Le compilateur Basic de Ray MacLaughlin (2)

Le cas des commandes Sedoric

par André C.

Etat de la question

Dans mon premier article (Le Basic Compiler de Ray McLaughlin, CEO-mag n°332 de décembre 2017), j'ai fait la promotion de ce magnifique outil, avec quelques tests à l'appui. J'avais signalé les limitations officielles de cet utilitaire. Depuis, j'ai eu l'occasion de l'utiliser à plusieurs reprises et j'ai découvert une limitation bien plus grave que celles signalées par Ray : Les commandes Sedoric ne sont pas honorées et déclenchent des erreurs de compilation...

Si l'on a besoin de vitesse d'exécution et que l'on veuille absolument utiliser des commandes Sedoric avec le compilateur de Ray, il existe deux solutions à ce problème :

1) Écrire normalement le programme Basic puis remplacer les commandes Sedoric par des sous-programmes en langage machine. Le compilateur de Ray devrait avaler ce programme car il sera complètement conforme à la syntaxe Basic.

2) A partir du même programme de départ, conserver les commandes Sedoric, mais repérer les parties grandes consommatrices de temps d'exécution, les transformer en sous-programmes et les mouliner individuellement avec le compiler de Ray. Quelques CALL et le tour est joué...

La première solution me semble de loin la plus élégante, la plus simple et surtout la plus universelle (réutilisable dans d'autres programmes). C'est cette solution que je vous propose aujourd'hui et qui, je l'avoue tout de suite, sera un échec… (provisoire).

Démonstration de l'utilisation de la commande CALL avec le compilateur Basic

Avant de se lancer dans les grands travaux, vérifions que la commande CALL est bien acceptée par le compilateur de Ray et qu'elle fonctionne parfaitement. Soit le classique programme Basic affichant un message sur la ligne service (voir les fichiers SALUT.TXT et SALUT.TAP dans Compiler2.zip sur la disquette accompagnant ce mag) : Il s'agit d'un chargeur Basic, qui implante un sous-programme en langage machine à partir de l'adresse #9801 (figure 1, ci-dessous). L'explication de ce sous-programme est donnée à la figure 2 page suivante.

```
100 DATA #53,#61,#6C,#75,#74,#20,#6C,#65
110 DATA #73,#20,#67,#61,#72,#73,#20,#21,#00
120 DATA #A0,#00,#B9,#01,#98,#F0,#06
130 DATA #99,#82,#BB,#C8,#D0,#F5,#60
120 FOR I=1 TO 31:READ V:POKE#9800+I,V:NEXT
130 CALL#9812
140 PING:END
```

Figure 1

Après avec vérifié que le programme natif SALUT.TAP (sans compilation) s'exécute sans problème, je le compile. Aucun souci particulier, je l'enregistre et l'exécute. La figure 3 (page suivante) vous montre le résultat qui est conforme à ce qui était attendu : affichage du message suivit d'un PING.

Conclusion : La commande CALL fonctionne correctement avec le compiler de Ray.

```
9801 53 61 6C 75 74 20 6C 65 73    Message "Salut les"
980A 20 67 61 72 73 20 21 00  " gars !" suivi d'un zéro
9812 A0 00     LDY #00     mettre zéro dans le registre Y
9814 B9 01 98  LDA 9801,Y  lire un octet du message
9817 F0 06     BEQ 981F    si c'est 0, fini, on termine en 981F
9819 99 82 BB  STA BB82,Y  sinon on copie l'octet en BB82+Y
981C C8        INY         Y=Y+1 pour indexer l'octet suivant
981D D0 F5     BNE 9814    reboucle tant que Y ne repasse pas à 0
981F 60        RTS         retourne au point d'appel
```

Figure 2

```
Salut les gars !                                    CAPS

Enter drive letter? A
A-ADDRESSES.BIN #501    #527
A-CODEA.BIN     #528    #592
A-FNDEFS.BIN    #593    #593
A-LIBRARY.BIN   #594    #BBD
A-DATA.BIN      #BBE    #C3A
A-LITERALS.BIN  #C3B    #C4F
A-ONLISTS.BIN   #C50    #C50
A-SCALARS.BIN   #C51    #C5F
A-ARRAYS.BIN    #C60    #C60

Enter name for file? PASSSBIN

The file will run automatically.
You may run it now by entering
A-PASSSBIN

Ready
PASSSBIN

Ready
```

Figure 3

Figure 4 : Chargeur “Simulation d’un LOAD”

```
100 DATA A5E98D4F98A5EA8D5098
200 DATA A200BD3B989535F003E8
300 DATA D0F6A926851BA998851C
400 DATA A234A0004CBDC4AD4F98
500 DATA 85E9AD509885EAA9B085
600 DATA 1BA9CC851C20C1C84C4F
700 DATA 4144224558454D504C45
800 DATA 30312E53435222000000
900 FOR I=1 TO 80:READ V:POKE#
    9800+I,V:NEXT
999 SAVE"ESSAI",A#9801,E#9850
```

Simulation d'un LOAD par une routine en langage machine

Comme d’habitude, je vous donne le chargeur Basic qui implante un sous-programme en langage machine à partir de #9801 (figure 4 et Essai.tap dans le fichier Compiler2.zip sur la disquette accompagnant ce mag).

Vous trouverez l'explication de ce sous-programme sur la figure 5 à la page suivante (il s’agit en fait du fichier source en assembleur Essai.asm disponible dans le fichier Compiler2.zip sur la disquette accompagnant ce mag).

Stratégie de ce programme

Elle consiste à placer la chaîne de caractères à exécuter dans le tampon clavier (TIB ou Terminal Input Buffer). Dans notre exemple, cette chaîne est LOAD"EXEMPLE01.SCR", suivie d’un zéro de fin de chaîne. Il faut ensuite ajuster le pointeur de lecture (TXTPTR ou TXT PoinTeR), juste devant le "L" de LOAD (la routine de lecture commence par incrémenter TXTPTR, qui pointera alors sur le "L") et enfin appeler l’interpréteur Basic en $C4BD.

Notez que tout autre chaîne de commande peut être évidemment utilisée à la place de celle de l’exemple.

Astuce : Dans LDA commande,X l’adresse ‘commande’ se trouve en $980E. Pour utiliser ce même sous-programme avec différentes chaînes de commandes, il suffira de DOKEr en $980E l’adresse de la chaîne à exécuter ($983D dans notre exemple). Ces différentes chaînes peuvent être placées à la suite les unes des autres, en fin de sous-programme, sans rien avoir à modifier à part l’adresse en $980E avec le DOKE#980E,...

Autre possibilité : vous pouvez remarquer que le nom de l’écran à charger est de la forme "9+3 caractères" et qu’il suffit donc de POKER tout autre nom de fichier dans la chaîne située à partir de $9842, en formatant si besoin avec des espaces.

Tout ça, c’est bien beau, mais le JMP $C4BD retourne au "Ready" ! Il faut donc prendre

Figure 5

```
; Sous-programme LOAD"xxxxxxxxx.xxx" en langage machine
; pour exécution avec CALL dans un programme Basic
   org $9801       ; Adresse d'implantation de ce sous-programme
   LDA $E9         ; Sauvegarde TXTPTR (pointe sur ":" ou #00 qui suit CALL)
   STA txtptr
   LDA $EA
   STA txtptr+1
   LDX #$00
boucle
   LDA commande,X ; Lit les octets de la commande jusqu'au #$00 de fin de
   STA $35,X      ; chaîne et les copie dans le tampon clavier
   BEQ suite      ; si l'octet copié est #$00, c'est fini
   INX            ; sinon, vise l'octet suivant
   BNE boucle     ; et reboucle (donc chaîne de 256 caractères au maximum)
suite
   LDA #LOW retour; Détourne l'adresse du vecteur 1B/1C (affichage du Ready)
   STA $1B        ; vers la suite de notre sous-programme à l'adresse retour
   LDA #HIGH retour    ; pour en continuer l'exécution sans rendre la main
   STA $1C
   LDX #$34       ; ajuste TXTPTR au début du tampon clavier juste avant le
   LDY #$00       ; début de la commande et continue à la lecture du tampon
   JMP $C4BD      ; clavier et exécution de la commande Sedoric LOAD
retour
   LDA txtptr     ; restaure le TXTPTR à l'adresse $E9-EA
   STA $E9
   LDA txtptr+1
   STA $EA
   LDA #$B0       ; remet en place le vecteur #CCB0 normal d'affichage du
   STA $1B        ; Ready
   LDA #$CC
   STA $1C
   JMP $C8C1      ; Exécution de la ligne Basic
txtptr
   db $00, $00
commande
   string 'LOAD"EXEMPLE01.SCR"'
   db $00
fin
```

deux précautions :

1) Sauvegarder le TXTPTR qui pointe sur le ":" ou sur le #00, situé juste après le CALL#9801, afin de reprendre le programme Basic à cet endroit-là. Rappel : Dans un programme Basic, les commandes sont séparées par " :" et un #00 est ajouté en fin de ligne après la dernière commande.

2) Détourner le vecteur d'affichage de "Ready" afin qu'il pointe sur la suite de notre sous-programme en langage machine. Lorsque l'interpréteur Basic re passera en mode direct, il tentera d'afficher "Ready" (JMP $CCB0), mais à la place exécutera JMP $9826 et continuera donc à cette adresse, qui se trouve dans notre sous-programme, juste après notre JMP $C4BD ! La deuxième partie de notre sous-programme commence alors. Il s'agit de reprendre l'exécution du programme Basic là où nous en étions restés, à savoir juste après le CALL#9801. J'ai pas mal galéré pour trouver le truc et pourtant il est hyper-simple ! En premier lieu, restaurons les dégâts que nous avons commis précédemment : Il faut remettre en place la valeur qu'avait le TXTPTR avant que nous ne le tripotions. Tant que nous y sommes restaurons la valeur normale

du vecteur d'affichage du Ready. Reste quoi ? Juste à faire un petit JMP $C8C1 pour retourner à l'exécution de la ligne Basic !

En pratique

J'ai utilisé l'assembleur Frasm qui m'est familier depuis que j'ai travaillé sur le code du Super-Oric de Fabrice F., dont le fichier source est justement conçu pour Frasm. Mais rien ne s'oppose à ce que vous utilisiez un autre assembleur.

La compilation du programme source Essai.asm avec l'assembleur Frasm conduit d'abord au fichier Essai.hex que je mouline avec Hex2oric.exe pour obtenir le fichier Essai.tap. Ce fichier est un chargeur Basic qui met en place, de #9801 à #9850, le code du sous-programme en langage machine, que je sauve sur une disquette Sedoric normale sous le nom Essai.bin.

Avant toute chose, je teste Essai.bin avec le petit Basic suivant :

```
10 CALL#9801:PING
20 GET R$
30 END
```

Figure 6

Je sauve ce petit Basic sous le nom Essai.bas en vue de le compiler plus tard.

Minute de vérité : Je mets en place Essai.bin, puis Essai.bas. Après le RUN, l'écran Exemple01.scr apparaît (figure 7),

Figure 7

suivit d'un joyeux PING.

Le programme attend gentiment que vous ayez fini d'examiner l'écran et ayez appuyé sur une touche, pour rendre la main et retourner au Ready.

Tout marche comme prévu et nous pouvons dons passer à la suite.

Test avec le Compiler de Ray

Après avoir ainsi vérifié que le programme natif Essai.bas (sans compilation) s'exécute sans problème (il faut évidemment avoir Essai.bin et Exemple01.scr sur la même disquette), je le mouline avec le compiler de Ray. Aucun souci particulier de ce côté. J'enregistre le résultat sous le nom Essai1.com que je transfère sur ma disquette Sedoric où se trouvent déjà Essai.bin et Exemple01.scr.

Après avoir rebouté sur cette disquette, je mets en place le sous-programme en langage machine Essai.bin puis le Basic compilé Essai1.com dont l'exécution se lance en AUTO.

Cata ! Syntax Error in 23792 !

A tout hasard je reloge Essai1.com en $9901 que je sauve sous le nom Essai2.com et teste à nouveau.

Même Syntax Error in 23792 !

Conclusion

Mon hypothèse est que la sauvegarde et la restauration du TXTPTR, qui marche si bien lorsqu'on lance le sous-programme en langage machine à l'aide d'un CALL dans un programme Basic normal, ne fonctionne plus avec un programme Basic compilé puisqu'il n'y a plus de texte à suivre pas à pas avec TXTPTR.

Ce fut un bel effort, mais il faut le reconnaître, une grossière erreur de conception de ma part ! Si vous avez une idée, n'hésitez pas à me contacter. J'aurais dû intituler cet article "Histoire d'un échec", mais je n'ai pas dit mon dernier mot…

à suivre…