Savez-vous utiliser le T.I.B. ?

par André Chéramy

C'est le B + A = BA de la programmation sur Oric. Ce truc est utilisé par exemple pour charger les logiciels composés de plusieurs morceaux (module de protection, page de copyright, éléments de programme en BASIC, routines en langage machine, etc.). Dans un programme BASIC, il suffit de chaîner les divers morceaux avec des CLOAD"" ou des LOAD"...", voire des LOAD implicites (NOM ou !NOM du fichier). Dans un fichier non BASIC, par exemple dans un module en langage machine, il est possible d'écrire une routine faisant appel aux sous-programmes de la ROM (pour CLOAD) ou de la RAM OVERLAY (pour les LOAD directs ou implicites). Mais cela est bien lourd, alors qu'il y a beaucoup plus simple : Il suffit d'écrire l'ordre BASIC ou SEDORIC dans le tampon clavier (T.I.P. pour Terminal Input Buffer) et d'appeler l'interpréteur. En outre, ce truc permet d'exécuter n'importe quelle commande BASIC ou SEDORIC!

METHODE INIST exemple : exécuter la commande DIR"*.BAS"

Un modèle est donné dans la banque n°0, en #C54A, lorsque Sédoric exécute les instructions de démarrage (INIST, nous reverrons plus loin le problème de la bogue "SET/OFF") : il suffit de placer les commandes Sédoric avec leurs paramètres dans le TIB, d'initialiser correctement TXTPTR et de faire appel à l'interpréteur en ROM. Cette méthode, très simple, a un inconvénient : on retourne au "Ready" (ce qui est voulu dans le cas de INIST). A l'aide de votre moniteur favori, implantez directement les 31 octets en gras à partir de #9801 :

```
9801  44 49 52 22 2A 2E 42 41 53 22 00   commande  'DIR"*.BAS"#00'
980C  A2 0B        LDX #0B               pour copier les 11 octets de cette commande
980E  BD 01 98     LDA 9801,X            lit les octets de 9801 à 980B
9811  95 35        STA 35,X              et les copie dans le T.I.B. de 35 à 3F
9813  CA           DEX                   octet précédent (par la fin)
9814  10 F8        BPL 980E              et reboucle tant qu'il en reste
9816  A2 34        LDX #34               XY pour ajuster TXTPTR à 0034
9818  A0 00        LDY #00               adresse C4BD de l'interpréteur
981A  20 BD C4     JSR C4BD              Atmos (prendre C4CD pour l'Oric-1)
981D  4C B5 FA     JMP FAB5              SHOOT (FA9B pour la ROM V1.0)
```

La 1ère ligne contient la commande à placer dans le T.I.B (#00 = fin de ligne de commande). Les 2ème à 6ème lignes contiennent le code pour effectuer cette copie, les 7ème à 9ème lignes, le code pour exécuter le contenu du T.I.B. et la 10ème ligne, le code pour exécuter un SHOOT (pour montrer qu'après avoir exécuté le T.I.B., l'interpréteur retourne directement au "Ready" sans exécuter ce SHOOT). La ligne 310 charge les 31 octets des DATA en RAM à partir de #9801.

SAVE"TIB1",A#9801,E#981F puis CALL#980C. Si tout va bien ce programme affiche le catalogue des fichiers "*.BAS" et retourne au "Ready" sans SHOOT (et oui !).

METHODE HENNINOT : même exemple, mais sans retourner au "Ready"

Une autre méthode, préconisée par Denis Henninot, permet de retourner dans votre programme en détournant le vecteur 1B/1C. En effet, l'affichage du message "Ready" se fait en #C4B4 (ROM) par un JSR 001A. Or, en 1A/1B/1C se trouve un JMP CCB0, adresse de la routine qui affiche "Ready". Il suffit de remplacer ce JMP CCB0 par JMP adresse de la suite de votre programme, et ça repart ! Cette méthode simple de mise en oeuvre permet aussi de bénéficier de la gestion des erreurs.

```
9801  44 49 52 22 2A 2E 42 41 53 22 00   commande  'DIR"*.BAS"#00'
980C  A2 00        LDX #00               autre méthode de copie : index initial #00
980E  BD 01 98     LDA 9801,X  lit les octets de 9801 jusqu'au #00  de fin de commande
9811  95 35        STA 35,X    et les copie dans le TIB de 35 à 3F
9813  F0 03        BEQ 9818    si octet copié est #00, c'est fini
9815  E8           INX         sinon, vise l'octet suivant
9816  D0 F6        BNE 980E    et reboucle (donc chaîne de 256 caractères au maximum)
```

```
9818  A9 27       LDA #27     LL de l'adresse de retour pour détournement du
981A  85 1B       STA 1B      vecteur 1B/1C (affichage du "Ready") vers la suite
981C  A9 98       LDA #98     du programme appelant
981E  85 1C       STA 1C      idem avec HH
9820  A2 34       LDX #34     pour ajuster TXTPTR juste avant le début de la commande
9822  A0 00       LDY #00     au début du tampon clavier et continue
9824  4C BD C4    JMP C4BD    à l'interpréteur Atmos (prendre C4CD pour l'Oric-1)
9827  A9 B0       LDA #B0     remet en place le vecteur normal
9829  85 1B       STA 1B      CCB0 (CBED pour l'Oric-1)
982B  A9 CC       LDA #CC     d'affichage du "Ready"
982D  85 1C       STA 1C      en 1B/1C et continue la suite du programme appelant :
982F  4C B5 FA    JMP FAB5    SHOOT (FA9B pour l'Oric-1)
```

Les adresses suivantes doivent bien sûr être ajustées en fonction de votre contexte :

```
9801  adresse de la commande (on peut implanter ailleurs)
980C  adresse d'exécution du sous-programme (CALL ou JSR à cette adresse)
9827  adresse de retour après passage sous Sédoric (suite de votre programme)
9831  fin du sous-programme (SAVE"TIP2",A#9801,E#9831)
```

Cette fois, le CALL#980C doit non seulement afficher la liste des fichiers "*.BAS", mais aussi faire entendre un joli SHOOT avant de retourner au "Ready" ! Sur ce même principe, vous pourrez utiliser la plupart des commandes BASIC ou SEDORIC dans tous vos programmes en langage machine.

WANTED ! (suite et fin)

Profitons de l'occasion pour revoir la bogue de INIST, liée à l'utilisation des paramètres SET et OFF. Dans ce qui suit, le signe "_" représente un espace. Dans le programme TIB2, vous pouvez remplacer les 10 caractères de la commande DIR"*.BAS" par les chaînes qui vont suivre. Si la programmation en langage machine vous est pénible, utilisez plus simplement le petit programme BASIC suivant :

```
100 A$="PR_OFF:DIR"+CHR$(0) : REM pour ajouter un #00 à la fin comme dans TIB2
110 TKEN A$ : STRUN A$
```

Voici quelques chaînes qui donnent une erreur : "PR_OFF____", "PR_OFF_:__", "PR_OFFDIR:", "PR_OFFXDIR". Celles qui suivent n'en donnent pas : "PR_OFF:___", "PR_OFF:DIR", "PR_OFF::::", "DIR:PR_OFF" (donc avec #00 juste après). Vous arriverez à la conclusion suivante : Il n'y a de "SYNTAX ERROR" que si le caractère qui suit OFF (idem pour SET) est autre que ":" ou #00.

On peut finalement retrouver le problème en mode direct avec PR_OFF_ ou en mode programme avec 100 PR_OFF_ suivi de RUN. De même, une ligne de programme innocente, mais si terminant par SET ou OFF donnera une "SYNTAX ERROR" après avoir été recopiée par des CTRL/A ayant dépassés la fin de la ligne (et donc ayant copié au moins un espace !). On peut s'arracher les cheveux ensuite pour comprendre pourquoi cette ligne donne obstinément une erreur alors que la syntaxe est correcte (l'espace est invisible). Les jolis alignements de REM que l'on trouve dans certains listings sont également à proscrire:

```
100 CLS          :' Ici pas de problème
200 PR SET       :' Mais cette ligne donnera une "SYNTAX ERROR"
```

Lorsqu'une commande PR, ERR, KEY ou ACCENT est détectée, la routine Sédoric #E94D recherche elle-même l'argument SET ou OFF, sans passer par l'interpréteur de commandes. Si elle trouve SET ou OFF, elle positionne la retenue respectivement à 1 ou à 0, puis elle met à jour TXTPTR en lui ajoutant 3. Et voilà la bogue, car si dans la plupart des cas TXTPTR pointe sur le signe de fin d'instruction (":") ou de fin de ligne de commande (#00) qui suit normalement SET ou OFF, il n'en est pas de même dans le cas particulier où il y a un autre caractère, quel qu'il soit, ne serait-ce qu'un innocent espace !

Je ne pense pas qu'il soit simple de corriger cette bogue. De plus, cela n'en vaut peut être pas la peine. Ajoutez donc une ligne dans votre manuel Sédoric : "Attention, pas d'espace après SET ou OFF, ceci inclut les espaces implicites des fenêtres de type LINPUT, utilisées par exemple dans la commande INIST".