RunOldies

BEAUMONT Christophe

COLLABORATORS			
	TITLE : RunOldies		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	BEAUMONT Christophe	November 2, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

RunOldies

1.1 RunOldies Documentation

```
                          RunOldies v1.1
      ©1995  BEAUMONT Christophe
         Tous droits réservés

                                    \|/
                                    @ @
----------------------------------oOO-(_)-OOo---------------------------------
```


1.2 historique

Tout d'abord, l'idée de ce programme m'est venu car énormément de ↩
démos ne
fonctionnent pas sur mon 4040. Je programme des utilitaires mais j'aime me
détendre parfois en regardant des démos.

Hélas,je suis souvent frustré de voir que des créations récentes,certaines
étant même primées, soient programmées de manière impropre et sans respect
du système. En effet,beaucoup de "démomakers" programment sur des systèmes
de base (1200 sans Fast) sans se soucier d'une compatibilité ascendante ou
bien ont conservés certaines techniques ancestrales valables sur 1200 mais

totalement incompatibles avec le 4040 et donc certainement avec les futurs Amigas.

En dehors des démos, RunOldies permet aussi l'utilisation de vieux jeux et de vieux utilitaires.

Vous trouverez des exemples de sources ASM et des conseils pour programmer vos démos dans
Spécial Programmeurs
.

Après avoir cherché, dans le domaine publique, des utilitaires capables de résoudre mon problème de compatibilité, j'en ai trouvé deux mais qui ne me donnaient pas satisfaction;par une utilisation énervante ou leur manque de respect envers le système. Pour ne pas porter préjudice à leurs auteurs,je ne citerais pas publiquement leurs noms.

Ainsi, est né RunOldies!

La version 1.0 n'est pas publique,elle a servit aux bêtas-tests permettant ainsi d'apporter certaines modifications pour la version 1.1.

RunOldies v1.1 1772 octets Hunk 0 = Hunk_Code 1636 Octets
Hunk_Reloc32
Hunk 1 = Hunk_Data 48 Octets

RunOldies.guide 31092 octets

1.3 utilisation

Pour fonctionner, ce programme nécessite le kickstart 3.0 et supérieur. En effet, il ne trouve pas vraiment d'utilité pour des systèmes inférieurs.

C'est une commande CLI/SHELL dont l'utilisation est très simple. En effet, le seul paramètre est le nom du programme à lancer.

RunOldies <programme>

Personnellement, je l'utilise à partir d'un fichier AmigaGuide. Ce qui me permet de pouvoir lancer automatiquement mes démos à partir d'une base de données.

L'auteur dégage toute responsabilité quant à l'utilisation de RunOldies et ne pourra être tenu pour responsable en quelque manière que ce soit.

Que fait RunOldies ?

Tout simplement, il downgrade le système de manière transparente pour l'utilisateur de la manière suivante :

- Déplacement du VBR, si celui-ci se trouve en FAST, il est mis à l'adresse 0 afin de pouvoir exécuter les interruptions de niveau 0 à 7 sur 4040 et supérieurs, ainsi que certaines cartes accélératrices.

- modifie l'état du pointeur, si vous utilisez dans votre WB un pointeur en haute résolution vous remarquerez que les sprites de plus de 16 pixels ont des blancs. Donc, celui-ci est mis en mode ECS.

- annulation des bords, si vous utilisez un programme comme BBlank qui affiche des bords noirs vous remarquerez un décalage dans les premières lignes rasters d'une copperliste. Donc, les bords sont mis à leur état d'origine.

- annulation des caches CPU, en effet certaines routines 3D causent des bugs car elles supportent mal les caches. Il en est de même pour les démomakers qui utilisent une synchro processeur par boucle au lieu d'une synchro VBlank ou par les CIAs ce qui cause une exécution trop rapide des effets et des modules, ce qui rend la démo horrible et non-visualisable.
  Pour les 4040, il désactive le mode CopyBack.

- changement du mode d'écran, pour ceux qui utilisent des écrans en mode DBLPAL, MULTISCAN, EURO72 ... il n'est plus nécessaire de changer le mode d'écran en PAL, le programme s'en occupe.

Toutes ces modifications sont restaurées dans leur état initial en sortie.
(Quand les démos rendent la main, ce qui n'est pas toujours le cas).

1.4 diffusion

RunOldies v1.1 est un programme SHAREWARE.
Son utilisation vous donne l'obligation morale de verser 15FF à son auteur pour encourager son travail et l'inciter à développer une version complète ainsi que d'autres programmes.

Il peut être distribué librement à condition de :

- ne pas en tirer un quelconque profit
- de le distribuer dans son intégralité
- de ne pas modifier la documentation ou le contenu du programme

Le source ASM est disponible auprès de l'auteur.

1.5 remerciements

Je remercie sincèrement FRANCK Nicolas pour m'avoir aidé dans la tâche laborieuse du 'bêta-test' dans lequel nous avons testés environ plus de 1200 démos,intros...
Environ 90% des "Oldies" fonctionnent. Les 10% restant s'avèrent être des programmes utilisant des adressages absolus trop bas, de mauvais segments pour la localisation des données en mémoire, du code auto-modifiable ....

Je remercie aussi par avance tous les distributeurs,BBS,et associations

qui diffuseront ce programme.

1.6 contacter

Si vous désirez m'envoyer votre contribution, échanger des sources ou bien témoigner votre intérêt à mes créations, écrivez-moi à l'adresse suivante:

BEAUMONT Christophe
16, rue de l'Orme
67400 ILLKIRCH-GRAFFENSTADEN
FRANCE

Attention! Pour les retours des disks, les réponses au courrier, il faudra me joindre impérativement une enveloppe affranchie au taux correspondant à l'envoi ainsi que vos coordonnées.

Ceux qui possèdent un minitel, un modem, peuvent me contacter sur:

3615 RTEL en b.a.l HERETIC

1.7 vis_viva

VIS VIVA (A.D.D.I.)
Association Des Développeurs Indépendants

VIS VIVA est une association comprenant des programmeurs, des graphistes, des musiciens, des scénaristes, et autres talents de l'informatique sur des machines comme l'Amiga de Commodore ou le PC.

Cette association a pour but de réaliser des logiciels de très bonne facture en vue de se faire dans le monde des logiciels un nom et une renommée de créateurs de qualité. Les logiciels que nous sortirons ainsi que les projets en cours sont soumis à des critères de qualité strictes afin d'éviter de sortir un logiciel ayant des lacunes évidentes.Par exemple un jeu ayant un très bon scénario,mais dont les graphismes ou la musique ne suivent pas la qualité de celui-ci, ou bien qui n'exploitent pas les capacités de la machine employée.

C'est dans le but de créér un label de qualité et de s'imposer sur le marché des logiciels que cette association existe. Nous recherchons des talents nouveaux, disponibles, et surtout très motivés.

Si vous souhaitez adhérer à VIS VIVA et participer à ses buts, veuillez adresser un courrier ainsi qu' une disquette avec vos créations personnelles pour nous faire connaître vos capacités dans votre domaine à l'adresse suivante :

ASSOCIATION VIS VIVA
MORDA Jean-Jacques BEAUMONT Christophe

```
    35, rue de la Vacquerie   16 rue de l'Orme
    59279   LOON-PLAGE    67400 ILLKIRCH
    FRANCE        FRANCE

  Vous pouvez aussi  nous  renvoyer  ce petit questionnaire avec
  votre courrier (recopié ou imprimé et découpé) :

 ------------------------------------------------------------------------

 NOM : .......................        PRENOM : ........................

 ADRESSE : ..................................................................

 CODE POSTAL : ........    VILLE : ..........................................

 TEL : (...) .../.../.../... (facultatif)

 TYPE D'ORDINATEUR: .........................................................

 PROGRAMMEUR : O    LANGAGE(S) UTILISE(S) : ..............................

 SCENARISTE  : O    DOMAINE DE PREDILECTION : ............................

 GRAPHISTE   : O    THEMES PREFERENTIELS : ...............................

 MUSICIEN    : O    STYLE OU INFLUENCES : ................................

INFORMATIONS COMPLEMENTAIRES : ..........................................
..........................................................................
..........................................................................

 ------------------------------------------------------------------------

  Un questionnaire représentant une demande d'adhésion vous sera
  renvoyé ainsi qu' une circulaire détaillant le fonctionnement,
  les buts et les raisons d'être de l'association.

  En souhaitant recevoir  vos nombreuses candidatures, nous vous
  souhaitons  de continuer à bien vous amuser  sur cette sublime
  et merveilleuse machine qu'est l'ordinateur.


             Les membres de VIS VIVA
```

1.8 programmeurs

Quelques conseils

Lancement par le CLI/WB

Eviter les assigns

Le Vector Base Register

Les caches CPU

1.9 conseils

Afin de rendre vos démos compatibles,agréables,conviviales voici ↩ certains
conseils:

- Utilisez la routine de Startup qui gère les messages envoyés par le WorkBench afin de permettre à vos démos d'être exécutées par le CLI et le WorkBench. (voir Lancement par le CLI/WB Une démo qui n'est pas prévue pour être lancée du WorkBench ↩ aura les effets suivants: une défaillance du logiciel et la mémoire utilisée ne serra pas rendue, sans compter les conflits engendrés par la suite qui amènent la plupart du temps à une profonde méditation du Gourou ou à un blocage du système.

- La majorité des Amigaïstes ont maintenant un disque dur, alors arrêtez de faire des démos en trackloading afin qu'elles puissent être installables sur celui-ci. La plupart des personnes aiment pouvoir regarder une démo et passer sur une application par la suite sans avoir à réinitialiser le système.

- Pour ceux qui n'ont pas de Fast, n'oubliez pas de déclarez vos segments afin d'éviter que vos démos ne s'installent entièrement en Fast pour ceux qui en ont. Je rappelle que le processeur ne peut avoir accès à certaines données (graphisme,samples...) que s'ils se trouvent en chip.
  Par exemple commencez votre code par : 1) Section MonCode,Code
  Celà placera votre code en priorité dans la Fast et l'exécution de vos routines y gagneront en rapidité.
  Pour les graphismes, modules: 2) Section Datas,Data_C
  Vos graphismes,modules,écrans seront placés en Chip et le processeur y aura accès (sinon le résultat est catastrophique).

- Faîtes en sorte que vos démos rendent la main au WorkBench et libère correctement la mémoire. Les démos qui quittaient avec un reset étaient à la mode de 1987 à 1989 mais maintenant c'est considéré comme une carence au niveau connaissances. Il en est de même pour les démos utilsant par exemple 1Mo de Ram et qui ne les rendent pas en sortie, résultat il faut souvent réinitialiser pour la récupérer.

- Evitez l'adressage absolu et le code auto-modifiable. Celà fait d'ailleurs parti des conventions de feu Commodore car le 68040 notamment n'apprécie pas du tout; par conséquent les futures versions aussi.

- Evitez comme certaines MégaDémos de devoir faire des ASSIGN pour accéder à la suite de la démo. Le système possède toutes les facilités nécessaires pour éviter ce genre de désagrément.
  (voir

```
                Eviter les assigns
                 )
```

- Ne faites pas de synchro processeur à partir d'une boucle dans une routine. Il est pourtant logique que cette boucle ne sera pas exécutée à la même vitesse sur un 68000 et un 68060.
  Certaines démos utilisent ce procédé notamment pour les routines players; le résultat c'est un module affreux (on a l'impression d'écouter 'L'ouverture de Guillaume Tell' en 78 tours).
  Donc utilisez plutôt une synchro VBlank, un timing par les CIAs ou le timer.device.

- Pour ceux qui utilisent des fichiers de configuration pour leurs programmes; ce n'est pas la peine de surcharger le répertoire S: pour rien, utilisez plutôt ce truc génial que sont les toolstypes de l'icône. (voir Bible de l'amiga p.509)

Voilà, c'est ce qui me vient à l'esprit pour le moment. Je suis certain d'en avoir oublié mais cette partie réservée aux programmeurs sera mise dans chacune de mes créations et remise à jour. J'espère ainsi qu'elle parviendra à un maximum de programmeurs et contribuera à une plus grande compatibilité des démos à l'avenir.

Et si mes conseils vous ont permis de vous améliorer, de régler certains problèmes que le manque de livres ne peut résoudre, et bien faîtes moi une place dans vos greetings ou bien envoyez-moi un mot.

Un peu de publicité !

Si vous aimez l'ensemble des sources, explications et exemples donnés dans cet article; sachez que je suis en train de programmer le deuxième volet de CONNEXION (magazine sur disque réservé aux programmeurs).
Le numéro 1, qui était un numéro-test, est sorti en Juillet 1994 mais le numéro 2 sera totalement différent avec une interface 100% intuitive.
Vous y trouverez une rubrique dédiée à la programmation ASM pour les démos et le système avec explications, sources commentées ...
Il y aura aussi une rubrique réservée aux graphistes et musiciens désireux de faire connaître leur talent ainsi qu'une rubrique pour des articles qui ne concerneront que la programmation, les techniques de création musicale, graphique. Enfin bref que des articles sérieux et permettant à chacun de débuter, découvrir ou progresser dans son domaine.
Autant dire que les messages personnels, les ragots... n'y auront pas leur place.
Ce magazine ne sera disponible qu'en me le commandant directement. La date de sortie sera annoncée sur RTEL, dans certains magazines Amiga et par le biais de certains BBS.
Si vous désirez y contribuer en m'envoyant vos créations, oeuvres etc...
Regardez les modalités dans

```
                Me Contacter
                .
```

1.10 cli_wb

Vous avez déjà surement lancés des programmes ou démos par un icône et qui donne le message suivant lorsqu'il retourne au WB :

" Défaillance du logiciel ..."

Mais lorsque vous lancez ce programme par le CLI tout se passe bien; la mémoire est restaurée et aucun message n'apparait.
Et bien souvent cette erreur est causée par l'absence de gestion des messages envoyés par le Workbench à votre programme.

Je vous conseille donc de vous référer à la deuxième édition de la "Bible de l'Amiga" au chapitre 3.4 p.504. (Editions Micro-Application)
En effet, en vertu de la loi du 11 Mars 1957, artcicle 40, il ne m'est pas permis d'en reproduire, même partiellement, le contenu. Vous y trouverez de plus amples informations.

Voici un exemple personnel que j'utilise au début de tous mes programmes. Il vous permettra de lancer correctement vos programmes du WorkBench ou bien du CLI/SHELL.

```
  INCDIR  "Sources:Includes/" ; Répertoire courant
  INCLUDE "exec/execbase.i" ; Charger les includes nécessaires
  INCLUDE "dos/dosextens.i" ; au programme

  ; Ici j'utilise un code de condition qui va me permettre de tester
  ; mes programmes à partir de l'assembleur. Si FINAL = 0, tout le
  ; code se trouvant entre IF et ENDIF ne sera pas assemblé donc il
  ; n'y aura pas de traitement de l'information.

FINAL = 0 ; =0 pour tester à partir de l'assembleur
    ; =1 pour l'assemblage final

  Section Startup,Code_P    ; Le segment sera placé en mémoire
          ; publique

 IF FINAL = 1       ; Condition d'assemblage

Main: lea Startup_Base(pc),a5 ; Pointe sur buffer
  move.l  $4.w,_StartSysBase(a5)  ; Adresse de exec dans buffer

  ; Tout d'abord on va récupérer l'adresse de la tâche de notre
  ; programme et on va tester sa provenance.

  move.l  _StartSysBase(a5),a6  ; Adresse de base d'Exec.Library
  move.l  ThisTask(a6),a4   ; Structure de la tâche dans A4
  tst.l pr_CLI(a4)    ; Lancement effectue par le CLI?
  bne.s Start     ; Si = 0 c'est par le Workbench

  ; Ici, le lancement a été effectué du WorkBench donc on va prendre
  ; en compte le message qui est envoyé au programme.

From_WBench:
  move.l  _StartSysBase(a5),a6  ; Adresse de base d'Exec.Library
  lea pr_MsgPort(a4),a0 ; Pointe sur structure message du port
  CALL  WaitPort    ; Attendre la réception d'un message
  lea pr_MsgPort(a4),a0 ; Pointe sur structure message du port
  CALL  GetMsg      ; Prendre en compte le message
```

```
  move.l  d0,_WBench_Msg(a5)  ; Sauve adresse du message

Start:  movem.l d0-d7/a0-a6,-(sp) ; Par sécurité, on sauvegarde tous
          ; les registres dans la pile

  bsr ...     ; là on saute à notre programme qui
          ; devra donc se finir par un RTS

  movem.l (sp)+,d0-d7/a0-a6 ; on restaure l'ensemble des registres

  ; Maintenant on va signaler que l'on a pris en compte le message
  ; envoyé par le WorkBench

Quit: move.l  _WBench_Msg(a5),d0  ; Place adresse du message dans D0
  beq.s Exit_To_Dos   ; Si = 0, pas de message car -> CLI
  move.l  _StartSysBase(a5),a6  ; Adresse d'Exec.Library
  CALL  Forbid      ; Interdit le multi-tâches
  move.l  _WBench_Msg(a5),a1  ; Pointe sur la structure message
  CALL  ReplyMsg    ; Envoie message de retour au Reply-Port

Exit_To_Dos:
  moveq.l #0,d0     ; D0 = 0
  rts       ; Et retour au WorkBench

  RSRESET

_StartSysBase:  rs.l  1
_WBench_Msg:  rs.l  1
Startup_Size: rs.w  0
Startup_Base: ds.b  Startup_Size  ; Réservation du buffer
 ENDIF          ; Fin de condition
```

1.11 assigns

Certaines mégadémos (ILYAD, PREY...) et aussi des programmes commerciaux nécessitent l'utilisation de la commande ASSIGN afin de pouvoir s'exécuter normalement en ayant accès à ses fichiers.
Bien que ces assignations peuvent être éffacée grâce à l'option REMOVE de la commande ASSIGN, elles n'en sont pas moins encombrantes et utilisent de la mémoire.

Pour éviter ceci, il existe 2 méthodes:

1) Lancement par icône grâce auquel on va récupérer les arguments transmis, trouver la structure Lock et rendre le répertoire courant. A partir de ce moment, tout fichier chargé et ne possédant pas un chemin complet le sera à partir du répertoire courant.
   ex: vous avez la démo XYZ qui se trouve dans le répertoire DATAS:DEMOS/MEGADEMOS/AGA/ et qui nécessite le fichier XYZ.datas et bien il vous suffira dans votre démo de ne donner que le nom du fichier à charger et celui-ci le sera directement à partir du répertoire dans lequel se trouve votre démo.
   Les avantages sont une économie d'octets dans votre démo, et la possibilité de la déplacer dans n'importe quel coin de votre

disque dur sans avoir à modifier des assignations. La seule condition, bien-sûr, c'est que vos datas se trouvent dans le même répertoire.
Maintenant si vous avez besoin d'un fichier de configuration se trouvant par exemple dans le répertoire S et bien vous n'aurez qu'à définir comme paramètres de chargement "S:Config_File" sans que celà modifie le répertoire courant qui sera toujours DATAS:DEMOS/MEGADEMOS/AGA/.

2) Pour un lancement effectué par le CLI, il est aussi possible de récupérer la structure lock et de désigner le répertoire dans lequel a été lancé votre programme comme répertoire courant. Le principe reste ensuite le même que celui pour le WB.

Je vais reprendre l'exemple de lancement par le CLI/WB pour montrer la méthode à utiliser.

```
FINAL = 0 ; =0 pour tester à partir de l'assembleur
    ; =1 pour l'assemblage final

  Section Startup,Code_P    ; Le segment sera placé en mémoire
          ; publique

 IF FINAL = 1       ; Condition d'assemblage

Main: lea Startup_Base(pc),a5 ; Pointe sur buffer
  move.l  $4.w,_StartSysBase(a5)  ; Adresse de exec dans buffer

  move.l  _StartSysBase(a5),a6  ; Adresse de base d'Exec.Library
  move.l  ThisTask(a6),a4   ; Structure de la tâche dans A4
  tst.l pr_CLI(a4)    ; Lancement effectue par le CLI?
  bne.s From_CLI    ; Si = 0 c'est par le Workbench

From_WBench:
  move.l  _StartSysBase(a5),a6  ; Adresse de base d'Exec.Library
  lea pr_MsgPort(a4),a0 ; Pointe sur structure message du port
  CALL  WaitPort    ; Attendre la réception d'un message
  lea pr_MsgPort(a4),a0 ; Pointe sur structure message du port
  CALL  GetMsg      ; Prendre en compte le message
  move.l  d0,_WBench_Msg(a5)  ; Sauve adresse du message

  ; Pour un lancement par icône on va récupérer le message envoyé au
  ; programme, pointer les arguments transmis, récupérer l'adresse
  ; de la structure Lock dans laquelle se trouve la description du
  ; répertoire et déclarer celui-ci comme répertoire courant.

  move.l  d0,a3     ; Placer le message en A3
  move.l  sm_ArgList(a3),a3 ; Pointer sur la liste des arguments
  move.l  wa_Lock(a3),d1    ; Récupérer structure lock
  CALLB _DosBase,CurrentDir ; Transformer en répertoire courant
  bra Start     ; On va plus loin

  ; Pour un lancement par le CLI, on va trouver la structure lock
  ; en ne mettant aucun paramètre dans D1 pour CurrentDir et celui-ci
  ; retournera en D0 l'adresse de l'ancienne structure lock qui est
```

```
  ; celle d'où à été lancé le programme. On va faire une copie du
  ; "shared-read-lock" qui autorisera la lecture dans le fichier à
  ; partir de plusieurs programmes et désigner le répertoire courant
  ; à partir du Lock récupéré.

From_CLI:
  moveq #0,d1     ; Pas de nouveau lock
  CALLB _DosBase,CurrentDir ; Transformer en répertoire courant
  move.l  d0,d1     ; Lock de l'ancien répertoire
  CALL  DupLock     ; Copier un shared-read-lock
  move.l  d0,d1     ; Nouveau lock dans D1
  CALL  CurrentDir    ; Transformer en répertoire courant
  move.l  d0,LockBase(a5)   ; Sauver adresse de l'ancien lock

Start:  movem.l d0-d7/a0-a6,-(sp) ; Par sécurité, on sauvegarde tous
          ; les registres dans la pile

  bsr ...     ; là on saute à notre programme qui
          ; devra donc se finir par un RTS

  movem.l (sp)+,d0-d7/a0-a6 ; on restaure l'ensemble des registres

  ; Ici par exemple, nous allons libérer notre répertoire courant et
  ; restaurer l'ancien afin de remettre le système dans son état
  ; initial.

  move.l  LockBase(a5),d1   ; Ancienne structure lock
  CALLB _DosBase,UnLock   ; Dévérouiller le répertoire courant

Quit: move.l  _WBench_Msg(a5),d0  ; Place adresse du message dans D0
  beq.s Exit_To_Dos   ; Si = 0, pas de message car -> CLI
  move.l  _StartSysBase(a5),a6  ; Adresse d'Exec.Library
  CALL  Forbid      ; Interdit le multi-tâches
  move.l  _WBench_Msg(a5),a1  ; Pointe sur la structure message
  CALL  ReplyMsg    ; Envoie message de retour au Reply-Port

Exit_To_Dos:
  moveq.l #0,d0     ; D0 = 0
  rts       ; Et retour au WorkBench

  RSRESET

_StartSysBase:  rs.l  1
_WBench_Msg:  rs.l  1
LockBase: rs.l  1
Startup_Size: rs.w  0
Startup_Base: ds.b  Startup_Size  ; Réservation du buffer
 ENDIF          ; Fin de condition
```

1.12 vbr

Qu'est ce que le VBR ?

Pour le traitement des interruptions et exceptions, ainsi que pour initialiser le compteur programme et le pointeur de pile, le 680x0 utilise

une zone de mémoire particulière appelée la table des vecteurs d'interruptions ou d'exceptions, située à partir de l'adresse 0. Dans de nombreuses applications, il peut être souhaitable de déplacer cette table à une autre adresse afin de pouvoir disposer de plusieurs jeux de vecteurs d'interruptions ou d'exceptions. Ce déplacement est possible grâce au VBR (Vector Base Register) dont le contenu n'est autre que l'adresse du premier vecteur de la table.

Pour savoir à quelle adresse se trouve le VBR, vous pouvez utiliser CPU-Control de Martin Berndt ainsi que ASM-One de TFA dans lequel le VBR est affiché dans le contrôle des statuts (Shift-X <return>).

Pour être concret, voici un exemple !

Beaucoup de démomakers utilisent le vecteur niveau 3 ($6C) pour leurs interruptions de la manière suivante :

```
 move.l  #MyInterrupt,$6C
```

Bien que cette méthode ne soit pas "propre", elle est cependant valable à condition que le VBR soit à 0. C'est le cas pour les A500, A600, A2000, A3000, A4030 mais pas sur le 4040 et supérieur, ainsi que certaines cartes accélératrices.
Le VBR se trouvant ailleurs , la plupart du temps en FAST, les routines sous interruptions de niveau 0 à 7 ne seront pas exécutées car il ne sera pas possible de trouver l'adresse de l'interruption dans la table des vecteurs.
Il est donc nécessaire de placer l'adresse de l'interruption à partir de l'adresse du VBR.
Cette méthode permettra à vos démos d'être compatibles avec tous les 680x0 actuels et futurs.

```
  ; On va considérer que A5 pointera en permanence sur le buffer
  ; et A4 sur l'adresse de base des Customs Chips.

Start:  lea Buffer_Base(pc),a5  ; Pointe sur adresse du buffer
  move.l  $4.w,SysBase(a5)  ; exec.library dans buffer
  lea $DFF000,a4    ; Adresse des customs-chips

  ...

  ; On va obtenir l'adresse de la table des vecteurs et la sauver
  ; dans un buffer par exemple.

  jsr GetVBR(pc)    ; Chercher VBR
  move.l  d0,VBRBase(a5)    ; Sauver résultat dans buffer

  ...

  ; On va placer l'adresse de la table des vecteurs dans A0 et
  ; l'utiliser comme un adressage indirect par registre avec
  ; déplacement. Par exemple, pour installer une interruption niveau
  ; 3 vous procéderez ainsi:

  move.w  #$7fff,$9a(a4)    ; Bloquer toutes les IT
```

```
  move.l  VBRBase(a5),a0    ; Adresse de la table des vecteurs
  move.l  $6c(a0),OldLevel3(a5) ; Sauver état vecteur niveau 3
  move.l  #MyInterrupt,$6c(a0)  ; Placer nouvelle interruption
  move.w  #$c020,$9a(a4)    ; Interruption VBlank autorisée

  ...

  ; On place l'adresse de la table des vecteurs dans A0 et on remet
  ; le vecteur niveau 3 dans son état original.

  move.w  #$7fff,$9a(a4)    ; Bloquer toutes les IT
  move.l  VBRBase(a5),a0    ; Adresse de la table des vecteurs
  move.l  OldLevel3(a5),$6c(a0) ; restaurer vecteur niveau 3
  ...

  ; Ici, on va tester s'il y a un CPU supérieur au 68000 car celui
  ; ci ne possède pas de VBR. Pour celà on va simplement tester le
  ; bit AFB_68010 car ci celui-ci est égal à un, c'est que le CPU
  ; est un 68010 ou supérieur. En effet, admettons que vous ayez
  ; un 68030 les résultats seront : AFB_68040 = 0
  ;           AFB_68030 = 1
  ;           AFB_68020 = 1
  ;           AFB_68010 = 1

MyInterrupt:
  movem.l d0-d7/a0-a6,-(sp) ; Sauvegarder les registres
  jsr Routine1(pc)    ; Saut à sous-routine
  jsr Routine2(pc)    ; Saut à sous-routine
  move.w  #$20,$9c(a4)    ; Demande interruption VBlank
  movem.l (sp)+,d0-d7/a0-a6 ; Restaurer les registres
  rte

GetVBR: move.l  a5,-(sp)    ; Sauver état de A5 dans la pile
  moveq #0,d0     ; D0 = 0
  move.l  SysBase(a5),a6    ; exec.library dans A6
  btst  #AFB_68010,AttnFlags+1(a6) ; 68010 ou supérieur ?
  beq .1      ; =0 -> 68000, on ne fait rien
  lea VBR_Exception(pc),a5  ; pointer sur routine d'exception
  CALL  Supervisor    ; passer en mode superviseur
.1: move.l  (sp)+,a5    ; restaurer état de A5
  rts       ; Quitter sous-routine

  ; movec vbr,Xn est une instruction privée. Vous devez donc être en
  ; mode Superviseur pour l'exécuter.
  ; Si votre assembleur ne reconnait pas l'instruction movec vbr,Xn
  ; remplacez là par dc.w $4e7a,$0801

VBR_Exception:
  movec vbr,d0      ; Adresse du VBR dans D0
  rte       ; Retour d'exception

  RSRESET

SysBase:  rs.l 1
VBRBase:  rs.l 1
OldLevel3:  rs.l 1
Buffer_Size:  rs.w 0
```

```
Buffer_Base:  ds.b Buffer_Size
```

1.13 cpu

```
Si vous souhaitez désactiver les caches utilisés par l'amiga comme
Instruction Cache, Data Cache, Instruction Burst, Data Burst, CopyBack,
il existe une méthode particulièrement simple. (C'est dans ces cas là
que l'on se rend compte que l'amiga OS est vraiment génial et se mettre à
apprendre le système est un véritable plaisir).


- Instruction Cache OFF

 moveq #0,d0
 move.l  d0,d1
 bset  #CACRB_EnableI,d1
 CALLB _SysBase,CacheControl

- Instruction Cache ON

 moveq #0,d0
 bset  #CACRB_EnableI,d0
 move.l  d0,d1
 CALLB _SysBase,CacheControl

- Data Cache OFF

 moveq #0,d0
 move.l  d0,d1
 bset  #CACRB_EnableD,d1
 CALLB _SysBase,CacheControl

- Data Cache ON

 moveq #0,d0
 bset  #CACRB_EnableD,d0
 move.l  d0,d1
 CALLB _SysBase,CacheControl

- Instruction Burst OFF

 moveq #0,d0
 move.l  d0,d1
 bset  #CACRB_IBE,d1
 CALLB _SysBase,CacheControl

- Instruction Burst ON

 moveq #0,d0
 bset  #CACRB_IBE,d0
 move.l  d0,d1
 CALLB _SysBase,CacheControl

- Data Burst OFF
```

```
 moveq #0,d0
 move.l  d0,d1
 bset  #CACRB_DBE,d1
 CALLB _SysBase,CacheControl

- Data Burst ON

 moveq #0,d0
 bset  #CACRB_DBE,d0
 move.l  d0,d1
 CALLB _SysBase,CacheControl

- CopyBack OFF

 moveq #0,d0
 move.l  d0,d1
 bset  #CACRB_CopyBack,d1
 CALLB _SysBase,CacheControl

- CopyBack ON

 moveq #0,d0
 bset  #CACRB_CopyBack,d0
 move.l  d0,d1
 CALLB _SysBase,CacheControl
```