets comportant la longueur et l'adresse en mémoire.
Cette adresse est interne au texte Basic lorsque l'attribution d'une 'valeur' est faite explicitement par le programme lui-même.
Par contre, la chaîne est rangée en haut de mémoire si elle est entrée sur INPUT ou en commande directe ou si elle est produite par une fonction spécialisée (CHR$, MID$, etc) ou par concaténation.
Le pointeur FRETOP (£A2,A3), haut de l'espace libre, est mis à jour.
Si la chaîne est redéfinie par la suite, la nouvelle 'valeur' prend place en dessous des autres et l'espace libre est réduit d'autant.
Voyons ce que donne la série d'instructions en mode immédiat:

```
CLEAR                                   FRETOP=HIMEM

                                      FRETOP    HIMEM
                                        ↓         ↓
A$="BLANC":B$="NOIR"                    NOIRBLANC

                          FRETOP                  HIMEM
                            ↓        < périmé   > ↓
T$=A$:A$=B$:B$=T$         BLANCNOIRBLANCNOIRBLANC
```

Imaginez cette permutation reproduite des centaines de fois au cours d'un tri et vous comprendrez que l'ordinateur soit obligé de s'arrêter pour se débarrasser des chaînes périmées.
Le tri peut devenir interminable à cause de cette opération de nettoyage.

Pour vous en convaincre, essayez le programme ci-dessous, après avoir sauvegardé le programme de démonstration.

Un tri par insertion est appliqué sur un tableau de cinq chaînes de caractères entrées au clavier. Voyez le résultat dans le cas le plus défavorable; chaînes initialement rangées dans l'ordre alphabétique inverse.

```
10 GOSUB 90:FOR I=1 TO 5
20 PRINT"entrez chaine ";I:INPUT A$(I)
30 NEXT I:PRINT
40 FOR I=2 TO 5:J=I-1
50 K=J+1:IF A$(K)=>A$(J) THEN 70
60 T$=A$(J):A$(J)=A$(K):A$(K)=T$:J=J-1:IF J>0 THEN 50
70 NEXT:FOR I=1 TO 5:PRINT A$(I):NEXT:GOSUB 90
80 PRINT DEEK(#A6)-DEEK(#A2);"octets utilises":END
90 PRINT:PRINT"FRETOP=";HEX$(DEEK(#A2)):PRINT:RETURN
```

Voyons ce qui se passe quand les chaînes de caractères figurent en DATA.
Précisons qu'en général, un tableau est destiné à être manipulé.
Il doit rester indépendant d'un programme, ne serait-ce que pour autoriser l'emploi de STORE et de RECALL.
Il est alors exclus de bloquer ses éléments dans le texte Basic.
Ajoutez les lignes suivantes:

```
15 READ A$(I):GOTO 30
100 DATA ZEBU,TAPIR,MOUTON,GAZELLE,AUTRUCHE
RUN
```

Le haut de la mémoire n'a pas été utilisé.
Tout se passe comme s'il y avait eu déplacement physique des chaînes en DATA, mais l'interpréteur s'est contenté de permuter les descripteurs des éléments à échanger.
Grâce à un peu de langage machine, nous allons appliquer cette technique à nos éléments en haut de mémoire.
Supprimez les lignes 15 et 100, remplacez la ligne 60 par:

```
60 CALL#400,A$(J),A$(K):J=J-1:IF J>0 THEN 50
```

et ajoutez les lignes:

```
5 FOR I=0 TO 31:READ C$:C=VAL("#"+C$):POKE#400+I,C:NEXT
ATMOS
1500 DATA 20,65,D0,20,88,D1,85,B8,84,B9,20,65,D0,20,88,D1
ORIC-1
1500 DATA 20,D9,CF,20,FC,D0,85,B8,84,B9,20,D9,CF,20,FC,D0
ATMOS et ORIC-1
1510 DATA A0,02,B1,B8,48,B1,B6,91,B8,68,91,B6,88,10,F3,60

RUN
```

L'occupation du haut de la mémoire est réduite au seul stockage des éléments du tableau.

```
        ATMOS              ORIC-1
0400-   JSR $D065          JSR $CFD9      CHKCOM, test virgule à TXTPTR
0403-   JSR $D188          JSR $D0FC      PTRGET, recherche 1ère variable
0406-   STA $B8            STA $B8        adr.dans $B6,B7 et dans A(-),Y(+)
0408-   STY $B9            STY $B9        transférée dans $B8,B9 ptr.temp
040A-   JSR $D065          JSR $CFD9      CHKCOM, test virgule à TXTPTR
040D-   JSR $D188          JSR $D0FC      PTRGET, recherche 2ème variable
0410-   LDY #$02           LDY #$02       adresse dans $B6,B7 (VARPNT).
0412-   LDA ($B8,),Y       LDA ($B8),Y
0414-   PHA                PHA
0415-   LDA ($B6),Y        LDA ($B6),Y    échange des descripteurs
0417-   STA ($B8),Y        STA ($B8),Y    sauvegarde intermédiaire
0419-   PLA                PLA            en pile
041A-   STA ($B6),Y        STA ($B6),Y
041C-   DEY                DEY
041D-   BPL $0412          BPL $0412
041F-   RTS                RTS
```

L'échange des descripteurs (SWAP pour les anglophones), plutôt que des chaînes de caractères elles-mêmes, présente un double avantage.
- L'efficacité du tri est améliorée par la réduction du nombre des mouvements nécessaires (déplacement de six octets seulement).
- L'arrêt pour nettoyage, si pénalisant, est évité car la consommation de mémoire due à l'échange est nulle.

Pour l'essayer avec les différentes méthodes, rechargez le programme de démonstration et modifiez-le comme suit:

```
110 CALL#400,A$(I),A$(J):E=E+1:GOSUB 50
220 CALL#400,A$(J),A$(K):J=J-1:IF J>0 THEN 210
340 CALL#400,A$(J),A$(K):GOSUB 50:J=J-P:IF J>0 THEN 330
460 CALL#400,A$(J),A$(K):GOSUB 50

1000 FOR I=0 TO 31:READ C$:C=VAL("#"+C$):POKE#400+I,C:NEXT
ATMOS
1500 DATA 20,65,D0,20,B8,D1,B5,B8,B4,B9,20,65,D0,20,B8,D1
ORIC-1
1500 DATA 20,D9,CF,20,FC,D0,B5,B8,B4,B9,20,D9,CF,20,FC,D0
ATMOS et ORIC-1
1510 DATA A0,02,B1,B8,48,B1,B6,91,B8,68,91,B6,B8,10,F3,60
```

CLEF DE TRI, INFORMATIONS ASSOCIEES

Rares sont les programmes dans lesquels l'opération de tri se limite à mettre en ordre une simple liste.
Aux éléments à ordonner sont généralement associées des informations, contenues dans des tableaux subordonnés ou dans d'autres colonnes du même tableau à deux dimensions (le plus souvent alphanumérique).
Prenons le cas d'un répertoire téléphonique où, à chaque patronyme correspondent un nom de ville et un numéro de téléphone.
On aura, par exemple:

```
A$(1)="MARTIN"      B$(1)="PARIS"      C$(1)="16(1)3332211"
A$(2)="DUPOND"      B$(2)="TOURS"      C$(2)="16(47)665544"
etc ...
ou bien:

A$(1,0)="MARTIN"    A$(1,1)="PARIS"    A$(1,2)="16(1)3332211"
A$(2,0)="DUPOND"    A$(2,1)="TOURS"    A$(2,2)="16(47)665544"
etc ...
```

Si la clef de tri est la première colonne (tri alphabétique des noms des individus), il est clair que les autres informations devront être arrangées simultanément, sinon on obtiendrait un fichier erroné et sans valeur.

Le programme de la page suivante construit un tableau des commandes Basic, de END à NEW.
C'est un tableau alphanumérique à deux dimensions dont les trois colonnes contiennent respectivement, le code de la commande, son nom et l'adresse du point d'entrée en ROM.
Un tri SHELL est proposé avec comme clef de tri l'une des trois colonnes.
Le tableau étant initialement rangé dans l'ordre croissant des codes, vous choisirez la clef 2 ou 3 pour le premier tri.
Le test porte sur les éléments clefs à la ligne 70. L'échange éventuel concerne l'ensemble des éléments associés (ligne 80).

```
10 GOSUB 200 'tableaux initiaux, code machine
20 CLS:GOSUB 300 'affichage eventuel
30 INPUT"Clef, 1.code 2.mot-cle 3.adresse";C:IF C<1 OR C>3 THEN 30

40 C=C-1:P=N:REM tri Shell
50 P=INT(P/2):IF P=0 THEN 100
60 FOR I=1+P TO N:J=I-P
70 K=J+P:IF A$(K,C)=>A$(J,C) THEN 90
80 FOR S=0 TO 2:CALL#400,A$(J,S),A$(K,S):NEXT:J=J-P:IF J>0 THEN 70
90 NEXT:GOTO 50
100 GOTO 20

200 REM code=A$(I,0) mot-cle=A$(I,1) adresse=A$(I,2)
210 TA=#C006:TM=#C0EA:Z=127:N=66:DIM A$(N,2)
220 FOR I=1 TO N:A$(I,0)=HEX$(Z+I):A$(I,1)="":REPEAT:L=PEEK(TM)
230 L$=CHR$(L+128*(L>Z)):A$(I,1)=A$(I,1)+L$:TM=TM+1:UNTIL L>Z
240 A$(I,2)=HEX$(DEEK(TA)+1):TA=TA+2:NEXT

250 FOR I=0 TO 31:READ C$:C=VAL("#"+C$):POKE#400+I,C:NEXT:RETURN
ATMOS
260 DATA 20,65,D0,20,88,D1,85,B8,84,B9,20,65,D0,20,88,D1
ORIC-1
260 DATA 20,D9,CF,20,FC,D0,85,B8,84,B9,20,D9,CF,20,FC,D0
ATMOS et ORIC-1
270 DATA A0,02,B1,B8,48,B1,B6,91,B8,68,91,B6,88,10,F3,60

300 INPUT"affichage, O/N";S$:IF S$<>"O"THEN 340
310 FOR I=1 TO N:PRINT A$(I,0),A$(I,1);
320 PRINT SPC(10-LEN(A$(I,1)));A$(I,2):IF KEY$<>"" THEN GET S$
330 NEXT
340 RETURN
```

TABLEAU ORDONNE, RECHERCHE DICHOTOMIQUE

Ce nom barbare désigne une méthode très naturelle de recherche d'un élément dans un tableau ordonné, analogue à celle employée pour trouver un mot dans un dictionnaire.
Le livre est ouvert au milieu puis, selon le classement alphabétique du mot, la recherche est poursuivie dans une partie du dictionnaire ou dans l'autre. Ainsi de suite jusqu'à localisation.
Ajoutez les lignes suivantes au programme précédent:

```
100 CLS:GOSUB 300:IF C<>1 THEN 30
110 INPUT"recherche, O/N";S$:IF S$<>"O" THEN 30
120 PA=1:PZ=N:INPUT"nom commande";N$  'recherche dichotomique
130 IF PA=>PZ THEN 180
140 M=INT((PA+PZ)/2)
150 IF N$>A$(M,1) THEN 170
160 PZ=M:GOTO 130
170 PA=M+1:GOTO 130
180 IF N$<>A$(PA,1) THEN PRINT"pas trouve":GOTO 110
190 PRINT A$(PA,1);SPC(10-LEN(A$(PA,1)));A$(PA,0),A$(PA,2):END
```

Notez une certaine similitude avec l'algorithme Quicksort (page 95) en ce qui concerne les découpages successifs du tableau.

TABLEAU ORDONNE, SUPPRESSION D'UN ELEMENT

L'élément à supprimer ayant été trouvé par dichotomie, on décale d'une position vers le haut du tableau tous ceux qui le suivent.
Le même traitement est appliqué aux renseignements associés.
L'information périmée disparaît par "écrasement".
Le nombre d'éléments N est mis à jour.
Remplacez la ligne 190 du programme précédent par les suivantes:

```
185 IF PA=N THEN 195
190 FOR I=PA TO N-1:FOR S=0 TO 2:A$(I,S)=A$(I+1,S):NEXT S,I
195 N=N-1:GOTO 20
```

INSERTION D'UN ELEMENT

L'insertion d'un élément dans un tableau ordonné est simple.
La capacité du tableau doit évidemment permettre cet ajout.
Conservez le programme précédent; il servira encore, en partie.
Entrez celui-ci (vous le supprimerez après l'essai):

```
500 DIM N$(10),A%(10):N=5:PRINT:DOKE#B0,DEEK(#AC)
510 DATA JEAN,8,MARC,18,YVES,12,JOEL,15,ERIC,10
520 FOR I=1 TO N:READ N$(I),A%(I):NEXT:GOSUB 550

530 FOR I=2 TO N:J=I-1:GOSUB 560:NEXT:GOSUB 550:N=N+1:N$(N)="PAUL"

540 INPUT"note de PAUL";A%(N):J=N-1:GOSUB 560:PRINT:GOSUB 550:END

550 FOR I=1 TO N:PRINT N$(I),A%(I):NEXT:PRINT:RETURN

560 K=J+1:IF A%(K)=<A%(J) THEN 590
570 T=A%(J):A%(J)=A%(K):A%(K)=T:T$=N$(J):N$(J)=N$(K):N$(K)=T$
580 J=J-1:IF J>0 THEN 560
590 RETURN

RUN 500
```

Un tri par insertion est effectué (ligne 530) pour classer cinq élèves d'après les notes obtenues (ordre décroissant).
Un sixième élève doit figurer dans le classement conformément à la note attribuée.
L'élément clef, la note, et l'information associée, le nom, placés initialement en fin de tableaux, sont insérés suivant la même procédure (ligne 540); comme s'il s'agissait de mettre en place le dernier élément du tableau précédemment oublié.

TRI AVEC INDEX

Lorsque les informations associées à la clef de tri sont importantes, les nombreux échanges ralentissent notablement le processus.
Le tri avec index permet de laisser ces renseignements à leur place initiale.
Soit, par exemple, les tableaux, clef=A$, info=B$, de dimension N.
Le tableau index de même dimension est construit en séquence.
Les différentes valeurs indiquent la position initiale des éléments à ordonner.

```
DIM X%(N)      FOR I=1 TO N:X%(I)=I:NEXT
```

Les éléments clef et index sont déplacés simultanément.
Quand le tri est terminé, le tableau index forme une espèce de table d'adressage indirect. La position initiale d'un élément A$(I) étant conservée en X%(I), l'information associée peut être récupérée pour affichage ou sortie sur imprimante.

```
FOR I=1 TO N:PRINT A$(I),B$(X%(I)):NEXT
```

Dans le programme qui suit, aucun des tableaux initiaux n'est modifié; le tableau clef est transféré dans le tableau temporaire T$ qui sera trié à sa place.
Les éléments index sont échangés en même temps que les éléments correspondants de T$ (ligne 110)
Grâce à l'index, l'ensemble des informations est récupéré pour affichage dans l'ordre déterminé par la clef de tri.
Notez que l'opération de tri est inutile pour les codes car le tableau est ordonné par construction suivant cette clef.

```
10 GOSUB 200 'tableaux initiaux, code machine
20 CLS:GOSUB 300 'affichage eventuel
30 INPUT"Clef, 1.code 2.mot-cle 3.adresse";C:IF C<1 OR C>3 THEN 30
40 FOR I=1 TO N:ON C GOSUB 400,410,420:X%(I)=I:NEXT

50 S=0:PA=1:PZ=N:REM tri Quicksort
60 M$=T$(INT((PA+PZ)/2)):I=PA:J=PZ
70 IF T$(I)<M$ THEN I=I+1:GOTO 70
80 IF T$(J)>M$ THEN J=J-1:GOTO 80
90 IF I>J THEN 130
100 IF I=J THEN 120
110 T=X%(I):X%(I)=X%(J):X%(J)=T:CALL#400,T$(I),T$(J)
120 I=I+1:J=J-1:IF I=<J THEN 70
130 IF I=>PZ THEN 150
140 TA(S)=I:TB(S)=PZ:S=S+1
150 PZ=J:IF PA<PZ THEN 60
160 IF S=0 THEN 20
170 S=S-1:PA=TA(S):PZ=TB(S):GOTO 60

200 REM code=C$ mot-cle=K$ adresse=A$ index=X%
210 TA=#C006:TM=#C0EA:Z=127:N=66:DIM C$(N),K$(N),A$(N),T$(N),X%(N)
220 FOR I=1 TO N:C$(I)=HEX$(Z+I):K$(I)="":REPEAT:L=PEEK(TM)
230 L$=CHR$(L+128*(L>Z)):K$(I)=K$(I)+L$:TM=TM+1:UNTIL L>Z
240 A$(I)=HEX$(DEEK(TA)+1):TA=TA+2:X%(I)=I:NEXT

250 FOR I=0 TO 31:READ C$:C=VAL("#"+C$):POKE#400+I,C:NEXT RETURN
ATMOS
260 DATA 20,65,D0,20,88,D1,85,B8,84,B9,20,65,D0,20,88,D1
ORIC-1
260 DATA 20,D9,CF,20,FC,D0,85,B8,84,B9,20,D9,CF,20,FC,D0
ATMOS et ORIC-1
270 DATA A0,02,B1,B8,48,B1,B6,91,B8,68,91,B6,88,10,F3,60

300 INPUT"affichage, O/N";S$:IF S$<>"O"THEN 340
310 FOR I=1 TO N:J=X%(I):PRINT C$(J),K$(J);
320 PRINT SPC(10-LEN(K$(J)));A$(J);IF KEY$<>"" THEN GET S$
330 NEXT
340 RETURN

400 T$(I)=C$(I):RETURN
410 T$(I)=K$(I):RETURN
420 T$(I)=A$(I):RETURN
```

CONCLUSIONS

Ces quelques pages avaient pour but de vous familiariser avec le tri en général, sujet difficile au premier abord.
Le choix d'une méthode est fonction de l'application envisagée et aussi de la mémoire vive disponible.
Les tableaux sont gourmands en mémoire. Ainsi, par exemple, un tableau à 4 colonnes pouvant contenir 500 fiches, 4 rubriques de 10 caractères, occupera plus de 26 K.octets.
Le tri par insertion convient pour les petits tableaux (une trentaine d'éléments). Il offre l'avantage d'être stable.
Le tri Shell sera choisi dans la plupart des cas.
En raison de ses performances, Quicksort pourra être préféré pour les grands tableaux (500 éléments ou plus). Il lui faut un espace de travail non négligeable (pile des paramètres).
La méthode étant choisie, reste le problème de l'adaptation au programme principal.
Les solutions pratiques sont innombrables et il y aurait encore beaucoup à dire. Notamment à propos des possibilités offertes par la concaténation ou le découpage des chaînes de caractères.
Concernant ces dernières, on aura souvent intérêt à utiliser la nique de l'échange des descripteurs (page 97).
Enfin, pour une plus grande rapidité, le tri proprement dit peut être assuré par un sous-programme en langage machine. Comme l'opération se situe dans un environnement Basic, relativement lent de nature, l'amélioration ne sera pas toujours spectaculaire.

TRI EN LANGAGE MACHINE

Les sous-programmes présentés sont une transcription aussi fidèle que possible de leurs équivalents en Basic (pages 93 et 95), avec une légère exception pour Quicksort, adaptation d'un programme publié dans une revue américaine.
La solution originale proposée par l'auteur pour la sauvegarde des paramètres, l'utilisation de la pile machine, a été conservée. Le contenu de la pile 6502 est transféré en 9E00-9EFF pendant le tri et restauré après l'opération.
Les deux routines permettent le tri simple ou avec index d'un tableau numérique ou alphanumérique.
Pour le tri avec index, un tableau à deux dimensions, de même nature que le tableau à ordonner, est construit. Il comporte deux colonnes seulement, la première représente l'index, la deuxième contient la clef de tri.
La syntaxe est la même, pour Shell ou pour Quicksort

tri normal: `CALL adresse de la routine, nom du tableau`
avec index: `CALL adresse de la routine, nom du tableau, I`

Vous trouverez le code hexadécimal dans les pages suivantes.
Je vous conseille d'entrer les deux routines en mémoire, l'une après l'autre, en 9800 et en 9B00, à l'aide de l'éditeur de langage machine.
Le code Oric-1 est plus volumineux. Sur cette machine, la routine système PTRGET (pages 108 et 116) n'est pas utilisable pour retrouver l'adresse d'un tableau, il a fallu la reconstituer; explication plus loin à propos de STORE et RECALL.

Le tri Shell deviendra un simple tri par insertion si vous apportez les modifications suivantes à l'aide du moniteur:

```
*A98A0
98A0-    A9 01        LDA #$01     fait Pas = 1
98A2-    85 42        STA $42
98A4-    A9 00        LDA #$00
98A6-    85 43        STA $43
98A8-    F0 05        BEQ $98AF    branchement forcé

*A9954
9954-    60           RTS          retour Basic
```

Quand les routines seront implantées en mémoire, vous les actionnerez à l'aide des petits programmes Basic intercalés dans le code.

A titre de comparaison, essayez-les avec le programme qui nous a servi précédemment pour le tri en Basic.
Sur Oric-1, aux lignes 50 et 240,
remplacer STR$(I) par MID$(STR$(I),2)

```
10 GOSUB 200 'tableaux initiaux
20 CLS:GOSUB 300 'affichage eventuel
30 INPUT"Clef, 1.code 2.mot-cle 3.adresse";C:IF C<1 OR C>3 THEN 30
40 INPUT"Methode, 1.Shell 2.Quicksort";M:IF M<1 OR M>2 THEN 40
50 FOR I=1 TO N:T$(I,0)=STR$(I):ON C GOSUB 400,410,420:NEXT

60 TRI=#9B00+#300*(M=1)
70 CALL TRI,T$,I:PING:GOTO 20

200 REM code=C$ mot-cle=K$ adresse=A$ clef=T$(I,1) index=T$(I,0)
210 TA=#C006:TM=#C0EA:Z=127:N=66:DIM C$(N),K$(N),A$(N),T$(N,1)
220 FOR I=1 TO N:C$(I)=HEX$(Z+I):K$(I)="":REPEAT:L=PEEK(TM)
230 L$=CHR$(L+128*(L>Z)):K$(I)=K$(I)+L$:TM=TM+1:UNTIL L>Z
240 A$(I)=HEX$(DEEK(TA)+1):TA=TA+2:T$(I,0)=STR$(I):NEXT:RETURN

300 INPUT"affichage, O/N";S$:IF S$<>"O"THEN 340
310 FOR I=1 TO N:J=VAL(T$(I,0)):PRINT C$(J),K$(J);
320 PRINT SPC(10-LEN(K$(J)));A$(J):IF KEY$<>"" THEN GET S$
330 NEXT
340 RETURN

400 T$(I,1)=C$(I):RETURN
410 T$(I,1)=K$(I):RETURN
420 T$(I,1)=A$(I):RETURN
```

SHELL ATMOS

```
9800:  20 65 D0 A9 40 85 2B 20 88 D1 A0 00 84 2B B4 3A    0674
9810:  20 E8 00 F0 0C 20 65 D0 C9 49 D0 27 C6 3A 20 E2    0DD8
9820:  00 A2 02 98 24 29 30 09 E8 A5 28 49 80 10 02 A2    12CC
9830:  05 85 3B 86 97 A0 04 B1 CE C9 02 F0 09 A5 3A 30    19A4
9840:  02 90 0F 4C 33 D3 C8 B1 CE D0 F8 C8 B1 CE C9 02    2288
9850:  D0 F1 C8 C8 38 B1 CE 85 66 E9 01 85 40 85 42 88    2BA9
9860:  B1 CE 85 67 E9 00 85 41 85 43 30 D7 24 3A 30 0C    322C
9870:  18 A5 CE 69 07 AA A5 CF 69 00 90 20 18 A5 CE 69    3952
9880:  09 85 60 A5 CF 69 00 85 61 A5 66 A4 67 20 C4 99    4096
9890:  86 66 84 67 18 8A 65 60 AA 98 65 61 86 50 85 51    4788
98A0:  46 43 A4 43 66 42 A6 42 98 D0 04 8A D0 01 60 18    4DC7
98B0:  A9 01 65 42 85 44 A9 00 65 43 85 45 38 A5 44 E5    5402
98C0:  42 85 46 A5 45 E5 43 85 47 18 A5 46 65 42 85 48    5A64
98D0:  A5 47 65 43 85 49 A5 46 A4 47 20 B8 99 86 52 85    616A
98E0:  53 A5 48 A4 49 20 B8 99 86 54 85 55 20 57 99 F0    68BC
98F0:  50 30 4E A4 97 88 B1 52 AA B1 54 91 52 8A 91 54    7051
9900:  88 10 F3 24 3A 10 25 88 A2 00 38 B5 52 E5 66 95    76B8
9910:  62 B5 53 E5 67 95 63 A2 02 C8 D0 EF A4 97 88 B1    8005
9920:  62 AA B1 64 91 62 8A 91 64 88 10 F3 38 A5 46 E5    882B
9930:  42 85 46 AA A5 47 E5 43 85 47 90 05 D0 88 8A D0    900C
9940:  88 E6 44 D0 02 E6 45 A5 40 C5 44 A5 41 E5 45 90    9849
9950:  03 4C BC 98 4C A0 98 24 3B 10 0E A5 52 A4 53 20    9DFB
9960:  7B DE A5 54 A4 55 4C 4C DF 70 1A A0 01 38 B1 54    A525
9970:  F1 52 AA 88 B1 54 F1 52 D0 03 8A F0 07 6A 51 52    AD43
9980:  51 54 09 01 60 A0 02 B1 52 99 70 00 B1 54 99 73    B311
9990:  00 88 10 F3 C8 A5 73 F0 17 AA C5 70 90 04 A6 70    BB0C
99A0:  F0 12 B1 74 D1 71 90 0C D0 0A C8 CA D0 F4 A5 73    C459
99B0:  C5 70 F0 03 6A 09 01 60 20 C4 99 8A 18 65 50 AA    CAD3
99C0:  98 65 51 60 85 E0 84 E1 A9 00 4C 56 D4             D16A
```

tri tableau alphanumérique

```
10 DIM A$(5):SHELL=#9800:QUICK=#9800
20 FORI=1 TO 5:L$="":FOR J=1 TO INT(RND(1)*7)+2
30 L$=L$+CHR$(INT(RND(1)*21+65))
40 NEXT J:A$(I)=L$:PRINT" ",I,A$(I):NEXT I
50 PRINT:GET S$:CALL SHELL,A$
60 FOR I=1 TO 5:PRINT" ",I,A$(I):NEXT:END
```

avec index

```
10 DIM A$(5,1):SHELL=#9800:QUICK=#9B00
20 FORI=1 TO 5:L$="":FOR J=1 TO INT(RND(1)*7)+2
30 L$=L$+CHR$(INT(RND(1)*21+65)):NEXT J
40 A$(I,0)=MID$(STR$(I),2):A$(I,1)=L$:PRINT" ",I,A$(I,1):NEXT
50 PRINT:GET S$:CALL QUICK,A$,I
60 FOR I=1 TO 5:PRINT I,A$(I,0),A$(I,1):NEXT:END
```

SHELL ORIC-1

```
9800: 20 D9 CF A9 40 85 2B 20 CD 99 A0 00 84 2B 84 3A   06F4
9810: 20 E8 00 F0 0C 20 D9 CF C9 49 D0 27 C6 3A 20 E2   0ECB
9820: 00 A2 02 98 24 29 30 09 E8 A5 28 49 80 10 02 A2   13BF
9830: 05 85 3B 86 97 A0 04 B1 CE C9 02 F0 09 A5 3A 30   1A97
9840: 02 90 0F 4C 9D D2 C8 B1 CE D0 F8 C8 B1 CE C9 02   2414
9850: D0 F1 C8 C8 38 B1 CE 85 66 E9 01 85 40 85 42 88   2D05
9860: B1 CE 85 67 E9 00 85 41 85 43 30 D7 24 3A 30 0C   3388
9870: 18 A5 CE 69 07 AA A5 CF 69 00 90 20 18 A5 CE 69   3AAE
9880: 09 85 60 A5 CF 69 00 85 61 A5 66 A4 67 20 C4 99   41F2
9890: 86 66 84 67 18 8A 65 60 AA 98 65 61 86 50 85 51   48E4
98A0: 46 43 A4 45 66 42 A6 42 98 D0 04 8A D0 01 60 18   4F23
98B0: A9 01 65 42 85 44 A9 00 65 43 85 45 38 A5 44 E5   555E
98C0: 42 85 46 A5 45 E5 43 85 47 18 A5 46 65 42 85 48   5BC0
98D0: A5 47 65 43 85 49 A5 46 A4 47 20 B8 99 86 52 B5   62C6
98E0: 53 A5 48 A4 49 20 B8 99 86 54 85 55 20 57 99 F0   6A18
98F0: 50 30 4E A4 97 88 B1 52 AA B1 54 91 52 8A 91 54   71AD
9900: 88 10 F3 24 3A 10 25 88 A2 00 38 B5 52 E5 66 95   7814
9910: 62 B5 53 E5 67 95 63 A2 02 C8 D0 EF A4 97 88 B1   8161
9920: 62 AA B1 64 91 62 8A 91 64 88 10 F3 38 A5 46 E5   8987
9930: 42 85 46 AA A5 47 E5 43 85 47 90 05 D0 88 8A D0   9168
9940: 88 E6 44 D0 02 E6 45 A5 40 C5 44 A5 41 E5 45 90   99A5
9950: 03 4C BC 98 4C A0 98 24 3B 10 0E A5 52 A4 53 20   9F57
9960: 73 DE A5 54 A4 55 4C 34 DF 70 1A A0 01 38 B1 54   A661
9970: F1 52 AA 88 B1 54 F1 52 D0 03 8A F0 07 6A 51 52   AE7F
9980: 51 54 09 01 60 A0 02 B1 52 99 70 00 B1 54 99 73   B44D
9990: 00 88 10 F3 C8 A5 73 F0 17 AA C5 70 90 04 A6 70   BC48
99A0: F0 12 B1 74 D1 71 90 0C D0 0A C8 CA D0 F4 A5 73   C595
99B0: C5 70 F0 03 6A 09 01 60 20 C4 99 8A 18 65 50 AA   CC0F
99C0: 98 65 51 60 85 E0 84 E1 A9 00 4C AE D3 20 E8 00   D405
99D0: 85 B4 20 86 D1 B0 03 4C E4 CF A2 00 86 28 86 29   DB66
99E0: 20 E2 00 90 05 20 86 D1 90 0B AA 20 E2 00 90 FB   E246
99F0: 20 86 D1 B0 F6 C9 24 D0 06 A9 FF 85 28 D0 0C C9   EB20
9A00: 25 D0 0F A9 80 85 29 05 B4 85 B4 8A 09 80 AA 20   F1CA
9A10: E2 00 86 B5 A6 9E A5 9F 86 CE 85 CF C5 A1 D0 04   FB51
9A20: E4 A0 F0 20 A0 00 B1 CE C8 C5 B4 D0 06 A5 B5 D1   0546
9A30: CE F0 16 C8 B1 CE 18 65 CE AA C8 B1 CE 65 CF 90   0F61
9A40: D7 A2 6B 2C A2 2A 4C 85 C4 60                     1432
```

tri tableau numérique réels

```
10 DIM A(5):SHELL=#9800:QUICK=#9B00
20 FOR I=1 TO 5:A(I)=RND(1):PRINT I,A(I):NEXT
30 PRINT:GET S$:CALL SHELL,A
40 FOR I=1 TO 5:PRINT I,A(I):NEXT:END
```

avec index

```
10 DIM A(5,1):SHELL=#9800:QUICK=#9B00
20 FOR I=1 TO 5:A(I,0)=I:A(I,1)=RND(1):PRINT" ",I,A(I,1):NEXT
30 PRINT:GET S$:CALL QUICK,A,I
40 FOR I=1 TO 5:PRINT I,A(I,0),A(I,1):NEXT:END
```

QUICKSORT ATMOS

```
9B00: 20 65 D0 A9 40 85 2B 20 88 D1 A0 00 84 2B 84 3A   0674
9B10: 20 E8 00 F0 0C 20 65 D0 C9 49 D0 42 C6 3A 20 E2   0DF3
9B20: 00 A2 02 98 24 29 30 0C E8 A0 02 A5 28 49 80 10   12E8
9B30: 03 A2 05 C8 85 3B 86 97 84 3D A0 02 18 B1 CE 65   1996
9B40: CE 08 38 E5 97 85 40 C8 B1 CE E9 00 28 65 CF 85   21F6
9B50: 41 C8 B1 CE C9 02 F0 09 A5 3A 30 02 90 0F 4C 33   2871
9B60: D3 C8 B1 CE D0 F8 C8 B1 CE C9 02 D0 F1 C8 C8 38   33EE
9B70: B1 CE E9 02 85 E0 88 B1 CE E9 00 85 E1 90 DF BA   3E3C
9B80: 86 5A BD 00 01 9D 00 9E E8 D0 F7 8A 9A 20 56 D4   4632
9B90: 86 E0 84 E1 38 A2 FE B5 42 95 5D F5 E2 95 44 95   5003
9BA0: 5F E8 D0 F3 A5 97 0A 65 E0 85 E0 90 02 E6 E1 CA   5A20
9BB0: A5 40 6A B0 01 CA 86 3C C6 97 10 22 A5 42 C5 40   6127
9BC0: A5 43 E5 41 90 18 BA D0 0D A6 5A 9A BD 00 9E 9D   6906
9BD0: 00 01 E8 D0 F7 60 A2 03 68 95 40 CA 10 FA A2 03   7071
9BE0: B5 40 95 44 CA 10 F9 18 A5 40 65 42 AA A5 41 65   77AB
9BF0: 43 6A A8 8A 6A 90 06 18 65 3D 90 01 C8 24 3B 30   7D2C
9C00: 16 70 04 09 01 25 3C 85 CE 84 CF A4 97 B1 CE 99   841A
9C10: D0 00 88 10 F8 30 03 20 7B DE A5 46 A4 47 20 D4   8AF0
9C20: 9C F0 0A 30 08 20 AF 9C D0 F2 20 BB 9C A5 44 A4   92EF
9C30: 45 20 D4 9C 30 F4 20 C7 9C 90 44 F0 37 A4 97 B1   9B52
9C40: 46 AA B1 44 91 46 8A 91 44 88 10 F3 A5 3A F0 24   A2EB
9C50: 88 A2 00 38 B5 44 E5 E0 95 50 B5 45 E5 E1 95 51   AB96
9C60: A2 02 C8 D0 EF A4 97 B1 50 AA B1 52 91 50 8A 91   B4A6
9C70: 52 88 10 F3 20 AF 9C 20 BB 9C 20 C7 9C B0 9B A5   BCD8
9C80: 46 C5 40 A5 47 E5 41 B0 1B A5 40 48 A5 41 48 A5   C400
9C90: 46 48 A5 47 48 BA D0 0C A2 03 B5 5B 95 40 CA 10   CABC
9CA0: F9 4C DE 9B A5 44 85 40 A5 45 85 41 4C BC 9B 38   D2B3
9CB0: A5 97 65 46 85 46 90 02 E6 47 60 18 A5 44 E5 97   DA01
9CC0: 85 44 B0 02 C6 45 60 A5 45 C5 47 90 06 D0 04 A5   E0EC
9CD0: 44 C5 46 60 24 3B 10 03 4C 4C DF 85 CE 84 CF 70   E79A
9CE0: 1A A0 01 38 B1 CE E5 D1 AA 88 B1 CE E5 D0 D0 03   F0FB
9CF0: 8A F0 07 6A 45 D0 51 CE 09 01 60 A0 02 B1 CE 99   F83E
9D00: 53 00 88 D0 F8 B1 CE F0 19 AA C5 D0 90 04 A6 D0   01B2
9D10: F0 14 B1 54 D1 D1 90 0E D0 0C C8 CA D0 F4 A0 00   0ACD
9D20: B1 CE C5 D0 F0 D4 6A 09 01 60                     1079
```

tri tableau numérique entiers

```
10 DIM A%(5):SHELL=#9800:QUICK=#9800
20 FOR I=1 TO 5:A%(I)=INT(RND(1)*32768)*(1+2*(RND(1)<.5))
30 PRINT" ",I,A%(I):NEXT
40 PRINT:GET S$:CALL QUICK,A%
50 FOR I=1 TO 5:PRINT" ",I,A%(I):NEXT:END
```

avec index

```
10 DIM A%(5,1):SHELL=#9800:QUICK=#9800
20 FORI=1 TO 5:A%(I,1)=INT(RND(1)*32768)*(1+2*(RND(1)<.5))
30 A%(I,0)=I:PRINT" ",I,A%(I,1):NEXT
40 PRINT:GET S$:CALL SHELL,A%,I
50 FOR I=1 TO 5:PRINT I,A%(I,0),A%(I,1):NEXT:END
```

```
9B00: 20 D9 CF A9 40 85 2B 20 2A 9D A0 00 B4 2B B4 3A  0655
9B10: 20 EB 00 F0 0C 20 D9 CF C9 49 D0 42 C6 3A 20 E2  0E47
9B20: 00 A2 02 98 24 29 30 0C E8 A0 02 A5 28 49 80 10  133C
9B30: 03 A2 05 C8 85 3B 86 97 84 3D A0 02 18 B1 CE 65  19EA
9B40: CE 08 38 E5 97 85 40 C8 B1 CE E9 00 28 65 CF 85  224A
9B50: 41 C8 B1 CE C9 02 F0 09 A5 3A 30 02 90 0F 4C 9D  292F
9B60: D2 C8 B1 CE D0 F8 C8 B1 CE C9 02 D0 F1 C8 C8 38  34AB
9B70: B1 CE E9 02 85 E0 88 B1 CE E9 00 85 E1 90 DF BA  3EF9
9B80: 86 5A BD 00 01 9D 00 9E E8 D0 F7 8A 9A 20 AE D3  4746
9B90: 86 E0 84 E1 38 A2 FE B5 42 95 5D F5 E2 95 44 95  5117
9BA0: 5F E8 D0 F3 A5 97 0A 65 E0 85 E0 90 02 E6 E1 CA  5B34
9BB0: A5 40 6A B0 01 CA 86 3C C6 97 10 22 A5 42 C5 40  623B
9BC0: A5 43 E5 41 90 18 BA D0 0D A6 5A 9A BD 00 9E 9D  6A1A
9BD0: 00 01 E8 D0 F7 60 A2 03 68 95 40 CA 10 FA A2 03  7185
9BE0: B5 40 95 44 CA 10 F9 18 A5 40 65 42 AA A5 41 65  78BF
9BF0: 43 6A A8 8A 6A 90 06 18 65 3D 90 01 C8 24 3B 30  7E40
9C00: 16 70 04 09 01 25 3C 85 CE 84 CF A4 97 B1 CE 99  852E
9C10: D0 00 88 10 F8 30 03 20 73 DE A5 46 A4 47 20 D4  88FC
9C20: 9C F0 0A 30 08 20 AF 9C D0 F2 20 BB 9C A5 44 A4  93FB
9C30: 45 20 D4 9C 30 F4 20 C7 9C 90 44 F0 37 A4 97 B1  9C5E
9C40: 46 AA B1 44 91 46 8A 91 44 88 10 F3 A5 3A F0 24  A3F7
9C50: 88 A2 00 38 B5 44 E5 E0 95 50 B5 45 E5 E1 95 51  ACA2
9C60: A2 02 C8 D0 EF A4 97 B1 50 AA B1 52 91 50 8A 91  B5B2
9C70: 52 88 10 F3 20 AF 9C 20 BB 9C 20 C7 9C B0 9B A5  BDE4
9C80: 46 C5 40 A5 47 E5 41 B0 1B A5 40 48 A5 41 48 A5  C50C
9C90: 46 48 A5 47 48 BA D0 0C A2 03 B5 5B 95 40 CA 10  CBC8
9CA0: F9 4C DE 9B A5 44 85 40 A5 45 85 41 4C BC 9B 38  D3BF
9CB0: A5 97 65 46 85 46 90 02 E6 47 60 18 A5 44 E5 97  DB0D
9CC0: 85 44 B0 02 C6 45 60 A5 45 C5 47 90 06 D0 04 A5  E1F8
9CD0: 44 C5 46 60 24 3B 10 03 4C 34 DF 85 CE 84 CF 70  E8BE
9CE0: 1A A0 01 38 B1 CE E5 D1 AA 88 B1 CE E5 D0 D0 03  F1EF
9CF0: 8A F0 07 6A 45 D0 51 CE 09 01 60 A0 02 B1 CE 99  F932
9D00: 53 00 88 D0 F8 B1 CE F0 19 AA C5 D0 90 04 A6 D0  02A6
9D10: F0 14 B1 54 D1 D1 90 0E D0 0C C8 CA D0 F4 A0 00  0BC1
9D20: B1 CE C5 D0 F0 D4 6A 09 01 60 20 E8 00 85 B4 20  13CE
9D30: 86 D1 B0 03 4C E4 CF A2 00 86 28 86 29 20 E2 00  1AD8
9D40: 90 05 20 86 D1 90 0B AA 20 E2 00 90 FB 20 86 D1  222D
9D50: B0 F6 C9 24 D0 06 A9 FF 85 28 D0 0C C9 25 D0 0F  2A94
9D60: A9 00 85 29 05 B4 85 B4 8A 09 80 AA 20 E2 00 86  31A2
9D70: 85 A6 9E A5 9F 86 CE 85 CF C5 A1 D0 04 C4 A0 F0  3C35
9D80: 20 A0 00 B1 CE C8 C5 B4 D0 06 A5 B5 D1 CE F0 16  458A
9D90: C8 B1 CE 18 65 CE AA C8 B1 CE 65 CF 90 D7 A2 6B  4FB5
9DA0: 2C A2 2A 4C 85 C4 60                             52A2
```

```
                        ORG $9800      Tri SHELL
                        OBJ $9800

                VALTYP  =$28 variable système, 00 = nombre, FF = chaîne
                INTFLG  =$29    "         "   , 00 = réel, 80 = entier
                SUBFLG  =$2B    "         "   , 00 = indice permis, 80 = non
                IDXFLG  =$3A drapeau, 00 = tri sans index, FF tri avec index
                TYPFLG  =$3B type var., 00 = entier, 80 = réel, 7F = chaîne
                N       =$40 nombre d'éléments (indice 0 non utilisé)
                P       =$42 pas
                I       =$44 )
                J       =$46 ) indices
                K       =$48 )
                PT0     =$50 pointeur élément clef indice 0
                PTJ     =$52    "       "      "     "    J
                PTK     =$54    "       "      "     "    K
                PX0     =$60 pointeur élément index indice 0
                PXJ     =$62    "       "      "     "    J
                PXK     =$64    "       "      "     "    K
                D       =$66 déplacement (N+1) * 2, 3 ou 5
                LJ      =$70 )descripteur A$(J): longueur
                AJ      =$71 )                   adresse
                LK      =$73 )descripteur A$(K): longueur
                AK      =$74 )                   adresse
                LENFLG  =$97 2, 3 ou 5 selon type (voir fin listing, MULT)
                LOWTR   =$CE contient adresse tableau trouvée par PTRGET
                STRNG2  =$E0 voir fin listing, MULT.
                CHRGET  =$E2 recueille dans A caractère à TXTPTR
                CHRGOT  =$E8      ;Oric-1
                CHKCOM  =$D065    ;$CFD9 test virgule à TXTPTR
                PTRGET  =$D188    ;$D0FC recherche variable, adr en LOWTR
                SUBERR  =$D333    ;$D29D bad subscript error
                MULT    =$D456    ;$D3AE voir fin listing.
                MOVFM   =$DE7B    ;$DE73 nombre réel en A(-),Y(+) -> FAC
                FCOMP   =$DF4C    ;$DF34 compare FAC à nombre en A(-),Y(+)


                        ;Recueil paramètres
                        ;LOWTR= adr.tableau

9800 20 65 D0 START     JSR CHKCOM vérifie présence virgule à TXTPTR
9803 A9 40              LDA #$40   SUBFLG = 40 ...
9805 85 2B              STA SUBFLG pour recherche tableau par ...
9807 20 88 D1           JSR PTRGET PTRGET. Adresse en LOWTR (#CE,CF).
980A A0 00              LDY #$00   SUBFLG ...
980C 84 2B              STY SUBFLG remis à zéro.   (Oric-1, voir page 125)
980E 84 3A              STY IDXFLG IDXFLG initialisé à zéro..

                        ;tri avec index?

9810 20 E8 00           JSR CHRGOT fin d'instruction (: ou 00)
9813 F0 0C              BEQ NOIND  oui, tri sans index.
9815 20 65 D0           JSR CHKCOM non, test virgule
9818 C9 49              CMP #"I"   caractère après virgule est-il 'I'?
981A D0 27              BNE ERR    non, erreur
981C C6 3A              DEC IDXFLG oui, positionne drapeau index.
981E 20 E2 00           JSR CHRGET déplace TXTPTR, caractère suivant.
```

```
                        ;type de variable

9821 A2 02     NOIND    LDX #$02        X = 02
9823 98                 TYA             A = 00
9824 24 29              BIT INTFLG      est-ce un nombre entier?
9826 30 09              BMI SETFLG      oui, LENFLG = 2 octets
9828 E8                 INX             X = 03
9829 A5 28              LDA VALTYP      nombre ou chaîne?
982B 49 80              EOR #$80        test 00 ou FF
982D 10 02              BPL SETFLG      chaîne, LENFLG = 3 octets
982F A2 05              LDX #$05        nombre réel, LENFLG = 5 octets

                        ;pos.drapeaux

9831 85 3B     SETFLG   STA TYPFLG      00 = entier, 80 = réel, 7F = chaîne
9833 86 97              STX LENFLG      2, 3 ou 5 octets.

                        ;nb. dimensions
                        ;doit être <3

9835 A0 04              LDY #$04        recherche nombre de dimensions
9837 B1 CE              LDA (LOWTR),Y   à adr. tableau + 4.
9839 C9 02              CMP #$02        est-ce 2?
983B F0 09              BEQ 2DIM        oui
983D A5 3A              LDA IDXFLG      non, alors IDXFLG ...
983F 30 02              BMI ERR         doit être 00, sinon erreur.
9841 90 0F              BCC DIM1        OK, 1 dimension
9843 4C 33 D3  ERR      JMP SUBERR      bad subscript error.

                        ;si index, 2e dim.
                        ;doit avoir 2 élém.

9846 C8        2DIM     INY
9847 B1 CE              LDA (LOWTR),Y   octet fort doit être 00
9849 D0 F8              BNE ERR         sinon erreur.
984B C8                 INY
984C B1 CE              LDA (LOWTR),Y   octet faible ...
984E C9 02              CMP #$02        doit être 02
9850 D0 F1              BNE ERR         sinon, erreur.

                        ;nb.élém 1e dim.
                        ;rangé en D

9852 C8        DIM1     INY             nombre total d'éléments, y compris
9853 C8                 INY             indice 0 (N+1)
9854 38                 SEC
9855 B1 CE              LDA (LOWTR),Y   octet faible
9857 85 66              STA D           rangé en D
9859 E9 01              SBC #$01        nb. d'éléments utilisés (1 à N)
985B 85 40              STA N           rangé en N
985D 85 42              STA P           et en P (pas)
985F 88                 DEY
9860 B1 CE              LDA (LOWTR),Y   octet fort total éléments
9862 85 67              STA D+1         en D+1.
9864 E9 00              SBC #$00
9866 85 41              STA N+1
9868 85 43              STA P+1
986A 30 D7              BMI ERR         err. si 0 élém.
```

```
                        ;rech. ptr clef(0)

986C 24 3A              BIT IDXFLG      IDXFLG = FF?
986E 30 0C              BMI R1          oui, deux dimensions.

                        ;1 dimension

9870 18                 CLC
9871 A5 CE              LDA LOWTR       adresse élément clef indice 0
9873 69 07              ADC #$07
9875 AA                 TAX                  = adresse tableau + 7
9876 A5 CF              LDA LOWTR+1
9878 69 00              ADC #$00
987A 90 20              BCC R2          branchement forcé.

                        ;2 dimensions

987C 18         R1      CLC
987D A5 CE              LDA LOWTR       adresse élément index indice 0
987F 69 09              ADC #$09
9881 85 60              STA PX0              = adresse tableau + 9
9883 A5 CF              LDA LOWTR+1
9885 69 00              ADC #$00
9887 85 61              STA PX0+1

                        ;calc.déplacement

9889 A5 66              LDA D           utilise routine multiplication
988B A4 67              LDY D+1
988D 20 C4 99           JSR MLT         déplacement = D * LENFLG
9890 86 66              STX D
9892 84 67              STY D+1         rangé en D.

                        ;PT0=PX0+D

9894 18                 CLC
9895 8A                 TXA             adresse élément clef indice 0
9896 65 60              ADC PX0
9898 AA                 TAX                  = adresse élém.index(0) + D
9899 98                 TYA
989A 65 61              ADC PX0+1

                        ;adresse A(0)

989C 86 50      R2      STX PT0         adresse clef (0)
989E 85 51              STA PT0+1       rangée en PT0.

                        ;nouveau pas
                        ;sortie si P=0

98A0 46 43      NEWP    LSR P+1         P = P/2 (valeur initiale P = N)
98A2 A4 43              LDY P+1
98A4 66 42              ROR P
98A6 A6 42              LDX P           si P = 0 ...
98A8 98                 TYA             fin du tri ...
98A9 D0 04              BNE LOOP
98AB 8A                 TXA
98AC D0 01              BNE LOOP
98AE 60                 RTS             sortie
```

```
                        ;I=1+P

98AF 18         LOOP    CLC
98B0 A9 01              LDA #$01
98B2 65 42              ADC P
98B4 85 44              STA I
98B6 A9 00              LDA #$00
98B8 65 43              ADC P+1
98BA 85 45              STA I+1

                        ;J=I-P

98BC 38         LOOP1   SEC
98BD A5 44              LDA I
98BF E5 42              SBC P
98C1 85 46              STA J
98C3 A5 45              LDA I+1
98C5 E5 43              SBC P+1
98C7 85 47              STA J+1

                        ;K=J+P

98C9 18         LOOP2   CLC
98CA A5 46              LDA J
98CC 65 42              ADC P
98CE 85 48              STA K
98D0 A5 47              LDA J+1
98D2 65 43              ADC P+1
98D4 85 49              STA K+1

                        ;adr.éléments J,K

98D6 A5 46              LDA J           recherche adresses
98D8 A4 47              LDY J+1
98DA 20 B8 99           JSR LOC         des éléments clef
98DD 86 52              STX PTJ
98DF 85 53              STA PTJ+1       d'indice J et K
98E1 A5 48              LDA K
98E3 A4 49              LDY K+1         rangées dans pointeurs
98E5 20 B8 99           JSR LOC
98E8 86 54              STX PTK         PTJ et PTK.
98EA 85 55              STA PTK+1

                        ;IF A(K)=>A(J) THEN
                        ;NEXT I

98EC 20 57 99           JSR COMPAR      comparaison des éléments
98EF F0 50              BEQ NEXTI       si A(K) = A(J)
98F1 30 4E              BMI NEXTI       si A(K) > A(J)
```

```
                        ;échange valeurs

98F3 A4 97              LDY LENFLG      échange des 'valeurs' clef ...
98F5 88                 DEY             sur 2, 3 ou 5 octets ...
98F6 B1 52      E1      LDA (PTJ),Y     selon LENFLG.
98F8 AA                 TAX             sauvegarde intermédiaire
98F9 B1 54              LDA (PTK),Y     en X.
98FB 91 52              STA (PTJ),Y
98FD 8A                 TXA
98FE 91 54              STA (PTK),Y
9900 88                 DEY
9901 10 F3              BPL E1          sortie boucle, Y = FF

                        ;tri avec index?

9903 24 3A              BIT IDXFLG      si IDXFLG = 0
9905 10 25              BPL NEXTJ       next J

                        ;rech.ptrs.index

9907 88                 DEY             recherche des pointeurs index
9908 A2 00              LDX #$00        PXJ et PXK
990A 38                 SEC
990B B5 52      E2      LDA PTJ,X       sont égaux à ...
990D E5 66              SBC D           PTJ ou PTK - déplacement D
990F 95 62              STA PXJ,X
9911 B5 53              LDA PTJ+1,X
9913 E5 67              SBC D+1
9915 95 63              STA PXJ+1,X
9917 A2 02              LDX #$02
9919 C8                 INY             en début boucle Y = FE,
991A D0 EF              BNE E2          sortie si Y = 0

                        ;et échange

991C A4 97              LDY LENFLG      échange valeurs index
991E 88                 DEY             idem valeurs clef ci-dessus.
991F B1 62      E3      LDA (PXJ),Y
9921 AA                 TAX
9922 B1 64              LDA (PXK),Y
9924 91 62              STA (PXJ),Y
9926 8A                 TXA
9927 91 64              STA (PXK),Y
9929 88                 DEY
992A 10 F3              BPL E3
```

```
                        ;J=J-P:IF J>0 THEN.

992C 38        NEXTJ    SEC
992D A5 46              LDA J
992F E5 42              SBC P
9931 85 46              STA J           après soustraction ...
9933 AA                 TAX             next I si J négatif ...
9934 A5 47              LDA J+1         ou J = 0.
9936 E5 43              SBC P+1
9938 85 47              STA J+1
993A 90 05              BCC NEXTI       sinon poursuite ...
993C D0 BB              BNE LOOP2       boucle interne (insertion)
993E 8A                 TXA
993F D0 88              BNE LOOP2

                        ;NEXTI jusqu'à I=N

9941 E6 44     NEXTI    INC I           incrémentation de I, variable FOR ...
9943 D0 02              BNE NXTI0       de la boucle principale.
9945 E6 45              INC I+1
9947 A5 40     NXTI0    LDA N
9949 C5 44              CMP I           comparaison avec N, ...
994B A5 41              LDA N+1         valeur de fin de boucle.
994D E5 45              SBC I+1
994F 90 03              BCC NP          si fin de boucle, nouveau pas
9951 4C BC 98           JMP LOOP1       sinon, next I.

                        ;on recommence avec
                        ;un nouveau pas

9954 4C A0 98  NP       JMP NEWP


                        ;comparaison
                        ;A(J) et A(K)

9957 24 3B     COMPAR   BIT TYPFLG      si bit 7 TYPFLG = 0
9959 10 0E              BPL CMP0        il ne s'agit pas de valeurs réelles

                        ;nombres réels

995B A5 52     CMPREL   LDA PTJ         nombre réel pointé par ...
995D A4 53              LDY PTJ+1       A(-) et Y(+)
995F 20 7B DE           JSR MOVFM       est mis en FAC
9962 A5 54              LDA PTK         nombre réel en mémoire ...
9964 A4 55              LDY PTK+1       pointé par A(-) et Y(+) ...
9966 4C 4C DF           JMP FCOMP       est comparé à FAC.

9969 70 1A     CMP0     BVS CMPSTR      bit 6 de TYPFLG = 1 (7F) -> chaînes
```

```
                        ;nombres entiers

9968 A0 01     CMPINT LDY #$01        comparaison
996D 38                SEC            de deux nombres 16 bits signés.
996E B1 54             LDA (PTK),Y
9970 F1 52             SBC (PTJ),Y    octets faibles ...
9972 AA                TAX
9973 88                DEY
9974 B1 54             LDA (PTK),Y
9976 F1 52             SBC (PTJ),Y    octets forts.
9978 D0 03             BNE ROR1
997A 8A                TXA            Z=1 si nombres égaux
997B F0 07             BEQ RTS1
997D 6A        ROR1    ROR A          ajuste indicateur de signe N
997E 51 52             EOR (PTJ),Y    selon valeurs relatives signées
9980 51 54             EOR (PTK),Y    EOR octets forts.
9982 09 01             ORA #$01       force indicateur Z à zéro
9984 60        RTS1    RTS            retour.

                        ;chaînes caractères
                        ;descripteurs en
                        ;LJ,AJ et LK,AK

9985 A0 02     CMPSTR LDY #$02        LJ = longueur A$(J) sur 1 octet
9987 B1 52     CPS1   LDA (PTJ),Y    AJ = adresse A$(J) sur 2 octets
9989 99 70 00          STA LJ,Y
998C B1 54             LDA (PTK),Y    LK = longueur A$(K) sur 1 octet
998E 99 73 00          STA LK,Y       AK = adresse A$(K) sur 2 octets
9991 88                DEY
9992 10 F3             BPL CPS1

                        ;comp.chaînes

9994 C8                INY            Y = 00
9995 A5 73             LDA LK         si LK = 0 et si LJ = 0 sortie Z = 1
9997 F0 17             BEQ CPS3       si LK = 0 et si LJ <> 0 sortie N = 0
9999 AA                TAX
999A C5 70             CMP LJ         compteur de boucle X = longueur
999C 90 04             BCC CPS2       la plus faible.
999E A6 70             LDX LJ
99A0 F0 12             BEQ ROR2
99A2 B1 74     CPS2    LDA (AK),Y     comparaison caractères ...
99A4 D1 71             CMP (AJ),Y     successifs
99A6 90 0C             BCC ROR2       A$(K) < A$(J), sortie N = 0
99A8 D0 0A             BNE ROR2       A$(K) > A$(J), sortie N = 0
99AA C8                INY            inc. index caractère suivant
99AB CA                DEX            déc. compteur boucle
99AC D0 F4             BNE CPS2       caractère suivant.
99AE A5 73             LDA LK
99B0 C5 70     CPS3    CMP LJ         tous caractères égaux
99B2 F0 03             BEQ RTS2       compare longueurs à nouveau.
99B4 6A        ROR2    ROR A          positionnement ...
99B5 09 01             ORA #$01       indicateurs Z et N.
99B7 60        RTS2    RTS            sortie.
```

```
                        ;Localisation, ADR
                        ;=PT0+indice*LENFLG

9988 20 C4 99 LOC     JSR MLT       sous-routine appelée par la section
998B 8A               TXA           chargée de la recherche des adresses
998C 18               CLC           des éléments A(J) et A(K) (98D6)
998D 65 50            ADC PT0       le déplacement, calculé par MLT, est
998F AA               TAX           ajouté à PT0, adresse du 1er élément
99C0 98               TYA           du tableau (non utilisé)
99C1 65 51            ADC PT0+1
99C3 60               RTS

                        ;mult.16 bits

99C4 85 E0     MLT    STA STRNG2
99C6 84 E1            STY STRNG2+1  voir remarque ci-dessous
99C8 A9 00            LDA #$00
99CA 4C 56 D4         JMP MULT

Oric-1 seulement
99CD 20 E8 00 GTARYP JSR CHRGOT  voir pages 125 et 131
```

Remarque:
En D44D sur Atmos, D3A5 sur Oric-1, on trouve une routine de multiplication de nombres entiers sur 16 bits qui sert uniquement pour les opérations sur les tableaux. (LOWTR),Y, rangé en #97,98, est multiplié par STRNG2. Résultat en X(-1), A et Y(+).
MLT, ci-dessus, utilise en partie cette routine; entrée en D456 sur Atmos, en D3AE sur Oric-1.
Le nombre d'octets occupés par la valeur de l'élément, LENFLG en #97, est multiplié par l'indice placé en STRNG2 (#E0,E1).

```
                        ORG $9B00       Tri QUICKSORT
                        OBJ $9B00

                VALTYP =$28     )
                INTFLG =$29     |
                SUBFLG =$2B     |   voir tri SHELL
                IDXFLG =$3A     |
                TYPFLG =$3B     )
                INTMSK =$3C     masque pour entiers
                ADJFLG =$3D     ajuste pointeur pivot
                PTZ    =$40     pointeur dernier élément
                PTA    =$42     pointeur premier élément
                J      =$44     variable temp. pour PTZ
                I      =$46     variable temp. pour PTA
                PXJ    =$50     temp.index J
                PXI    =$52     temp.index I
                STMP   =$54     temp.pour chaînes
                SPT    =$5A     sauvegarde pointeur de pile S
                SAVE   =$5B     sauvegarde PTA et PTZ
                LENFLG =$97     )  voir tri SHELL
                LOWTR  =$CE     )
                DSCTMP =$D0     descripteur temporaire
                STRNG2 =$E0     )
                CHRGET =$E2     |  voir tri SHELL
                CHRGOT =$E8     )
                STACK  =$100    pile
                STKSV  =$9E00   9E00-9EFF, zone sauvegarde pile
                CHKCOM =$D065   ;CFD9   )
                PTRGET =$D188   ;D0FC   |
                SUBERR =$D333   ;D29D   |  voir tri SHELL
                MULT   =$D456   ;D3AE   |
                MOVFM  =$DE7B   ;DE73   |
                FCOMP  =$DF4C   ;DF34   )

                        ;recueil paramètres

9B00 20 65 D0 START     JSR CHKCOM
9B03 A9 40              LDA #$40
9B05 85 2B              STA SUBFLG
9B07 20 88 D1           JSR PTRGET          voir tri SHELL
9B0A A0 00              LDY #$00
9B0C 84 2B              STY SUBFLG
9B0E 84 55              STY IDXFLG

                        ;tri avec index?

9B10 20 E8 00           JSR CHRGOT
9B13 F0 0C              BEQ NOIND
9B15 20 65 D0           JSR CHKCOM
9B18 C9 49              CMP #"I"            voir tri SHELL
9B1A D0 42              BNE ERR
9B1C C6 55              DEC IDXFLG
9B1E 20 E2 00           JSR CHRGET
```

```
                        ;type de variable

9B21 A2 02     NOIND    LDX #$02
9B23 98                 TYA
9B24 24 29              BIT INTFLG
9B26 30 0C              BMI SETFLG
9B28 E8                 INX
9B29 A0 02              LDY #$02          voir tri SHELL
9B2B A5 28              LDA VALTYP
9B2D 49 80              EOR #$80
9B2F 10 03              BPL SETFLG
9B31 A2 05              LDX #$05
9B33 C8                 INY

                        ;pos.drapeaux

9B34 85 3B     SETFLG   STA TYPFLG
9B36 86 97              STX LENFLG
9B38 84 3D              STY ADJFLG     0, 2 ou 3 pour ajuster ptr. pivot

                        ;ptr.dernier élém.

9B3A A0 02              LDY #$02       recherche de l'adresse du dernier
9B3C 18                 CLC            élément clef.
9B3D B1 CE              LDA (LOWTR),Y
9B3F 65 CE              ADC LOWTR      Le déplacement servant à déterminer
9B41 08                 PHP            l'adresse du tableau suivant, 3ème
9B42 38                 SEC            et 4ème octet de l'en-tête, est
9B43 E5 97              SBC LENFLG     ajouté à l'adresse du tableau en
9B45 85 40              STA PTZ        LOWTR.
9B47 C8                 INY            L'adresse obtenue est diminuée de
9B48 B1 CE              LDA (LOWTR),Y  2, 3 ou 5 (LENFLG) selon le type
9B4A E9 00              SBC #$00       de variable.
9B4C 28                 PLP
9B4D 65 CF              ADC LOWTR+1    Le résultat est inscrit en PTZ,
9B4F 85 41              STA PTZ+1      pointeur du dernier élément.

                        ;nb.dimensions

9B51 C8                 INY
9B52 B1 CE              LDA (LOWTR),Y
9B54 C9 02              CMP #$02
9B56 F0 09              BEQ 2DIM          voir tri SHELL
9B58 A5 3A              LDA IDXFLG
9B5A 30 02              BMI ERR
9B5C 90 0F              BCC DIM1
9B5E 4C 33 D3  ERR      JMP SUBERR

9B61 C8        2DIM     INY
9B62 B1 CE              LDA (LOWTR),Y
9B64 D0 F8              BNE ERR
9B66 C8                 INY               voir tri SHELL
9B67 B1 CE              LDA (LOWTR),Y
9B69 C9 02              CMP #$02
9B6B D0 F1              BNE ERR
```

```
                        ;prépare mult.
                        ;le dim-2 * LENFLG

9B6D C8         DIM1    INY             nombre d'éléments 1ère dimension
9B6E C8                 INY             (6ème et 7ème ou 8ème et 9ème octets
9B6F 38                 SEC             de l'en-tête) diminué de 2 est rangé
9B70 B1 CE              LDA (LOWTR),Y   en STRNG2 (#E0,E1) pour la multipli-
9B72 E9 02              SBC #$02        cation ultérieure par LENFLG (9B80)
9B74 85 E0              STA STRNG2
9B76 88                 DEY
9B77 B1 CE              LDA (LOWTR),Y
9B79 E9 00              SBC #$00
9B7B 85 E1              STA STRNG2+1
9B7D 90 DF              BCC ERR         1ère dim. doit être > 0.

                        ;sauvegarde pile
                        ;et ptr.S

9B7F BA                 TSX
9B80 86 5A              STX SPT         sauvegarde pointeur de pile
9B82 BD 00 01 D1        LDA STACK,X
9B85 9D 00 9E           STA STKSV,X     sauvegarde contenu de la pile
9B88 E8                 INX             en 9E00-9EFF.
9B89 D0 F7              BNE D1

                        ;init.S, exéc mult.
                        ;résult. ds STRNG2

9B8B 8A                 TXA             S = 0. La pile sert maintenant à
9B8C 9A                 TXS             sauv. début et fin des s/tableaux
9B8D 20 56 D4           JSR MULT        mult. STRNG2 par LENFLG
9B90 86 E0              STX STRNG2      résultat dans STRNG2.
9B92 84 E1              STY STRNG2+1    (voir fin listing tri SHELL).

                        ;calcul.ptr.déb.PTA
                        ;sauv. PTZ et PTA
                        ;dans SAVE

9B94 38                 SEC             PTA = PTZ - STRNG2
9B95 A2 FE              LDX #$FE
9B97 B5 42      D2      LDA PTZ+2,X     les deux pointeurs sont sauvegardés
9B99 95 5D              STA SAVE+2,X    dans SAVE (4 octets).
9B9B F5 E2              SBC STRNG2+2,X
9B9D 95 44              STA PTA+2,X
9B9F 95 5F              STA SAVE+4,X
9BA1 E8                 INX
9BA2 D0 F3              BNE D2

                        ;STRNG2 =
                        ;longueur s/tableau

9BA4 A5 97              LDA LENFLG      longueur s/tableau =
9BA6 0A                 ASL A           LENFLG * 2
9BA7 65 E0              ADC STRNG2      + STRNG2.
9BA9 85 E0              STA STRNG2
9BAB 90 02              BCC D3          Résultat dans STRNG2.
9BAD E6 E1              INC STRNG2+1
```

```
                        ;ajuste INTMSK
                        ;pour entiers
                        ;et LENFLG
                        ;début tri

9BAF CA         D3      DEX             INTMSK prend la valeur #FF ou FE
9BB0 A5 40              LDA PTZ         selon que PTZ est pair ou impair.
9BB2 6A                 ROR A           En combinaison avec ADJFLG,
9BB3 B0 01              BCS ODD         ajuste l'adresse du pivot (midélém)
9BB5 CA                 DEX             pour recueil d'une valeur entière
9BB6 86 3C      ODD     STX INTMSK      (voir page suivante).
9BB8 C6 97              DEC LENFLG      déc. LENFLG pour indexation (9C0B)
9BBA 10 22              BPL MIDEL       branchement forcé (début tri).

                        ;comp.ptrs.

9BBC A5 42      CMPTR   LDA PTA         sous-tableau traité?
9BBE C5 40              CMP PTZ
9BC0 A5 43              LDA PTA+1       (PTA >= PTZ?)
9BC2 E5 41              SBC PTZ+1
9BC4 90 18              BCC MIDEL       non.

                        ;pile épuisée?

9BC6 BA                 TSX             terminé?
9BC7 D0 0D              BNE NEXT        non.

                        ;oui, restaure
                        ;et retour Basic

9BC9 A6 5A              LDX SPT         restauration de la pile
9BCB 9A                 TXS
9BCC BD 00 9E R1        LDA STKSV,X
9BCF 9D 00 01           STA STACK,X
9BD2 E8                 INX
9BD3 D0 F7              BNE R1
9BD5 60                 RTS

                        ;désempile ptrs
                        ;nouv.s/tableau

9BD6 A2 03      NEXT    LDX #$03        renseigne
9BD8 68         N1      PLA             PTA et PTZ
9BD9 95 40              STA PTZ,X
9BDB CA                 DEX
9BDC 10 FA              BPL N1

                        ;I=PTA, J=PTZ

9BDE A2 03      MIDEL   LDX #$03        initialise I et J
9BE0 B5 40      M1      LDA PTZ,X
9BE2 95 44              STA J,X
9BE4 CA                 DEX
9BE5 10 F9              BPL M1
```

```
                        ;rech.élém.milieu

9BE7 18                 CLC
9BE8 A5 40              LDA PTZ
9BEA 65 42              ADC PTA
9BEC AA                 TAX
9BED A5 41              LDA PTZ+1       adresse du pivot
9BEF 65 43              ADC PTA+1
9BF1 6A                 ROR A           = (PTA+PTZ) / 2
9BF2 A8                 TAY
9BF3 8A                 TXA
9BF4 6A                 ROR A
9BF5 90 06              BCC VARTYP
9BF7 18                 CLC             ajustée par ADJFLG
9BF8 65 3D              ADC ADJFLG      si résultat impair
9BFA 90 01              BCC VARTYP
9BFC C8                 INY

                        ;type de variable

9BFD 24 3B      VARTYP  BIT TYPFLG      nombre réel?
9BFF 30 16              BMI PUTFAC      oui, mis en FAC
9C01 70 04              BVS STRINT      branchement si chaîne
9C03 09 01              ORA #$01        ajuste l'accumulateur
9C05 25 3C              AND INTMSK      si nombre entier.

9C07 85 CE      STRINT  STA LOWTR       adresse 'valeur' à comparer
9C09 84 CF              STY LOWTR+1     rangée en LOWTR.
9C0B A4 97              LDY LENFLG
9C0D B1 CE      S1      LDA (LOWTR),Y   nombre entier ou descripteur
9C0F 99 D0 00           STA DSCTMP,Y    de chaîne rangé dans DSCTMP.
9C12 88                 DEY
9C13 10 F8              BPL S1
9C15 30 03              BMI CPAI        branchement forcé.

9C17 20 7B DE PUTFAC    JSR MOVFM       nombre réel en A(-),Y(+) mis en FAC

                        ;compare midelem
                        ;à A(I) et A(J)
                        ;boucles

9C1A A5 46      CPAI    LDA I           comparaison pivot
9C1C A4 47      CP1     LDY I+1         avec élément A(I)
9C1E 20 D4 9C           JSR COMPAR      par appel de COMPAR
9C21 F0 0A              BEQ CPAJ        branchement si
9C23 30 08              BMI CPAJ        midelem <= A(I)
9C25 20 AF 9C           JSR ADD         inc.I (I=I+2/3/5).
9C28 D0 F2              BNE CP1         branchement forcé.
9C2A 20 BB 9C CP2       JSR SBC         déc.J (J=J-2/3/5).
9C2D A5 44      CPAJ    LDA J           comparaison pivot
9C2F A4 45              LDY J+1         avec élément A(J)
9C31 20 D4 9C           JSR COMPAR      par appel de COMPAR
9C34 30 F4              BMI CP2         branch. si midelem < A(J)
9C36 20 C7 9C           JSR CMPIJ       compare I et J
9C39 90 44              BCC POS         branchement si J < I.
9C3B F0 37              BEQ NOCH        pas d'échange si I = J.
```

```
                        ;échange A(I)
                        ;et A(J)

9C3D A4 97              LDY LENFLG
9C3F B1 46      EXIJ    LDA (I),Y
9C41 AA                 TAX             sauvegarde intermédiaire en X
9C42 B1 44              LDA (J),Y
9C44 91 46              STA (I),Y
9C46 8A                 TXA
9C47 91 44              STA (J),Y
9C49 88                 DEY
9C4A 10 F3              BPL EXIJ        en sortie de boucle Y = FF

                        ;et évent.index

9C4C A5 3A              LDA IDXFLG      tri avec index?
9C4E F0 24              BEQ NOCH        non

                        ;rech.ptrs.index

9C50 88                 DEY             Y = FE
9C51 A2 00              LDX #$00
9C53 38                 SEC
9C54 B5 44      RX      LDA J,X         PXI = I - longueur s/tableau
9C56 E5 E0              SBC STRNG2
9C58 95 50              STA PXJ,X
9C5A B5 45              LDA J+1,X
9C5C E5 E1              SBC STRNG2+1    PXJ = J - longueur s/tableau
9C5E 95 51              STA PXJ+1,X
9C60 A2 02              LDX #$02
9C62 C8                 INY
9C63 D0 EF              BNE RX

                        ;échange index

9C65 A4 97              LDY LENFLG      même procédure que
9C67 B1 50      EXIN    LDA (PXJ),Y     pour les éléments clef
9C69 AA                 TAX
9C6A B1 52              LDA (PXI),Y
9C6C 91 50              STA (PXJ),Y
9C6E 8A                 TXA
9C6F 91 52              STA (PXI),Y
9C71 88                 DEY
9C72 10 F3              BPL EXIN

                        ;I=I+1
                        ;J=J-1
                        ;compare J et I

9C74 20 AF 9C   NOCH    JSR ADD
9C77 20 B8 9C           JSR SBC
9C7A 20 C7 9C           JSR CMPIJ
9C7D B0 98              BCS CPAI        branchement si J >= I.
```

```
                        ;compare I à PTZ

9C7F A5 46     POS      LDA I
9C81 C5 40              CMP PTZ
9C83 A5 47              LDA I+1
9C85 E5 41              SBC PTZ+1
9C87 B0 18              BCS DNOT     branchement si I >= PTZ

9C89 A5 40     POS1     LDA PTZ      sinon
9C8B 48                 PHA          empilage PTZ
9C8C A5 41              LDA PTZ+1
9C8E 48                 PHA
9C8F A5 46              LDA I        et I
9C91 48                 PHA
9C92 A5 47              LDA I+1
9C94 48                 PHA

                        ;pile pleine?

9C95 BA                 TSX
9C96 D0 0C              BNE DNOT     non

                        ;oui, récup.PTA et
                        ;PTZ et recommencer

9C98 A2 03              LDX #$03
9C9A B5 5B     SNP1     LDA SAVE,X
9C9C 95 40              STA PTZ,X
9C9E CA                 DEX
9C9F 10 F9              BPL SNP1
9CA1 4C DE 9B           JMP MIDEL

                        ;PTZ=J

9CA4 A5 44     DNOT     LDA J
9CA6 85 40              STA PTZ
9CA8 A5 45              LDA J+1
9CAA 85 41              STA PTZ+1
9CAC 4C BC 9B           JMP CMPTR    sous-tableau suivant

                        ;**************

                        ;sous/routines

                        ;I=I+2/3/5

9CAF 38        ADD      SEC
9CB0 A5 97              LDA LENFLG
9CB2 65 46              ADC I
9CB4 85 46              STA I
9CB6 90 02              BCC RTS1
9CB8 E6 47              INC I+1
9CBA 60        RTS1     RTS
```

```
                        ;J=J-2/3/5

9CBB 18        SBC      CLC
9CBC A5 44               LDA J
9CBE E5 97               SBC LENFLG
9CC0 85 44               STA J
9CC2 B0 02               BCS RTS2
9CC4 C6 45               DEC J+1
9CC6 60        RTS2      RTS

                        ;comp. I et J
                        ;à la sortie
                        ;C=0 si J<I
                        ;Z=1 si J=I
                        ;C=1 si J>=I

9CC7 A5 45     CMPIJ     LDA J+1
9CC9 C5 47               CMP I+1
9CCB 90 06               BCC RTS3
9CCD D0 04               BNE RTS3
9CCF A5 44               LDA J
9CD1 C5 46               CMP I
9CD3 60        RTS3      RTS

                        ;compar. pivot et
                        ;A(I) ou A(J).

9CD4 24 3B     COMPAR    BIT TYPFLG      signe TYPFLG?
9CD6 10 03               BPL CMP0        positif, entiers ou chaînes.

                        ;réels

9CD8 4C 4C DF  CMPREL    JMP FCOMP       réel en A(-),Y(+) comparé à FAC.

9CDB 85 CE     CMP0      STA LOWTR       adresse A(I) ou A(J)
9CDD 84 CF               STY LOWTR+1     en LOWTR.
9CDF 70 1A               BVS CMPSTR      chaînes si bit 6 TYPFLG = 1

                        ;entiers

9CE1 A0 01     CMPINT    LDY #$01        comparaison entiers signés
9CE3 38                  SEC
9CE4 B1 CE               LDA (LOWTR),Y   (voir tri SHELL, 996B)
9CE6 E5 D1               SBC DSCTMP+1
9CE8 AA                  TAX
9CE9 88                  DEY
9CEA B1 CE               LDA (LOWTR),Y
9CEC E5 D0               SBC DSCTMP
9CEE D0 03               BNE ROR1
9CF0 8A                  TXA
9CF1 F0 07               BEQ RTS4
9CF3 6A        ROR1      ROR A
9CF4 45 D0               EOR DSCTMP
9CF6 51 CE               EOR (LOWTR),Y
9CF8 09 01               ORA #$01
9CFA 60        RTS4      RTS
```

```
                        ;chaînes

9CFB A0 02     CMPSTR LDY #$02
9CFD B1 CE     CPS1   LDA (LOWTR),Y     descripteur A(I) ou A(J)
9CFF 99 53 00         STA STMP-1,Y      rangé dans STMP.
9D02 88               DEY
9D03 D0 F8            BNE CPS1

9D05 B1 CE            LDA (LOWTR),Y     comparaison avec chaîne pivot
9D07 F0 19            BEQ CDSC          en DSCTMP.
9D09 AA               TAX
9D0A C5 D0            CMP DSCTMP
9D0C 90 04            BCC CPS2          (voir tri SHELL, 9994)
9D0E A6 D0            LDX DSCTMP
9D10 F0 14            BEQ ROR2
9D12 B1 54     CPS2   LDA (STMP),Y
9D14 D1 D1            CMP (DSCTMP+1),Y
9D16 90 0E            BCC ROR2
9D18 D0 0C            BNE ROR2
9D1A C8               INY
9D1B CA               DEX
9D1C D0 F4            BNE CPS2
9D1E A0 00            LDY #$00
9D20 B1 CE            LDA (LOWTR),Y
9D22 C5 D0     CDSC   CMP DSCTMP
9D24 F0 D4            BEQ RTS4
9D26 6A        ROR2   ROR A
9D27 09 01            ORA #$01
9D29 60               RTS

Oric-1 seulement
9D2A 20 E8 00 GTARYP JSR CHRGOT  voir pages 125 et 131
 ...

ATTENTION
Toute la page #9E (9E00-9EFF) est réservée à la sauvegarde de la pile.


Quicksort par C.BONGERS
NIBBLE n°4/1982
```

STORE ET RECALL POUR ORIC1

ENREGISTREMENT ET LECTURE DE TABLEAUX SUR CASSETTE

Il m'a semblé opportun, au terme de cette partie consacrée aux tableaux, de proposer cette commodité aux possesseurs d'Oric-1.

La transposition des routines de l'Atmos n'a pas été aussi facile qu'il y paraissait au premier abord, la ROM ayant été sensiblement remaniée dans le but d'intégrer les nouvelles commandes.
Les modifications concernent les routines CSAVE et CLOAD mais aussi PTRGET, recherche de l'adresse d'une variable.
La fonction originelle de PTRGET (en D0FC sur Oric-1) est de trouver une variable simple ou indicée nommément désignée.
Dans la nouvelle version, cette routine permet également de retrouver l'en-tête d'un tableau. A cet effet, avant l'appel, SUBFLG (2B) est positionné à #40. Des tests supplémentaires de ce drapeau (par BVS et BNE) détournent PTRGET de manière appropriée. A la sortie, l'adresse du tableau est détenue par le pointeur LOWTR (#CE,CF).

En bref, la routine PTRGET, n'étant pas utilisable sur Oric-1 pour retrouver un tableau, a dû être reconstituée dans ce but exclusif.
Le résultat est GTARYP à la page 131.
Elle est présentée sous forme de sous-routine séparée car elle remplace aussi PTRGET dans les routines de tri SHELL et QUICKSORT (pages 108 et 116) où le même problème se posait. Elle est placée tout à la fin.
Les instructions concernant SUBFLG sont sans objet. Elles ont été conservées en vue de maintenir le même arrangement du listing pour les deux systèmes.
En examinant attentivement le début de GTARYP (99CD ou 9D2A), vous remarquerez une différence mineure: JSR CHRGOT au lieu de CHRGET. Il s'agit simplement d'ajuster correctement TXTPTR sur le nom du tableau.

Puisque nous sommes entre nous, profitons-en pour évoquer deux défauts de l'Oric-1.
L'un concerne le mauvais positionnement de HIMEM à l'initialisation.
On peut le corriger aisément par HIMEM#9800 ou DOKE#A6,#9800.
L'autre, plus détestable, provoque l'altération du pointeur Basic VARTAB (#9C,9D), début de la zone des variables, après chargement d'un programme en langage machine. Cela oblige à charger le code machine AVANT un programme Basic ou sinon, avant lancement de ce dernier, à redonner à VARTAB sa valeur correcte.

Revenons à STORE/RECALL.
La routine peut être appelée indifféremment en commande directe ou par un programme; le magnétophone étant en marche dans la position requise, enregistrement ou lecture.
La syntaxe est la suivante:

- Pour STORE: CALL#A000,CSAVE nom du tableau,"appellation" (,S)

- Pour RECALL: CALL#A000,CLOAD nom du tableau,"appellation" (,S)

Le tableau devra être dimensionné au préalable pour recevoir les éléments en provenance de la cassette.

Remarque:
Dans le programme d'essai de la page suivante, les éléments du tableau A$, obtenus par concaténation, sont placés en haut de mémoire.

```
A000:  20 D9 CF C9 B7 F0 07 C9 B6 F0 6B 4C E4 CF 08 20    0940
A010:  F8 A0 20 BA E6 A9 24 20 C6 E5 A2 09 B5 5D 20 C6    11D3
A020:  E5 CA D0 F8 B5 35 F0 06 20 C6 E5 E8 D0 F6 20 C6    1C89
A030:  E5 A2 00 CA D0 FD 20 63 E5 A9 BC A0 E5 20 76 E5    2674
A040:  20 6E E5 20 2D A1 20 A7 E5 24 28 10 22 A0 00 B1    2C50
A050:  10 F0 17 AA A0 02 B1 10 99 D0 00 88 D0 F8 E8 CA    34DF
A060:  F0 08 B1 D1 20 C6 E5 C8 D0 F5 20 4A A1 90 DE 20    3E4A
A070:  04 E8 28 4C F9 E7 20 95 D5 08 20 F8 A0 20 63 E5    463C
A080:  A9 03 A0 E5 20 76 E5 20 96 E6 20 30 E6 C9 24 D0    4E77
A090:  F9 A2 09 20 30 E6 95 5D CA D0 F8 20 30 E6 F0 05    5700
A0A0:  95 49 E8 D0 F6 95 49 20 F0 E6 8A D0 D0 A0 02 B1    60DD
A0B0:  CE C5 5F C8 B1 CE E5 60 B0 06 20 04 E8 4C 83 C4    69B0
A0C0:  20 2D A1 20 D8 E4 24 28 10 27 A0 00 B1 10 F0 1C    6F6A
A0D0:  20 F0 D4 A0 00 AA E8 CA F0 08 20 30 E6 91 D1 C8    78A2
A0E0:  D0 F5 A0 02 B9 D0 00 91 10 88 D0 F8 20 4A A1 90    811E
A0F0:  D9 20 04 E8 28 4C F9 E7 20 5D A1 20 D9 CF A0 03    88E0
A100:  B1 CE 85 60 88 B1 CE 85 5F D0 02 C6 60 C6 5F A5    91F1
A110:  29 48 A5 28 48 20 25 E7 68 85 28 68 85 29 A5 63    97D6
A120:  D0 04 05 64 F0 03 4C E4 CF 20 CA E6 60 18 A5 CE    9FC0
A130:  65 5F 85 61 A5 CF 65 60 85 62 A0 04 B1 CE 20 F6    A7C3
A140:  D1 85 5F 84 60 85 10 84 11 60 18 A9 03 65 10 85    ADA4
A150:  10 90 02 E6 11 A8 A5 11 C4 61 E5 62 60 20 E2 00    B469
A160:  85 B4 20 86 D1 B0 03 4C E4 CF A2 00 86 28 86 29    BBCA
A170:  20 E2 00 90 05 20 86 D1 90 0B AA 20 E2 00 90 FB    C2AA
A180:  20 86 D1 B0 F6 C9 24 D0 06 A9 FF 85 28 D0 0C C9    CB84
A190:  25 D0 0F A9 80 85 29 05 B4 85 B4 8A 09 80 AA 20    D22E
A1A0:  E2 00 86 B5 A6 9E A5 9F 86 CE 85 CF C5 A1 D0 04    DBB5
A1B0:  E4 A0 F0 20 A0 00 B1 CE C8 C5 B4 D0 06 A5 B5 D1    E5AA
A1C0:  CE F0 16 C8 B1 CE 18 65 CE AA C8 B1 CE 65 CF 90    EFC5
A1D0:  D7 A2 6B 2C A2 2A 4C 85 C4 60                      F496
```

```
10 DIM D$(20),A$(20):FOR I=1 TO 20:READ D$(I
20 A$(I)=D$(I)+D$(I):NEXT:CLS
30 PRINT"magneto position enreg.? O/N":GOSUB 100
40 CALL#A000,CSAVE A$,"T",S
50 PRINT:PRINT:CLEAR:DIM A$(20)
60 PRINT"rembobinez, position lecture? O/N":GOSUB 100
70 CALL#A000,CLOAD A$,"T",S
80 CLS:PRINT"et voila le tableau!":PRINT
90 FOR I=1 TO 20:PRINT A$(I):NEXT:END
100 GET S$:IF S$<>"O"THEN 100
110 PRINT"attendez ...":RETURN
150 DATA DA,PAN,BE,BO,ZI,PA,DI,LO,PI,CA
160 DATA DO,ME,TU,FE,GA,TON,LU,MI,NE,TA
```

```
ORIC-1      ORG $A000      STORE/RECALL
            OBJ $A000

TMP     =$10     pointeur temporaire
VALTYP  =$28     drapeau, 00=nombre, FF=chaîne
INTFLG  =$29     drapeau, 00=nombre réel, 80=nombre entier
BUF     =$35     début du buffer d'entrée
DEF     =$5F     début d'enregistrement
FIN     =$61     fin d'enregistrement
AUTO    =$63     drapeau, 00=non AUTO, <>00=AUTO
BASIC   =$64     drapeau, 00=Basic, <>00=code machine
ARYTB   =$9E     pointeur, début des tableaux
STEND   =$A0     pointeur, fin zone stockage variables
VARNM   =$B4     2 octets, nom dernière variable utilisée
LTR     =$CE     pointeur, nombr.usages, ici adr.tableau
DT      =$D0     3 octets, descripteur temporaire
CHRGET  =$E2     inc.TXTPTR et recueille caractère en A
CHRGOT  =$E8     recueille caractère à TXTPTR en A
MEMERR  =$C483   erreur 'OUT OF MEMORY'
ERRMSG  =$C485   affiche erreur,  X=index dans table
CHKCOM  =$CFD9   vérifie présence virgule à TXTPTR
SYNTER  =$CFE4   erreur de SYNTAXE
ISLETC  =$D186   teste la présence d'une lettre dans A.
GETARY  =$D1F6   adr.1er élém. en A(-),Y(+) selon nb.dim
STRSPA  =$D4F0   prépare install. chaine, desc.en DT (#D0)
GARBAG  =$D595   nettoyage mémoire
LOAD    =$E4D8   chargement programme ou données
CLRSTL  =$E563   nettoie la ligne supérieure
STOUT1  =$E56E   affiche message début ligne supérieure
STOUT2  =$E576   affiche le nom de l'enregistrement
SAVE    =$E5A7   enregistrement programme ou données
OUTBYT  =$E5C6   sort contenu de A vers magnétophone
RDBYTE  =$E630   recueille en A octet lu sur cassette
GETSYN  =$E696   synchronisation
DELAY   =$E6BA   temporisation
VIAON   =$E6CA   déconnect.clavier pour E/S magnétophone
CHKNAM  =$E6F0   vérifie nom sur cassette
GTPRMS  =$E725   lecture paramètres CLOAD ou CSAVE
EXIT    =$E7F9   sortie via test mode immédiat (#A9)
VIAOFF  =$EB04   reconnection clavier.


A000 20 D9 CF START  JSR CHKCOM   virgule? si oui carac.suivant
A003 C9 B7           CMP #$B7     CSAVE?
A005 F0 07           BEQ STORE    oui, STORE
A007 C9 B6           CMP #$B6     CLOAD?
A009 F0 6B           BEQ RECALL   oui, RECALL
A00B 4C E4 CF        JMP SYNTER   aucun des deux, erreur de syntaxe
```

```
A00E 08          STORE  PHP
A00F 20 F8 A0           JSR GTINFO    Nb octets tableau dans DEP
A012 20 BA E6           JSR DELAY
A015 A9 24              LDA #$24
A017 20 C6 E5           JSR OUTBYT
A01A A2 09              LDX #$09      enregistre
A01C B5 5D       ST0    LDA $5D,X     en-tête
A01E 20 C6 E5           JSR OUTBYT    info
A021 CA                 DEX
A022 D0 F8              BNE ST0
A024 B5 35       ST1    LDA BUF,X     enregistre
A026 F0 06              BEQ ST2       nom
A028 20 C6 E5           JSR OUTBYT
A02B E8                 INX
A02C D0 F6              BNE ST1
A02E 20 C6 E5 ST2       JSR OUTBYT
A031 A2 00              LDX #$00      temporisation
A033 CA          TEMPO  DEX
A034 D0 FD              BNE TEMPO
A036 20 63 E5           JSR CLRSTL    efface ligne supérieure
A039 A9 BC              LDA #$BC
A03B A0 E5              LDY #$E5
A03D 20 76 E5           JSR STOUT2    affiche "Saving"
A040 20 6E E5           JSR STOUT1    affiche nom
A043 20 2D A1           JSR GTPTRS    début et fin tableau
A046 20 A7 E5           JSR SAVE      enregistrement tableau
A049 24 28              BIT VALTYP    chaînes?
A04B 10 22              BPL STFIN     non, terminé
A04D A0 00       STSTR  LDY #$00
A04F B1 10              LDA (TMP),Y   recueil
A051 F0 17              BEQ SNXSTR    et
A053 AA                 TAX           enregistrement
A054 A0 02              LDY #$02      éléments
A056 B1 10       STST0  LDA (TMP),Y   alphanumériques
A058 99 D0 00           STA DT,Y
A05B 88                 DEY
A05C D0 F8              BNE STST0
A05E E8                 INX
A05F CA          STST1  DEX
A060 F0 08              BEQ SNXSTR
A062 B1 D1              LDA (DT+1),Y
A064 20 C6 E5           JSR OUTBYT
A067 C8                 INY
A068 D0 F5              BNE STST1
A06A 20 4A A1 SNXSTR    JSR NXTDSC    descripteur suivant
A06D 90 DE              BCC STSTR
A06F 20 04 E8 STFIN     JSR VIAOFF    reconnection clavier
A072 28                 PLP
A073 4C F9 E7           JMP EXIT
```

```
A076 20 95 D5 RECALL JSR GARBAG    nettoie haut mémoire
A079 08              PHP
A07A 20 F8 A0        JSR GTINFO
A07D 20 63 E5 REC0   JSR CLRSTL
A080 A9 03           LDA #$03
A082 A0 E5           LDY #$E5
A084 20 76 E5        JSR STOUT2    affiche "Searching"
A087 20 96 E6        JSR GETSYN    synchronisation
A08A 20 30 E6 REC1   JSR RDBYTE
A08D C9 24           CMP #$24
A08F D0 F9           BNE REC1
A091 A2 09           LDX #$09
A093 20 30 E6 REC2   JSR RDBYTE    lit info
A096 95 5D           STA $5D,X     en-tête
A098 CA              DEX           et le
A099 D0 F8           BNE REC2      range
A09B 20 30 E6 REC3   JSR RDBYTE    lit nom
A09E F0 05           BEQ REC4      le range
A0A0 95 49           STA $49,X     en $49
A0A2 E8              INX
A0A3 D0 F6           BNE REC3
A0A5 95 49     REC4  STA $49,X
A0A7 20 F0 E6        JSR CHKNAM    test nom
A0AA 8A              TXA
A0AB D0 D0           BNE REC0      pas bon, recherche
A0AD A0 02           LDY #$02
A0AF B1 CE           LDA (LTR),Y   test
A0B1 C5 5F           CMP DEP       capacité
A0B3 C8              INY           tableau
A0B4 B1 CE           LDA (LTR),Y   accueil
A0B6 E5 60           SBC DEP+1
A0B8 B0 06           BCS RECOK     OK
A0BA 20 04 E8        JSR VIAOFF
A0BD 4C 83 C4        JMP MEMERR    sortie, OUT OF MEMORY
A0C0 20 2D A1 RECOK  JSR GTPTRS    début et fin
A0C3 20 D8 E4        JSR LOAD      chargement du tableau
A0C6 24 28           BIT VALTYP    chaînes?
A0C8 10 27           BPL RFIN      non, terminé
A0CA A0 00     RSTR  LDY #$00
A0CC B1 10           LDA (TMP),Y   lecture
A0CE F0 1C           BEQ RNXSTR    des
A0D0 20 F0 D4        JSR STRSPA    éléments
A0D3 A0 00           LDY #$00      chaînes
A0D5 AA              TAX           de
A0D6 E8              INX           caractères
A0D7 CA        REC5  DEX           et
A0D8 F0 08           BEQ REC6      rangement
A0DA 20 30 E6        JSR RDBYTE    en mémoire
A0DD 91 D1           STA (DT+1),Y
A0DF C8              INY
A0E0 D0 F5           BNE REC5
A0E2 A0 02     REC6  LDY #$02
A0E4 B9 D0 00 REC7   LDA DT,Y
A0E7 91 10           STA (TMP),Y
A0E9 88              DEY
A0EA D0 F8           BNE REC7
A0EC 20 4A A1 RNXSTR JSR NXTDSC    descripteur suivant
A0EF 90 D9           BCC RSTR
A0F1 20 04 E8 RFIN   JSR VIAOFF    reconnection clavier
A0F4 28              PLP
A0F5 4C F9 E7        JMP EXIT
```

```
A0F8 20 5D A1 GTINFO JSR GTARYP     recherche adresse tableau
A0FB 20 D9 CF        JSR CHKCOM     virgule?
A0FE A0 03           LDY #$03
A100 B1 CE           LDA (LTR),Y
A102 85 60           STA DEP+1      déplacement
A104 88              DEY            pour
A105 B1 CE           LDA (LTR),Y    tableau suivant
A107 85 5F           STA DEP        diminué de 1
A109 D0 02           BNE D1         =longueur du tableau
A10B C6 60           DEC DEP+1      rangée
A10D C6 5F     D1    DEC DEP        en DEP
A10F A5 29           LDA INTFLG
A111 48              PHA            sauvegarde
A112 A5 28           LDA VALTYP     drapeaux
A114 48              PHA
A115 20 25 E7        JSR GTPRMS     recueil param.CLOAD ou CSAVE
A118 68              PLA
A119 85 28           STA VALTYP     récupération
A11B 68              PLA            drapeaux
A11C 85 29           STA INTFLG
A11E A5 63           LDA AUTO       AUTO?
A120 D0 04           BNE ERR        oui, erreur
A122 05 64           ORA BASIC      BASIC?
A124 F0 03           BEQ TAPOK      oui, OK
A126 4C E4 CF ERR    JMP SYNTER     erreur syntaxe
A129 20 CA E6 TAPOK  JSR VIAON      déconnection clavier
A12C 60              RTS

A12D 18        GTPTRS CLC
A12E A5 CE           LDA LTR        début tableau
A130 65 5F           ADC DEP        + longueur
A132 85 61           STA FIN        = fin
A134 A5 CF           LDA LTR+1      rangée en FIN
A136 65 60           ADC DEP+1
A138 85 62           STA FIN+1
A13A A0 04           LDY #$04
A13C B1 CE           LDA (LTR),Y    nombre dimensions en A
A13E 20 F6 D1        JSR GETARY     recherche adresse 1er élément
A141 85 5F           STA DEP        rangée en
A143 84 60           STY DEP+1      DEP
A145 85 10           STA TMP        et en
A147 84 11           STY TMP+1      TMP
A149 60              RTS

A14A 18        NXTDSC CLC
A14B A9 03           LDA #$03
A14D 65 10           ADC TMP        adresse
A14F 85 10           STA TMP        descripteur
A151 90 02           BCC N1         augmentée de trois
A153 E6 11           INC TMP+1      pour
A155 A8        N1    TAY            descripteur suivant
A156 A5 11           LDA TMP+1
A158 C4 61           CPY FIN        comparée
A15A E5 62           SBC FIN+1      à FIN
A15C 60              RTS
```

```
A15D 20 E2 00 GTARYP JSR CHRGET   incrémente TXTPTR, caractère en A
A160 85 B4           STA VARNM    1er caractère du nom
A162 20 86 D1        JSR ISLETC   doit être une lettre?
A165 B0 03           BCS OK1      oui, OK
A167 4C E4 CF        JMP SYNTER   non, erreur de syntaxe
A16A A2 00     OK1   LDX #$00     X=0, servira plus tard
A16C 86 28           STX VALTYP   drapeaux VALTYP et INTFLG
A16E 86 29           STX INTFLG   initialisés à zéro
A170 20 E2 00        JSR CHRGET   caractère suivant
A173 90 05           BCC CHR2     un chiffre, c'est bon
A175 20 86 D1        JSR ISLETC   est-ce une lettre?
A178 90 0B           BCC SUFFIX   non, doit être un suffixe
A17A AA        CHR2  TAX          2ème caractère du nom dans X
A17B 20 E2 00  AFTER JSR CHRGET   et après?
A17E 90 FB           BCC AFTER    chiffre, on s'en fout
A180 20 86 D1        JSR ISLETC   est-ce une lettre?
A183 B0 F6           BCS AFTER    oui, on s'en fout
A185 C9 24     SUFFIX CMP #"$"    est-ce '$'
A187 D0 06           BNE SUFF0    non, alors peut-être '%'
A189 A9 FF           LDA #$FF     oui, chaîne
A18B 85 28           STA VALTYP   VALTYP=FF
A18D D0 0C           BNE ORA2     branchement forcé
A18F C9 25     SUFF0 CMP #"%"     est-ce '%'
A191 D0 0F           BNE SCHR2    non, alors nombre réel
A193 A9 80           LDA #$80     oui, nombre entier
A195 85 29           STA INTFLG   INTFLG=80
A197 05 B4           ORA VARNM    bit 7 1er carac. nom mis à un
A199 85 B4           STA VARNM    ASCII négatif
A19B 8A        ORA2  TXA          et deuxième
A19C 09 80           ORA #$80     caractère aussi
A19E AA              TAX          toujours en X
A19F 20 E2 00        JSR CHRGET   avance TXTPTR
A1A2 86 B5     SCHR2 STX VARNM+1  rangement 2ème carac. du nom
A1A4 A6 9E     FNDARY LDX ARYTB   recherche adresse tableau
A1A6 A5 9F           LDA ARYTB+1  dont le nom est en VARNAM
A1A8 86 CE     FND0  STX LTR      adresse début zone des tableaux
A1AA 85 CF           STA LTR+1    dans LOWTR, pointeur de travail
A1AC C5 A1           CMP STEND+1  comp.LOWTR à STREND fin exploration
A1AE D0 04           BNE FND1     ce n'est pas fini.
A1B0 E4 A0           CPX STEND    oct.forts égaux, comp.oct.faibles.
A1B2 F0 20           BEQ DATER+1  fini, pas trouvé, OUT OF DATA
A1B4 A0 00     FND1  LDY #$00     recherche du nom
A1B6 B1 CE           LDA (LTR),Y  premier caractère
A1B8 C8              INY
A1B9 C5 B4           CMP VARNM
A1BB D0 06           BNE NOTFND   pas bon, tableau suivant
A1BD A5 B5           LDA VARNM+1  deuxième caractère
A1BF D1 CE           CMP (LTR),Y
A1C1 F0 16           BEQ FOUND    tableau trouvé, adr.en LOWTR
A1C3 C8        NOTFND INY         le déplacement
A1C4 B1 CE           LDA (LTR),Y  recueilli à l'adresse
A1C6 18              CLC          pointée par LOWTR+2
A1C7 65 CE           ADC LTR      est ajouté à LOWTR
A1C9 AA              TAX          le résultat en X(-),A(+)
A1CA C8              INY          est l'adresse
A1CB B1 CE           LDA (LTR),Y  du tableau suivant
A1CD 65 CF           ADC LTR+1    nouvel essai
A1CF 90 D7           BCC FND0     branchement forcé.
A1D1 A2 6B           LDX #$6B     inutile, vestige oublié, dommage!
A1D3 2C A2 2A  DATER BIT $2AA2    X=2A pour
A1D6 4C 85 C4  ERR1  JMP ERRMSG   erreur OUT OF DATA
A1D9 60        FOUND RTS          sortie.
```

QUATRIEME PARTIE
DEPLACEMENT D'UN PROGRAMME BASIC

La possibilité de faire tourner un programme Basic n'importe où, en mémoire vive, est séduisante.
Entre autres, elle peut faciliter la cohabitation du Basic et du langage machine. Le code étant logé à partir de £500, devant le programme Basic, ce dernier conserve l'entière disposition du haut de la mémoire. Tout risque d'interférence est éliminé.
On peut imaginer d'autres applications. Ce n'est pas notre propos.
La discussion a pour but principal de vous faire acquérir une meilleure connaissance de l'implantation d'un programme et du rôle de ses pointeurs.
Son intérêt annexe mais non négligeable est de proposer quelques solutions aux problèmes de communication entre les deux langages.
Potentiellement, le Basic est très souple dans le domaine de la relogeabilité. Quelle que soit sa position en mémoire, le texte Basic peut être déchiffré par l'interpréteur, à condition que ce dernier puisse le retrouver, c'est-à-dire que les pointeurs soient correctement ajustés.
Avant de poursuivre, quelques rappels seront utiles.

LE PROGRAMME BASIC ET SES POINTEURS

Les lignes du programme sont rangées en mémoire dans l'ordre des numéros.
La fin de ligne est marquée par un octet nul, zéro (00).
Les mots-clés sont codés sur un octet.
La contexture d'une ligne est la suivante:

L H	L H		
30 05	0A 00	... instructions ...	00
adresse ligne suivante (#530)	numéro de ligne (10)		fin de ligne

Les deux premiers octets sont *les octets de chaînage* (link bytes).
L'ensemble des lignes constitue ce que l'on appelle généralement le texte Basic. Il est obligatoirement précédé d'un zéro (00) et se termine par trois zéros consécutifs (00 00 00).
Les zones occupées par le programme et son espace de travail, délimitées par les différents pointeurs, sont schématisées ci-dessous:

chaînes de caractères	← MENSIZ A6,A7	(HIMEM) début chaînes HIMEM-1
	← FRETOP A2,A3	bas des chaînes (fin espace libre+1)
espace libre	← STREND A0,A1	début espace libre (fin zone variables+1)
tableaux	← ARYTAB 9E,9F	
variables simples	← VARTAB 9C,9D	début stockage variables (fin programme+1)
00 00 00	← 3 octets nuls	
programme	← TXTTAB 9A,9B	début progrm. norm. #501.
00	← octet nul	(#500)

- TXTTAB (#9A,9B) détient l'adresse du début de programme. Il est réglé à #501 à l'initialisation du système.
- VARTAB (#9C,9D) détient l'adresse du début de la zone de stockage des variables. C'est aussi l'adresse+1 de la fin du texte Basic.

A ce titre, il sert de marqueur, non pas pour l'exécution ou le listage (ce sont les trois zéros) mais en cas de modification du programme ou pour les opérations d'enregistrement ou de lecture sur cassette.
HIMEM peut être modifié par l'utilisateur.
Les pointeurs ARYTAB et STREND sont faits égaux à VARTAB et FRETOP est fait égal à HIMEM par les commandes RUN ou CLEAR. Au cours de l'exécution du programme, ils sont ajustés par l'interpréteur en fonction de son activité.
Pour que le programme soit à nouveau opérationnel après son installation à un emplacement inhabituel, il est nécessaire d'ajuster les pointeurs, d'inscrire la valeur 00 à TXTTAB-1 et de mettre à jour les octets de chaînage.

Nous allons aborder le sujet selon trois approches différentes susceptibles de convenir chacune à une application particulière.
La première technique consiste à charger le programme Basic à son nouvel emplacement directement à partir de la cassette.
La seconde permet de déplacer un programme dans les deux sens et dans toute l'étendue de la mémoire vive utilisable.
Enfin, la dernière est une variante de la précédente. Elle rend le programme autonome pour ses déplacements.

MODIFICATIONS DES ADRESSES DE CHARGEMENT

Les pointeurs #2A9,2AA et #2AB,2AC pour l'Atmos, #5F,60 et #61,62 pour l'Oric-1, reçoivent respectivement les adresses de début et de fin du programme à enregistrer ou à charger en mémoire.
Avec d'autres informations (vitesse, langage, etc..), ces adresses font partie de l'en-tete enregistré sur la bande avant le programme lui-même.
Dans le cas d'un programme Basic, les contenus de TXTTAB et de VARTAB sont simplement recopiés.
L'opération CLOAD commence par la lecture de l'en-tête.
Les pointeurs susnommés sont renseignés pour le chargement du programme dans la zone d'implantation adéquate. Si l'on désire la modifier, il suffit de changer le contenu des pointeurs à ce moment-là, avant la poursuite de l'opération de chargement.
C'est ce que réalise l'utilitaire de la page suivante qui procède aussi, après chargement du texte Basic, à l'ajustement des pointeurs et des octets de chaînage et à l'inscription de la valeur 00 à TXTTAB-1.
Il fait largement appel au langage machine car il s'agit d'intervenir au cours de l'exécution de la routine CLOAD. La partie Basic se contente de recueillir le nouveau début de programme et la vitesse de transfert et d'inscrire la valeur 00 au début du buffer d'entrée pour éviter la vérification du nom.

Commencez par taper le court programme ci-dessous puis sauvegardez-le sur cassette, en Slow par exemple. Il servira de témoin:

```
10 A=#9A:FOR I=0 TO 6:READ PTR$:P=A+I*2:PRINT PTR$,HEX$(P);
20 PRINT",";MID$(HEX$(P+1),2),HEX$(DEEK(P)),DEEK(P):NEXT:END
30 DATA TXTTAB,VARTAB,ARYTAB,STREND,FRETOP,FRESPC,MEMSIZ
```

Faites NEW ou débranchez puis rebranchez votre ordinateur avant d'entrer ce programme:

```
ATMOS
10 FOR I=0 TO #37:READ C$:C=VAL("#"+C$):POKE#400+I,C:NEXT
20 INPUT"nouveau TXTTAB";A:DOKE#70,A
30 INPUT"Slow ou Fast, S/F";S$:POKE#24D,0-1*(S$="S")
40 POKE#35,0:CALL#400

100 DATA 08,20,6A,E7,20,7D,E5,20,AC,E4,38,A5,70,AA,ED,A9
105 DATA 02,85,72,A5,71,A8,ED,AA,02,85,73,86,9A,8E,A9,02
110 DATA 84,98,8C,AA,02,8A,D0,02,C6,71,C6,70,A0,00,98,91
115 DATA 70,A6,72,A4,73,4C,A4,E8

ORIC-1
10 FOR I=0 TO #79:READ C$:C=VAL("#"+C$):POKE#400+I,C:NEXT
20 INPUT"nouveau TXTTAB";A:DOKE#70,A
30 INPUT"Slow ou Fast, S/F";S$:POKE#67,0-1*(S$="S")
40 POKE#35,0:CALL#400

100 DATA 08,20,CA,E6,20,63,E5,A9,03,A0,E5,20,76,E5,20,96
105 DATA E6,20,30,E6,C9,24,D0,F9,A2,09,20,30,E6,95,5D,CA
110 DATA D0,F8,20,30,E6,F0,05,95,49,E8,D0,F6,95,49,20,F0
115 DATA E6,8A,D0,D0,20,63,E5,A9,12,A0,E5,20,76,E5,20,6E
120 DATA E5,38,A5,70,AA,E5,5F,85,72,A5,71,A8,E5,60,85,73
125 DATA 86,5F,86,9A,84,60,84,98,18,A5,61,65,72,85,61,A5
130 DATA 62,65,73,85,62,8A,D0,02,C6,71,C6,70,A0,00,98,91
135 DATA 70,20,EB,E4,20,6F,C5,4C,BC,E7

RUN
```

Après avoir entré la nouvelle adresse de début et indiqué la vitesse de transfert adhoc, lancez votre magnétophone pour la lecture du programme témoin que vous venez d'enregistrer.
Le chargement terminé, faites LIST puis RUN.
Remarque:
Tel qu'il est écrit, cet utilitaire s'applique exclusivement au chargement d'un programme Basic enregistré normalement (TXTTAB=#501).
Le nouveau TXTTAB doit être supérieur à #500 et il faut ménager un espace suffisant en dessous de HIMEM.
Par sécurité, vous pouvez ajouter une ligne 25 comme celle-ci:

```
25 IF A<#501 OR A>#8000 THEN 20
```

Après chargement à sa nouvelle adresse, le programme sera sauvegardé puis rechargé à nouveau de façon normale. Le lancement devra être précédé des instructions:
DOKE#9A,nouveau TXTTAB puis POKEDEEK(#9A)-1,0

Le listing de désassemblage du code machine, sommairement commenté, est reproduit à la page suivante.
La différence entre les deux systèmes est importante.
Pour une meilleure compréhension, vous devrez sans doute désassembler les sous-routines concernées en ROM.

ATMOS

```
0400-  PHP
0401-  JSR $E76A VIA ON
0404-  JSR $E57D Searching
0407-  JSR $E4AC lit en-tete
040A-  SEC       calcul
040B-  LDA $70    déplacement
040D-  TAX       différence
040E-  SBC $02A9 entre
0411-  STA $72    nouveau
0413-  LDA $71    et ancien
0415-  TAY       TXTTAB
0416-  SBC $02AA inscrite
0419-  STA $73    en 72,73
041B-  STX $9A    TXTTAB et
041D-  STX $02A9  début
0420-  STY $9B    enregistr.
0422-  STY $02AA  ajustés.
0425-  TXA       valeur 00
0426-  BNE $042A  inscrite
0428-  DEC $71    à TXTTAB-1
042A-  DEC $70
042C-  LDY #$00   suite de
042E-  TYA        l'opération
042F-  STA ($70),Y est assurée
0431-  LDX $72    par
0433-  LDY $73    routine
0435-  JMP $E8A4  en E8A4
```

ORIC-1

```
0400-  PHP
0401-  JSR $E6CA  VIA ON
0404-  JSR $E563  reproduction
0407-  LDA #$03   intégrale
0409-  LDY #$E5   du début
040B-  JSR $E576  de la
040E-  JSR $E696  sous-routine
0411-  JSR $E630  responsable
0414-  CMP #$24   du
0416-  BNE $0411  chargement
0418-  LDX #$09   section
041A-  JSR $E630  E4A8-E4E4
041D-  STA $5D,X  lecture
041F-  DEX        en-tête
0420-  BNE $041A
0422-  JSR $E630
0425-  BEQ $042C  interruption
0427-  STA $49,X  pour
0429-  INX        modification
042A-  BNE $0422  des
042C-  STA $49,X  pointeurs
042E-  JSR $E6F0
0431-  TXA        reprise
0432-  BNE $0404  chargement
0434-  JSR $E563  du
0437-  LDA #$12   programme
0439-  LDY #$E5   en 471
043B-  JSR $E576  JSR $E4EB
043E-  JSR $E56E
0441-  SEC        calcul
0442-  LDA $70     déplacement
0444-  TAX        différence
0445-  SBC $5F     entre
0447-  STA $72     nouveau
0449-  LDA $71     et ancien
044B-  TAY        TXTTAB
044C-  SBC $60     inscrite
044E-  STA $73     en 72,73
0450-  STX $5F     TXTTAB et
0452-  STX $9A     début
0454-  STY $60     enregistr.
0456-  STY $9B     ajustés
0458-  CLC
0459-  LDA $61     ajustement
045B-  ADC $72     de
045D-  STA $61     VARTAB
045F-  LDA $62
0461-  ADC $73
0463-  STA $62
0465-  TXA        valeur 00
0466-  BNE $046A  inscrite
0468-  DEC $71     à
046A-  DEC $70     TXTTAB-1
046C-  LDY # 30
046E-  TYA
046F-  STA ($70),Y
0471-  JSR $E4EB  chargement
0474-  JSR $C56F  chaînage
0477-  JMP $E7BC  sortie
```

Remarque: La divergence entre les listings Atmos et Oric-1 est due principalement au fait que, sur Atmos le chargement de l'en-tête et celui du programme proprement dit sont effectués par deux sous-routines séparées, alors que sur Oric-1 c'est la même sous-routine qui en a la charge, ce qui oblige à la reproduire en partie.

DEPLACEMENT D'UN PROGRAMME EN MEMOIRE

C'est la seconde hypothèse envisagée.
Le cahier des charges est le suivant:
- Un programme Basic est présent en mémoire à l'adresse habituelle.
- On désire le déplacer pour une raison quelconque, tout en conservant la possibilité de le remettre en place aussitôt.
- Les zones d'implantation doivent pouvoir se chevaucher.

Le dernier impératif complique le premier problème à résoudre, celui du transfert du texte Basic, et impose des restrictions:
- moyen de transfert indépendant du programme à déplacer.
- obligation de recourir au langage machine.
- nécessité de disposer de deux sous-routines de transfert; l'une qui commence par la fin du bloc pour les déplacements vers le haut de la mémoire, l'autre par le début pour les déplacements vers le bas.
Deux routines du système, les seules spécialisées dans le transfert d'octets, présentent justement ces particularités.

- BLTU, en C3F4 Atmos, C3F8 Oric-1, est appelée par l'interpréteur pour déplacer la "queue" du programme vers le haut de la mémoire en vue de l'insertion d'une ligne Basic.
Elle débute par un test de la mémoire vive disponible; intéressant pour ce qui nous occupe.
Nous avons déjà rencontré cette routine. Elle sert, amputée dudit test, sous le nom de MOVUP, à l'éditeur de langage machine et au moniteur.
(voir pages 25 et 67, paramètres à passer et pointeurs correspondants).

- MOVE, en EDC4 Atmos, EC0C Oric-1, est employée par le système pour déplacer le jeu de caractères en fonction du mode d'affichage.
Cette routine commence le transfert par le début du bloc.

pointeurs	Atmos	0C	0D	0E	0F	10	11
	Oric-1	200	201	202	203	204	205
paramètres		adr.départ		destination		nb d'octets	
		L	H	L	H	L	H

Le programme de la page suivante met en oeuvre l'une ou l'autre de ces deux routines suivant le sens de déplacement.
Pour simplifier les choses, le bloc transféré comprend le 00 précédant le texte Basic; il s'étend donc des adresses (TXTTAB)-1 à (VARTAB)-1.
Après le déplacement, pointeurs et octets de chaînage sont ajustés.

Le code machine est relogeable. Vous pouvez l'installer n'importe où mais il faut penser aux interférences possibles pendant les déplacements.
Vous l'implanterez à l'aide de l'ELM ou du moniteur, ou bien vous le mettrez en DATA et utiliserez un chargeur Basic classique.
S'il est inscrit en page 4, par exemple, après chargement du programme Basic à manipuler, la procédure sera la suivante:

CALL#400,ADR où ADR=nouveau début de programme-1 (TXTTAB)-1

Le programme Basic déplacé est immédiatement opérationnel dans ses nouveaux pointeurs. Il est susceptible d'être à nouveau déplacé dans un sens ou dans l'autre.
Il pourra ainsi voyager dans tout l'espace utilisateur, entre #500 et HIMEM (jusqu'à #B3FF avec GRAB).

```
ATMOS     ORIC-1

20 65 D0  20 D9 CF        JSR CHKCOM   test, virgule à TXTPTR?
20 53 E8  20 9D E7        JSR MAKINT2  C=0, ADR ds LINNUM (33,34)
84 0E     8C 02 02        STY DEST     et Y,A  inscrite dans DEST
85 0F     8D 03 02        STA DEST+1   0E,0F ou 202,203 pour MOVE
AA        AA              TAX          octet fort ADR dans X.
A5 9A     A5 9A           LDA TXTTAB   adr.déb. zone à transférer
69 FF     69 FF           ADC #$FF     (celle du 00 précédant
85 CE     85 CE           STA LOWTR    le début du programme)
85 0C     8D 00 02        STA DEP      placée ds LOWTR pour BLTU
A5 9B     A5 9B           LDA TXTTAB+1 et dans DEP pour MOVE.
69 FF     69 FF           ADC #$FF     LOWTR(CE,CF) et DEP(0C,0D
85 CF     85 CF           STA LOWTR+1  ou 200,201) = (TXTTAB)-1
85 0D     8D 01 02        STA DEP+1    après l'addition C=1,
A5 9C     A5 9C           LDA VARTAB   contenu VARTAB placé dans
85 C9     85 C9           STA HIGHTR   HIGHTR (C9,CA) pour BLTU.
E5 CE     E5 CE           SBC LOWTR    calcul
85 10     8D 04 02        STA NMB        nombre d'octets
A5 9D     A5 9D           LDA VARTAB+1   à transférer
85 CA     85 CA           STA HIGHTR+1   inscrit dans NMB
E5 CF     E5 CF           SBC LOWTR+1    (10,11 ou 204,205)
85 11     8D 05 02        STA NMB+1    pour MOVE.
18        18              CLC          C=0 pour addition qui suit
98        98              TYA          LINNUM (ADRL)  -> A
65 10     6D 04 02        ADC NMB      fin de programme + 1
85 C7     85 C7           STA HIGHDS     = ADR + NMB dans
85 00     85 00           STA $00        HIGHDS (C7,C8) pour BLTU
8A        8A              TXA            et dans 00,01, pointeur
65 11     6D 05 02        ADC NMB+1      temporaire.
85 C8     85 C8           STA HIGHDS+1   Sera nouveau VARTAB si
85 01     85 01           STA $01        transfert possible.
C4 CE     C4 CE           CPY LOWTR    détermine sens du
9A        8A              TXA            déplacement (haut ou bas)
E5 CF     E5 CF           SBC LOWTR+1    pour choix routine
B0 09     B0 09           BCS BLTU       BLTU ou MOVE
E0 05     E0 05     MOVE  CPX #$05     ADR < #500 ?
90 27     90 27           BCC ERR      oui, sortie OUT OF MEMORY
20 C4 ED  20 0C EC        JSR MOVE     non, transfert vers le bas
F0 07     D0 07           BEQ FIN      forcé, BNE pour ORIC-1
A5 C7     A5 C7     BLTU  LDA HIGHDS   A = (HIGHDS)
A4 C8     A4 C8           LDY HIGHDS+1 Y = (HIGHDS+1), pour BLTU
20 F4 C3  20 FB C3        JSR BLTU     déplacement vers le haut
18        18        FIN   CLC          régle pointeurs début et
A5 33     A5 33           LDA LINNUM   fin de programme.
69 01     69 01           ADC #$01     ....
85 9A     85 9A           STA TXTTAB   TXTTAB = ADR + 1.
A5 34     A5 34           LDA LINNUM+1 ....
69 00     69 00           ADC #$00     ....
85 9B     85 9B           STA TXTTAB+1 ....
A5 00     A5 00           LDA $00      VARTAB = (00,01)
85 9C     85 9C           STA VARTAB   ....
A5 01     A5 01           LDA $01      ....
85 9D     85 9D           STA VARTAB+1 ....
20 5F C5  20 6F C5        JSR LINKSET  ajuste octets de chaînage
4C 0F C7  4C 3A C7        JMP CLEARC   ajuste autres Ptrs, sortie.
4C 7C C4  4C 83 C4  ERR   JMP MEMERR   message OUT OF MEMORY
```

PROTECTION DES PROGRAMMES BASIC

Dans le numéro 4 de la revue THEORIC, je proposais une application de ce que nous venons de voir à la protection d'un programme Basic.
Après déplacement, un module en langage machine, chargé de l'action protectrice et du lancement, est placé devant le programme à protéger.
L'ensemble est sauvegardé en bloc, en mode AUTO.
Après rechargement en mémoire, un mot de passe est exigé avant lancement ou avant l'entrée de toute commande.
L'opération est réalisée par le programme reproduit ci-dessous.
Le code responsable du déplacement (lignes 100-160) est légèrement différent du précédent. La routine MOVE est reconstituée pour la compatibilité entre les deux systèmes. Le reste du code est commenté à la page suivante.

```
 100 REM Deplacement programme Basic
 110 DATA 20,65,D0,20,53,E8,AA,A5,9A,69,FF,85,CE,A5,9B,69,FF,85,
CF,A5,9C
 120 DATA 85,C9,E5,CE,85,10,A5,9D,85,CA,E5,CF,85,11,18,9B,65,10,
85,C7,85,00
 130 DATA 8A,65,11,85,C8,85,01,C4,CE,8A,E5,CF,B0,25,8A,C9,05,90,
42,48,A2,00
 140 DATA A0,00,C4,10,D0,04,E4,11,F0,0E,B1,CE,91,33,C8,D0,F1,E6,
CF,E6,34,E8
 150 DATA D0,EA,68,85,34,D0,07,A5,C7,A4,C8,20,F4,C3,18,A5,33,69,
01,85,9A,A5
 160 DATA 34,69,00,85,9B,A5,00,85,9C,A5,01,85,9D,20,5F,C5,4C,0F,
C7,4C,7C,C4

 200 REM Mise en place protection
 210 DATA A6,9A,8E,B7,04,E0,3A,A4,9B,8C,BF,04,98,E9,05,90,EC
 220 DATA 8A,E9,13,8D,D1,04,98,E9,00,8D,D2,04,8A,E9,37,85,00
 230 DATA 98,E9,00,85,01,A0,35,B9,B5,04,91,00,88,10,F8,60

 300 REM Protection, reglage pointeurs
 310 DATA 38,A9,FF,85,9A,E9,16,85,1B,A9,FF,85,9B,E9,00,85,1C
 320 DATA AD,AB,02,85,9C,AD,AC,02,85,9D,20,FF,FF,4C,08,C7

 330 REM Mot de passe  M O T
 340 DATA 20,B0,CC,A0,12,20,E8,C5,D1,1B,D0,F7,C8,C0,15,D0,F4,60,
4D,4F,54

 500 FOR I=0 TO #EA:READ A$:C=VAL("#"+A$):POKE #400+I,C:NEXT
 510 CLS:PRINT:PRINT"chargez programme a proteger, puis":PRINT
 520 PRINT"- CALL#400,ADR (transfert)"
 530 PRINT"ADR=(nouv.TXTTAB)-1, doit etre > #538":PRINT
 540 PRINT"- CALL#483 (mise en place protection)"
 550 PRINT:PRINT"sauvegarde type langage machine"
 560 PRINT"CSAVE";CHR$(34);CHR$(34);",ADEEK(#9A)-55,AUTO"
 570 PRINT:PRINT"Si lancement AUTO echoue (ATMOS),"
 580 PRINT"CALL#E8E5"
 590 IF PEEK(#C3F4)=32 THEN NEW

1000 REM modif ORIC-1
1010 DOKE#401,#CFD9:DOKE#404,#E79D:DOKE#463,#C3F8
1020 DOKE#47B,#C56F:DOKE#47E,#C73A:DOKE#481,#C483
1030 DOKE#4D4,#C733:DOKE#4D7,#CBED:DOKE#4DC,#C5F8
1040 DOKE#4C9,#EAEA:DOKE#4CE,#EAEA:NEW
```

MISE EN PLACE PROTECTION

```
0483-   A6 9A       LDX $9A
0485-   8E B7 04    STX $04B7
0488-   E0 3A       CPX #$3A
048A-   A4 9B       LDY $9B
048C-   8C BF 04    STY $04BF
048F-   98          TYA
0490-   E9 05       SBC #$05
0492-   90 EC       BCC $0480
0494-   8A          TXA
0495-   E9 13       SBC #$13
0497-   8D D1 04    STA $04D1
049A-   98          TYA
049B-   E9 00       SBC #$00
049D-   8D D2 04    STA $04D2
04A0-   8A          TXA
04A1-   E9 37       SBC #$37
04A3-   85 00       STA $00
04A5-   98          TYA
04A6-   E9 00       SBC #$00
04A8-   85 01       STA $01
04AA-   A0 35       LDY #$35
04AC-   B9 B5 04    LDA $04B5,Y
04AF-   91 00       STA ($00),Y
04B1-   88          DEY
04B2-   10 F8       BPL $04AC
04B4-   60          RTS
```

Cette sous-routine est appelée après le déplacement du programme (CALL#483, ligne 540). Elle inscrit en #4B7 et #4BF les octets faible et fort du nouveau TXTTAB. Elle calcule ensuite l'adresse de la boucle de recueil du mot de passe et l'inscrit en #4D1,4D2. Le module protecteur (54 octets), ainsi renseigné, est placé devant le 00 qui précède le texte Basic.

PROTECTION, REGLAGE POINTEURS

```
04B5-   38          SEC
04B6-   A9 FF       LDA #$FF
04B8-   85 9A       STA $9A
04BA-   E9 16       SBC #$16
04BC-   85 1B       STA $1B
04BE-   A9 FF       LDA #$FF
04C0-   85 9B       STA $9B
04C2-   E9 00       SBC #$00
04C4-   85 1C       STA $1C
04C6-   AD AB 02    LDA $02AB
04C9-   85 9C       STA $9C
04CB-   AD AC 02    LDA $02AC
04CE-   85 9D       STA $9D
04D0-   20 FF FF    JSR $FFFF
04D3-   4C 08 C7    JMP $C708
```

Dérivation du JMP STROUT en #1A,1B,1C sur la sous-routine de recueil du mot de passe. Ajustement des pointeurs TXTTAB et VARTAB (inutile pour ce dernier sur Oric-1) et lancement du programme si mot de passe correct.

SETPTRS, Oric-1 C733 (RUN)

MOT DE PASSE

```
04D6-   20 B0 CC    JSR $CCB0
04D9-   A0 12       LDY #$12
04DB-   20 E8 C5    JSR $C5E8
04DE-   D1 1B       CMP ($1B),Y
04E0-   D0 F7       BNE $04D9
04E2-   C8          INY
04E3-   C0 15       CPY #$15
04E5-   D0 F4       BNE $04DB
04E7-   60          RTS

04E8-   4D 4F 54    ASCII
```

STROUT (Oric-1, CBED) Affichage Ready. Boucle de comparaison des caractères entrés (recueil par INCHR, Oric-1 C5F8) avec ceux du mot de passe en vigueur (ici M O T, ASCII 4D 4F 54). Entrée initiale au début de la boucle. Affichage préalable de Ready par JSR STROUT après un arrêt.

PROGRAMME AUTOMOTEUR

Avec la technique précédente, le programme Basic est dans la situation d'une caravane sur un terrain de camping. Le véhicule tracteur doit se trouver à proximité si l'on désire pouvoir lever le camp rapidement.

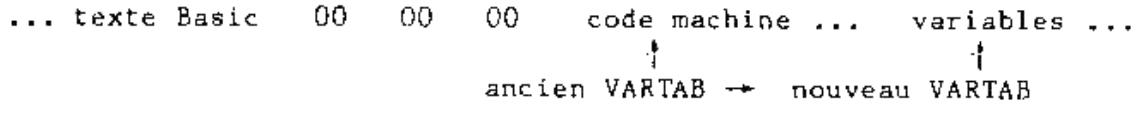
L'autonomie de mouvement du programme est souhaitable. Son déplacement étant, par exemple, l'une des options d'un menu initial.
Pour cela, l'élément moteur doit être solidaire du texte Basic.
Deux difficultés apparaissent qui seront aisément surmontées.
- Il faut éviter d'écraser le moteur pendant le transfert; on le mettra à l'abri.
- L'éxécution ne doit pas être interrompue; il suffit de garder trace de la position du pointeur de texte (TXTPTR) par rapport au début de programme (TXTTAB).

Comment assurer la motorisation?
Une solution consiste à conserver le code machine en DATA.
Le moment venu, il est inscrit par une boucle FOR-NEXT, dans un endroit sûr, avant exécution.
Ce n'est pas satisfaisant; pas très efficace, les boucles Basic sont lentes, ni esthétique.
Comme si l'on décidait de laisser la caravane attelée à son remorqueur.

Il existe un moyen plus élégant.
Parmi les techniques d'interfaçage entre Basic et langage machine, l'une convient ici à merveille. Il s'agit de la méthode dite transparente.
Elle nous permettra de rendre l'élément moteur à la fois invisible et bien plus nerveux. Pour en finir avec une comparaison osée, disons que notre caravane sera remplacée par un camping-car.

LA METHODE TRANSPARENTE

Essayons de rendre plus claire cette obscure méthode bizarrement nommée.
Elle est fort intéressante.
VARTAB en est la vedette.
C'est à partir de l'adresse détenue par ce pointeur que l'interpréteur inscrit ou recherche ses variables. Toute l'information située en deçà est tabou, considérée comme faisant partie du texte Basic.
La plus légère modification du programme entraîne un décalage de cette information et l'ajustement correspondant de VARTAB (et aussi la perte des variables par un CLEAR systématique, ARYTAB et STREND = VARTAB).
La routine CSAVE utilise TXTTAB et VARTAB pour délimiter le bloc de code à enregistrer.
Par chance, l'exécution ou le listage ignorent VARTAB.
Pour les routines concernées, la fin du programme est marquée uniquement par les trois zéros.
On peut donc décaler artificiellement VARTAB vers le haut et installer un sous-programme en langage machine dans l'espace créé entre les trois zéros du texte Basic et la table des variables.
Précisons que cette opération suppose la mise à jour (réajustement sur VARTAB) de ARYTAB et STREND par un RUN ou CLEAR ultérieur.



Les avantages sont évidents.
- Le code machine est parfaitement à l'abri de toute altération due au fonctionnement du programme.
- Il est considéré comme faisant partie intégrante du texte Basic et sera avec lui sauvegardé sur cassette.
- Il est invisible au listage, d'où le nom de la méthode.

Voyons les inconvénients.
- A moins d'etre absolument certain que le programme ne sera pas modifié, cette méthode exige que le code machine soit relogeable c'est-à-dire sans adressage absolu interne. C'est un inconvénient sérieux en temps normal.

Il n'est pas genant dans notre cas particulier puisque le code, étant appelé à etre déplacé, devait etre relogeable de toutes façons.
- Sans etre vraiment compliquée, l'implantation du code à la suite du texte Basic demande certaines précautions.

Nous allons étudier les diverses procédures possibles à l'occasion de l'adaptation de son élément moteur à un programme Basic.

Le code peut etre placé en DATA et inscrit à l'aide d'une boucle.
S'il s'agit d'un programme nouveau, la procédure est simple.
Le code est implanté à partir de l'adresse #503,
VARTAB est ajusté (nouveau VARTAB = #503 + nombre d'octets)
puis la valeur 00 est inscrite en #501 et #502.

ATMOS

```
100 DATA 38,A5,E9,E5,9A,85,00,A5,EA,E5,9B,85,01,A5,9C,E9
105 DATA 34,85,02,A5,9D,E9,00,85,03,A0,33,B1,02,99,50,00
110 DATA 88,10,F8,20,80,CD,E8,86,E9,84,EA,20,53,E8,84,0E
115 DATA 85,0F,AA,A5,9A,69,FF,85,CE,85,0C,A5,9B,69,FF,85
120 DATA CF,85,0D,A5,9C,85,C9,E5,CE,85,10,A5,9D,85,CA,E5
125 DATA CF,85,11,18,98,65,10,85,C7,85,02,8A,65,11,85,C8
130 DATA 85,03,C4,CE,8A,E5,CF,90,03,4C,55,00,E0,05,90,03
135 DATA 4C,50,00,4C,7C,C4,20,C4,ED,F0,07,A5,C7,A4,C8,20
140 DATA F4,C3,E6,33,D0,02,E6,34,18,A5,33,85,9A,65,00,85
145 DATA E9,A5,34,85,9B,65,01,85,EA,A5,02,85,9C,A5,03,85
150 DATA 9D,20,5F,C5,20,0F,C7,4C,C1,C8

200 A=#503:FOR I=0 TO #A9:READ C$:C=VAL("#"+C$)
210 POKE A+I,C:NEXT:DOKE#9C,A+170:DOKE#501,0
```

ORIC-1

```
100 DATA 38,A5,E9,E5,9A,85,00,A5,EA,E5,9B,85,01,A5,9C,E9
105 DATA 34,85,02,A5,9D,E9,00,85,03,A0,33,B1,02,99,50,00
110 DATA 88,10,F8,20,F4,CC,E8,86,E9,84,EA,20,9D,E7,8C,02
115 DATA 02,8D,03,02,AA,A5,9A,69,FF,85,CE,8D,00,02,A5,9B
120 DATA 69,FF,85,CF,8D,01,02,A5,9C,85,C9,E5,CE,8D,04,02
125 DATA A5,9D,85,CA,E5,CF,8D,05,02,18,98,6D,04,02,85,C7
130 DATA 85,02,8A,6D,05,02,85,C8,85,03,C4,CE,8A,E5,CF,90
135 DATA 03,4C,55,00,E0,05,90,03,4C,50,00,4C,83,C4,20,0C
140 DATA EC,D0,07,A5,C7,A4,C8,20,F8,C3,E6,33,D0,02,E6,34
145 DATA 18,A5,33,85,9A,65,00,85,E9,A5,34,85,9B,65,01,85
150 DATA EA,A5,02,85,9C,A5,03,85,9D,20,6F,C5,20,3A,C7,4C
155 DATA AD,C8

200 A=#503:FOR I=0 TO #B1:READ C$:C=VAL("#"+C$)
210 POKE A+I,C:NEXT:DOKE#9C,A+178:DOKE#501,0
```

Après exécution du programme précédent, vérifiez la disparition du Basic par LIST puis entrez AU CLAVIER le programme témoin modifié:

```
5 INPUT"deplacement, O/N";S$:IF S$="O" THEN 100

10 A=#9A:FOR I=0 TO 6:READ PTR$:P=A+I*2:PRINT PTR$,HEX$(P);
20 PRINT",";MID$(HEX$(P+1),2),HEX$(DEEK(P)),DEEK(P):NEXT:END
30 DATA TXTTAB,VARTAB,ARYTAB,STREND,FRETOP,FRESPC,MEMSIZ
ATMOS
100 PRINT"nouvelle adresse (TXTTAB-1)":CALLDEEK(#9C)-170:GOTO 10
ORIC-1
100 PRINT"nouvelle adresse (TXTTAB-1)":CALLDEEK(#9C)-178:GOTO 10

RUN
```

La procédure est plus compliquée quand il s'agit d'accoler le code machine à un programme existant.
Débranchez puis rebranchez votre ordinateur.
Commencez par entrer le programme témoin ci-dessus. Vous lui ajouterez les lignes 63000 et suivantes.
La ligne 63000 décale VARTAB du nombre d'octets à insérer.
A la ligne 63005, CLEAR fait ARYTAB et STREND égaux à VARTAB.
Il faut préciser que toute nouvelle variable simple est inscrite à l'adresse pointée par ARYTAB, la zone des tableaux ayant été préalablement décalée de sept octets vers le haut. Après l'inscription, ARYTAB et STREND sont augmentés de sept.
L'instruction DOKE#B0,DEEK(#AC) amène le pointeur de DATA (DATPTR #B0,B1) dans cette section de programme (OLDTXT #AC,AD contient l'adresse des deux-points marquant la fin de l'instruction qui vient d'être exécutée)

```
ATMOS
63000 DOKE#9C,DEEK(#9C)+170
63005 CLEAR:A=DEEK(#9C)-170:DOKE#B0,DEEK(#AC)
63010 FOR I=0 TO #A9:READ C$:C=VAL("#"+C$):POKE A+I,C:NEXT:END

63100 DATA Atmos, voir page precedente
.....
ORIC-1
63000 DOKE#9C,DEEK(#9C)+178
63005 CLEAR:A=DEEK(#9C)-178:DOKE#B0,DEEK(#AC)
63010 FOR I=0 TO #B1:READ C$:C=VAL("#"+C$):POKE A+I,C:NEXT:END

63100 DATA Oric-1, voir page precedente
.....
```

Faites RUN 63000 puis supprimez les lignes 63000 et la suite.

Lorsqu'on dispose d'un moyen extérieur, éditeur de langage machine ou moniteur par exemple, l'opération est facilitée.
Voici la procédure, le programme étant en mémoire:
- Rechercher la valeur de VARTAB, ?DEEK(#9C) noter l'adresse
- Décaler VARTAB par DOKE#9C,DEEK(#9C)+170 (+178 pour Oric-1)
- Appeler l'éditeur ou le moniteur et inscrire le code à partir de l'adresse notée précédemment.

Les lignes de code avec somme de contrôle, obtenues par l'ELM, sont reproduites à la page suivante. L'adresse d'implantation sera identique si le programme témoin a été tapé exactement comme il figure ci-dessus.

ATMOS

```
05EE:  38 A5 E9 E5 9A 85 00 A5 EA E5 9B 85 01 A5 9C E9    0989
05FE:  34 85 02 A5 9D E9 00 85 03 A0 33 B1 02 99 50 00    0F66
060E:  88 10 F8 20 80 CD E8 86 E9 84 EA 20 53 E8 84 0E    1815
061E:  85 0F AA A5 9A 69 FF 85 CE 85 0C A5 9B 69 FF 85    210B
062E:  CF 85 0D A5 9C 85 C9 E5 CE 85 10 A5 9D 85 CA E5    2AB9
063E:  CF 85 11 18 98 65 10 85 C7 85 02 8A 65 11 85 C8    3163
064E:  85 03 C4 CE 8A E5 CF 90 03 4C 55 00 E0 05 90 03    3867
065E:  4C 50 00 4C 7C C4 20 C4 ED F0 07 A5 C7 A4 C8 20    404F
066E:  F4 C3 E6 33 D0 02 E6 34 18 A5 33 85 9A 65 00 85    4804
067E:  E9 A5 34 85 9B 65 01 85 EA A5 02 85 9C A5 03 85    4FB0
068E:  9D 20 5F C5 20 0F C7 4C C1 C8                      545C
```

ORIC-1

```
05EE:  38 A5 E9 E5 9A 85 00 A5 EA E5 9B 85 01 A5 9C E9    0989
05FE:  34 85 02 A5 9D E9 00 85 03 A0 33 B1 02 99 50 00    0F66
060E:  88 10 F8 20 F4 CC E8 86 E9 84 EA 20 9D E7 8C 02    18CD
061E:  02 8D 03 02 AA A5 9A 69 FF 85 CE 8D 00 02 A5 9B    1FD4
062E:  69 FF 85 CF 8D 01 02 A5 9C 85 C9 E5 CE 8D 04 02    27F5
063E:  A5 9D 85 CA E5 CF 8D 05 02 18 98 6D 04 02 85 C7    2F3D
064E:  85 02 8A 6D 05 02 85 C8 85 03 C4 CE 8A E5 CF 90    36F7
065E:  03 4C 55 00 E0 05 90 03 4C 50 00 4C 83 C4 20 0C    3B6E
066E:  EC D0 07 A5 C7 A4 C8 20 FB C3 E6 33 D0 02 E6 34    44E9
067E:  18 A5 33 85 9A 65 00 85 E9 A5 34 85 9B 65 01 85    48AF
068E:  EA A5 02 85 9C A5 03 85 9D 20 6F C5 20 3A C7 4C    52EC
069E:  AD C8                                              5461
```

Le code source est reproduit page ci-contre.
Il ressemble à celui de la page 138 avec quelques différences.

- Section 1 - La position du pointeur TXTPTR dans le texte (déplacement par rapport à TXTTAB) est sauvegardée dans TMP1.

- Section 2 - Les routines de déplacement (partie active, section 5) sont mises à l'abri dans le buffer d'entrée (à partir de l'adresse #50). Le début du bloc concerné (51 octets) est (VARTAB) - 52. Le recours aux routines du système minimise le code à protéger.

- Section 3 - La nouvelle adresse est recueillie par appel de INPTRQ (Input Request), variante de INLIN. Son exploitation reste la même.
Pour ne pas atteindre les routines de transfert en #50, évitez les espaces inutiles en entrant la nouvelle adresse.

- Section 4 - Le branchement sur la routine adéquate (MOVUP ou MOVDN) est assuré par un saut à l'adresse idoine dans le buffer (#50 ou 55).

- Section 5 - La mise à jour des pointeurs est complétée par celle du pointeur de texte. L'exécution du programme est poursuivie après le déplacement grâce à la sortie par NEWSTT, exécution d'une nouvelle instruction.

REMARQUE IMPORTANTE

Le programme se retouve en l'état qui suivrait une instruction CLEAR.
Les variables définies précédemment sont perdues, le pointeur de DATA est replacé en début de programme (RESTORE) et la pile est réinitialisée. Pour cette dernière raison, la routine machine ne peut pas être appelée à partir d'un sous-programme Basic (GOSUB-RETURN).

```
TMP1    =$00
TMP2    =$02
DEP     =$0C    ;$200
DEST    =$0E    ;$202
NMB     =$10    ;$204
LINNUM  =$33
MOVDN   =$50
MOVUP   =$55
TXTTAB  =$9A
VARTAB  =$9C
HIGHDS  =$C7
HIGHTR  =$C9
LOWTR   =$CE
TXTPTR  =$E9
BLTU    =$C3F4 ;$C3F8
MEMERR  =$C47C ;$C483
LNKSET  =$C55F ;$C56F
CLEARC  =$C70F ;$C73A
NEWSTT  =$C8C1 ;$C8AD
INPTRQ  =$CD80 ;$CCF4
MKINT2  =$E853 ;$E79D
MOVE    =$EDC4 ;$EC0C

        ;sauv.pos.TXTPTR

        SEC
        LDA TXTPTR
        SBC TXTTAB
1       STA TMP1
        LDA TXTPTR+1
        SBC TXTTAB+1
        STA TMP1+1

        ;rout.depl.ds BUF

        LDA VARTAB
        SBC #$34
        STA TMP2
        LDA VARTAB+1
        SBC #$00
2       STA TMP2+1
        LDY #$33
        LDA (TMP2),Y
        STA MOVDN,Y
        DEY
        BPL -8

        ;nouv.adr.TXTTAB-1
        ;rens.pointeurs

        JSR INPTRQ
        INX
        STX TXTPTR
        STY TXTPTR+1
        JSR MKINT2
        STY DEST
3       STA DEST+1
        TAX
        LDA TXTTAB
        ADC #$FF
        STA LOWTR
        STA DEP
        LDA TXTTAB+1
        ADC #$FF
        STA LOWTR+1
        STA DEP+1
        LDA VARTAB
        STA HIGHTR
        SBC LOWTR
        STA NMB
        LDA VARTAB+1
        STA HIGHTR+1
        SBC LOWTR+1
        STA NMB+1
        CLC
        TYA
        ADC NMB
        STA HIGHDS
        STA TMP2
        TXA
        ADC NMB+1
        STA HIGHDS+1
        STA TMP2+1

        ;sens deplacement

        CPY LOWTR
        TXA
        SBC LOWTR+1
        BCC +3
4       JMP MOVUP
        CPX #$05
        BCC +3
        JMP MOVDN
        JMP MEMERR

        ;rout.deplacement

        JSR MOVE
        BEQ +7  ;BNE Oric-1
        LDA HIGHDS
        LDY HIGHDS+1
        JSR BLTU
        INC LINNUM
        BNE +2
        INC LINNUM+1
        CLC
        LDA LINNUM
5       STA TXTTAB
        ADC TMP1
        STA TXTPTR
        LDA LINNUM+1
        STA TXTTAB+1
        ADC TMP1+1
        STA TXTPTR+1
        LDA TMP2
        STA VARTAB
        LDA TMP2+1
        STA VARTAB+1
        JSR LNKSET
        JSR CLEARC
        JMP NEWSTT
```

INDEX

ADRESSES DU SYSTEME

MEMOIRE VIVE

0C-11	PMOVE (Atmos)	,137,138,145
12,13	CURBAS	,10,28,36
1A	JMP STROUT	,140
28	VALTYP	,108,109,116,117,127,128,129,130,131
29	INTFLG	,108,109,116,117,127,131
2B	SUBFLG	,108,116,125
2F	FLG	,10,38,59,83
33,34	LINNUM	,138,145
35	BUF	,10,34,59,74
5F,60	déb.enr.Oric-1	,127,130,133,136
61,62	fin " "	,127,130,133,136
63	AUTO ou non "	,127,130
64	Basic/LM "	,127,130
67	Slow/Fast "	,135
97,98	RES+2 (LENFLG)	,108,109,112,115,116,117,118,121,122,123
9A,9B	TXTTAB	,133,134,135,136
9C,9D	VARTAB	,133,134,136,142,143
9E,9F	ARYTAB	,133,134,141,143
A0,A1	STREND	,133,134,141,143
A2,A3	FRETOP	,133,134
A6,A7	MEMSIZ	,133,134
AC,AD	OLDTXT	,143
B0,B1	DATPTR	,143,144
B4,B5	VARNAM	,127,131
B6,B7	VARPNT	,97
C7,C8	HIGHDS	,10,59,67,138,145
C9,CA	HIGHTR	,10,59,67,138,145
CE,CF	LOWTR	,10,59,67,138,145
D0,D1,D2	DSCTMP	,116,123,124,127,128,129
E0,E1	STRNG2	,108,115,116,118,121,
E2	CHARGET	,10,34,59,60,78,79,108,116,131
E8	CHARGOT	,59,61,66,108,115,116,124
E9,EA	TXTPTR	,10,59,60,77,78,144,145
FF-110	FBUF	,10,38
100	STACK	,59,60,83
200-205	PMOVE Oric-1	,137,138,145
20C	CAPLCK	,59,60
230	RTI Oric-1	,59,60
24A	RTI Atmos	,59,60
24D	Slow/FastAtmos	,135
268	CURROW	,10,36
269	CURCOL	,10,28,36,75
26A	MODE0	,73
26D,26E	SCRNAD Oric-1	,10,12
26F	NLIN "	,10,12
278,279	VDUL2 Atmos	,10,12
27A,27B	VDUL1 "	,10,12
27C,27C	NCHR "	,10,12
27E	NLIN "	,10,12
2A9,2AA	déb.enr.Atmos	,134,136
2AB,2AC	fin " "	,134,136
2F1	PRTFLG	,10,16
2F5,2F6	EXCLV	,53,59,60

MEMOIRE MORTE

Atmos Oric-1

C3F4	C3F8	BLTU,	,10,28,59,67,137,138,145
C47C	C483	MEMERR	,138,145
	C485	ERRMSG	,127,131
C55F	C56F	LINKSET	,136,145
C592	C5A2	INLIN	,10,59,60,77
C5E8	C5F8	INCHR	,10,12,16,20,59,71,140
C708	C733	SETPTRS	,140
C70F	C73A	CLEARC	,53,59,60,138,144,145
C726	C751	STKINI	,53,59
C816		LPRTON	,10,16
C82F		LPRTOF	,10,16
C8C1	C8AD	NEWSTT	,144,145
CBF0	CBAF	CRDO	,10,14,16,22,59,62,63,65,66,72,76
CCB0	CBED	STROUT	,10,140
CCCE	CC0A	CLS	,10,12,14
CCD1		CURTGL	,37
CCD4	CC0D	OUTSPC	,10,16,18,22,38,59,63,65,74,80
CCD9	CC12	OUTDO	,10,16,18,22,24,26,36,38,40,59,60,63,65,70,71, 74,76,77,80,81,82
CD80	CCF4	INPTRQ	,10,34,145
D065	CFD9	CHKCOM	,97,108,116,120
	CFE4	SYNTER	,127,131
D188	D0FC	PTRGET	,97,116,125
	D186	ISLETC	,131
D333	D29D	SUBERR	,108,109
	D4F0	STRSPA	,129
	D595	GARBAGE	,129
D993	D8F5	BINHEX	,10,38,59,83
DE7B	DE73	MOVFM	,108,113,116,120
DF4C	DF34	FCOMP	,108,113,116,123
E4AC		INLED	,136
E57D		SRCHNG	,136
E76A	E6CA	VIAON	,136
E853	E79D	MAKINT2	,138,145
E8A4		LOAD	,136
	E804	VIAOFF	,127,128,129
E94C	E813	HEXGET	,53,58,59,60,66,69,79
EB78	E905	RDKEY	,10,16
EDC4	EC0C	MOVE	,137,138,139,145
F75A	F729	PRCAPS	,59,60
F865	F82F	STOUT	,10,38,59,83
F8B2	F882	RESET	,10,12,59,60
F8B8	F888	RESET0	,10,12,59,60
F90E	F8D1	CURSON	,59,71
FA9F	FA85	PING	,10,12,14,22,34,59,67,72,77,79

Oric-1
routines cassette
voir pages 127 et 136

TABLE DES MATIERES

LES PROGRAMMES DE CE LIVRE SUR CASSETTES

Les principaux programmes décrits dans cet ouvrage sont disponibles en "version originale", enregistrés sur cassettes.

Pour tout renseignement, veuillez contacter la société :

ISOSOFT FRANCE
BP 22
49130 Les Ponts-De-Cé

ISBN 2-904266-04-6
Prix : 120 F TTC

Voici le deuxième tome du Manuel de référence Atmos-Oric 1. Il parachève l'étude de ces ordinateurs d'une manière résolument pratique à l'aide d'exemples commentés permettant d'acquérir connaissances, habileté et pratique.

Chacune des trois parties qui le composent contient de nombreux programmes présentés de façon la plus claire : listing désassemblé en page gauche, commentaires en regard en page droite. Le lecteur pourra ainsi apprendre à manipuler les pointeurs Basic ou découvrir "mille et une" méthodes de tri, sujet peu traité dans la littérature informatique.

Les lecteurs du premier tome en retrouveront le style agréable et le langage accessible ; ceux abordant directement cet ouvrage suivront avec passion les chemins de la découverte et de l'approfondissement tracés par André.

Dépôt légal 3e trimestre 1985