Screen Saver *Hp 49*

Totalement inutile, mais véritablement indispendable : remplacez l'extinction automatique de votre Hp 49 par de belles veilles d'écrans...

Tous les programmes de cet article sont réalisés dans le mode RPN. Pour basculer dans ce mode, appuyer sur [MODE], [F2], [Bas] et [ENTER] à deux reprises pour valider le tout.

• le b-a-ba •

Une fois de plus, la Hp 49 se distingue de sa soeur aînée par des fonctions très intéressantes qui permettent de modifier la durée avant mise en veille de la machine, d'exécuter un programme lors d'une extinction et une autre exécution lors d'un redémarrage à chaud. Mais comment gérer tout ça ?
Pour régler la durée avant mise en veille, il suffit de se placer dans le répertoire racine HOME et créer une variable nommée **'TOFF'** dans laquelle on stocke un entier binaire qui correspond à la valeur en ticks (1 tick = 1/8192ème de seconde). Pour exemple, si on choisit une mise en veille au bout de 10 secondes, on saisit cette séquence dans ligne de commande : `# 14000h 'TOFF' STO`. La valeur 14000 en hexadécimal correspond à 81920 en décimal (ou 8192x10). Par défaut (quand la variable **TOFF** n'existe pas), la machine se met en veille au bout de cinq minutes (l'équivalent de `# 258000h`).
Pour lancer un programme à la place de l'arrêt automatique de la machine (géré par la valeur stockée dans la variable **TOFF**), il suffit de stocker un objet dans HOME nommé **STARTOFF**. Cet objet est généralement un programme comme cet exemple : `« CLLCD "A bientôt !" 4 DISP 1000 .5 BEEP 2 WAIT OFF »`. Attention, ce programme remplace complètement l'extinction de la machine, c'est à dire le `OFF`. C'est donc pourquoi il est ajouté en fin de programme pour que la machine s'éteigne effectivement. Petit détail : lors de la remise sous tension de la machine, l'écran ne se rafraîchit pas, c'est à dire que nous avons toujours le message affiché...
Pour lancer un programme juste après un redémarrage à chaud, le principe est le même que pour **STARTOFF** sauf que cette fois on stocke l'objet (qui est toujours un programme) dans la variable **STARTUP**. Les programmeurs s'en sert bien souvent pour initialiser quelques flags bien utiles, et les plus fantaisistes s'en servent pour afficher un message de bienvenue...
Maintenant que vous savez tout, profitons de ces caractéristiques intéressantes pour créer deux petites veilles d'écran 100% assembleur.

• détails de programmation •

Puisque cette fois nous travaillons directement sur la Hp 49, utilisons l'assembleur intégré MASD qui utilise aussi la syntaxe d'Asmflash, et qui présente des caractéristiques supplémentaires vraiment intéressantes.
Pour commencer, des mnémoniques font appel directement à des routines assembleur. Exemple, `SAVE` correspond à `GOSBVL 0679B` pour la sauvegarde des registres et pointeurs. En voici quelques-uns très intéressants pour notre programme :
`SAVE` : `GOSBVL 0679B` (sauve registres et pointeurs),
`LOADRPL` : `GOVLNG 05143` (récupère registres et pointeurs et retourne au Rpl),
`OUT=C=IN` : `GOSBVL 0020F` (test de touche par un OUT=C et un C=IN),
`SCREEN` : `GOSBVL 2677C` (récupère l'adresse de début de l'écran courant).
Pour la gestion des touches, la structure est la suivante pour la touche [ENTER] :

```
LC 001 OUT=C=IN ?CBIT=1.0 -> SAUT
```

Pour la touche [ON], il suffit juste d'utiliser cette ligne :

```
OUT=C=IN ?CBIT=1.15 -> SAUT
```

Les boucles présentent une structure bien pratique en se qui concerne les sauts conditionnels (en fonction de la retenue). Voici l'exemple d'une pause telle qu'on la retrouvera dans le listing de la source :

```
LC 7FF { C-1.X UPNC }
```

Ici, tant que la retenue ne passe pas à 1, on boucle la partie entre accolades.
On retrouvera les derniers détails dans la source de la veille.

• la balle •

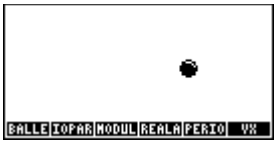
On pourrait imaginer des veilles d'écran comme on peut en voir sur ordinateur avec des algorithmes compliqués, mais nous nous limiterons à un grand classique : une balle qui rebondit sur les bords de l'écran en tournant. La balle partirait du centre de l'écran, prendrait une trajectoire à 45°et changerait de direction une fois arrivée sur l'un des bords de l'écran. L'animation s'arrêterait suite à un appui sur [ON].
Avant tout, définissons les différentes images des balles pour simuler la rotation : il y a quatre images et chaque image présente une taille de 9x9.

Le programme en langage machine n'a pas besoin d'externals de préparation. En voici la source commentée :

```
"SAVE ST=0.15 %sauve registres et pointeurs, et interdit les interruptions
SCREEN R0=A.A %adresse de début d'écran courant
LC 3D R1=C.B ST=0.5 %position en X, sens +
LC 17 R2=C.B ST=0.6 %position en Y, sens +
GOSUB BALLE
$830EF0EF0FF1FF1DF1AF06C0830 %première balle
$830EF0EF0F71F71F71EB0ED0830 %seconde balle
$8306C0EB0F71FF1FF1EF0EF0830 %troisième balle
$8306F0AF0DF1DF1DF1EF0EF0830 %quatrième balle
*BALLE
C=RSTK R3=C.A %adresse première balle
*KEY
OUT=C=IN ?CBIT=1.15 -> FIN %test de [ON]
A=R0.A D0=A A=0.W LC 76
{ DAT0=A.W D0+16 C-1.B UPNC } %efface écran
GOSUB AFF %affiche la balle
LC 7FF { C-1.X UPNC } %petite pause
GOTO KEY %et on boucle
*FIN
GOSBVL 267AD %initialisation de [ON]
LOADRPL %récupère registres et pointeurs, et retourne au RPL
*AFF %routine d'affichage
A=R1.B %prend X
?ST=1.5 -> { A-1.B } %sens - => on décrémente
?ST=0.5 -> { A+1.B } %sens + => on incrémente
```

```
R1=A.B %sauve X
LC 7A
?A=C.B -> { ST=1.5 } %passage en sens - sur X
?A=0.B -> { ST=0.5 } %passage en sens + sur X
A=R2.B %prend Y
?ST=1.6 -> { A-1.B } %sens - => on décrémente
?ST=0.6 -> { A+1.B } %sens + => on incrémente
R2=A.B %sauve Y
LC 2F
?A=C.B -> { ST=1.6 } %passage en sens - sur Y
?A=0.B -> { ST=0.6 } %passage en sens + sur Y
C=0.A C=R2.B %prend Y
C+C.A B=C.A CSL.A C+B.A %on se place sur la bonne ligne
A=0.A A=R1.B ASRB.B ASRB.B %puis sur la bonne colonne
A+C.A C=R0.A A+C.A D0=A %et on pointe l'écran
C=0.A C=R1.B A=C.B ASRB.B ASRB.B A+A.B A+A.B
C-A.B D=C.A C+C.A A=C.A CSL.A %décalage en bits sur un quartet
A+A.A C-A.A C-D.A %on multiplie 27 par le numéro de la balle
A=R3.A A+C.A D1=A %on pointe la bonne balle
LC 08 B=C.B %boucle d'affichage d'une balle
{ A=DAT1.X C=D.B %initialisation
*TST
C-1.B GOC DECAL A+A.X GOTO TST %décalage de la balle
*DECAL
DAT0=A.X D1+3 D0+34 B-1.B UPNC } %et on se déplace
RTN
@"
```

Pour compiler cette source, il suffit de la poser sur le niveau 1 de la pile et d'utiliser la commande **ASM**. Stockez ensuite le code obtenu dans la variable **STARTOFF** et lancez-là. Si tout fonctionne normalement, voici la balle à l'instant t :



Pour qui préfère saisir directement la chaîne hexadécimale (sans espace ni saut) et la compiler avec **->H**, voici ce qu'il vous faut :

```
CCD20 3F100 8FB97 6084F 8FC77 6281A F0031 D381A
60984 53171 81A60 A8467 C6083 0EF0E F0FF1 FF1DF
1AF06 C0830 830EF 0EF0F 71F71 F71EB 0ED08 30830
6C0EB 0F71F F1FF1 EF0EF 08308 306F0 AF0DF 1DF1D
F1EF0 EF083 00781 AF0B8 FF020 0808B F2381 AF101
30AF0 31671 50716 FA6E5 5F7D1 032FF 7A3E5 CF65C
F8FDA 7628D 34150 81A61 18655 0A6C8 7550B 6481A
60131 A7966 50855 96C50 84581 A6128 6650A 6C876
50B64 81A60 231F2 96650 85696 C5084 6D281 A61AC
6D5F2 C9D08 1A611 81960 81960 CA81A F18CA 130D2
81A61 9AEA8 19608 1960A 64A64 B62D7 C6DAF 2C4E2
EB81A F13CA 13131 80AE5 1533A EBA6E 490A3 466FF
15031 7216F 16F16 1A6D5 8D01
```

Si vous mettez 5 secondes dans la variable **TOFF** (# A000h pour être précis) et que vous attendez un instant, la balle se mettra en mouvement. Pour l'arrêter, un simple appui sur la touche [ON] règle ça.

• big screen •

Voici un programme qu'on peut stocker dans **STARTOFF** et qui peut servir de veille d'écran. Il agrandit la zone d'état de la Hp 49 sur la totalité de l'écran. Cette source n'est pas commentée : à vous de découvrir quelles en sont les ficelles...

```
"!RPL !NO CODE
::
TURNMENUOFF
CODE
SAVE ST=0.15
SCREEN LC 0021E A+C.A D0=A
LC 00660 A+C.A D1=A
LC 0F B=C.B
*BOUCLE
A=DAT0.B LC 03
{ DAT1=A.B D1-34 C-1.B UPNC }
D0-16 AD1EX LC 00078 A+C.A D1=A
A=DAT0.W LC 03
{ DAT1=A.W D1-34 C-1.B UPNC }
D0-16 AD1EX LC 00078 A+C.A D1=A
A=DAT0.W LC 03
{ DAT1=A.W D1-34 C-1.B UPNC }
D0-2 D1+32
B-1.B GOC SUITE
GOTO BOUCLE
*SUITE
OUT=C=IN ?CBIT=1.15 -> FIN
GOTO SUITE
*FIN
GOSBVL 267AD
LOADRPL
ENDCODE
;
@"
```

• une petite conclusion •

Les veilles d'écran peuvent très bien personnaliser votre machine, bien souvent grâce à de superbes animations (même si cet article vous présente des animations bas de gamme...). Mais sur une calculatrice puissante comme la Hp 49, les batteries ont une autonomie assez réduite en utilisation intense. Quand on utilise une veille d'écran comme celle qu'on vient de décrire, le microprocesseur tourne à plein régime et les piles en souffrent d'autant. A utiliser avec parcimonie !

Philippe Pamart
[phpamart@nordnet.fr]

Introduction à MASD *Hp 49*

Chers (futurs) codeurs, voici quelques rudiments d'explications sur l'assembleur intégré à la Hp 49 : un outil à maîtriser absolument.

• à quoi ça sert ? •

Un assembleur permet de compiler une suite d'instructions (appelées des mnémoniques) qui correspondent directement à des codes machine en hexadécimal, d'où l'appelation "langage machine". Pour ceux qui ne savent pas, le microprocesseur utilisé est le Saturn, cadencé à 4 MHz avec une architecture 4 bits, ce n'est donc pas un microprocesseur de première jeunesse...

L'assembleur est un langage qui utilise des registres de 64 bits (4 de calcul et 5 de sauvegarde), des pointeurs (2 au total !), et différents flags. Et on fait tout avec ça ! MASD, l'assembleur en question, est intégré dans la rom de la Hp 49 et nécessite une library supplémentaire livrée qui accompagne la rom. Cette library est une table de conversion externals entre les mnémoniques des points d'entrée et leurs adresses. Exemple : TURNMENUOFF correspond à l'adresse # 2F034h. Ce qui permet de rendre plus agréable la saisie des sources.

L'assembleur est le seul langage qui permet d'atteindre réellement les limites de la machine, voir les différents logiciels de qualité qui circulent sur internet.

• le tout début •

Le but du jeu n'est pas de vous apprendre l'assembleur sur Hp 49, mais de vous montrer les possibilités de MASD. Il faut savoir que ce logiciel permet de travailler avec des externals et/ou de l'assembleur. Mais commençons par la structure type d'une source assembleur :

```
"RPL
@"
```

C'est le programme le plus simple que l'on puisse faire (et qui ne fait rien) : il effectue un retour en Rpl. La routine `RPL` correspond à la séquence `A=DAT0.A D0+5 PC=(A)`. Voici une autre structure (qui ne fait toujours rien) qui a l'avantage de sauvegarder et récupérer les registres en début et en fin de programme, ceci avant de retourner au Rpl. Cette structure est utilisée dans 90% des cas :

```
"SAVE
LOADRPL
@"
```

A partir de cette source, vous pouvez insérer les tâches à effectuer entre le SAVE et le LOADRPL, comme dans les sources présentées dans l'article sur les screen savers. Notez que chaque fin de source s'achève par le caractère **@**, ce qui est nécessaire pour dire au logiciel que notre chaîne de caractères est bien une source à assembler. Enfin, pour assembler on utilise la commande **ASM** qui se trouve dans la libary de développement (touche [APPS] pour y accéder).

• la syntaxe •

• Il est très vivement conseillé de consulter la documentation de MASD qui se trouve sur le cdrom. Néanmoins, observons quelques petits changements qui vont vous faciliter la vie. Tout d'abord quelque soit l'opération ou le test à effectuer, la notation est la suivante : `opération.champ`, comme par exemple `A=0.B` où `A=0` est l'opération et `B` le champ. Cette notation est aussi valable pour les manipulations de drapeaux comme par exemple : `ST=1.7`.

• Toujours au sujet des opérations, on peut dorénavant contracter les opérations telles que `A=A+C.A` en `A+C.A`, ou `C=C-A.X` en `C-A.X`, ou `C=C+C.W` en `C+C.W`, ou `D1=D1-34` en `D1-34`. Vous aurez bien évidemment compris le principe de contraction.

• Les sauts se font de manière classique en appelant un nom de label. La nouveauté est que l'on peut avoir des labels dits "locaux" qui appartiennent uniquement à la source en cours (sachant que l'on peut appeler des macros). Un label local présente la syntaxe suivante : `*.nom`. Seul l'ajout d'un point permet de définir la différence entre un label local et un label global. Cela sert essentiellement à accélerer la compilation des sources.

• Les routines internes peuvent être directement appelées par leurs adresses, mais aussi par leurs mnémoniques correspondant. Exemple : `GOVLNG 26AAF` correspond à `GOVLNG aLineXor`. Attention, il faut absolument disposer de la table des mnémoniques pour bénéficier de cet artifice.

• Et vous l'aurez sans doute observé dans l'ensemble des sources présentées, on peut maintenant apposer plusieurs instructions sur la même ligne. Pour réaliser une seule tâche avec plusieurs instructions, c'est très agréable de les écrire sur une même ligne. L'inconvénient, c'est qu'on perd beaucoup en lisibilité donc en compréhension.

• Et comme d'ordinaire, les commentaires se placent en fin de ligne après le caractère `;`.

• un petit exemple •

L'exemple classique est l'insruction SWAP qui échange le contenu du premier niveau de la pile avec le second. Dans la foulée, testons si les deux premiers niveaux de la pile ne sont pas vides. Le programme est très simple :

```
"C=DAT1.A D1+5 A=DAT1.A
?A=0.A -> { D1-5 GOTO .FIN }
DAT1=C.A D1-5 DAT1=A.A
*FIN
RPL
@"
```

Sur la première ligne on charge l'adresse des objets placés sur les niveaux 1 et 2 de la pile. Sur la deuxième ligne on teste l'absence d'un objet sur le niveau 2 de la pile : si c'est le cas on saute à la fin du programme. Sinon on échange les deux premiers niveaux de la pile, en ligne trois. Et enfin on quitte.

Sur le même principe, amusez-vous à réaliser un ROT ou un UNROT, chose très simple quand on calque cet exemple.

• le mot de la fin •

Cette brève présentation donne un léger aperçu des capacités de MASD en assembleur. Dans le prochain numéro, nous verrons toujours avec MASD la structure d'une source qui contient des externals de l'assembleur mélangés, avec des exemples à la clef.

Philippe Pamart
[phpamart@nordnet.fr]