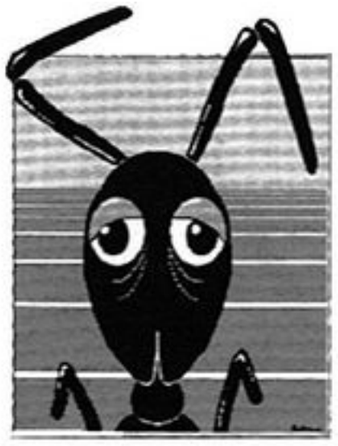
LE MAGAZINE DES PROGRAMMEURS SUR AMIGA - N°23 JUIN 1991

AMIGA NEWS-TECH

Dessin de couverture 24 bit au format TIFF (Mac) tiré d'un CD-Rom (Rick Doyle édité par Gazelle Technologies Inc), converti au format IFF à l'aide de ImageLink puis traité en 16 couleurs à l'aide de TAD. Voir Amiga Revue n°34 de Juin.

Ce numéro 23 de l'Amiga News-Tech a été tiré à 3000 exemplaires (2804 abonnés enregistrés le 10.05.91).
Ce magazine est une publication Commodore Revue Sarl. 5, rue de la Fidélité - 75010 PARIS.



EDITO

Voilà déjà quatre mois que nous sommes ensemble. L'heure des premiers renouvellements d'abonnement à sonné. Je ne vous ferai pas l'affront de vous dire que nous espérons tous, ici à la rédaction, que vous serez nombreux à renvoyer le bon prévu à cet effet. Cela nous confortera dans notre idée qu'un journal entièrement dédié à la programmation sur l'Amiga manquait réellement, et surtout que l'Amiga NewsTech EST ce journal.
Je profite juste du peu de place qui me reste, pour vous transmettre les excuses de Jerôme Etienne, qui n'a pas pu terminer son article sur la 3D face cachées, à cause d'une panne de son Amiga. Cet article devrait paraître dans le prochain numéro si les dieux sont avec nous. En remplacement, le Coin des Démo-Makers vous propose une petite routine de rasters en mouvement.

Maintenant, j'aimerais avoir votre avis sur un projet que nous avons pour augmenter le volume d'articles dans l'ANT.
Etant donné qu'il nous est impossible, financièrement parlant, d'ajouter des pages, nous avons pensé à changer la formule d'abonnement : considérant que presque la moitié du journal est occupée par du listing, on peut gagner de la place en ne publiant plus que les articles, les listings étant, eux, sur la disquette (comme le fait d'ailleurs Amiga World).

Nous sommes toutefois conscients que ce serait pour certains un manque important, pour d'autres un investissement assez conséquent, et c'est pourquoi nous vous demandons votre avis. Répondez-nous nombreux par courrier ou par Minitel (3615 code ANT, BAL ANT) afin que nous puissions juger du bon aloi de cette proposition.

Vivement les vacances !

Stéphane Schreiber

OURS

L'Amiga NewsTech est une publication de Commodore Revue SARL, société au capital indéterminé sise au 5, rue de la Fidélité, 75010 Paris, France, Europe, Monde, Voie Lactée, Univers. Son numéro de téléphone est le
(1) 42 469 290

Le Directeur de la Publication est toujours Jean-Yves Primas. C'est près de lui qu'il faut gueuler si vous voulez des pages supplémentaires : son numéro à lui, c'est le
(1) 42 471 216
C'est juste au fond du couloir, mais il a sa ligne à lui tout seul, avec le téléphone sans fil, c'est plus classe.

Le Rédacteur en Chef, Monsieur Crapiton père, s'appelle toujours Yves Huitric. Il s'occupe également de la PAO sur son Amiga 2000 (équipé tout de même d'une carte 68030 et d'un Flicker Fixer) avec Professional Page v2.0, dont il attend impatiemment la version française, avis à Franck..

Le Rédacteur en Chef Adjoint, c'est Stéphane Schreiber. C'est lui qu'il faudra appeler au numéro de la rédaction si d'aventure vous aviez des questions à poser, des suggestions à faire, des articles à proposer, une soeur à marier..

Les malades qui ont collaboré à ce numéro sont (dans l'odre d'apparition à l'écran) : Denis Jarril, François Lionet, François Gueugnon, Frédéric Mazué, Jerôme Etienne, Loïc Far, Pascal Amiable, Max et votre serviteur (grand jeu gratuit : devinez qui je suis et gagnez ma considération ée).

Les abonnements sont gérés par la société MCM Europe, dont les sièges tout court et social sont au 1, Quai Jean-Baptiste Clément, 94140 Alfortville. Téléphone :
(1) 43 987 210
C'est eux qu'il faut appeler si le chien du facteur à bouffé le journal et pissé sur la disquette.

Professional Page sortant des fichiers PostScript, il nous est facile de donner nos disquettes à monsieur Color Way, qui se fait un plaisir de flasher nos pages directement sur sa rutilante Lino. C'est d'autant plus pratique qu'il habite à la même adresse que nous : 5, rue de la Fidélité, 75010 Paris. Téléphone :
(1) 40 708 787

Quant à l'imprimeur, il s'appelle N.I.C, habite Reims, travaille vite et bien, et est joignable au
(16) 26 075 787

Voilà. Vous saurez tout quand je vous aurai dit que l'ANT et Commodore Revue SARL sont totalement indépendante de la société Commodore France, que notre numéro de commision paritaire est "en cours", que l'ANT n'est pas membre inscrit à l'ODJ, que notre mascotte est la fourmi Denise, que notre porte-bonheur est un charmant bambin du nom d'Axel et que je fais une grosse bise à Gaëlle, Manon et Carole.

ExecBase : BON

Lorsque l'on a en projet de développer un gros programme en C, il est souvent utile d'y ajouter quelques routines en assembleur, histoire de gagner du temps et/ou de la place, ou bien tout simplement parce qu'on ne peut pas faire autrement.

Ce cas est de loin le plus facile, l'interface avec l'assembleur étant tout naturel en langage C. A peine aura-t-on besoin de se souvenir de quelle manière les paramètres sont transmis (ce qui a d'ailleurs déjà été expliqué dans le précédent numéro de l'ANT, dans cette même rubrique, qui traitait alors de l'utilisation de l'amiga.lib en assembleur). Le cas inverse (à savoir : appeler des routines écrites en C ou accéder à des données définies dans ce langage) n'est pas beaucoup plus difficile, pour peu que l'on prenne la peine de s'y intéresser.

BREF RAPPEL

A l'attention de ceux qui ne nous rejoindraient qu'à partir de ce numéro, un bref mais court rappel de ce qu'il faut faire pour appeler une routine assembleur depuis le C ne mangera pas de pain.

Par définition, le langage C passe tous les paramètres des fonctions par la pile, empilés en ordre inverse de leur déclaration. Pour un "long" ou un pointeur, un mot long (32 bits) est empilé, tandis que pour un "int" (16 bits, certains préfèrent dire "short" ou "word") ou un "char" (8 bits), c'est un mot qui est empilé. Quant à la valeur de retour, s'il y a en une, elle est contenue dans le registre d0 au sortir de la fonction. Quand on sait ça, la suite n'est plus qu'un jeu d'enfant.

Voici tout de même un petit exemple, comme ça, pour la forme.

En C :

```
/* extrait d'un programme C appelant une routine assembleur
*/
...
/* déclaration de la fonction à appeler */
extern USHORT RoutineAsm(ULONG Par1, USHORT Par2);

void main(int argc, char **argv)
{
ULONG a;
USHORT b, c;
        ...
        c = RoutineAsm(a, b);
        ...
}
```

Et en assembleur :

```
* Routine assembleur appelé depuis le C

        XDEF    _RoutineAsm     ; Exporte le label pour le
linker

; structure de la pile à l'appel de la fonction :
;  (sp) -> adresse de retour (RTS)
; 4(sp) -> paramètre "Par2" (USHORT=16 bits)
; 6(sp) -> paramètre "Par1" (ULONG =32 bits)

_RoutineAsm:
        move.w  4(sp),d0 ; d0=Par2=b
        move.l  6(sp),d1 ; d1=Par1=a
        ...              ; traitement de la fonction
```

```
        move.w  d1,d0    ; d0.w=valeur de retour (USHORT)
        rts
```

Quelque remarques qui n'ont pas été faites précédemment.
Côté C, la fonction à appeler doit être déclarée "extern" pour que le linker sache qu'il la trouvera ailleurs que dans le progamme compilé. D'autre part, il est de bon ton de lui déclarer un prototype, histoire d'être sûr de ne pas se gourrer dans les paramètres...

Côté assembleur, plusieurs petites choses sont à prendre en compte. D'abord, le source doit évidemment être assemblé en objet et non en exécutable (n'importe quel macro-assembleur digne de ce nom fait celà très bien), ça tombe sous le sens.

De plus, le point d'entrée de la fonction doit être "exporté" grâce à la directive XDEF, sinon le linker se plaindra de ne pas trouver ce qu'il cherche. Ensuite, à cause d'une convention du C, son nom doit être précédé du caractère souligné (_). Enfin, c'est la fonction appelante qui a la charge de restaurer la pile, et non la fonction appelée, comme je l'ai encore vu récemment dans un magazine que je ne citerai pas mais qui parle beaucoup des 16/32 bits...

Permettez-moi d'ouvrir ici une parenthèse utile : il est tout-à-fait possible qu'une même routine assembleur soit appelée à la fois par le C et par une autre routine assembleur. Que faire dans ce cas ? Il serait en effet superflu - et en tout bien ennuyeux - de devoir empiler les paramètres et réajuster soi-même la pile alors que les registres du processeur sont tout de même faits pour ça ! La petite "astuce" (et encore !) suivante y remédie, en utilisant justement le coup du souligné :

```
        XDEF    _RoutineAppelableDePartout              ;
pour le linker

_RoutineAppelableDePartout:
        move.l  4(sp),d0 ; paramètre 1
        move.l  8(sp),d1 ; paramètre 2
RoutineAppelableDePartout:
        ...              ; traitement normal
        ...              ; de la routine
        rts              ; et retour à l'envoyeur
```

ainsi, cette routine pourra être appelé depuis le C par :

```
extern VOID RoutineAppelableDePartout(ULONG, ULONG);

        RoutineAppelableDePartout(a, b);
```

et depuis l'assembleur par :

```
        move.l  #a,d0
        move.l  #b,d1
        jsr     RoutineAppelableDePartout       ;  sans  le
souligné !
```

Sympathique, non ?

DU C DANS L'ASSEMBLEUR

Voyons maintenant le cas inverse. Il n'est pas rare qu'une routine assembleur appelée par une fonction C ait à son tour besoin d'appeler une autre fonction C et/ou d'accéder à des données globales du programme, elles-mêmes définies en C... Ce qui, encore une fois, n'est pas si terrible que ça en a l'air.

Il faut évidemment se rappeler la convention du passage des paramètres, mais également ne pas oublier que la pile DOIT être restaurée dans son état initial. Vite, vite, un exemple.

En C :

```
/* extrait de programme appelant et
appelé par une routine assembleur */

extern VOID RoutineAsm(VOID)      /*    déclaration    de    la
routine Asm */
VOID RoutineC(ULONG);    /* prototypes des fonctions C */
...

VOID main(int argc, char **argv)
{
        ...
        RoutineAsm();
        ...
}

VOID Routine(ULONG a)
{
        ...      /* On peut faire ce qu'on veut ! */
}
```

Et en assembleur :

```
        XDEF    _RoutineAsm      ; Exporte ce label
        XREF    _RoutineC        ; Importe ce label

_RoutineAsm:                ; Pas de paramètres ici
...                 ; traitement normal de la routine
move.l  d2,-(sp) ; Empile le paramètre 'a' (ULONG)
jsr     _RoutineC        ; Saut à la routine en C
addq.l  #4,sp    ; Corrige la pile (ULONG=4 octets)
...                 ; Suite de la routine
rts                 ; Retour au C
```

Jusque là, rien de vraiment compliqué : on s'est contenté d'utiliser la directive XREF pour indiquer à l'assembleur que le label _RoutineC (sans oublier le souligné) se trouve dans un autre module, extérieur à celui qu'il traite. Enfin, pour appeler la fonction, on a empilé le paramètre (32 bits), sauté avec JSR puis additionné à SP le nombre d'octets empilé, c'est-à-dire quatre. Enfantin.

Accéder aux données globales du programme C est tout aussi simple, bien qu'il faille différencier deux cas :
- le programme est compilé en modèle SMALL ;
- le programme est compilé en modèle LARGE.

Dans le premier cas (SMALL), les données gloables sont référencées par l'intermédiaire du registre A4 (et seulement par lui !). On aura par exemple :

```
        move.l    _XXX(a4),a0
```

Ceci limite les données à 32 Ko - à cause de la manière dont le 68000 code ce mode d'adressage - mais produit un code plus compact et plus rapide.

Dans le second cas (LARGE), les données gloables sont adressées directement par leur adresse effective :

```
        move.l    _XXX,a0
```

Là, on n'est plus limité (?) qu'à 16 Mo de données, mais le code produit est plus gros et moins rapide. C'est une question de choix.

Vérifiez donc dans le manuel de votre compilateur quel type de code il produit (normalement, il doit savoir faire les deux !), mais si vous me permettez un conseil, utilisez toujours le modèle SMALL : 32 Ko de données, c'est déjà énorme ! Et si d'aventure vous en aviez besoin de plus, allouez-les dynamiquement avec AllocMem(), c'est beaucoup plus rentable !

par Denis Jarril

LANGAGE AMOS : AMT 4ème

Comme promis le mois dernier, voici le format des banques de musique. Accrochez-vous, car c'est assez complexe ! La place me manquera ce mois pour vous proposer un programme permettant de mettre plusieurs musiques dans une seule banque. Pour le mois prochain, c'est promis.

Les banques de musique sont entièrement relogeables. De plus, elles sont structurées en trois grandes parties, totalement indépendantes : les instruments ; les musiques, qui ne sont qu'une suite des numéros des patterns à jouer ; les patterns, contenant les notes. Comme d'habitude, je vais vous décrire le format sous la forme d'un listing assembleur :

```
* Début de la banque :
    dc.b    "Music   "      * 8 caractères
    Start:  dc.l    Instruments-Start
    dc.l    Musiques-Start
    dc.l    Patterns-Start
    dc.l    0               * Libre pour plus tard !
* Partie contenant tous les instruments de la banque.
Instruments:
    dc.w    Nombre_d'instruments

* Pour chacun des instruments (".inst"=numéro de l'instrument)
    REPT    Nombre_d'instruments
* Adresse de la partie d'attaque des échantillons
    dc.l    Attaque.inst-Instruments
* Adresse de la partie boucle. si pas de boucle, pointe sur un petit
* sample nul avant tous les autres...
    dc.l    Boucle.inst-Instruments
* La longueur des échantillons est exprimée en mots
* (pour être toute prête à poker dans les circuits...
    dc.w    Longueur_Attaque.inst
    dc.w    Longueur_Boucle.inst
* Niveau de volume individuel de l'instrument
    dc.w    Volume
    dc.w    Longueur_Totale.inst
* En Ascii tout bête:
    dc.b    Nom_de_l'instrument_sur_16_octets
    ENDR

* Vient maintenant l'échantillon vide que pointent les "boucles" vides...
    dc.w    0,0

* Ensuite, à la queue-leu-leu, les échantillons des différents instruments...
    REPT    Nombre_d'instruments
Attaque.inst:
    dcb.b   Sample....
* Si elle est définie:
Boucle.inst:
    dcb.b   Sample....
    ENDR

* Maintenant, la partie contenant les musiques...
* ".mus" est le numéro de la musique.
Musiques:
    dc.w    Nombre_de_musiques

    REPT    Nombre_de_musiques
    dc.l    Musique.mus-Musiques
    ENDR

    REPT    Nombre_de_musiques
Musique.mus:
    dc.w    Tempo
    dc.w    Liste_patterns_voix_0 - Musique.mus
    dc.w    Liste_patterns_voix_1 - Musique.mus
    dc.w    Liste_patterns_voix_2 - Musique.mus
    dc.w    Liste_patterns-voix_3 - Musique.mus
    dc.w    0               * Pour extension...

* Suit maintenant, pour chacune des voix séparément, les numéros des patterns à jouer...
Liste_patterns_voix_0:
    dc.w    ,,,,,
Liste_patterns_voix_1:
    dc.w    ,,,,,
Liste_patterns_voix_2:
    dc.w    ,,,,,
Liste_patterns_voix_3:
    dc.w    ,,,,,
    ENDR

* La dernière partie à voir contient les définitions des patterns. Dans ce qui
* suit, ".pat" contient le numéro du pattern. Je ne vais pas  vous décrire le
* format interne de chaque pattern, ce qui nous entraînerait trop loin...
Patterns:
    dc.w    Nombre_de_patterns

    REPT    Nombre_de_patterns
    dc.w    Liste_des_notes_voix_0.pat - Patterns
    dc.w    Liste_des_notes_voix_1.pat - Patterns
    dc.w    Liste_des_notes_voix_2.pat - Patterns
    dc.w    Liste_des_notes_voix_3.pat - Patterns
    ENDR

* Et maintenant, toutes les listes de notes, à la suite...
    REPT    Nombre_de_patterns
Liste_de_notes_voix_0.pat:
    dc.w    ,,,,,
Liste_de_notes_voix_1.pat:
    dc.w    ,,,,,
Liste_de_notes_voix_2.pat
    dc.w    ,,,,,
Liste_de_notes_voix_3.pat
    dc.w    ,,,,,
    ENDR
```

Quelques remarques, comme elles me viennent :
- tout est relogeable : contrairement à SoundTracker, le player n'a pas besoin de modifier les adresses dans la banque avant de jouer la musique ;
- le nombre d'instruments est illimité (heu, si, à 65536 !) ;
- le nombre de patterns est lui aussi illimité (même remarque) ;
- avec un peu de programmation, on peut gagner beaucoup de place : plusieurs musiques peuvent partager les mêmes instruments, les mêmes patterns ;
- ce format de banque (et le format des patterns, dont je ne vous ai pas parlé) permet d'émuler plusieurs programmes musicaux ;
- il manque un réel éditeur exploitant les possibilités de ces banques (ne me le répétez pas, je le sais !).

FORMAT SUR DISQUE

Les banques de musique sont sauvées telles quelles sur la disquette, en rajoutant simplement le petit entête AMOS :

```
dc.b    "AmBk"
dc.w    Numéro_de_la_banque
dc.l    $80000000 + Longueur_de_la_banque
```

NB : S80000000 indique à AMOS de charger la banque en mémoire CHIP.

PETITE APPLICATION

Pour répondre à de très nombreuses demandes (au moins deux), voici comment savoir si une musique est terminée. Il existe une fonction non documentée dans le manuel, qui retourne l'adresse de la zone de données de l'extension musicale :

```
=MUBASE
```

A partir de cette adresse, on peut connaître beaucoup de choses, dont les compteurs de chacune des voix jouées. La procédure MUSTOPPED ci-jointe retourne la valeur 0 si la musique est arrêtée, et une valeur différente de 0 (l'addition des valeurs des 3 compteurs) lorsque qu'une musique est en cours. Vous devez bien sûr, pour en voir l'effet, charger une banque de musique en mode direct.

CIAO

Bien, c'est tout pour aujourd'hui. Je laisse la parole (enfin, le clavier) à mon chien. Autant vous prévenir tout de suite, elle est complètement mégalo, vous vous en rendrez vite compte. Elle va certainement vous dire que c'est elle qui a programmé AMOS, et qu'elle est en train de terminer le compilateur, etc. Je m'élève en faux contre toutes ces affirmations ! De toute façon, pour la punir, ce soir pas de Royal-Canin ! On verra bien qui des deux aura raison à la fin...

```
'-------------------------------------------------------------------
' Démonstration MUSTOPPED
'-------------------------------------------------------------------
' Load "Musique.Abk"
'
Do
    A$=Inkey$
    Exit If A$=" "
    If A$="0" : Music Off : End If
    If A$="1" : Music 1 : End If
    MUSTOPPED
    Home : Centre " "+Str$(Param)+" "
Loop
Music Off
Edit
'
Procedure MUSTOPPED
    A=Leek(Mubase+4)
    If A
        For V=0 To 3
            F=F+Peek(A+V*40+21)
        Next
    End If
End Proc[F]
```

DAISY : NOS AMIS LES BETES

Salut ! Daisy au clavier. Laissez moi me présenter : je suis le chien de l'individu innommable dont vous venez de terminer l'article. Merci à Stéphane Schreiber de me permettre de m'exprimer enfin (NDLR : de rien !).

J'appartiens donc à la gente canine, et suis une superbe femelle de race indéfinie mais néanmoins harmonieuse, d'un caractère agréable et constant, avec une préférence pour les pantalons bruns qui ont plus de goût... Le malheur a voulu que j'atterrisse chez mon maître actuel. Je ne sais ce qu'il a pu vous raconter, mais moi je vais vous dire la vérité vraie : je suis l'unique auteur de l'AMOS ! Hé oui, aussi incroyable que cela puisse paraître, François Lionet n'est qu'un imposteur !
Dès qu'il a découvert mes talents en programmation, il m'a attachée au clavier de l'Amiga et m'a obligée à produire 1 Ko de langage machine par jour ! Si je n'avais pas atteint ce quota, je n'avais pas droit à ma nourriture.
Pour éviter de sombrer dans la dépression nerveuse, j'ai entrepris la programmation d'un bon nombre de démOS dès qu'il avait le dOS tourné. Tous les mois, si je suis encore en vie, je viendrai vous en propOSer une.
Mes démOS ont en commun leur thème: l'OS, ce met délicieux que -hélas - je ne peux goûter très souvent !
Avant toute choses, allez donc voir le listing de notre première démOS, là, quelque part, ci-contre.
Pour faire fonctionner cette démOS, vous devez charger en mode direct une banque de musique. Choisissez-la de préférence plutôt rhytmée. Eteignez la lumière et fixez l'écran pendant quelques heures : vous aurez gagné un aller simple pour l'hôpital psychiatrique le plus proche.

Quelques mots sur cette superbe programmation.

Pour obtenir les effets de moirage, la démOS utilise une illusion d'optique que vous pouvez facilement recréer : dessinez des lignes parallèles proches sur une feuille de papier et sur un papier calque. Faites bouger la seconde feuille sur la première : les moires apparaissent aussitôt. On peut recréer un tel effet sur Amiga, en utilisant le DUAL-PLAYFIELD.
La première partie du programme ouvre deux écrans dual-playfield d'une taille bien supérieure à celle de l'affichage. Les écrans sont ensuite cachés (SCREEN HIDE) pour procéder au dessin des cercles. Un petit écran est ouvert pour faire patienter, et la musique est démarrée.
Les deux écrans ne sont en fait que des vu-mètres. Le scrolling est obtenu très facilement en modifiant les valeurs des offsets des écrans (CHANNEL TO SCREEN OFFSET). La voix 0 est affectée à la coordonnée en X de l'écran 0, la voix 1 à la coordonnée Y, les voix 2 et 3 aux coordonnées X et Y de l'écran 1. Tout le travail est effectué sous interruption grâce à deux chaînes AMAL.
Lesquelles chaînes AMAL ne diffèrent très peu : ne change que le numéro des voix dans l'instruction =Vu(). Plutôt que de réécrire la chaîne en entier, il vaut mieux rechercher les instructions Vu() dans la première chaîne (INSTR), et modifier les numérOS des voix avec l'instruction MIDS. Vous pourrez même automatiser le processus pour plus de deux canaux.

Une fois AMAL initialisé, le programme dessine dans l'un des écrans de nombreux cercles concentriques, puis la chOSe la plus importante de la démOS : l'OS ! Cette partie prend 23 secondes (le dessin de cercles n'est pas des plus rapides en amOS : j'ai du garder les routines de François).

L'animation est lancée par AMAL ON. Les deux écrans bougent alors automatiquement. Le programme n'a plus qu'à modifier les couleurs des écrans avec FADE, et à tester l'appui sur une touche du clavier.
Vous pouvez insérer cette démo dans une autre, et faire, par exemple, un scrolling de texte dans un troisième écran : l'animation étant faite par AMAL, vous pouvez continuer à travailler en amOS.

Bon, j'espère avoir remonté le niveau de la rubrique AMOS de l'ANT. Il faut quand même avouer qu'on voit clairement, en lisant mes programmes, qu'il n'y a aucune comparaison entre Lui et Moi ! Le mois prochain si tout va bien, je vous propOSerai une nouvelle démOS.

Améliorer le déroulement des démos

La plupart des démos ont une animation au 1/50 ième de seconde. Le moindre petit défaut dans le déroulement saute immédiatement aux yeux.

Les deux petites procédures qui suivent vont vous permettre de bloquer tous les autres programmes de l'Amiga (dont celui du lecteur de disquette) ; vous pourrez ainsi récupérer à votre profit toute la puissance de la machine.

ATTENTION : cette routine bloque TOUT l'Amiga, y compris le système de validation des disquettes. Vous ne devez pas appeler F0RBID avant d'être sûr que la lumière du lecteur de disquette est éteinte !

```
Procedure F0RBID
F=Execall(-132)
End Proc
Procedure PERMIT
F=Execall(-138)
End Proc
```

Appelez F0RBID au début de votre programme et rétablissez le multitache à la fin grâce à PERMIT. La seule entrée de votre programme sera alors la souris. Le clavier, les ports, TOUT restera figé !

Notes : F0RBID s'écrit avec un zéro à la place du "O", pour éviter que l'AMOS ne le prenne pour un For-Next...
Si vous faites un FORBID sur une machine plus complexe qu'un A500, je vous conseille fortement d'effectuer un RESET à la fin du programme : les auteurs du système de l'Amiga déconseillent fortement l'utilisation prolongée de cette instruction.

```
'-------------------------------------------
' Daisy-DemOS numero 1
'-------------------------------------------
' Par Daisy Lionet
'-------------------------------------------
'
' Initialisation des ecrans
Hide On
Screen Open 1,352,430,2,Lowres
Curs Off : Flash Off : Cls 0
Screen Open 0,352,430,2,Lowres
Curs Off : Flash Off : Cls 0
Screen Display 0,,30,,340
Dual Playfield 0,1
Screen Offset 0,352,64
Screen Offset 1,352,64
Colour 1,0 : Colour 9,0
'
Screen Open 2,320,8,2,0
Curs Off : Colour 1,$F40
Centre "... 23 secondes SVP ..."
'
View
Music 1
'
' Initialisation AMAL
Channel 0 To Screen Offset 0
Channel 1 To Screen Offset 1
'
A0$=A0$+"       Let  R3=-1;  Let  R4=-1;  Let
R5=3;"
A0$=A0$+"Loop: Let R2=Vu(0); If R2>0 Jump X"
A0$=A0$+"      If R0=0 Jump F"
A0$=A0$+"      Let R0=R0-R5; If R0>0 Jump F"
A0$=A0$+"      Let R0=0; Jump F;"
A0$=A0$+"X:    Let R0=R2;"
A0$=A0$+"      Let R3=-2*R3+R3;"
A0$=A0$+"F:    Let X=R3*R0/2*2+353;"
A0$=A0$+"      Let R2=Vu(1); If R2>0 Jump Y"
A0$=A0$+"      If R1=0 Jump G"
A0$=A0$+"      Let R1=R1-R5; If R1>0 Jump G"
A0$=A0$+"      Let R1=0; Jump G;"
A0$=A0$+"Y:    Let R1=R2;"
A0$=A0$+"      Let R4=-2*R4+R4"
A0$=A0$+"G:    Let Y=R4*R1+64;"
A0$=A0$+"      Pause; Jump Loop"
Amal 0,A0$
P=Instr(A0$,"Vu(0)") : Mid$(A0$,P)="Vu(2)"
P=Instr(A0$,"Vu(1)") : Mid$(A0$,P)="Vu(3)"
Amal 1,A0$
'
' Dessine des cercles a l'ecran
Screen 0
OS[176,200,40,10,15]
For C=1 To 256 Step 2
    Circle 176,200,C
Next
Screen Copy 0 To 1
'
' On y va
Amal On
Screen Close 2
'
' Modifie les couleurs
Do
    Read C1,C2
    If C1=-1 : Restore : Read C1,C2 : End If
    Fade 25,,C1,,,,,,,C2
    Repeat
        Exit If Inkey$<>"",2
    Until Colour(1)=C1 and Colour(9)=C2
Loop
'
' Snif, fini
Default
Edit
'-------------------------------------------
Data $46F,$FFF
Data $F0,$F
Data $0,$FFF
Data $F0F,$F0
Data $8F6,$56
Data -1,-1
'-------------------------------------------
Procedure OS[X,Y,SX,SY,C]
Bar X-SX,Y-SY To X+SX,Y+SY
    For R=1 To C
        Circle X-SX,Y-SY,R
        Circle X-SX,Y+SY,R
        Circle X+SX,Y-SY,R
        Circle X+SX,Y+SY,R
    Next
End Proc
```

By Daisy LIONET

LANGAGE AREXX : LA VOIE RO

Dans le précédent article, nous avons vu deux types d'utilisation d'ARexx. La plus simple consiste en un programme indépendant effectuant des opérations telles que la formalisation personnelle d'un répertoire.

Un peu plus compliquée est la suivante, qui traite des éléments d'un programme, éventuellement totalement étranger, d'une manière indépendante de la fonction et de l'utilisation prévue de ce programme. Aujourd'hui nous terminerons la revue élémentaire des possibilités d'ARexx, en évoquant sa capacité de communication entre des programmes écrits en ARexx par vous-même ou par d'autres auteurs, écrits en quelque langage que ce soit mais prévus pour émettre ou recevoir des commandes ARexx à partir de scripts écrits par vous ou quelqu'un d'autre, ou bien conçus de telle façon que l'usage d'ARexx soit totalement transparent pour l'utilisateur. Toutes les combinaisons des trois possibilités précédentes étant réalisables.

Cette capacité de communication, réalisée au moyen de "ports", sortes de boîtes aux lettres, permet d'étendre le concept de sous-programme couramment utilisé à l'intérieur d'un programme, et de le généraliser à tout programme capable de gérer des ports en autorisant non seulement le lancement de programmes externes, mais encore en leur "passant" les arguments nécessaires à leur fonctionnement et en recevant de ceux-ci, par le même moyen, les valeurs qu'ils ont pu élaborer.

INSTALLATION D'AREXX

L'installation d'ARexx ne présente pas de difficulté. La disquette fournie contient tous les éléments nécessaires à l'installation tant sur disquette que sur disque dur, avec lancement automatique ou manuel selon le désir de chacun.

Plusieurs progammes doivent être installés dans différents répertoires.

- libraries spécifiques à ARexx dans le répertoire LIBS:

```
Rexxarplib.library 47228
Rexxmathlib.library 7076
Screenshare.library 1796
Rexxsupport.library 2528
Rexxsyslib.library 33300
```

soit un total de 91928 octets.

- programmes spécifiques à ARexx dans le répertoire C:

```
Rexxmast        2348
Hi              304
Loadlib         712
Rx              968
Rxc             440
Rxset           640
Tco             332
Tcc             332
Te              268
Ts              268
WaitForPort     356
```

soit un total de 6968 octets.

Les librairies n'ont pas besoin d'être toutes résidentes et peuvent être chargées sélectivement, selon les nécessités du programme.

Dans le répertoire C: la commande principale est RexxMast, qui lance le fonctionnement d'ARexx. Cette commande peut être lancée à partir d'un CLI ou d'un Shell pour le lancement manuel, ou bien être placée dans votre Startup-Sequence pour un lancement automatique. L'arrêt du processus résident d'ARexx s'effectue de la même façon avec la commande Rxc.

Il est également utile d'installer un certain nombre d'utilitaires, non indispensables au fonctionnement d'ARexx, mais dont l'emploi facilite la mise en oeuvre, la réalisation et la mise au point des programmes. Il s'agit principalement d'un Shell, qui évite beaucoup de frappe et qui est préférable au CLI standard, et d'un bon éditeur. Un traitement de texte peut aussi être utilisé, pourvu qu'il ait une sauvegarde en Ascii lisible par n'importe qui. A moins qu'ED, l'éditeur indigène de l'Amiga, n'ait beaucoup changé, il est prudent d'éviter son emploi, à cause de sa fâcheuse tendance à "rater" la "validation" de la disquette, ce qui conduit à la requête d'utilisation de DiskDoctor - auquel d'ailleurs nous préférons DiskSalv du Domaine Public (Fish).

L'installation de ces programmes, spécifiques à ARexx ou autres, ne présente aucune difficulté. Cependant, pour vérifier que tout fonctionne correctement, nous vous suggérons de lancer le test suivant :
- si votre installation est du type "manuelle", à partir du CLI/Shell, lancez la commande RexxMast. Si le programme n'est pas déjà installé, une courte présentation (à effacement automatique) vous prévient de sa mise en action. Sinon, une autre présentation disant "ARexx server already active" vous prévient que tout est déjà en place. Vous pouvez essayer ici la manoeuvre consistant à mettre ARexx en action, puis à le désactiver en utilisant la commande Rxc. Cette dernière commande est muette, mais la commande suivante RexxMast remet l'ensemble en activité.
- recherchez ensuite sur la disquette ARexx, le programme nommé "Calc.rexx". Lorsque vous l'avez trouvé, placez-vous dans le répertoire le contenant, puis envoyez la commande Rx Calc 5+3 et attendez le résultat (qui doit apparaître sous forme de "8").

Explications : Calc.rexx est un programme qui effectue l'opération que vous lui proposez et retourne le résultat. Il n'est pas nécessaire de donner explicitement le suffixe .rexx. Ensuite, pour lancer le programme, il a fallu lancer Rx, qui est un programme que nous avons placé dans le répertoire C: au moment de l'installation. Nous lui avons passé les arguments nécessaires pour que Calc puisse effectuer son travail, à savoir les valeurs numériques ainsi que l'opération à effectuer.

Nous reverrons cela en détails plus loin. Essayez si vous voulez d'autres opérations. Il n'est bien sûr pas nécessaire de se placer dans le répertoire où se trouve le programme. On aurait très bien pu écrire Rx df1:rexx/calc (3+2)/4 et le résultat aurait été "1.25".

Notons en passant que le programme Calc contient moins de 100 octets, commentaires y compris !

Les essais terminés nous pouvons passer à ARexx lui-même.

AREXX, LE LANGAGE

Comme pour tout langage, l'écriture de programmes en ARexx doit respecter une orthographe et une syntaxe précises. L'orthographe est réservée à l'écriture des mots spécifiques, que ce soient des instructions ou des fonctions.

Les instructions sont composées d'au moins un mot-clé placé en tête et de paramètres qui le suivent dans un ordre défini pour cette instruction. D'autres mots-clés peuvent être trouvés dans le corps de l'instruction. Par exemple :

```
IF j>5 THEN a=3
```

Les fonctions sont composées d'un mot-clé suivi de parenthèses dans lesquelles on écrit les arguments qui doivent être envoyés à la fonction, laquelle retournera, ou non, un résultat selon le mode d'appel direct ou par l'emploi de CALL. Par exemple :

```
ABS(-5*3)
```

Enfin, les commandes sont des ordres donnés dans le corps d'un programme ARexx et qui ne peuvent pas être classées dans l'une des deux catégories précédentes. La commande n'est pas évaluée par l'interpréteur ARexx mais est envoyée telle quelle à un "hôte" extérieur. Pour que l'interpréteur n'effectue pas une tentative qui conduirait immédiatement à une erreur, la commande est entourée d'apostrophes ('). L'hôte le plus courant est l'AmigaDOS, mais il n'est pas le seul possible.
Pour envoyer une commande DOS, il suffit d'écrire la commande DOS dans son écriture habituelle, entourée d'apostrophes :

```
'delete df1:nomprogramme'
```

provoquera l'effacement du fichier "nomprogramme" de la disquette présente dans le lecteur DF1: sans autre participation d'ARexx que celle d'avoir envoyé la commande.

AUTRES SPECIFICITES

Comme dans tout langage, l'interpréteur ARexx évalue les expressions selon une hiérarchie et un ordre allant de la gauche vers la droite lors de la lecture d'une ligne de programme.

YALE (II) par F. Gueugnon

Les opérations mathématiques sont interprétées selon les règles usuelles de l'algèbre.

Les mots spécifiques à ARexx peuvent être indifféremment écrits en majuscules ou en minuscules, mais l'interpréteur les passera obligatoirement en majuscules.

Il peut y avoir plus d'une instruction par ligne si elles sont séparées par un point virgule. La fin de ligne est une condition implicite de fin d'instruction.

Lorsque une instruction est plus longue qu'une ligne, le symbole de concaténation est la virgule, qui indique à l'interpréteur que la fin de ligne n'est pas la fin de l'instruction.

CLASSIFICATION

Les instructions peuvent être classées en 2 catégories :
- les instructions de programmation : boucles, tests, etc. ;
- les instructions d'exécution : echo, parse, queue, etc. ;

Les fonctions peuvent être classées en 2 catégories :
- les fonctions système : ADDLIB(), ADDRESS(), etc. ;
- les fonctions effectuant une opération : ARG(), DATATYPE(), etc ;

METHODE

Nous ne présenterons pas le langage instruction par instruction, fonction par fonction car cela fait partie du livret fourni avec ARexx . Mais par le biais d'exemples choisis nous montrerons l'utilisation et l'intérêt de celles qui sont originales par rapport à d'autres langages ou l'usage d'une des autres dont l'emploi est original.

PREMIERS PAS EN AREXX

Lors de l'écriture d'un programme ARexx, la première instruction en tout début de programme DOIT être un commentaire. Un commentaire, que ce soit le premier ou un autre dans le corps du programme, est constitué de ce que l'on veut pourvu qu'il soit entouré des symboles "/*" en tête et */ en fin (tout comme en C). Par exemple :

```
/* Ceci est la méthode pour calculer ARGtgH(87) */
```

Les symboles /* et */ peuvent être imbriqués, pourvu qu'ils aillent par paire. Cette pratique permet de mettre un titre au programme que l'on écrit. Un peu d'ordre ne nuit pas ! En fin de programme, pour retourner au système hôte ou à l'appelant, il faut placer l'instruction EXIT. Notons que cette instruction est valable partout dans le programme et qu'elle peut apparaître au milieu d'une boucle ou d'un test, par exemple.

ENTREES-SORTIES

En général, l'exécution d'un programme produit à partir d'entrées, un résultat (sorties) qu'on souhaite connaître. Il faut donc communiquer avec le programme et que celui-ci communique avec l'utilisateur.

Pour afficher un résultat quelconque sur l'écran, il suffit d'écrire l'instruction suivante SAY. Lorsque l'on veut écrire une chaine de caractères non interprétables à l'écran, on utilise à nouveau les apostrophes :

```
SAY 'Ceci est la méthode pour élever A au carré : A**A'
```

écrira texto à l'écran :

```
Ceci est la méthode pour élever A au carré : A**A
```

On peut aussi demander la résolution d'une opération :

```
SAY 5**3
```

qui devrait donner "125".

```
SAY ' le résultat de 5**3 est:' 5**3
```

Dans ce dernier exemple, on voit apparaître le mélange de chaines de caractères dont certaines seront écrites telles quelles et d'autres seront interprétées. L'espace blanc entre les deux chaînes est nommé opérateur de concaténation. Dans ce cas, il est implicite, mais il y en a d'autres selon que l'on veut un espace entre les chaînes (cas présent) ou pas, par exemple :

```
SAY 'ceci est le résultat'||'du calcul'
```

C'est l'opérateur de concaténation || qui provoquera la sortie à l'écran de :

```
ceci est le résultat du calcul
```

sans espace entre "résultat" et "du".

L'instruction SAY a pour synonyme ECHO, qui lui est interchangeable. L'utilisation des apostrophes peut être remplacée par celle des guillemets.

Nous avons vu lors du test de l'installation d'ARexx, que nous avons invoqué le programme Calc.rexx au moyen de "Rx", qui est la commande de mise en fonctionnement d'ARexx, et en écrivant derrière le nom du programme invoqué, la liste des arguments que l'on veut lui passer.

Il existe deux méthodes pour passer des arguments à un programme.

La première, celle que nous avons vue, consiste à invoquer le programme en lui donnant immédiatement les arguments dont il a besoin.

La deuxième consiste à invoquer le programme seulement et à attendre que celui-ci pose la question. Bien entendu, il ne le fera que si les instructions correspondantes sont présentes dans le programme.

Dans les deux cas on utilisera à l'intérieur du programme appelé, une instruction paramétrée selon le besoin.

L'INSTRUCTION PARSE

Cette instruction est aussi puissante qu'intéressante. Elle s'écrit :

```
PARSE UPPER source variable_1 variable_2 variable_3 ... variable_N
```

"Parse", en anglais, signifie analyser. Cette instruction analyse donc ce qui suit le mot réservé PARSE.

- UPPER est une instruction qui met toute chaîne de caractères à laquelle elle est appliquée, en majuscules (en anglais : "uppercase"). Elle doit suivre immédiatement PARSE lorsqu'elle est utilisée ;

- source fournit la chaîne de caractères à analyser ;

- variable_1, variable_2, etc. sont les variables qui reçoivent les "morceaux" de la chaîne d'entrée disséqués par PARSE.

Par exemple, en considérant que la variable "essai" contient la chaîne suivante :

```
essai = 'a= 3 + 5'
```

à la suite de la commande

```
PARSE UPPER essai toto chiffre1 signe deuxièmechiffre
```

la variable toto contiendra la sous-chaine "A=", chiffre1, "3", signe, "+" et deuxiemechiffre, "5". La source est ici la variable essai.

Nous détaillerons dans le prochain article cette instruction PARSE, aux nombreuses possibilités.

A RETENIR

- un programme commence TOUJOURS par un commentaire, entouré de /* et */ en nombre apparié ;
- l'exécution d'un programme se termine par EXIT, quelque soit l'endroit où se trouve cette instruction. En particulier c'est la dernière ligne d'un programme, mais on peut la trouver fréquemment dans une boucle de test du genre "IF ... THEN EXIT" ;
- le point-virgule (;) termine une instruction ;
- la virgule (,) indique que l'instruction se continue à la ligne suivante.

OU SE PROCURER AREXX ?

Ceux qui n'ont pas la chance de posséder un Amiga 3000, sur lequel ARexx est fourni en standard, désirent sans doute connaître le moyen de se procurer ce langage... Il est l'oeuvre de l'américan Bill Hawes. Vous pouvez lui commander directement à l'adresse ci-dessous, pour un prix d'environ 50 dollars ou de convaincre votre revendeur habituel de se le procurer.

Bill Hawes
Whishful Thinking Development Corp.
P.O. Box 308
Maynard MA, 01754 USA

Une politique de remise à jour est soutenue, sous réserve que vous renvoyiez votre "registration card". La dernière version d'ARexx est la 1.15, qui fonctionne aussi bien avec les Workbench 1.2, 1.3 et 2.0.

ToolBox : SAS 5.10 LMK

Dans le remarquable environnement de développement du SAS se trouve un utilitaire que peu utilisent (j'ai fait un sondage auprès d'au moins deux personnes!) par manque de connaissance de ses possibilités. Pourtant voilà un programme bien agréable pour faciliter la vie des fainéants.

Je veux bien entendu parler de Lmk. "Mais à quoi peut-il bien servir", se demandent tous les lecteurs en coeur. Eh bien cela sert à compiler et linker d'un coup un ou plusieurs programmes C, sans s'occuper de rien. Autrement dit, finies les fastidieuses saisies de commandes de compilations et de linkage depuis le CLI, on peut enfin aller tranquillement boire un café pendant que l'Amiga travaille.
Lmk, dans l'environnment 5.10, peut être appelé par son nom depuis le CLI ou depuis le Workbench. En ce qui concerne le lancement depuis le workbench, vous verrez qu'il n'existe pas d'icône nommée "Lmk". En fait, il faut cliquer sur l'icône "Build", ce qui a finalement pour effet de d'invoquer Lmk.

C'EST UN MALIN

Quelque soit la façon dont LMK est lancé, tous les sources présents dans le repertoire courant sont compilés et linkés ensemble suivant les options par défaut définies dans le fichier SASCOPTIONS, qui doit siéger dans le tiroir préférences de votre Workbench.
Si vous faites quelques essais, vous constaterez que Lmk est un gros malin car il ne compile que les sources qui ne l'ont pas déjà été, où qui viennent seulement d'être modifiés. Si vous voulez forcer la recompilation de tous les sources, il faut dans ce cas appeler Lmk depuis le CLI avec l'option adaptée, c'est-à-dire :

```
lmk -a
```

Mais ceci, déjà agréable en soit, est fort peu de choses par rapport à ce qu'il est possible de faire dès que l'on crée un "makefile".

LE MAKEFILE

C'est un fichier texte qui contient toutes les commandes que LMK se chargera de faire incurgiter au compilateur et au linker. Bien sûr, il y a une syntaxe à respecter, mais d'abord le nom du fichier lui-même a son importance. Trois possibilités :

```
lmkfile
prout.lmk    (fichier quelconque avec l'extension .lmk)
makefile
```

Lorsque Lmk est lancé, les fichiers sont recherchés dans le répertoire courant, suivant selon l'ordre ci-dessus. Ce n'est que si aucun fichier n'est trouvé que les options du SASCOPTIONS sont en dernier recours utilisées.

UN EXEMPLE DE MAKEFILE

Commençons par quelque chose de très simple afin de nous familiariser.

```
# ceci est un commentaire
prog: prog.o
        Blink FROM LIB:c.o prog.o TO prog LIB LIB:lc.lib LIB:amiga.lib

prog.o prog.c
        lc -cist -v -y ptog.c

# ceci est un autre commentaire
```

Quelques petites explications s'imposent :
- la première ligne est un commentaire. On peut en mettre autant que l'on veut à condition qu'ils commencent en début de ligne et par le signe dièse (#) ;
- la deuxième ligne "prog: prog.o" signifie à Lmk que l'éxécutable "prog" dérivera de l'objet "prog.o" ;
- ensuite vient la commande de linkage proprement dite. Attention, dans la notice du SAS est précisé que cette ligne doit commencer par un espace. C'est faux, il en faut DEUX au moins. Nobody's perfect ! Passons...
- ensuite, la ligne "prog.o: prog.c" signifie que l'objet "prog.o" dérivera du source "prog.c" ;
- et enfin, la commande de compilation qui devra, vous l'aviez déjà compris, elle aussi débuter par deux espaces au moins.

ALLONS PLUS LOIN

En effet, dans l'exemple précédant n'est utilisé qu'un seul source. Un esprit chagrin comme le votre, ô lecteur adulé, peut encore dire que finalement ça ne valait pas le coup de perdre son temps à créer un makefile pour si peu. C'est vrai, mais si vous êtes un programmeur averti, vous êtes organisé et vous aimez ranger les sources appartenant à un même projet dans un répertoire spécifique et fractionner votre source en plusieurs morceaux pour des questions de clarté et de commodité. De plus, vous n'avez rien à foutre des fichiers objets produit par le compilateur et vous reprendriez bien un deuxième café... Voyez plutôt :

```
essai: ram:sources/essai.o
        blink FROM LIB:c.o ram:sources/essai.o TO essai
        LIB LIB:lc.lib LIB:amiga.lib
        delete ram:sources/essai.o

essai.o: ram:sources/essai.c
        lc -cist  -v -y essai.c

essai1: ram:sources1/essai1.o
        blink FROM LIB:c.o ram:sources1/essai1.o TO essai1
        LIB LIB:lc.lib LIB:amiga.lib
        delete ram:sources1/essai1.o

essai.o: ram:sources1/essai1.c
        lc -cist  -v -y essai1.c
```

Vous voulez compiler d'un coup les programmes "essai.c" et "essai1.c" dont les sources sont rangés dans les répertoires respectifs "sources" et "sources1". L'exemple ci-dessus fait tout cela, et vous vous retrouvez en fin de compte avec les programmes éxécutables "essai" et "essai1" dans le répertoire courant.

```
essai.o: ram:sources/essai.c
        lc -cist  -v -y essai.c
```

s'interprete comme ceci : Lmk voit où le source se situe et passe une directive include au compilateur, afin que celui-ci puisse trouver le fichier et ainsi de suite.
En plus, et c'est extrêmement sympathique, vous avez vu qu'il est parfaitement possible d'introduire des commandes AmigaDOS dans un makefile. C'est ce qui a été fait pour effacer les objets produits. Il n'y a déjà plus beaucoup de limites à ce qu'il est possible de trafiquer grâce à cette possibilité. Mais tout de suite, nous allons éliminer un problème auquel vous risquez de vous heurter. Vous voulez linkez un programme comme suit :

```
Blink FROM LIB:c.o prog1.o prog2.o TO prog
LIB LIB:lcm.lib LIB:lc.lib LIB:amiga.lib
SMALLCODE SMALLDATA NODEBUG VERBOSE
```

Cette commande, que l'on pourrait taper d'un seul jet dans le CLI, est trop longue pour tenir sur une seule ligne. Or il n'est pas possible, dans un makefile, d'aller simplement à la ligne car pour Lmk, chaque début de ligne correspont implicitement à une commande. Pour résoudre ce problème, nous utiliserons le caractère backslash (\), la commande devenant :

```
Blink FROM LIB:c.o prog1.o prog2.o TO prog \
LIB LIB:lcm.lib LIB:lc.lib LIB:amiga.lib \
SMALLCODE \
SMALLDATA \
VERBOSE \
```

ce qui est tout de même plus clair.

Imaginons maintenant qu'un programme "prog.c" ait besoin d'un fichier

de données nommé "data.h", par exemple. Vous pouvez bien sûr écrire dans le source "prog.c" la directive "#include <data.h", mais il est plus pratique de faire ainsi dans le makefile:

```
prog.c: prog.c data.h
lc -cist -y -y ...etc
```

et l'inclusion du fichier data.h se fera automatiquement.

TOUJOURS PLUS

Bon d'accord, il y a d'éternels mécontents parmi vous qui trouvent qu'il est bien fastidieux d'écrire à chaque fois la commande de compilation dans le makefile. c'est pour vous que les macros existent.
Voici un exemple de ce qu'il est possible de faire avec les macros :

```
OPT = -m0t -cist -v -y
OBJ = prog1.o prog2.o prog3.o prog4.o
REP = dh0:sas/sources/
LIB = LIB:lcm.lib LIB:lc.lib LIB:amiga.lib

prog: $(OBJ)
Blink FROM LIB:c.o $(OBJ) to prog $(LIB)

prog1.o: $(REP)prog1.c
lc $(OPT) prog1.c

prog2.o: $(REP)prog2.c
lc $(OPT) prog2.c

prog3.o: $(REP)prog3.c
lc $(OPT) prog3.c

prog4.o: $(REP)prog4.c
lc $(OPT) prog4.c
```

Vous l'avez compris, les macros peuvent être utilisées pour tout ce que l'on veut:
- options de compilation ;
- spécification de répertoire ;
- regroupement d'objet ;
- tout ce qui peut vous passer par la tête !

Je pense que bon nombre d'entre vous ont déjà découvert beaucoup de choses à propos de Lmk, mais je ne vous quitterai pas sans vous montrer comment il est possible de définir un règle de transformation sources-objets. L'exemple ci-dessus devient :

```
OPT = -m0t -cist -v -y
OBJ = prog1.o prog2.o prog3.o prog4.o
REP = dh0:sas/sources/
LIB = LIB:lcm.lib LIB:lc.lib LIB:amiga.lib

prog: $(OBJ)
Blink FROM LIB:c.o $(OBJ) to prog $(LIB)

.c.o:
lc $(OPT) $*

prog1.o: $(REP)prog1.c

prog2.o: $(REP)prog2.c

prog3.o: $(REP)prog3.c

prog4.o: $(REP)prog4.c
```

Vous l'aurez compris, la ligne

```
.c.o:
lc $(OPT) $*
```

définit une règle de compilation par défaut de tous les sources, ce qui permet de raccourcir sensiblement le makefile et n'empêche de toute façon pas de forcer une compilation particulière avec une ligne de commande spécifique.

Voilà. Le but de cet article était de vous montrer quelques unes des possibilité de l'hyper-puissant Lmk, mais nous sommes très loin de ses limites, comme vous pourrez vous en rendre compte maintenant que vous vous y êtes familiarisé, en épluchant la notice. Décidément le SAS est bien le meilleur compilateur C sur Amiga !

LIVRES ...

J'en vois déjà qui se disent : était-ce bien raisonnable de confier le test de la "Bible" à celui-là ? Eh bien oui, car je vais être tout à fait objectif et le premier à être heureux de pouvoir annoncer que nous disposons enfin de bon bouquins en français !

Tout, d'abord la bible. Elle a plus que doublé de volume et suit le plan suivant : le hardware; Exee; AmigaDOS; les formats IFF; les librairies façon "autodocs"; l'Amiga 3000. La première impression en feuilletant le bouquin : c'est encore une traduction de la version allemande. Abzolument rekretaple ! A croire que personne ne serait capable d'écrire en France ? Une prière à l'endroit de Micro-Application : que vous refusiez toujours d'éditer des auteurs français, c'est votre droit le plus strict. Mais par pitié, offrez-vous les services d'un traducteur sachant de quoi il parle (ca existe !), et non ceux d'un jeune diplômé boutonneux tout juste bon à traduire l'annuaire de Munich.
On voit vite qu'il y a quantité de programmes d'exemples en C ou en assembleur, beaucoup plus que dans l'ancienne version, les programmes assembleurs étant écrits avec un macro-assembleur (les allemands auraient-ils perdu leur cher Sekaka ?). Bon, la première impression est plutôt bonne. Voyons de suite le chapitre "Hardware". Rien de nouveau sous le soleil, ça ressemble comme deux gouttes d'eau à l'ancienne version. Ca y ressemble tellement que toutes les bêtises sur le blitter sont toujours là, fidèles au poste. Passons.

Le chapitre Exec : beaucoup plus détaillé que la version précédente, mais avec une présentation des choses pas très accessible au débutant. A mon avis, il est plus facile de comprendre les RKM en anglais que la Bible en français. Tiens, les conditions d'entrée de AddTAsk sont toujours fausses (registres A0,A1,A2 au lieu de A1,A2,A3). Est-ce VRAIMENT une nouvelle version ?

Le chapitre AmigaDOS : tenez vous bien, on nous dit que la dos.library est résidente en ROM contrairement à intuition.library, qu'il faut charger depuis une disquette ! Je laisse ça à vos méditation gouroutative... En revanche, on ne nous dit pas que la dos.library est écrite en BCPL, donc non standard et non modifiable par SetFunction(). Il est fait allusion aux DosPackets et puis plus loin il est dit qu'il ne sont pas documentés ! Bref, rien d'intéressant.

Les Devices : Cette fois, c'est un très bon chapitre avec beaucoup de programmes d'exemples. Un très bon point.

Les format IFF : Assez bien détaillés. Encore un bon chapitre.

Les librairies : très mauvais et je le prouve ! Vous voulez ouvrir une fenêtre et vous allez voir à la fonction OpenWindow(). La structure NewWindow est décrite mais les flags et les IDCMP sont seulement énumérés avec leus valeurs hexadécimales dont on a rien à faire. Il faut donc jouer aux devinettes pour connaitre les effets de ces flags. En fait pas totalement, car les IDCMP sont décrits au chapitre de l'input.device. Pas très logique, comme démarche. Mais si vous cherchez ici quelle est la différence entre un GADGETUP et un GADGETDOWN, mieux vaut allez vous rhabiller.

L'Amiga 3000 : dans ce chapître, on vous apprend seulement à vous servir du nouveau Workbench, mais comme vous n'êtes pas des demeurés et que vous possédez le manuel de votre Amiga (NDLR : qui est en français), vous n'en avez que faire. Par contre la workbench.library, qui existe sur le 3000, est passée sous silence, ce qui est plus que dommage.
Conclusion : nous avons là un livre très moyen qui manque totalement de rigueur et qui ne justifie pas son prix.

Bien débuter en langage machine. Ils ont du se tromper... C'est "mal débuter en langage machine" qu'il aurait fallu l'appeler. Le plan suivi est farfelu. Très tôt, on bassine le pauvre débutant avec les questions métaphysique de la WOM du 1000. On recommande l'utilisation du Sekaka et on apprend à arrêter sous Seka un programme avec l'instruction ILLEGAL ! Pourquoi pas tout simplement un RTS ?
A la fin du livre se trouve un chapitre intitulé "devenir un professionnel de l'Amiga". Il y est enfin fait référence aux fichiers Include, mais pour nous dire "qu'il faut transformer les fichiers include.h du langage C" pour pouvoir s'en servir ! Et alors ? A quoi ils servent, les include.i des macro-assembleurs ? C'est vrai j'oubliais que les allemands de connaissent que Seka.

La Bible édité par Micro Application - 53, rue du Fb Poissonnière - 75010 PARIS. Prix : 340F.

par Frédéric MAZUE

LE COIN DES DEMO-MAKERS :

Et voilà : un drive en panne vous prive du plaisir de manipuler des objets 3D en faces pleines. Nous souhaitons évidemment un prompt rétablissement à ce pauve Amiga, et pour vous faire patienter, vous proposons une routine que Jerôme Etienne nous avait envoyée pour l'évaluation de ses qualités.

*Vous aurez donc le plaisir, la joie et l'avantage de découvrir dans les pages qui suivent, un programme de Jerôme Etienne et un article de votre serviteur. Rassurez-vous, le programme est suffisamment long et commenté pour m'éviter de dire trop de bêtises...*De quoi s'agit-il donc ? D'un effet de Copper que tous les démo-makers ont utilisé un jour ou l'autre:
- des rasters horizontaux et en mouvement suivant une courbe sinusoïdale.

PRINCIPE DU PROGRAMME

Le principe de cette effet est assez simple à réaliser. Il consiste à recalculer la CopperList à chaque VBL, en prenant soin d'y inscrire chaque ligne de raster dans le bon ordre afin de conserver à la chose son côté esthétique.

On gère 15 barres composées de 16 lignes de rasters (cf. constante nb_coul). Au départ, on construit dans "tab_y_wave" une table des différentes coordonnées possibles pour chaque barre, suivant la fonction

$$y = \sin(x) * raywave + oy\_wave$$

avec x variant de 0 à 359 degrés. L'avantage d'une telle table est qu'on pourra aller y piocher les coordonnées des barres plutôt que d'avoir à recalculer un sinus à chaque itération de la boucle principale.

Une seconde table, au label "cur_coor_wave", contient la coordonnée actuelle de chacune des "nb_bar" barres. La première barre dans la table est celle qui semble la plus "éloignée" à l'écran. Ainsi, lorsque l'on affichera toutes les barres l'une après l'autre, elles viendront peu à peu se superposer pour donner cette impression de profondeur.

La boucle principale n'a plus qu'à attendre la VBL, à effacer entièrement la CopperList de manière à ce que les barres ne laissent pas de "traces" à l'écran puis à modifier la coordonnée de chaque barre en lui ajoutant la vitesse de déplacement actuelle (paramétrable avec les flèches gauche et droite du clavier) et à piquer la nouvelle coordonnée ainsi obtenue dans la table construite plus tôt. La barre est ensuite redessinée sur toute sa hauteur dans la CopperList. On boucle ainsi jusqu'à ce que toutes les barres aient été traitées.

Voilà. Un dernier mot sur la routine de sinus/cosinus : ne vous étonnez pas de voir qu'il est nécessaire d'appeler, dans la partie initialisation du programme, une routine qui la prépare. Pour éviter d'avoir à se taper les 360 valeurs de sinus possible, on n'en a initialisé que 90 (ce qui est quand même déjà pas mal !) à partir desquelles on va déduire les 270 restantes. C'est tout.

L'ESPOIR FAIT VIVRE

Normalement et si tout va bien, l'Amiga de Jerôme Etienne devrait être rapidement réparé - sinon une boutique que je ne nommerai pas pourrait bien voir l'aigle jaune et noir qui lui sert de logo se prendre deux baffes sur le coin du bec - et il devrait être capable de vous présenter dans le prochain numéro la suite de son initiation à la 3D façon démo. D'ici là, bonne fin de mois en notre compagnie.

```
; ************************************
; * Wave                             *
; * Affiche et déplace des barres de *
; * rasters copper                   *
; ************************************
; * par J. Etienne pour ANT          *
; ************************************

; ************************************
; * Définition des registres hard    *
; ************************************
custom  EQU $dff000
ciaapra EQU $bfe001
key     EQU $bfec01

color   EQU $180
color00 EQU color+00*2
color17 EQU color+17*2
color18 EQU color+18*2
color19 EQU color+19*2
dmacon  EQU $096
cop1lc  EQU $080
copjmp1 EQU $088
vposr   EQU $004
diwstrt EQU $08e
diwstop EQU $090
ddfstrt EQU $092
ddfstop EQU $094
bplcon0 EQU $100
bplcon1 EQU $102
bplcon2 EQU $104

; ************************************
; * Fonctions d'exec.library         *
; ************************************
_LVOOpenLibrary=-552
_LVOCloseLibrary=-414
_LVOForbid=-132
_LVOPermit=-138

; ************************************
; * Petite macro utile               *
; ************************************
CALLSYS MACRO
        jsr _LVO1(a6)
        ENDM

; ************************************
; * Variables                        *
; ************************************
nb_coul EQU $10 ; doit être multiple de 4
nb_line EQU 200 ; doit être multiple de 4
cl_size EQU nb_line*8+4
nb_bar  EQU 15

space_between_line=7
raywave EQU 80
oy_wave EQU 100
max_speed_wave=10

; ************************************
; * Début du programme               *
; * on s'approprie le hardware       *
; ************************************
Start   move.l  $4.w,a6
        lea custom,a5   ; adresse des chips dans a5
        lea bar_colors,a4   ; adresse de la palette dans a4

        CALLSYS Forbid  ; bloque le multitâche

        bsr compute_sin_tab; calcule la table des sinus
        bsr build_cl    ; construit la CopperList
        bsr init_wave   ; initialise les barres

        move.w  #$03e0,dmacon(a5)   ; dma bloqué
        move.l  #newcopper,cop1lc(a5)  ; init coplist
        move.w  #0,copjmp1(a5)
        move.w  #$8380,dmacon(a5)   ; dma copper et bitplane on

; ************************************
; * Boucle principale - fin avec ESC *
; ************************************
Loop    move.l  vposr(a5),d2    ;
        and.l   #$0001ff00,d2   ; attend le VBL
        cmp.l   #$00010100,d2   ;   (vsync)
```

```
        bne.s   Loop    ;

        lea     cl_adr,a0 ; adr de la coplist origine
        addq.l  #6,a0     ; pointe la première barre
        moveq   #nb_line/4-1,d0     ; taille de la liste -1
        moveq   #0,d1     ; inscrit des 0
loop_clear_cl:
        move.w  d1,(a0)
        move.w  d1,8(a0)
        move.w  d1,$10(a0)
        move.w  d1,$18(a0)
        lea     $20(a0),a0          ; a0 pointe la barre suivante
        dbf     d0,loop_clear_cl

        lea     cur_coor_wave(pc),a1
        moveq   #nb_bar-1,d2
each_bar:
        move.w  (a1),d3
        add.w   speed_wave(pc),d3
        cmp.w   #tab_y_wave_size,d3
        blt.s   not_change_cur_coor_wave
        sub.w   #tab_y_wave_size,d3
not_change_cur_coor_wave:
        move.w  d3,(a1)+

        moveq   #0,d0
        lea     tab_y_wave(pc),a0
        move.b  (a0,d3.w),d0        ; coordonnée de la barre

        lea     cl_adr,a0 ; adr de la coplist
        lsl.w   #3,d0     ; calcule l'adr du y à modifer
        addq.w  #6,d0
        add.l   d0,a0

        move.w  00(a4),$00(a0)      ; Répéter nb_coul fois
        move.w  02(a4),$08(a0)
        move.w  04(a4),$10(a0)
        move.w  06(a4),$18(a0)
        move.w  08(a4),$20(a0)
        move.w  10(a4),$28(a0)
        move.w  12(a4),$30(a0)
        move.w  14(a4),$38(a0)
        move.w  16(a4),$40(a0)
        move.w  18(a4),$48(a0)
        move.w  20(a4),$50(a0)
        move.w  22(a4),$58(a0)
        move.w  24(a4),$60(a0)
        move.w  26(a4),$68(a0)
        move.w  28(a4),$70(a0)

        dbf     d2,each_bar

; clavier
        move.b  key,d0    ;
        not     d0        ; teste le clavier
        ror.b   #1,d0     ;

        cmp.b   #$4e,d0   ; flèche gauche ?
        bne.s   not_inc_speed_wave
        cmp.w   #max_speed_wave,speed_wave
        bge.s   not_inc_speed_wave
        addq.w  #1,speed_wave
        bra     Loop

not_inc_speed_wave:
        cmp.b   #$4f,d0   ; flèche droite ?
        bne.s   not_dec_speed_wave
        cmp.w   #0,speed_wave
        ble.s   not_dec_speed_wave
        subq.w  #1,speed_wave
        bra     Loop

not_dec_speed_wave:
        cmp.b   #$45,d0   ; ESCape ?
        bne     Loop

; ************************************
; * Fin du programme. Retour au CLI  *
; ************************************
Ciao    lea     gfxname(pc),a1
        moveq   #0,d0
        CALLSYS OpenLibrary
        move.l  d0,a1
        move.l  38(a1),cop1lc(a5)
        clr.w   copjmp1(a5)
        CALLSYS CloseLibrary

        move.w  #$83e0,dmacon(a5)   ; réactivation des canaux dma
        CALLSYS Permit

fin     moveq   #0,d0     ; Retour au CLI
        rts

; ************************************
; * Initialise le tableau des barres *
; ************************************
init_wave:
        lea     cur_coor_wave(pc),a0
        moveq   #nb_bar-1,d0
        moveq   #0,d1

loop_init_cur_coor_wave:
        move.w  d1,(a0)+
        add.l   #space_between_line,d1
        dbf     d0,loop_init_cur_coor_wave

        lea     tab_y_wave(pc),a1
        move.l  #360-1,d2
        moveq   #0,d1
loop_init_tab_y_wave:
        move.l  d1,d0
        bsr     sin
        muls    #raywave,d0
        lsl.l   #1,d0
        swap    d0
        add.b   #oy_wave,d0
        move.b  d0,(a1)+
        addq.l  #4,d1
        dbf     d2,loop_init_tab_y_wave
        rts

; ************************************
; * Construit la CopperList          *
; ************************************
build_cl:
        lea     cl_adr,a0 ; adr de la coplist origine
        move.w  #nb_line-1,d0       ; taille de la liste -1
        moveq   #0,d1     ; ligne de depart
loop_build_cl:
        move.b  d1,(a0)+  ; wait y
        move.b  #$0f,(a0)+          ; wait x
        move.w  #$fffe,(a0)+        ; wait mask
        move.w  #color00,(a0)+      ; move.w adr
        move.w  #0,(a0)+  ; move.w data
        addq.w  #1,d1     ; 1 ligne traite en +
        dbf     d0,loop_build_cl
        move.l  #$fffffffe,(a0)+    ; la fin de la liste
        rts

; ************************************
; * Retourne le sinus de d0 dans d0  *
; ************************************
cos:    add.w   #360,d0   ; principe: cos(d0) = sin(d0+360)
sin:    cmp.w   #360*4,d0 ; angle sup ou equ a 360*4
        blt.s   sin1
        sub.w   #360*4,d0
        bra.s   sin
sin1:   tst.w   d0        ; negatif ?
        bpl     sin2
        add.w   #360*4,d0
        bra.s   sin1
sin2:   lea     sin_tab_adr(pc),a0
        lsl.l   #1,d0
        move.w  (a0,d0.w),d0        ; d0 = cos ou sin
        rts

; ************************************
; * Calcule la tables des sinus à    *
; * partir des valeurs déjà connues  *
; ************************************
compute_sin_tab:
sin_tab_size=360*2*4
        lea     sin_tab,a0
        lea     sin_tab_adr(pc),a1
        move.w  #360-1,d0
copy_sin_tab:
        move.w  (a0)+,(a1)+
        dbf     d0,copy_sin_tab

; calcule le reste de la table
```

```
;       sintab  90-180
        lea     sin_tab_adr(pc),a0 ; a1 pointe sur le degr 180
        lea     (360+360+1)*2(a0),a1
        move.w  #$7fff,360*2(a0)    ; place degr 90
        move.l  #360-1,d0 ; copie 90 valeurs
copy_90_180:                ; principe: sin 0= sin 180
        move.w  (a0)+,-(a1)         ; sin 1= sin 179
        dbf     d0,copy_90_180      ; sin 2= sin 178 etc...

;       sintab  180-270
        lea     sin_tab_adr(pc),a0 ; a0 pointe sur 0 degr
        lea     720*2(a0),a1        ; a1 pointe sur 180 degr
        move.l  #360-1,d0 ; copie 90 valeurs
        moveq   #0,d2
copy_180_270:
        moveq   #1,d1       ;
        swap    d1          ;/d1=$10000
        move.w  (a0)+,d2    ;
        sub.l   d2,d1       ;
        move.w  d1,(a1)+    ;d1= $10000-d2
        dbf     d0,copy_180_270

;       sintab  270-359
        lea     sin_tab_adr(pc),a0 ; a0 pointe sur 0 degr
        lea     720*2(a0),a0        ; a1 pointe sur le degr 360
        lea     (360+360)*2(a0),a1
        move.w  #$8000,360*2(a0)    ; place degr 270
        lea     2(a0),a0
        move.l  #359-1,d0 ; copie 89 valeurs
copy_270_359:               ; principe: sin 360=sin 180
        move.w  (a0)+,-(a1)         ; sin 359= sin 181
        dbf     d0,copy_270_359     ; sin 358= sin 182 etc...
        rts

; ************************************
; * Variables                        *
; ************************************
speed_wave          dc.w 1
cur_coor_wave       dcb.w nb_bar*2,0

tab_y_wave          dcb.b 180*2
end_tab_y_wave      EQU *
tab_y_wave_size     EQU end_tab_y_wave-tab_y_wave

gfxname dc.b        "graphics.library",0
        even

; ************************************
; * Palette des barres de couleur    *
; ************************************
bar_colors:
        dc.w        $0100,$0300,$0500,$0700
        dc.w        $0900,$0b00,$0d00,$0f00
        dc.w        $0e00,$0c00,$0a00,$0800
        dc.w        $0600,$0400,$0200,$0000

; ************************************
; * valeurs des sin*32768 de 0 à 90  *
; 4 valeurs pr 1 deg                 *
; ************************************
sin_tab dc.w        $0000,$008E,$011D,$01AC
        dc.w        $023B,$02CA,$0359,$03E8
        dc.w        $0477,$0506,$0595,$0624
        dc.w        $06B2,$0741,$07D0,$085E
        dc.w        $08ED,$097C,$0A0A,$0A99
        dc.w        $0B27,$0BB6,$0C44,$0CD2
        dc.w        $0D60,$0DEF,$0E7D,$0F0B
        dc.w        $0F99,$1026,$10B4,$1142
        dc.w        $11D0,$125D,$12EA,$1378
        dc.w        $1405,$1492,$151F,$15AC
        dc.w        $1639,$16C6,$1752,$17DF
        dc.w        $186B,$18F8,$1984,$1A10
        dc.w        $1A9C,$1B28,$1BB3,$1C3F
        dc.w        $1CCA,$1D55,$1DE0,$1E6B
        dc.w        $1EF6,$1F81,$200B,$2096
        dc.w        $2120,$21AA,$2234,$22BD
        dc.w        $2347,$23D0,$2459,$24E2
        dc.w        $256B,$25F4,$267C,$2704
        dc.w        $278C,$2814,$289C,$2923
        dc.w        $29AB,$2A32,$2AB9,$2B3F
        dc.w        $2BC6,$2C4C,$2CD2,$2D58
        dc.w        $2DDD,$2E63,$2EE8,$2F6D
        dc.w        $2FF1,$3076,$30FA,$317E
        dc.w        $3202,$3285,$3309,$338B
        dc.w        $340E,$3491,$3513,$3595
        dc.w        $3617,$3698,$3719,$379A
        dc.w        $381B,$389B,$391B,$399B
        dc.w        $3A1A,$3A9A,$3B19,$3B97
        dc.w        $3C16,$3C94,$3D12,$3D8F
        dc.w        $3E0C,$3E89,$3F06,$3F82
        dc.w        $3FFE,$407A,$40F5,$4170
        dc.w        $41EB,$4265,$42DF,$4359
        dc.w        $43D2,$444B,$44C4,$453D
        dc.w        $45B5,$462C,$46A4,$471B
        dc.w        $4791,$4808,$487E,$48F3
        dc.w        $4969,$49DE,$4A52,$4AC6
        dc.w        $4B3A,$4BAE,$4C21,$4C94
        dc.w        $4D06,$4D78,$4DEA,$4E5B
        dc.w        $4ECC,$4F3C,$4FAC,$501C
        dc.w        $508B,$50FA,$5169,$51D7
        dc.w        $5244,$52B2,$531F,$538B
        dc.w        $53F7,$5463,$54CE,$5539
        dc.w        $55A4,$560E,$5677,$56E0
        dc.w        $5749,$57B2,$5819,$5881
        dc.w        $58E8,$594F,$59B5,$5A1B
        dc.w        $5A80,$5AE5,$5B49,$5BAD
        dc.w        $5C11,$5C74,$5CD6,$5D39
        dc.w        $5D9A,$5DFC,$5E5C,$5EBD
        dc.w        $5F1D,$5F7C,$5FDB,$603A
        dc.w        $6098,$60F5,$6152,$61AF
        dc.w        $620B,$6267,$62C2,$631C
        dc.w        $6377,$63D0,$642A,$6482
        dc.w        $64DB,$6532,$658A,$65E1
        dc.w        $6637,$668D,$66E2,$6737
        dc.w        $678B,$67DF,$6832,$6885
        dc.w        $68D7,$6929,$697A,$69CB
        dc.w        $6A1B,$6A6B,$6ABA,$6B08
        dc.w        $6B57,$6BA4,$6BF1,$6C3E
        dc.w        $6C8A,$6CD5,$6D20,$6D6B
        dc.w        $6DB5,$6DFE,$6E47,$6E8F
        dc.w        $6ED7,$6F1E,$6F65,$6FAB
        dc.w        $6FF0,$7035,$707A,$70BE
        dc.w        $7101,$7144,$7186,$71C8
        dc.w        $7209,$724A,$728A,$72CA
        dc.w        $7308,$7347,$7385,$73C2
        dc.w        $73FF,$743B,$7476,$74B1
        dc.w        $74EC,$7526,$755F,$7598
        dc.w        $75D0,$7607,$763E,$7675
        dc.w        $76AB,$76E0,$7715,$7749
        dc.w        $777C,$77AF,$77E2,$7813
        dc.w        $7845,$7875,$78A5,$78D5
        dc.w        $7903,$7932,$795F,$798C
        dc.w        $79B9,$79E5,$7A10,$7A3B
        dc.w        $7A65,$7A8E,$7AB7,$7AE0
        dc.w        $7B07,$7B2E,$7B55,$7B7B
        dc.w        $7BA0,$7BC5,$7BE9,$7C0C
        dc.w        $7C2F,$7C52,$7C73,$7C94
        dc.w        $7CB5,$7CD5,$7CF4,$7D12
        dc.w        $7D31,$7D4E,$7D6B,$7D87
        dc.w        $7DA3,$7DBD,$7DD8,$7DF2
        dc.w        $7E0B,$7E23,$7E3B,$7E52
        dc.w        $7E69,$7E7F,$7E95,$7EA9
        dc.w        $7EBE,$7ED1,$7EE4,$7EF7
        dc.w        $7F08,$7F19,$7F2A,$7F3A
        dc.w        $7F49,$7F58,$7F66,$7F73
        dc.w        $7F80,$7F8C,$7F97,$7FA2
        dc.w        $7FAD,$7FB6,$7FBF,$7FC8
        dc.w        $7FD0,$7FD7,$7FDD,$7FE3
        dc.w        $7FE9,$7FED,$7FF1,$7FF5
        dc.w        $7FF8,$7FFA,$7FFB,$7FFC

sin_tab_adr:
        dcb.b       sin_tab_size,0

; ************************************
        section     chip,data_c

newcopper:
        dc.w        diwstrt,$2981,diwstop,$29c1
        dc.w        ddfstrt,$0038,ddfstop,$00d0
        dc.w        bplcon0,$0000,bplcon1,$0000
        dc.w        bplcon2,$0000
cl_adr: dcb.b       nb_line*8
        dc.l        -2

; ************************************
```

UTILITAIRE : LE TRAQUEUR DE VIRUS

En cette période printanière, les virus, anciens ou nouveaux, reviennent faire des dégats. Je sais bien que les chasseurs de virus courrent les rues du domaine public, mais quand je songe au nombre de disquettes infectées que les lecteurs m'envoient, je me dis qu'il y a encore des choses à faire.

Voyez la situation en face : beaucoup d'utilitaires, comme VirusX pour ne citer que le plus célèbre, signalent la présence d'un boot suspect. Mais s'agit-il d'un boot nécéssaire au démarrage d'un jeu ou d'une démo, ou bien encore d'un nouveau virus ? Angoissante question, je me remercie, une fois de plus, de l'avoir posée.

LE PROBLEME

Il s'agit cette fois de vérifier si un programme boot ou un programme normal a l'intention de faire joujou avec les vecteurs reset de votre Amiga chéri. Ceci posé, comment faire ?
Cela ne peut pas se faire facilement en regardant ce genre de programmes "à l'oeil nu", car il sont en principe codés de diverses manières, de façon à déjouer les regards indiscrets.

LA SOLUTION

Elle consiste en un programme dont la tâche sera de charger et lancer un programme ou un boot suspect puis à vérifier l'état des vecteurs reset chaque fois qu'une instruction du suspect sera éxécuté. Pour parvenir à ce résultat, nous utiliserons une particularité des microprocesseurs de la génération 68000 : le mode Trace.

PETIT RAPPEL

Nous allons revoir brièvement ce qui à déjà été dit dans l'article "Guru Interceptor" du CR N°28. Les microprocesseurs de la génération 68000 fonctionnent suivant deux modes : le mode utilisateur et le mode superviseur, chaque mode utilisant sa propre pile.
Le mode utilisateur est le mode normalement actif lorsqu'un programme est éxécuté. Le mode utilisateur n'a "pas le droit" d'utiliser la gamme complète du jeu d'instructions. Les instructions interdites sont appelées instructions privilégiées.
D'autre part, en mode utilisateur, le programmeur n'a le droit d'accéder qu'au poids faible du registre d'état SR. Cette partie poids faible est nommée sous-registre ou registre CCR et contient les flags arithmétiques : Zéro, Carry, etc.
Le mode superviseur est en principe réservé au système d'exploitation de l'ordinateur. Lorsque le microprocesseur fonctionne sur ce mode, il est possible d'utiliser le jeu d'instruction au complet. De même, il est possible d'accéder à la totalité du registre SR, dont le poids fort contient des flags représentant l'état du microprocesseur.

QUE SE PASSE-T-IL SI...

Que se passe-t-il si l'on essaie d'utiliser une instruction privilégiée depuis le mode utilisateur ? Eh bien dans ce cas, le microprocesseur considère qu'il s'agit d'une faute du programme, se met de lui-même en mode superviseur, puis saute dans le vecteur de bas de mémoire correspondant à la situation.
Dans ces vecteurs de bas de mémoire, sont installés les convertisseurs de Trap d'Exec, qui débouchent finalement sur le Gourou si la tâche coupable d'avoir utilisée une instruction privilégiée n'a pas auparavant signalé la présence d'un convertisseur de Trap qui lui est propre. C'est ce procédé que nous avons utilisé dans le Gourou Interceptor.

LE BIT TRACE

Dans le registre SR existe un bit appelé T. Ce bit est normalement à 0. S'il est à 1, chaque fois que depuis le mode utilisateur, le microprocesseur éxécute une instruction, il passe en mode superviseur puis saute dans le vecteur d'exception d'adresse S24. Voilà donc une excellente possibilité de contrôler pas à pas si un programme va attaquer le Reset.

ACTIVATION DE LA TRACE

Ca c'est facile. Exec nous offre la possibilité de passer sans se fatiguer en mode superviseur grâce à la routine SuperState, dont le principe de fonctionnement est de déclencher une exception, de la récupérer à peu près comme expliqué plus haut, puis de revenir au programme. Bref, nous voilà maintenant en mode superviseur. Nous avons accès au registre SR dans sa totalité et c'est un jeu d'enfant que d'activer la trace en plaçant le bit T puis de revenir en mode utilisateur à l'aide de la routine UserState, qui fait exactement l'inverse de la précédente.

Mais attention : nous ne pouvons nous limiter à cela car maintenant, le microprocesseur va sauter dans le vecteur d'adresse $24 dès la première instruction éxécutée, et nous aurions droit à un Gourou N°9.

Peut-on intercepter ce Gourou comme nous l'avions déjà fait en installant un convertisseur de Trap sur notre tâche ? Hélas non, car dans ce cas, et c'est le seul, Exec ne regarde pas si nous avons installé un convertisseur de Trap, mais envoie systématiquement le Gourou.
Il est donc malheureusement nécéssaire de modifier nous-mêmes le vecteur de saut, en le détournant sur notre routine de test. C'est bien sûr dans cette routine que nous allons vérifier si les vecteur reset sont attaqués.
Il est regrettable d'être obligé de procéder ainsi, car toutes les tâches sont de ce fait en mode trace, ce qui ralentit considérablement la machine et limite l'utilisation de notre programme à un test pur et non à un emploi permanent.

LE PROGRAMME

C'est maintenant tout simple. Il peut être appelé depuis le CLI ou le Workbench.
1° cas : appel depuis le Workbench ou depuis le CLI SANS paramètres. Dans ce cas, il est fait appel au trackdisk.device pour charger les secteurs boot en df0:. On vérifie que la disquette est bien autoboot, on installe la routine de traçage comme expliqué plus haut, puis on saute dans le programme boot. Si ce boot contient un virus, une alerte vous en informera immédiatement. Vous aurez le choix de continuer ou de tuer le virus de façon certaine, mais attention : en mémoire seulement. Il ne faut donc pas rebooter la disquette infectée. Vous devrez la nettoyer avec un utilitaire quelconque, ou tout simplement la commande "Install" du CLI (il faut également que je vous prévienne que si c'est le boot d'une démo que vous testez, il est probable que vous aurez la visite du Gourou. Mais dans tous les cas, s'il s'agit d'un virus, vous serez prévenu avant).
2° cas : vous lancez le programme depuis le CLI en donnant comme paramètre un nom de fichier. Si ce fichier existe, et s'il est éxécutable, on le charge en tant que segment puis l'on saute dans le programme. En cas de virus coquille, l'alerte apparait.

UN ESSAI

Le petit programme virus n'en n'est bien sûr pas un, mais se contente de mettre une valeur dans un vecteur reset. Vous pourrez vérifier ainsi sans risque que tout fonctionne bien, en tapant depuis le CLI :

```
Traqueur Virus
```

Notre programme est capable de tester tous les programmes "normaux" mais ne peut pas fonctionner avec les commandes du répertoire C: de la disquette Workbench, ceci à cause de la conception même de ces programmes.

Mais si vous êtes un fidèle lecteur, vous disposez déja du programme "Sentinelle", qui, lui, convient très bien.

En fait, le programme de ce mois est surtout intéressant pour tester les boots.

LA ROUTINE RESET

Toujours si vous êtes un fidèle lecteur, vous verrez que la routine reset du programme est très différente de celle recommandée dans l'article "Faisons Reset avec le sourire". Pourquoi ? Simplement parce que l'ancienne routine n'était valable que pour un Amiga avec microprocesseur 68000. La routine de ce mois convient à tout Amiga. Elle est directement issue de la dernière revision des Amiga Rom Kernal Manual.

Enfin une dernière chose : si vous avez installé le RAD: ne craignez rien, le Traqueur le reconnait et le réinstalle dès qu'il rend la main (NDLR : ce qui n'était pas le cas de la Sentinelle, la honte soit sur Frédéric jusqu'à la 13 ième génération !).

LE TRAQUEUR DE VIRUS

```
*--- Programme:traqueur.a
*--- Fonction: traquer les virus de boot ou les virus
*---        coquilles au plus profond des programmes
*--- Utilisation:  - depuis le workench, boot en DF0:
*---        - depuis le CLI:
*---        - boot si pas de paramètre
*---        - teste le programme donné en paramètre
*--- Remarques:- Ne détruit pas le RAD:
*---        - Ne peut tester une commande DOS because BCPL
*--- Auteur: F MAZUE pour ANT le 16/02/91
*--- Assembleur: Devpac2 (the top !)

 incdir "include:"
 include "exec/execbase.i"
 include "exec/libraries.i"
 include "exec/tasks.i"
 include "exec/io.i"
 include "exec/memory.i"
 include "exec/devices.i"
 include "exec/exec_lib.i"
 include "intuition/intuition.i"
 include "intuition/intuition_lib.i"
 include "libraries/dos.i"
 include "libraries/dos_lib.i"
 include "libraries/dosextens.i"
 include "devices/trackdisk.i"
 include "misc/easystart.i"     ;pour workbench

main
 clr.l Segment
 clr.l Tampon
 clr.l rad
 clr.l prog

 movem.l d0/a0,-(sp)

*--- tester existence de rad:
 move.l _SysBase,a6
 move.l KickMemPtr(a6),a3
 cmp.l #$7fdb2,a3
 bne norad
 add.l #$104,a3
 cmpi.l #$52414400,(a3)         ;Si le RAD: est présent, le mot RAD
                                ; (52414400) est ici pointé par a3
 bne norad
 move.l #$1,rad
 lea rad_data,a3                ;sauvegarde des vecteurs
 move.l KickMemPtr(a6),(a3)+    ;reset du RAD:
 move.l KickTagPtr(a6),(a3)+
 move.l KickCheckSum(a6),(a3)

norad
 clr.l CoolCapture(a6)          ;nettoyage des vecteurs reset
 clr.l ColdCapture(a6)          ;
 clr.l KickMemPtr(a6)           ;
 clr.l KickTagPtr(a6)           ;
 clr.l KickCheckSum(a6)         ;

 moveq #0,d0                    ;ouverture
 lea dosname,a1                 ;de
 CALLEXEC OpenLibrary           ;la
 move.l d0,_DOSBase             ;dos.library

 sub.l a1,a1
 CALLEXEC FindTask              ;recherche de sa propre adresse de tâche
 move.l d0,Task

 move.l $24,OldTrap             ;l'execption Trace du 68000
 move.l #trap,$24               ;passe par cette adresse

 move.l d0,a0                   ;ici l'on teste si le programme
 tst.l pr_CLI(a0)               ;est lancé depuis le CLI
 beq WorkBench                  ;ou depuis le workbench

 movem.l (sp)+,d0/a0            ;récupération des paramètres
 move.b #$0,-1(a0,d0.w)         ;remplace CR par un NUL
 subq #1,d0                     ;y a-t-il un paramètre ?
 beq wb1                        ;non donc aucun fichier à charger...

 move.l #$1,prog                ;flag pour choix d'alerte.
 move.l a0,file                 ;nom de fichier sauvé
 move.l d0,len_file             ;longueur du nom de fichier sauvée

 move.l a0,d1                   ;nom de fichier valide ?
 move.l #MODE_OLDFILE,d2
 CALLDOS Open
 tst.l d1
 beq NoFich                     ;et bien non !

 CALLDOS Close

 move.l file,d1                 ;fichier éxécutable ?
 CALLDOS LoadSeg
 move.l d1,Segment
 tst.l d1
 beq NoExec                     ;et bien non !

 move.l d1,8(a3)
 add.l d1,d1
 add.l d1,d1                    ;conversion bcpl
 addq #$4,d1                    ;cf article BackStart, CR 30 Janvier 91
 move.l d1,a5                   ;valeur de saut au programme dans a5

 bra GO                         ;est lent...

WorkBench
 addq #8,sp                     ;ajustement de la pile
wb1
 move.l #$400,d0                ;1024 octets pour le boot
 move.l #(MEMF_CLEAR|MEMF_CHIP),d1 ;en chip mem
 CALLEXEC AllocMem              ;sont réservé
 move.l d0,Tampon
 beq NoMem

 lea ReplyPort,a1               ;ici est initialisé
 move.l #portname,LN_NAME(a1)   ;un (gros) port
 move.l Task,MP_SIGTASK(a1)     ;
 CALLEXEC AddPort

 lea DiskIO,a1                  ;DiskIO est une structure ExtIORequest
                                ;à laquelle on attache le port de reply
                                ;précédement créé.

 move.l #ReplyPort,MN_REPLYPORT(a1)

 move.l #$0,d0                  ;unité 0 donc df0:
 clr.l d1                       ;flag toujours nul
 lea td_name,a0                 ;nom du device
 CALLEXEC OpenDevice            ;device ouvert

 lea DiskIO,a1
 move.w #CMD_CLEAR,IO_COMMAND(a1)  ;commande clear pour
                                ;forcer juste après la
                                ;lecture à chaque fois

 move.l #$400,IO_LENGTH(a1)     ;$400 octets = 2 secteurs
 move.l Tampon,IO_DATA(a1)      ;adresse buffer
 move.l #0,IO_OFFSET(a1)        ;à partir du secteur 0
 CALLEXEC DoIO

 lea DiskIO,a1
 move.w #CMD_READ,IO_COMMAND(a1);commande lire
 move.l #$400,IO_LENGTH(a1)     ;$400 octets = 2 secteurs
 move.l Tampon,IO_DATA(a1)      ;adresse buffer
 move.l #0,IO_OFFSET(a1)        ;à partir du secteur 0
 CALLEXEC DoIO      ;lire

 move.b IO_ERROR(a1),FlagDisk   ;problème ?

 lea DiskIO,a1                  ;ici
 move.w #TD_MOTOR,IO_COMMAND(a1);l'on arrête
 move.l #$0,IO_LENGTH(a1)       ;le
 CALLEXEC DoIO                  ;moteur

 lea ReplyPort,a1               ;effacement du port
 CALLEXEC RemPort               ;

 lea DiskIO,a1                  ;fermeture du device
 CALLEXEC CloseDevice

 tst.b FlagDisk
 bne NoFormat                   ;si le secteur boot n'a pu être lu

checksum                        ;ici on fait le calcul du checksum
 move.l Tampon,a5               ;du boot que l'on vient de lire
 move.l 4(a5),d2                ;
 clr.l 4(a5)                    ;si le résultat est égal au
 move.l a5,a0                   ;checksum lu, c'est qu'il s'agit
 move.w #$00ff,d1               ;bien d'un boot
 moveq #$00,d0
checksum1
 add.l (a0)+,d0
 bcc.s checksum2
 addq.l #1,d0
checksum2
 dbf d1,checksum1
 not.l d0
 cmp.l d2,d0

 bne NoBoot
 add.l #$0C,a5                  ;adresse de saut dans A5

GO
 CALLEXEC SuperState            ;on passe en mode Superviseur
 or.w #$8000,sr                 ;on place le bit Trace
 CALLEXEC UserState             ;et on revient au mode user

 pea exit                       ;on empile l'adresse de fin
 move.l a5,-(sp)                ;on empile l'adresse de saut
 lea Dummy_A0,a0                ;on fabrique un fausse chaîne
                                ;de paramètres
                                ;(c'est utile uniquement dans le cas du
                                ;traitement d'un fichier éxécutable
 move.l #$1,d0                  ;
 rts                            ;c'est parti !

exit
 CALLEXEC SuperState            ;mode Superviseur
```

```
 and.w #$7fff,sr                ;on enlève le bit Trace
 CALLEXEC UserState             ;mode User
 tst.l rad                      ;on reinstalle les vecteurs reset
 beq .exitnorad                 ;du RAd: si nécéssaire
 bsr repare_rad                 ;
.exitnorad                      ;on quitte
 tst.l Tampon                   ;proprement
 beq .exitnm                    ;
 move.l Tampon,a1               ;
 move.l #$400,d0                ;
 CALLEXEC FreeMem               ;
.exitnm                         ;
 tst.l Segment                  ;
 beq .exitnoseg                 ;
 move.l Segment,d1              ;
 CALLDOS UnLoadSeg              ;
.exitnoseg                      ;
 move.l _DOSBase,a1             ;
 CALLEXEC CloseLibrary          ;
 move.l OldTrap,$24             ;
 moveq #0,d0                    ;
 rts                            ;

trap                            ;routine exécutée en mode trace
 move.l a6,-(sp)
 move.l $4,a6                   ; teste si les vecteurs Reset sont
 tst.l ColdCapture(a6)          ;attaqués
 bne virus
 tst.l CoolCapture(a6)
 bne virus
 tst.l KickMemPtr(a6)
 bne virus
 tst.l KickTagPtr(a6)
 bne virus
 tst.l KickMemPtr(a6)
 bne virus
 move.l (sp)+,a6
 rte                            ;ce n'était pas le cas

virus                           ;un des vecteurs reset est attaqué
 clr.l ColdCapture(a6)          ;on efface tout
 clr.l CoolCapture(a6)          ;
 clr.l KickMemPtr(a6)           ;
 clr.l KickTagPtr(a6)           ;
 clr.l KickCheckSum(a6)         ;

 move.l (sp)+,a6
 move.l #alert,2(sp)            ;on trafique subtilement la pile
 rte                            ;et on retourne en mode User déclencher
                                ;l'alerte

*--- Ici est l'envoi de l'alerte en cas de virus
*--- Aucun commentaire n'est nécéssaire
alert
 CALLEXEC SuperState
 and.w #$7fff,sr
 CALLEXEC UserState

 move.l #0,d0
 lea intname(pc),a1
 CALLEXEC OpenLibrary
 move.l d0,_IntuitionBase

 move.l #RECOVERY_ALERT,d0
 lea alerte_boot,a0
 tst.l prog
 beq .alert1
 lea alerte_coquille,a0
.alert1
 move.l #100,d1
 CALLINT DisplayAlert
 bne reset

 move.l _IntuitionBase,a1
 CALLEXEC CloseLibrary
 rts

reset                           ;routine Reset béton qui tue n'importe
                                ;quel virus en mémoire
 CALLEXEC Disable
 move.l #0,ColdCapture(a6)      ;a6 pointe ici sur execbase
 move.l #0,CoolCapture(a6)
 move.l #0,KickMemPtr(a6)
 move.l #0,KickTagPtr(a6)
 move.l #0,KickCheckSum(a6)

*--- Routine Reset officile (cf. RKM dernière version)
ABSEXECBASE         EQU 4       ;Pointer to the Exec library base
MAGIC_ROMEND        EQU $01000000;End of Kickstart ROM
MAGIC_SIZEOFFSET    EQU -$14    ;Offset from end of ROM to Kickstart size
V36_EXEC            EQU 36      ;Exec with the ColdReboot() function
TEMP_ColdReboot     EQU -726    ;Offset of the V36 ColdReboot function

_ColdReboot:
                    move.l    ABSEXECBASE,a6
                    cmp.w     #V36_EXEC,LIB_VERSION(a6)
                    blt.s     old_exec
                    jmp       TEMP_ColdReboot(a6) ;Let Exec do it...
                    ;NOTE: Control flow never returns to here

;--- manually reset the Amiga
old_exec:
        lea.l   GoAway(pc),a5       ;address of code to
        execute
        jsr     _LVOSupervisor(a6) ;trap to code at (a5)...
        ;NOTE: Control flow never returns to here

;--- MagicResetCode - DO NOT CHANGE-
        CNOP    0,4                 ;IMPORTANT! Longword align!
GoAway: lea.l   MAGIC_ROMEND,a0     ;(end of ROM)
        sub.l   MAGIC_SIZEOFFSET(a0),a0 ;(end of ROM)-(ROM size)=PC
        move.l  4(a0),a0            ;Get Initial Program Counter
        subq.l  #2,a0               ;now points to second RESET
        reset                       ;first RESET instruction
        jmp     (a0)                ;CPU Prefetch executes this
        ;NOTE: the RESET and JMP instructions must share a longword!

alerte_coquille
 dc.b 0,150,15,'ATTENTION les vecteurs Reset sont touchés !!!',0,1
 dc.b 0,150,30,"Il s'agit probablement d'un VIRUS COQUILLE",0,1
 dc.b 0,150,45,'Left = Tuer le virus de façon certaine',0,1
 dc.b 0,150,60,'Right = Continuer à vos risques et périls',0,1
 dc.b 0,150,80,"PS:  guru's name is F MAZUE   ( Yeaah !)",0,0

alerte_boot
 dc.b 0,150,15,'ATTENTION les vecteurs Reset sont touchés !!!',0,1
 dc.b 0,150,30,"Il s'agit probablement d'un VIRUS dans le BOOT",0,1
 dc.b 0,150,45,'Left = Tuer le virus de façon certaine',0,1
 dc.b 0,150,60,'Right = Continuer à vos risques et périls',0,1
 dc.b 0,150,80,"PS:  guru's name is F MAZUE   ( Yeaah !)",0,0

repare_rad
 move.l _SysBase,a6
 lea rad_data,a3
 move.l (a3)+,KickMemPtr(a6)
 move.l (a3)+,KickTagPtr(a6)
 move.l (a3),KickCheckSum(a6)
 rts

NoFich
 pea exit
 CALLDOS Output
 move.l #nofich_msg,d2
 move.l #(fin_nofich_msg-nofich_msg),d3
 CALLDOS Write
 rts

NoExec
 pea exit
 CALLDOS Output
 move.l len_file,d3
 move.l file,d2
 CALLDOS Write

 CALLDOS Output
 move.l #noexec_msg,d2
 move.l #(fin_noexec_msg-noexec_msg),d3
 CALLDOS Write
 rts

NoMem
 pea exit
 CALLDOS Output
 beq .exit
 move.l #nomem_msg,d2
 move.l #(fin_nomem_msg-nomem_msg),d3
 CALLDOS Write
.exit
 rts

NoFormat
 pea exit
 CALLDOS Output
 beq .exit
 move.l #nofor_msg,d2
 move.l #(fin_nofor_msg-nofor_msg),d3
 CALLDOS Write
.exit
 rts

NoBoot
 pea exit
 CALLDOS Output
 beq .exit
 move.l #noboot_msg,d2
 move.l #(fin_noboot_msg-noboot_msg),d3
 CALLDOS Write
.exit
 rts
```

```
****************************************
dosname DOSNAME
intname INTNAME
td_name TD_NAME

_DOSBase    dc.l 0
_IntuitionBase dc.l 0
Dummy_A0    dc.b $0A,0
Task        dc.l 0
OldTrap     dc.l 0
Segment     dc.l 0
Tampon      dc.l 0
FlagDisk    dc.b 0,0
Pile        dc.l 0
file        dc.l 0
len_file    dc.l 0
prog        dc.l 0
rad         dc.l 0
rad_data    dcb.l 3,0
portname    dc.b 'monport',0
       even

nomem_msg
dc.b $0A,"Il n'y a pas suffisament de mémoire chip"
dc.b $0A,'pour lire les secteurs boot'
dc.b $0A,$0A,0
fin_nomem_msg
even

nofich_msg
dc.b $0A,"Il n'y a pas d'abonné au numéro"
dc.b $0A,'que vous avez demandé'
dc.b $0A,$0A,0
fin_nofich_msg
even

noexec_msg
dc.b $20,"n'est pas un éxécutable"
dc.b $0A,$0A,0
fin_noexec_msg
even

nofor_msg
dc.b $0A,"Il y a un problème, cette disquette"
dc.b $0A,"ne doit pas être formatée correctement"
dc.b $0A,$0A,0
fin_nofor_msg
even

noboot_msg
dc.b $0A,"Cette disquette n'est pas AutoBoot !!"
dc.b $0A,$0A,0
fin_noboot_msg
even

DiskIO      dcb.b IOTD_SIZE;structure IOrequest étendue
ReplyPort   dcb.b MP_SIZE  ;taille d'un message port
```

```
*--- Programme: virus.a
*--- Fonction: faux virus pour essayer le programme traqueur
*--- Auteur: F MAZUE pour ANT le 16/02/91
*--- Assembleur: Devpac2 (the top !)

 incdir "include:"
 include "exec/execbase.i"

virus
 movea.l $4,a6
 move.l d0,CoolCapture(a6)
 rts
```


CONCOURS LE

Le gagnant du concours de ce mois-ci nous propose un scrolling en "rouleau", comme on peut en voir dans les démos. Une petite routine sympathique, juste de quoi empocher notre DevKit en récompense ! Mais laissons donc la parole à Gérard lui-même.

Voici un scroll overscan d'un bel effet. L'effet de rouleau est rendu par le choix de l'affichage des lignes du plan dans le Copper, à partir d'une table (TD1). Les dégradés de couleurs sont également lus dans deux tables (TD2 et TD3) puis affichés par le Copper. Ceux qui veulent personnaliser le programme peuvent ajouter des couleurs de fond dans la table TD4. Apres avoir créé la liste Copper, le programme lit le texte, affiche la lettre en cours si nécessaire, puis décale le plan-scroll de deux pixels et d'une ligne vers le haut, et enfin recopie la ligne 1 dans la ligne 74.

La vitesse de défilement peut être aisément modifiée en chageant les paramètres 28(a3) et 33(a3) dans la table DONNEES_SCROLL.

Le temps pris par l'exécution du programme est peu important, ce qui permet de l'inclure dans une démo sans problème particulier.

Le fichier FONT.BP est une image au format bitplane RAW (pas d'IFF !) composée d'un plan qui contient des lettres de 32 pixels de large sur 30 de haut, chaque ligne contenant 10 lettres. L'ordre est indiqué dans LISTE1 (voir schéma). Pour utiliser vos propres fontes, il faut respecter ce format ou modifier les données de TABLE ainsi que les données de BLIT1.

```
* CBM 61 le 19-mars-1991 pour ANT *

       section PROG,CODE

       MOVE.B  #$87,$BFD100    ; Stop drive
       MOVE    #$B000,D0   ; And wait
       DBRA    D0,*
       move.l  #run,$80    ; Mode supervisor
       trap    #0
       move.l  #0,d0
       rts

run BSR INIT
       MOVE.L  4,A6
       LEA     GfxName(pc),A1 ; Ouvre Gfx lib
       JSR     -408(a6)    ; Openlibrary(A6)
       MOVE.L  D0,GfxBase
       MOVE.L  Gfxbase(pc),A0
       MOVE.L  $32(A0),oldcopper
       MOVE.L  #copper,$32(A0); Nouveau copper
       LEA     $DFF000,A5
       MOVE.L  $6C,SaveIrq
       MOVE    #$8540,$96(A5)
       MOVE    #$7FFF,$9C(A5)
       MOVE    #$C020,$9A(A5)
       MOVE.L  #MAIN,$6C   ; le prog est dans l'interruption
.WAIT  BTST    #6,$BFE001  ; attend bouton gauche souris
       BNE.S   .WAIT
       LEA     $dff000,a5
       MOVE    #$8540,$96(A5)
       MOVE    #$7FFF,$9C(A5)
       MOVE    #$C020,$9A(A5)
       MOVE.L  SaveIrq(pc),$6C; Restore context
       MOVE.L  GfxBase(pc),A1
       MOVE.L  oldcopper(pc),$32(A1)
       MOVE.L  4,A6
       JSR     -414(a6)    ; CloseLibrary(A6)
       RTE     ; Sort de TRAP #0

SaveIrq        dc.L 0
gfxbase        dc.L 0
oldcopper      dc.L 0
GFXNAME        DC.B 'graphics.library',0
               even

MAIN   MOVEM.L D0/D1/A0/A1/A5/A6,-(A7)
       LEA     $DFF000,A0
       MOVE.L  $4,A6
```

```
        MOVE    $1C(A0),D1
        AND     $1E(A0),D1
.INT2   BTST    #5,D1
        BEQ.S   .fin
        bsr     SCROLL
        MOVEM.L $90(A6),A1/A5
.FIN    PEA     -36(A6)         ; sort interruption
        JMP     (A5)

DONNEES SCROLL
        DC.L    plan+$86c       ; (A3) Adresse pour nouvelle lettre
        DC.L    plan+4          ; 4(A3) Ad source pour decalage 2 pixels
horizontaux
        DC.L    plan+2          ; 8(A3) ad dest pour decalage horizontal
et vertical
        DC.L    plan+$38        ; 12(A3) ad source pour decalage vertical
        DC.L    plan+$F9C       ; 16(A3) ad dest pour decalage vertical
        DC.L    plan            ; 20(a3) Debut plan scroll
        DC.L    plan1           ; 24(A3) AD font
        DC.W    $E9F0           ; 28(A3) Valeur bltcon ou $f9f0 et 36 en 33(a3)
        DC.W    0               ; 30(A3) Compteur pour fin scroll
        DC.B    0               ; 32(A3) Compteur pour nouvelle lettre
        DC.B    18              ; 33(A3) NB de pixels avant new lettre

LISTE1  DC.B 'ABCDEFGHIJKLMNOPQRSTUVWXYZ!., 0123456789'
        EVEN

SCROLL  MOVEM.l D0-A7,-(SP)
        LEA     DONNEES_SCROLL(pc),A3
        MOVE.L  #$FFFFFFFF,$DFF044
        CLR     $DFF042
        ADDQ.B  #1,32(A3)       ; Ajoute 1 compteur pixels
        MOVE.B  33(A3),D3
        CMP.B   32(A3),D3       ; si egal
        BEQ.S   .SUIT           ; nouvelle lettre
        BRA.S   BLIT2           ; autrement scroll
.SUIT   CLR.B   32(A3)
        LEA     TABLE(PC),A6    ; tableau ad lettres
        LEA     TEXTE(PC),A2    ; texte du scroll
        ADD     30(A3),A2       ; lettre actuelle
        MOVE.L  24(A3),A0       ; Ad font
        ADDQ    #1,30(A3)       ; Compteur+1
        CMP.W   #FINTEXTE-TEXTE,30(A3) ; teste si fin texte
        BNE.S   .Suit1          ; non continue
        CLR     30(A3)          ; oui recommence au debut
.Suit1  MOVEQ   #0,D0
        MOVE.B  (A2),D1         ; Met lettre ds D1
        LEA LISTE1,A1           ; Table lettres pour comparer
.CHER   MOVE.B  (A1),D2         ; Prend une lettre
        CMP.B   D1,D2           ; est-ce celle la ?
        BEQ.S   .OK             ; Trouve !!
        ADDQ.B  #1,D0           ; Position de la lettre pour calcul
        LEA     1(A1),A1        ; Incremente pour lettre suivante
        BRA.S   .CHER           ; continue de chercher
.OK     ROL     #1,D0           ; multiplie par 2
        ADD     D0,A6           ; OFFSET lettre dans table
        ADD     (A6),A0         ; AD lettre dans font
BLIT1   MOVE.L  A0,$DFF050      ; tansfert une nouvelle lettre
        MOVE.L  (A3),$DFF054
        MOVE.L  #$240032,$DFF064    ; Modulo font,plan
        MOVE    #$9f0,$DFF040
        MOVE    #30*64+2,$dff058
BLIT2   MOVE.L  4(A3),$DFF050   ; Fait avancer le scroll de 2 pixels
        MOVE.L  8(A3),$DFF054
        MOVE.L  #$020002,$DFF064
        MOVE    28(A3),$DFF040
        MOVE    #75*64+26,$dff058
BLIT3   MOVE.L  12(A3),$DFF050 ; et decale une ligne vers le haut
        MOVE.L  8(A3),$DFF054
        MOVE.L  #$060006,$DFF064
        MOVE    #$9f0,$DFF040
        MOVE    #74*64+24,$dff058
BLIT4   MOVE.L  20(A3),$DFF050 ; Permet a la ligne qui sort en haut
        MOVE.L  16(A3),$DFF054 ; de l'ecran de revenir en bas
        CLR.L   $DFF064
        MOVE    #1*64+27,$dff058    ; Blit 1 ligne
        MOVEM.l (SP)+,D0-A7
        RTS
L1=1320
L2=2560
L3=3800
L4=5000

; Valeurs a partir de 'PLAN' des lettres
TABLE   DC.W 0,4,8,12,16,20,24,28,32,36
        DC.W L1+0,L1+4,L1+8,L1+12,L1+16,L1+20,L1+24,L1+28,L1+32,L1+36
        DC.W L2+0,L2+4,L2+8,L2+12,L2+16,L2+20,L2+24,L2+28,L2+32,L2+36
        DC.W L3+0,L3+4,L3+8,L3+12,L3+16,L3+20,L3+24,L3+28,L3+32,L3+36
        DC.W L4+0,L4+4,L4+8,L4+12,L4+16,L4+20,L4+24,L4+28,L4+32,L4+36
TEXTE   DC.B ' HI!  HERE IS   CBM 61   PROUDLY  PRESENT HIS  ANT INTRO'
FINTEXTE EVEN

; Donnees des longs de lignes 54=1 ligne 108=2 etc.
        DC.W  54,54
```

Disposition de la fonte :


```
TD1     DC.W    162,108,108,108,54,108,54,54,108   ; Pointe la ligne
        DCB     30,54                              ; desiree avec le copper
        DC.W    108,54,108,54,108,108,108,162,270
TD5     DC.W    54,54
TD2     DC.W    $A80,$A80,$Ba0,$Ba0,$DB0,$DB0,$DC0,$DC0,$EE0,$EE0
        DCB     30,$FF0
        DC.W    $EE0,$EE0,$DC0,$DC0,$DB0,$DB0,$BA0,$BA0,$A80,$A80
TD3     DC.W    $A00,$A00,$900,$900,$800,$800,$700,$700,$600,$600
        DCB     30,$500
        DC.W    $600,$600,$700,$700,$800,$800,$900,$900,$A00,$A00

; Couleurs du fond pour donner votre "look" personnel
TD4     DCB     50,0

INIT    LEA     COPPERA,A0 ; Pointe liste copper
        MOVE    #49,D0     ; Nombre de lignes -1
        MOVE    #$8001,D1  ; Numero depart
        LEA     PLAN,A1    ; Ad depart
        MOVE.L  A1,D3
        MOVE.L  D3,D4
        ADD.L   #$144,D3   ; Premiere ligne $144=54*6
        ADD.L   #$f66,D4   ; Derniere ligne $f66=54*73
        LEA     TD1,A1     ; Table donnees des lignes endroit
        LEA     TD2,A2     ; Couleurs endroit
        LEA     TD3,A3     ; Couleurs envers
        LEA     TD4,A4     ; Couleurs fond ,pas utilise ici
        LEA     TD5,A5     ; Table donnees des lignes envers
.1      MOVE    D1,(A0)+
        ADDI    #$0100,D1
        MOVE    #$FFFE,(A0)+
        MOVE    #$E2,(A0)+
        MOVE    D3,(A0)+
        ADD     (A1)+,D3
        MOVE    #$E6,(A0)+
        MOVE    D4,(A0)+
        SUB     -(A5),D4
        MOVE    #$182,(A0)+
        MOVE    (A2),(A0)+
        MOVE    #$184,(A0)+
        MOVE    (A3)+,(A0)+
        MOVE    #$186,(A0)+
        MOVE    (A2)+,(A0)+; Force la couleur jaune si 2 plans
        MOVE    #$180,(A0)+
        MOVE    (A4)+,(A0)+
        DBF     D0,.1
        MOVE.L  #PLAN,D0    ; Valide plan dans copper
        SWAP    D0
        LEA     AD_SCROLL,A1
        MOVE    D0,2(A1)
        MOVE    D0,6(A1)
        RTS

        section data,code_c

COPPER  DC $92,$18,$94,$D8,$108,4,$180,0,$10A,4,$120,0,$122,0
        DC $8E $8041,$90,$30D1,$104,0
AD_SCROLL:
        DC $E0,0,$E4,0,$100,$2200

COPPERA DS 14*50
        DC $100,$200,$ffff,$fffe

PLAN    DS.B    $1040
PLAN1   INCBIN  FONT.BP
        END
```

par Gérard GARCIA

ExecBase II : LE 68000 de A à Z

Voici l'avant dernière partie de notre étude des instructions du 68000. Reportez-vous aux deux précédents numéros de l'ANT pour plus de détails sur la manière de lire les tableaux.

Le mois prochain, nous entamerons une petite mais courte série sur l'utilité et l'utilisation de certaines instructions obscures du 68000, comme LINK, CHK, RTR ou encore DBPL. Mais ceci est une autre histoire...

CMPA (comparaison d'adresse)

```
Syntaxe CMPA <ea>,An
Taille  Mot, mot long

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         1  0  1  1 REGISTRE OP-MODE  ADRESSE EFFECTIVE

        REGISTRE : numéro du registre d'adresse (0-7)
        OP-MODE : 011=mot, 111=mot long

Flags   N : 1 si résultat négatif, 0 sinon
        Z : 1 si résultat nul, 0 sinon
        V : 1 si débordement, 0 sinon
        C : 1 si retenue, 0 sinon
        X : inchangé
```

CMPI (comparaison immédiate)

```
Syntaxe CMPI #donnée,<ea>
Taille  Octet, mot, mot long

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  0  0  0  1  1  0  0 TAILLE ADRESSE EFFECTIVE

        TAILLE : 00=octet, 01=mot, 10=mot long
        La donnée est codée dans le mot (TAILLE=00 ou 01) ou dans le
        mot long (TAILLE=10) suivant le code opératoire.

Flags   N : 1 si résultat négatif, 0 sinon
        Z : 1 si résultat nul, 0 sinon
        V : 1 si débordement, 0 sinon
        C : 1 si retenue, 0 sinon
        X : inchangé
```

CMPM (comparaison mémoire)

```
Syntaxe CMPM (Ay)+,(Ax)+
Taille  Octet, mot, mot long

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         1  0  1  1    Ax     1 TAILLE 0  0  1    Ay

        Ax : registre source
        Ay : registre destination
        TAILLE : 00=octet, 01=mot, 10=mot long

Flags   N : 1 si résultat négatif, 0 sinon
        Z : 1 si résultat nul, 0 sinon
        V : 1 si débordement, 0 sinon
        C : 1 si retenue, 0 sinon
        X : inchangé
```

DBcc (test, décrémentation et branchement)

```
Syntaxe DBcc Dn,<label>
Taille  Mot

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  1  0  1  CONDITION   1  1  0  0  1 REGISTRE

        REGISTRE : numéro du registre de données (0-7)
        Le déplacement (16 bits) est codé dans le mot suivant le code opératoire.

Flags   inchangés
```

DIVS (division signée)

```
Syntaxe DIVS <ea>,Dn
Taille  Mot

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         1  0  0  0 REGISTRE  1  1  1 ADRESSE EFFECTIVE

        REGISTRE : numéro du registre de données (0-7)

Flags   N : 1 si quotient    négatif, 0 sinon. Indéfini si V=1
        Z : 1 si quotient nul, 0 sinon. Indéfini si V=1
        V : 1 si débordement, 0 sinon
        C : toujours 0
        X : inchangé
```

DIVU (division non signée)

```
Syntaxe DIVU <ea>,Dn
Taille  Mot

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         1  0  0  0 REGISTRE  0  1  1 ADRESSE EFFECTIVE

        REGISTRE : numéro du registre de données (0-7)

Flags   N : 1 si quotient négatif, 0 sinon. Indéfini si V=1
        Z : 1 si quotient nul, 0 sinon. Indéfini si V=1
        V : 1 si débordement, 0 sinon
        C : toujours 0
        X : inchangé
```

EOR (ou exclusif)

```
Syntaxe EOR Dn,<ea>
Taille  Octet, mot, mot long

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         1  0  1  1 REGISTRE OP-MODE  ADRESSE EFFECTIVE

        REGISTRE : numéro du registre de données (0-7)
        OP-MODE : 100=octet, 101=mot, 110=mot long

Flags   N : 1 si résultat négatif, 0 sinon
        Z : 1 si résultat nul, 0 sinon
        V : toujours 0
        C : toujours 0
        X : inchangé
```

EORI (ou exclusif immédiat)

```
Syntaxe EORI #donnée,<ea>
Taille  Octet, mot, mot long

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  0  0  0  1  0  1  0 TAILLE ADRESSE EFFECTIVE

        TAILLE : 00=octet, 01=mot, 10=mot long
        La donnée est codée dans le mot (TAILLE=00 ou 01) ou dans le
        mot long (TAILLE=10) suivant le code opératoire.

Flags   N : 1 si résultat négatif, 0 sinon
        Z : 1 si résultat nul, 0 sinon
        V : toujours 0
        C : toujours 0
        X : inchangé
```

EORI to CCR (ou exclusif immédiat avec CCR)

```
Syntaxe EORI #donnée,CCR
Taille  Octet

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  0  0  0  1  0  1  0  0  0  1  1  1  1  0  0

        La donnée (8 bits) est codée dans le mot suivant le code opératoire.

Flags   N : 1 si le bit 3 de la donnée est 1, inchangé sinon
        Z : 1 si le bit 2 de la donnée est 1, inchangé sinon
        V : 1 si le bit 1 de la donnée est 1, inchangé sinon
        C : 1 si le bit 0 de la donnée est 1, inchangé sinon
        X : 1 si le bit 4 de la donnée est 1, inchangé sinon
```

EORI to SR (ou exclusif immédiat avec SR)

```
Instruction privilégiée.

Syntaxe EORI #donnée,SR
Taille  Mot

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  0  0  0  1  0  1  0  0  1  1  1  1  1  0  0

        La donnée (16 bits) est codée dans le mot suivant le code opératoire.

Flags   N : 1 si le bit 3 de la donnée est 1, inchangé sinon
```

```
        Z : 1 si le bit 2 de la donnée est 1, inchangé sinon
        V : 1 si le bit 1 de la donnée est 1, inchangé sinon
        C : 1 si le bit 0 de la donnée est 1, inchangé sinon
        X : 1 si le bit 4 de la donnée est 1, inchangé sinon
```

EXG (échange de registres)

```
Syntaxe EXG Rx,Ry
Taille  Mot long

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         1  1  0  0    Rx     1    OP-MODE        Ry

        OP-MODE   :   01000=registres   de   données,   01001=registres
d'adresse, 10001=registre de données et registre d'adresse
        Rx  :  numéro  du  premier  registre  (registre  de  donnée  si
OP-MODE=10001)
        Ry : numéro du second registre

Flags   Inchangés
```

EXT (extension de signe)

```
Syntaxe EXT Dn
Taille  Mot, mot long

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  1  0  0  1  0  0  OP-MODE  0  0  0 REGISTRE

        OP-MODE : 010=extension au mot, 011=extension au mot long
        REGISTRE : numéro du registre de données (0-7)

Flags   N : 1 si résultat négatif, 0 sinon
        Z : 1 si résultat nul, 0 sinon
        V : toujours 0
        C : toujours 0
        X : inchangé
```

ILLEGAL (instruction illégale)

```
Syntaxe ILLEGAL
Taille  -

Format  $4AFC

Flags   Non affectés

Notes   cette instruction génère une exception de vecteur 4.
```

JMP (saut inconditionnel)

```
Syntaxe JMP <ea>
Taille  -

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  1  0  0  1  1  1  0  1  1 ADRESSE EFFECTIVE

Flags   Inchangés
```

JSR (saut à un sous programme)

```
Syntaxe JSR <ea>
Taille  -

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  1  0  0  1  1  1  0  1  0 ADRESSE EFFECTIVE

Flags   Inchangés
```

LEA (chargement d'adresse effective)

```
Syntaxe LEA <ea>,An
Taille  Mot long

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  1  0  0 REGISTRE  1  1  1 ADRESSE EFFECTIVE

        REGISTRE : numéro du registre d'adresse (0-7)
Flags   Inchangés
```

LINK (création de liens avec la pile)

```
Syntaxe LINK An,#donnée
Taille  -

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  1  0  0  1  1  1  0  0  1  0  1  0 REGISTRE

        REGISTRE : numéro du registre d'adresse (0-7)
Flags   Inchangés
```

LSL, LSR (décalage logique)

```
Syntaxe LSx Dx,Dy
        LSx #donnée,Dy
        LSx <ea>
        X=L (Left, gauche) ou R (Right, droite)

Format  Décalage d'un registre :
        15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         1  1  1  0  NOMBRE   D TAILLE I  0  1 REGISTRE

        Décalage d'une adresse :
        15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         1  1  1  0  0  0  1  D  1  1 ADRESSE EFFECTIVE

        NOMBRE : nombre de décalages ou numéro du registre de données
        le contenant (0-7)
        D : 0=décalage à gauche, 1=décalage à droite
        TAILLE : 00=octet, 01=mot, 10=mot long
        I : 0=NOMBRE contient le nombre de décalages (0-7, 0=8)
            1=NOMBRE indique le registre contenant le nombre de
décalages (modulo 64)

Flags   N : 1 si résultat négatif, 0 sinon
        Z : 1 si résultat nul, 0 sinon
        V : toujours 0
        C : 1 si le dernier bit sorti de l'opérande était 1
        X : idem C
```

MOVE (transfert de données)

```
Syntaxe MOVE <ea>,<ea>
Taille  Octet, mot, mot long

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  0 TAILLE ADRESSE EFFECTIVE ADRESSE EFFECTIVE

        TAILLE : 00=octet, 01=mot, 10=mot long

Flags   N : 1 si résultat négatif, 0 sinon
        Z : 1 si résultat nul, 0 sinon
        V : toujours 0
        C : toujours 0
        X : inchangé
```

MOVE to CCR (transfert vers CCR)

```
Syntaxe MOVE <ea>,CCR
Taille  Octet

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  1  0  0  0  1  0  0  1  1 ADRESSE EFFECTIVE

Flags   N : 1 si le bit 3 de la source est 1, 0 sinon
        Z : 1 si le bit 2 de la source est 1, 0 sinon
        V : 1 si le bit 1 de la source est 1, 0 sinon
        C : 1 si le bit 0 de la source est 1, 0 sinon
        X : 1 si le bit 4 de la source est 1, 0 sinon
```

MOVEA (transfert vers un registre d'adresse)

```
Syntaxe MOVEA <ea>,An
Taille  Mot, mot long

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  0 TAILLE REGISTRE 0  0  1 ADRESSE EFFECTIVE

        TAILLE : 11=mot, 10=mot long

Flags   Inchangés
```

MOVE USP (transfert de ou vers USP)

```
Instruction privilégiée

Syntaxe MOVE USP,An
        MOVE An,USP
Taille  Mot long

Format  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
         0  1  0  0  1  1  1  0  0  1  1  0  D REGISTRE

        D : 0=registre vers USP, 1=USP vers registre
        REGISTRE : numéro du registre d'adresse (0-7)

Flags   Inchangés
```

MOVE from SR (transfert depuis SR)

Instruction privilégiée sur les 68010, 68020 et 68030

Syntaxe MOVE SR,<ea>
Taille Mot

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 0 0 0 0 1 1 ADRESSE EFFECTIVE

Flags Inchangés

MOVE to SR (transfert vers SR)

Instruction privilégiée

Syntaxe MOVE <ea>,SR
Taille Mot

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 0 1 1 0 1 1 ADRESSE EFFECTIVE

Flags N : 1 si le bit 3 de l'opérande est 1, 0 sinon
Z : 1 si le bit 2 de l'opérande est 1, 0 sinon
V : 1 si le bit 1 de l'opérande est 1, 0 sinon
C : 1 si le bit 0 de l'opérande est 1, 0 sinon
X : 1 si le bit 4 de l'opérande est 1, 0 sinon

MOVEM (transfert multiple)

Syntaxe MOVEM registres,<ea>
Taille Mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 1 D 0 0 1 T ADRESSE EFFECTIVE

D : 0=registres vers mémoire, 1=mémoire vers registres
T : 0=mot, 1=mot long
Le masque des registres concernés est codé dans le mot suivant le code opératoire

Flags Inchangés

MOVEP (transfert vers un périphérique)

Syntaxe MOVEP Dx,d(Ay)
MOVEP D(Ay),Dx
Taille Mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 0 0 0 REGISTRE OP-MODE 0 0 1 REGISTRE

OP-MODE : 100=mot mémoire vers registre, 101=mot long mémoire vers registre, 110=mot registre vers mémoire, 111=mot long registre vers mémoire.
Le déplacement (16 bits) est codé dans le mot suivant le code opératoire.

Flags Inchangés

MOVEQ (transfert immédiat rapide)

Syntaxe MOVEQ #donnée,Dx
Taille Octet, mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 1 1 REGISTRE 0 DONNEE 8 BITS

REGISTRE : numéro du registre de donnée (0-7)
DONNEE : donnée signée sur 8 bits

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : toujours 0
C : toujours 0
X : inchangé

MULS (multiplication signée)

Syntaxe MULS <ea>,Dn
Taille Mot

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1 1 0 0 REGISTRE 1 1 1 ADRESSE EFFECTIVE

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : toujours 0
C : toujours 0
X : inchangé

MULU (multiplication non signée)

Syntaxe MULU <ea>,Dn
Taille Mot

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1 1 0 0 REGISTRE 0 1 1 ADRESSE EFFECTIVE

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : toujours 0
C : toujours 0
X : inchangé

NBCD (complémentation décimale avec X)

Syntaxe NBCD <ea>
Taille Octet

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 1 0 0 0 0 0 ADRESSE EFFECTIVE

Flags N : indéfini
Z : 0 si résultat non-nul, inchangé sinon
V : indéfini
C : 1 si retenue, 0 sinon
X : idem C

NEG (négation)

Syntaxe NEG <ea>
Taille Octet, mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 0 1 0 0 TAILLE ADRESSE EFFECTIVE

TAILLE : 00=octet, 01=mot, 10=mot long

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : 1 si débordement, 0 sinon
C : 1 si retenue, 0 sinon
X : idem C

NEGX (négation avec X)

Syntaxe NEGX <ea>
Taille Octet, mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 0 0 0 0 TAILLE ADRESSE EFFECTIVE

TAILLE : 00=octet, 01=mot, 10=mot long

Flags N : indéfini
Z : 1 si résultat non-nul, inchangé sinon
V : indéfini
C : 1 si retenue, 0 sinon
X : idem C

NOP (pas d'opération)

Syntaxe NOP
Taille -

Format $4E71

Flags Inchangés

NOT (complémentation à 1)

Syntaxe NOT <ea>
Taille Octet, mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 0 1 1 0 TAILLE ADRESSE EFFECTIVE

TAILLE : 00=octet, 01=mot, 10=mot long

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : toujours 0
C : toujours 0
X : inchangé

TOOLBOX II : Combien ça fait ?

Délaissant ce mois-ci l'étude des langages de programmation par l'intermédiaire des logiciels du Domaine Public, nous allons nous intéresser à l'un des outils les plus indispensables pour un programmeur, j'ai nommé la calculatrice.

En effet, malgré toute la puissance de vos chères machines, chacun d'entre vous a dû avoir besoin, à un moment ou à un autre, d'effectuer un calcul d'expressions ou une conversion de nombres en binaire ou hexa lors du développement d'un de ses programmes. Et pour ce faire, rien ne vaut une calculatrice. Celle fournie en standard avec l'Amiga est certes très fonctionnelle, mais elle manque singulièrement de capacités dès que l'on veut effectuer autre chose qu'une addition ou une multiplication. J'ai donc parcouru pour vous ma collection de logiciels du domaine public, afin de vous proposer une sélection de trois calculatrices dont les fonctionnalités sont complémentaires. Nous commencerons cette étude par un outil très performant pour la conversion multiformat et la manipulation d'expressions logiques.

Hexalator V1.00

Une des premières utilisations de cette calculatrice est de permettre la conversion de format d'un nombre entier quelconque. Les formats de conversions sont des plus classiques puisqu'il s'agit de Binaire, Octal, Décimal et Héxadécimal. Pour convertir un nombre, rien de plus simple, puisqu'il suffit de sélectionner le format d'entrée, de taper le nombre et de sélectionner le format de sortie.
La deuxième classe de fonctions disponible est également très commune puisqu'il s'agit des quatres opérateurs "+","-","*","/". Je ne m'étendrai pas sur l'utilisation de ces opérateurs, mais rappellerai simplement que cette calculatrice utilise la notation polonaise inversée, chère à Hewlett Packard, qui risque de dérouter les personnes habituées à la notation classique.
Dernière série d'opérateurs utilisables sur cette calculatrice, les opérateurs logiques. On trouvera les opérateurs suivant "AND" (et logique) "OR" (ou logique) "XOR" (ou exclusif) et enfin le seul opérateur unaire "NOT". On regrettera l'abscence du "NAND" dont l'utilité est plus qu'évidente en logique.

Enfin, on trouve quatre autres touches permettant de rappeler les derniers nombres saisis, d'effectuer un décalage à gauche (pour corriger une éventuelle erreur de frappe) et d'échanger X et Y lors d'opérations.
En conclusion, il s'agit d'une bonne petite calculatrice logique qui mérite de figurer à portée de main en permanence.

HP 11C

La calculatrice suivante ravira les habitués de la notation polonaise inverse, puisque je vous propose une copie conforme de la très célèbre HP-11C de la firme HEWLETT PACKARD. Pour ceux qui ne connaitraient pas ce type de notation, un petit rappel s'impose.
La notation polonaise inverse est basée sur le principe d'empilage et de dépilage des données et des opérateurs. Afin de vous faire mieux comprendre la différence entre cette notation et la notation arithmétique classique, voici quelques exemples.

Pour réaliser les opérations suivantes :

```
5 + 3          vous tapez 5 3 +
5 * (2 + 4)    vous tapez 5 2 4 + *
2 * sin(0.5)   vous tapez 0.5 sin 2 *
```

Lorsque vous désirez saisir deux nombres consécutivement, vous devez les séparer en tapant <ENTER>. Ce qui donne pour les exemples ci-dessus :

```
5 <ENTER> 3 +
5 <ENTER 2 <ENTER> 4 + *
0.5 SIN 2 *
```

(dans ce dernier cas, on ne tape pas <ENTER> puisque l'on n'a pas deux nombres qui se suivent).

Dans le cas de notre calculatrice, vous avez la possibilité d'empiler au maximum 4 données ou résultats d'opérations successivement. Cette limitation est parfois gênante, mais comme vous le verrez vite en l'utilisant, on arrive toujours à passer outre.
Nous n'allons bien entendu pas passer en revue l'ensemble des fonctions de cette calculatrice, l'article au complet n'y suffirait pas. Néanmoins, j'aimerais vous signaler quelques points importants. D'abord, on distinguera deux modes de fonctionnement : au démmarage, la HP 11C se trouve en mode calcul. Vous pouvez ainsi taper toutes les expressions que vous désirez. Les fonctions alternées pour chaque touche sont accessibles par l'appui sur "f" ou "g" suivi de la fonction à exécuter. En sélectionnant la fonction "P/R", on passe la calculatrice en mode programmation, et il est alors possible de réaliser un programme quelconque avec le jeu d'instructions de la calculatrice. J'avoue que c'est assez amusant.

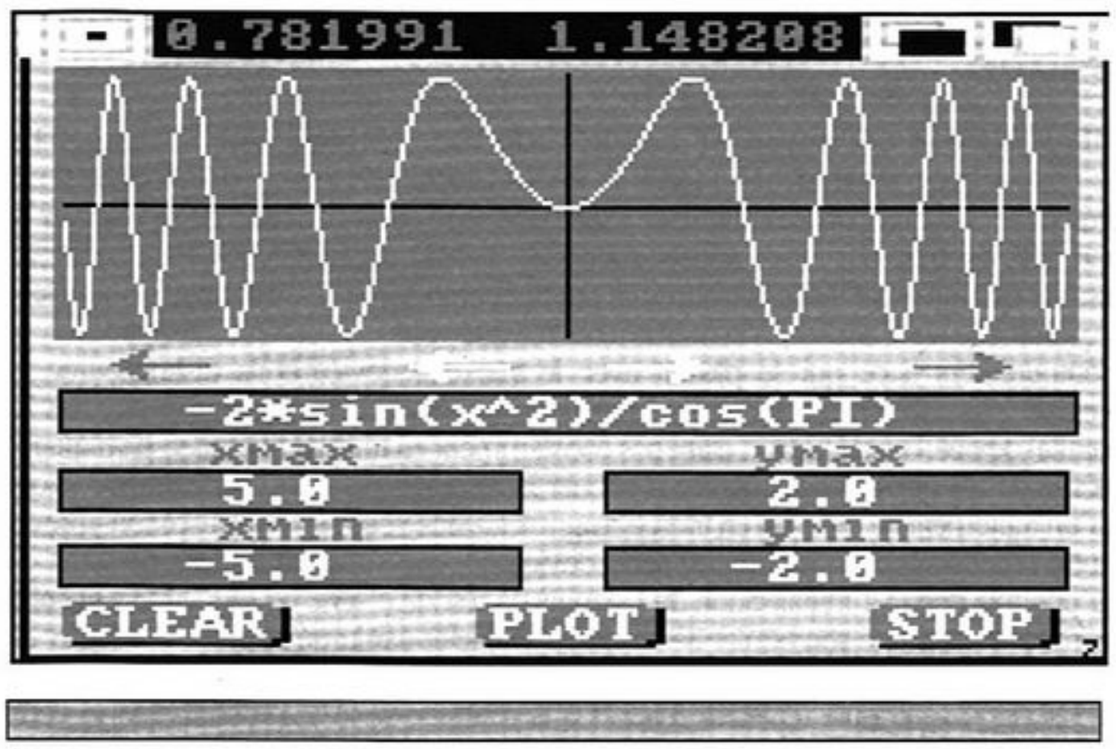
Pour le reste, je vous encourage à consulter la documentation (en anglais) jointe au logiciel. Sont égalements présentes sur la disquette, les sources du logiciel que je vous conseille d'étudier, c'est très bien fait et très instructif. Dernier point avant de passer à l'étude de la dernière calculette, la plupart des fonctions sont accessibles directement au clavier, ce qui est beaucoup plus rapide qu'à la souris.

Calc V 3.0

Pour les réfractaires à la notation polonaise inverse, j'ai sélectionné une calculatrice qui représente à mon avis le must dans ce domaine. En effet, elle comporte trois modes de traitement spécifiques.
Le mode "scientific" comprend toutes les fonctionnalités d'une vrai calculatrice du commerce.
En mode "programmer", elle permet la manipulation logique et la conversion de nombres en hexadécimal, décimal, octal et binaire (tous les opérateurs classiques y sont). Ces opérations peuvent s'effectuer sur 8, 16 ou 32 bits de précision.
Enfin, en mode "graph", elle permet de tracer rapidement l'allure d'une fonction saisie dans le champ approprié, par exemple -2*sin(x/2)*cos(x^2), dans un intervalle de votre choix. Une autre possibilité offerte par ce mode est le suivi du contour de cette courbe, afin d'obtenir la valeur en un point donné de la courbe.

A noter pour terminer que cette calculatrice est iconifiable et que la documentation qui l'accompagne est très complète et dans un anglais abordable.
Le mois prochain, nous reprendrons notre tour d'horizon des langages de programmation du domaine public avec l'AMXLisp, un langage LISP orienté objets dont vous parlera mon éminent confrère, spécialiste en Intelligence Artificielle, Patrick Chazé.

Herr Doktor Von GlutenStimmelImDorf
Domaines publics en gros, demi-gros et détail



SubWAY : Y'A DE LA RESOURCE (II)

Résumons-nous : la disk.resource et la cia.resource n'ont plus de secrets pour vous (ou si peu). Il est donc maintenant temps d'aller voir du côté des ports série, parallèle et joystick ce qui s'y passe.

Avant toute chose, je tiens à saluer le hasard, qui pour une fois fait bien les choses : le mois dernier, je vous parle rapidement des CIAs 8520, en précisant que le but de l'article n'était pas de faire un cours sur ce composant, et voilà-t-y pas que toc ! une nouvelle série naît dans l'ANT, écrite par mon ami Loïc Far et dédiée au Hardware, qui commence par une étude de ces fameux 8520... Si ça continue comme ça, je vais finir par croire en Dieu.
Mais refermons la parenthèse et rouvrons celle des resources. Quelques lecteurs assidus m'ont téléphoné pour se plaindre du passage un tantinet "rapide" sur la disk.resource, car ils auraient bien aimé apprendre à programmer efficacement les drives de leur Amiga, histoire de faire des démos qui se chargent vite et bien.

Alors d'abord, je n'ai pas honte d'avouer que moi, je ne sais pas comment on les programme directement, les drives. Désolé, mais je préfère et de loin utiliser ce fabuleux système que les gens de chez Commodore-Amiga nous ont pondu. Chacun son truc.
De plus, au sein d'une démo, on n'a pas besoin des resources : on attaque directement le hard sans demander son avis à personne. De toute façon, vu que l'on rend rarement la main au système après, on se fout un peu de tout remettre en état. Sinon, il reste toujours le Reset (le premier qui me saute directement en Rom, je te me le... Bref, j'en fais de la confiture pour routiers sentimentaux).
NDLR : La nouvelle rubrique "Hardware" s'attaquera très bientôt à ce passionnant sujet que sont les lecteurs de disquette.

LA MISC.RESOURCE

Celle-ci est réservée à ceux qui ont besoin d'une vitesse élevée et constante pour leurs transmissions série et/ou parallèles. Le "misc" dans "misc.resource" est l'abréviation de "miscellaneous", que nous traduirons approximativement par "accessoire".
Après avoir ouvert cette resource à l'aide de OpenResource() (cf. mois dernier), deux routines sont mises la disposition du programmeur :
MR_ALLOCMISCRESOURCE() et MR_FREEMISCRESOURCE(). Jeu : devinez à quoi elles servent...
Que peut-on donc faire avec cette resource ? C'est très simple : on s'approprie l'usage des ports parallèle et série ainsi que de leurs différents bits de contrôle. Un petit rappel au passage, l'interface série RS-232 fait partie de Paula (registres custom SERPER et SERDAT) et elle est contrôlée par le port A du CIA-B (adresse $BFD000) ; l'interface Centronics parallèle est incluse dans le port B du CIA-A pour les bits de données et est également contrôlée par le port B du CIA-B. Oui, je sais : c'est un véritable bordel. Mais avec un peu de patience et beaucoup de méthode, on y arrive.

La routine MR_ALLOCMISCRESOURCE attend pour paramètre dans d0, le code de la resource à allouer. On trouve les différents codes possibles dans le fichier Include "resources/misc.i" :

```
MR_SERIALPORT       EQU     0
MR_SERIALBITS       EQU     1
MR_PARALLELPORT     EQU     2
MR_PARALLELBITS     EQU     3
```

MR_SERIALPORT demande l'accès à SERPER et SERDAT, MR_SERIALBITS au CIA-A, MR_PARALLELPORT et PARALLEL_BITS aux CIA-A et B. Il est préférable, lorsque l'on désire utiliser l'interface série et/ou parallèle, de se réserver le port ET les bits, par deux appels successifs à MR_ALLOCMISCRESOURCE, histoire qu'une autre tâche ne se trouve pas bien embêtée si d'aventure elle s'amusait à vouloir elle aussi accéder à la même resource que vous.
Contrairement à ce qui se passait avec le trackdisk.device et la disk.resource, il n'existe pas forcément de conflit entre la misc.resource et le serial.device ou le parallel.device, ces derniers n'étant pas constamment en train de travailler. Il va quand même de soi que si vous essayez d'allouer la resource parallèle, par exemple, alors qu'un traitement de texte est en train d'imprimer, vous vous verrez refuser son accès.
Autre point à signaler, un bug du serial.device fait que s'il a été sollicité ne serait-ce qu'une fois depuis l'initialisation de l'Amiga, la resource série ne sera jamais libérée, même s'il a terminé son boulot depuis belle-lurette et gai-luron réunis. Pour régler ce problème, le RKM "Libraries & Devices" propose d'utiliser la routine FlushDevice() ci-dessous, avant toute tentative d'accès à la resource série. La routine que je vous présente est la version assembleur de celle publiée dans le RKM, page 907.

```
;
; FlushDevice.s
; Pour demander gentiment au serial.device (ou un autre !)
; de libérer ce qu'il a alloué (mémoire, resources...).
;
; En entrée : a1 pointe le nom du device à virer
; En sortie : rien
;
            INCLUDE   "exec/execbase.i"
            INCLUDE   "exec/exec_lib.i"

FlushDevice:
            CALLEXEC  Forbid
            movea.l   a6,a0
            lea       DeviceList(a0),a0
            CALLEXEC  FindName
            tst.l     d0
            beq.s     NotFound
            movea.l   d0,a1
            CALLEXEC  RemDevice
NotFound    CALLEXEC  Permit
            rts
```

Ainsi, avant que d'essayer d'ouvrir la misc.resource dans le but de vous approprier l'interface sérielle, il vous faudra faire :

```
            INCLUDE   "devices/serial.i"

            ...
            lea       sdname(pc),a1
            bsr       FlushDevice
            ...

sdname      SERIALNAME
            even
```

Ce qui n'est tout de même pas grand chose, non ?

LA POTGO.RESOURCE

Celle-là, elle se charge de gérer les... ports souris et joystick ! En fait, on l'utilise lorsqu'un contrôleur proportionnel, par exemple un paddle ou un joystick analogique, est connecté à l'un de ces ports. Là encore, vous ne m'en voudrez pas, mais le but de cet article n'est pas de faire un court complet sur les joysticks analogiques.
Grâce à la potgo.resource, on peut accéder directement et sans danger aux registres hardware POT0DAT ($DFF012), POT1DAT ($DFF014), POTGO ($DFF034), POTGOR ($DFF016) et au port A du CIA-A ($BFE001).
La potgo.resource dispose de trois fonctions pour s'approprier certains bits de ces registres. POTGOR pouvant contrôler à la fois le port gauche et le port droit, il en faut en effet préciser à la resource quels bis nous intéressent.

```
bits = AllocPotBits(mask)
d0                  d0
```

Cette fonction tente donc de réserver les bits décrits dans le paramètre "mask". En retour, "bits" doit être égal à "mask", sinon c'est que quelque chose s'est mal passé.

```
FreePotBits(bits)
d0
```

Devinez quoi ? Cette fonction libère les bits alloués avec la précédente, oui. Le paramètre "bits" est évidemment celui renvoyé par AllocPotBits().

```
WritePotGo(data, bits)
d0     d1
```

Enfin, voici la routine qui vous permettra une valeur quelconque dans le registre hardware POTGO. Le paramètre "data" contient le mot de 16 bits à y écrire, et le paramètre "bits" est celui renvoyé par AllocPotBits. La raison en est que cette valeur servira de masque au système, pour être sûr que nous n'attaquiez que les bits qui vous sont réservés.

C'EST FINI

Bon, on arrêtera là les dégâts. Pour être sincère, je n'ai abordé ces deux dernières resources que par soucis d'exhaustivité... Il est clair, je pense, qu'en dehors des deux CIAs, il vaut mieux laisser les devices s'occuper du hardware. C'est nettement plus facile et, au moins, sans danger !

Par MAX

EXECBASE III : LE DOS & SES DEVICES

Une des possibilités les plus intéressantes de l'AmigaDOS est de permettre la définition de périphériques logiques destinés à faciliter la progression de l'utilisateur dans les méandres de ses répertoires.

Tous ceux qui programment le système de l'Amiga le savent maintenant : il n'y existe qu'une seule adresse fixe, celle du pointeur sur la structure ExecBase. Toutes les autres structures importantes dont allouées dynamiquement à l'initialisation de la machine et peuvent donc se trouver n'importe où en mémoire, suivant la configuration actuelle (extensions, périphériques...). C'est pourquoi Exec est entièrement construit autour du principe fort simple mais très astucieux des listes et des noeuds : pour un type donné de structure, chacune indique l'adresse de sa "fille" (la suivante) et éventuellenemt, dans le cas des noeuds, celle de sa "mère" (la précédente). Le seul inconvénient de ce système est que si la liste n'est pas triée suivant au moins un critère, il faut la parcourir entièrement pour y retrouver un élément donné. Ceux qui ont déjà tâté des listes chaînées et autres arbres binaires savent de quoi je veux parler.
L'AmigaDOS, malgré le handicap certain que lui procure sa parenté BCPL, est construit de la même manière. Dans le cas qui nous intéresse, celui des devices, la structure DosLibrary contient un pointeur le premier de la liste, qui pointe sur le second, lui-même pointant sur le troisième, etc.
Arrivé ici, il convient de faire la distinction entre les devices au sens Exec et les devices au sens AmigaDOS. Pour le premier, un device est une couche logicielle (un ensemble de routines) destinée à gérer une partie du hardware : trackisk.device, audio.device; etc.

AmigaDOS quant à lui distingue trois types de devices :
- *les physiques*. Ce sont les lecteurs de disquette DFx:, l'interface sérielle SER: et parallèle PAR:, etc. ;
- *les logiques*. Ce sont les répertoires qui ont été assignés avec la commande Assign, comme LIBS: ou S: ;
- *les volumes*. Il s'agit ni plus ni moins que du nom réel de la disquette et/ou du disque dur, comme Workbench:, Work: ou Extras:.

La relation entre les deux types est que les devices AmigaDOS font appel aux devices Exec. Ainsi, DF0: utilise le trackdisk.device, DH0: le hddisk.device ou le scsi.device... Certains font plutôt appel aux Handlers que l'on trouve dans le répertoire L: de la disquette Workbench : le Aux-Handler pour AUX:, le Speak-Handler pour Speak:, etc. Ce sont alors ces handlers qui se chargent de communiquer avec le ou les devices et bibliothèques adéquats, la seule exception étant RAM: qui ne communique avec rien d'autre que la mémoire, et n'a donc pas besoin de device particulier.

PARCOURIR LES LISTES

Comme on l'a dit plus haut, tous les devices AmigaDOS chargés en mémoire sont chainés entre eux, afin de permettre au système de les trouver facilement.
Ainsi, pour savoir quels sont les devices présents à un moment donné - par exemple, dans le but de bâtir un FileRequester proposant un gadget pour chaque device - il suffit de parcourir la liste en filtrant éventuellement les données jugées inintéressantes.
Le début de cette liste se trouve dans la structure DosLibrary. On y trouve un pointeur sur une structure RootNode contenant elle-même un pointeur BCPL sur une structure DosInfo, dans laquelle on trouvera, enfin, un pointeur BCPL sur la structure DeviceList du premier device chargé. Cette structure DeviceList est en fait soit une structure DeviceNode, soit une structure DevInfo, suivant le type du device concerné. En C, on appelle celà une union.
Toutes ces structures sont définies dans "libraries/dosextens.i" et "libraries/filehandler.i". On y trouve au même offset un mot long indiquant le type du device, qui peut être :
- **DLT_DEVICE (0) :** device physique ;
- **DLT_DIRECTORY (1) :** device logique ;
- **DLT_VOLUME (2) :** volume.

Le meilleur moyen de parcourir cette liste est donc de le faire en trois étapes, une pour chaque type de device. C'est ce que fait le programme proposé ci-dessous.

LE PROGRAMME

Son but est d'afficher dans la fenêtre CLI ou Shell courante, la liste de chaque type de devices présents au moment de son exécution. L'affichage se fait en trois passes, histoire de bien séparer les trois types. Pour les devices physiques (type DLT_DEVICE), le nom du device Exec ou du handler utilisé est également affiché.
Curieusement, c'est l'assembleur qui se prête le mieux à ce genre de manipulation, notamment à cause de sa facilité à jouer avec les pointeurs et les chaines de caractères BCPL. Le même programme en C aurait obligé à faire appel à des routines de conversion BSTR vers CSTR.
En bonus, vous y verrez comment gérer l'interruption utilisateur par l'appui sur CTRL/C. Rien, à part peut-être ma belle-mère, n'est plus énervant qu'un programme qui ne s'arrête pas quand on le lui demande.

```
        incdir  "include:"
        include "libraries/dos.i"
        include "libraries/dosextens.i"
        include "libraries/filehandler.i"
        include "libraries/dos_lib.i"
        include "exec/exec_lib.i"

        opt o+,ow-
; ***** Macros
EXEC    MACRO
        move.l  $4.w,a6
        jsr _LVO1(a6)
        ENDM

DOS     MACRO
        move.l  (a5),a6
        jsr _LVO1(a6)
        ENDM
; ***** Définition des variables
        rsreset
DosBase rs.l    1
stdin   rs.l    1
stdout  rs.l    1
devinfo rs.l    1
VARSIZE rs.w    0
; ***** C'est parti !
Start   lea VARS(pc),a5
        moveq   #RETURN_FAIL,d7

        lea dosname(pc),a1
        moveq   #0,d0
        EXEC    OpenLibrary
        move.l  d0,(a5)
        beq.s   NoDos

        DOS Input
        move.l  d0,stdin(a5)
        beq.s   Exit

        DOS Output
        move.l  d0,stdout(a5)
        beq.s   Exit

        move.l  (a5),a0 ; DosLibrary
        move.l  dl_Root(a0),a0  ; DosLibrary->RootInfo
        move.l  rn_Info(a0),a0  ; RootInfo->DosInfo
        adda.l  a0,a0   ; Conversion BCPL
        adda.l  a0,a0   ;       ""      ""
        move.l  di_DevInfo(a0),d0   ; 1er device
        beq.s   Exit    ; Pas de device ?!?
        lsl.l   #2,d0   ; Conversion BCPL
        move.l  d0,devinfo(a5)

        bsr.s   ListDevices
        bsr ListDirectories
        bsr ListVolumes

        moveq   #RETURN_OK,d7

Exit    move.l  (a5),a1
        EXEC    CloseLibrary

NoDos   move.l  d7,d0
        rts
; ***** Liste les devices de type DLT_DEVICE
ListDevices:
        lea msg1(pc),a0
        bsr PrintBSTR

        move.l  devinfo(a5),a2
DevicesLoop:
        bsr CheckBreak

        cmpi.l  #DLT_DEVICE,dn_Type(a2)
        bne.s   NextDevice

        movea.l dn_Name(a2),a0  ; Nom du device dans a0
        adda.l  a0,a0   ; Conversion BPCL
        adda.l  a0,a0   ;       ""      ""
        bsr PrintBSTR   ; Affiche le nom

        lea tab(pc),a0
        bsr PrintBSTR

        move.l  dn_Handler(a2),d0
        beq.s   NoHandler   ; Certains n'ont pas de Handler
        lsl.l   #2,d0
        movea.l d0,a0
        bsr PrintBSTR
        bra.s   NewLine

NoHandler:
        move.l  dn_Startup(a2),d0
        cmpi.l  #21,d0
        bcs.s   NewLine
        lsl.l   #2,d0
        move.l  d0,a0
        move.l  fssm_Device(a0),a0
        adda.l  a0,a0
        adda.l  a0,a0
        bsr PrintBSTR

NewLine lea crlf(pc),a0 ; Passage à la ligne
        bsr PrintBSTR

NextDevice:
        move.l  dn_Next(a2),d0  ; Device suivant dans d0
        lsl.l   #2,d0   ; Conversion BCPL
        movea.l d0,a2
        bne.s   DevicesLoop ; Boucle jusqu'au dernier
```

suite page 31

SUB.way II : 50 % Intuition, 50 %

Après avoir fait connaissance avec les gadgets booléeens et les gadgets de chaînes, nous allons étudier ce mois-ci les gadgets proportionnels. Et pour illustrer tout celà, je vous propose un programme de palette pour faire varier les couleurs du WorkBench.

Le gadget proportionnel est, à mon avis, l'objet le plus flexible et le plus intuitif pour la saisie de données numériques, que l'on puisse trouver sur Amiga. Qu'ils soient fonction d'une ou plusieurs variables, ces gadgets permettent de saisir une valeur discrète comprise entre deux bornes fixées par le programmeur. Par exemple, dans un programme permettant de changer les couleurs de l'écran, les bornes inférieure et supérieure sont respectivement 0 et 15, l'intervalle entre deux valeurs (le pas de "déplacement") étant de 1.

Mais l'intérêt d'un tel gadget ne s'arrête pas là, puisqu'il permet également de donner à l'utilisateur des informations sur la valeur de telle ou telle donnée, ou une proportion de cette donnée par rapport à une autre. Par exemple, sur mon traitement de texte, au moment où j'écris cet article, le gadget proportionnel situé sur le coté m'indique en permanence la taille de mon document et la position du curseur dans celui-ci, et ce de manière beaucoup plus intuitive qu'une série de chiffres.

Un gadget proportionnel est, pour Intuition, un gadget de type spécial, comme l'était le gadget de saisie de chaîne de caractères du mois dernier. Nous allons donc associer à la structure gadget classique une structure spéciale dénommée "PropInfo".

La première chose à faire pour créer ce type de gadget est d'indiquer dans le champ "GadgetType" le flag PROPGADGET.

Ensuite, on indique dans le champ "SpecialInfo" l'adresse de la structure "PropInfo" associée au gadget. Cette structure "Propinfo" contient les données suivantes :

```
struct PropInfo
{
  USHORT Flags;     /* Flags généraux de gestion du gadget */
  USHORT HorizPot;  /* Position horizontale du curseur */
  USHORT VertPot;   /* Position verticale du curseur */
  USHORT HorizBody; /* Valeur de l'incrément Horizontal */
  USHORT VertBody;  /* Valeur de l'incrément vertical */
  USHORT CWidth;    /* Largeur réelle du container */
  USHORT CHeight;   /* Hauteur réelle du container */
  USHORT HPotRes,VPotRes;     /* Incréments du potentiometre */
  USHORT LeftBorder;          /* position horizontale du container */
  USHORT TopBorder; /* position verticale du container */
};
```

Voyons maintenant comment sont utilisés les différents champs de cette structure. Nous prendrons comme exemple le potentiomètre gérant une des couleurs de la palette.

La première chose qu'il faut indiquer à Intuition dans le champ "Flag", c'est de n'autoriser la variation que suivant la verticale. Pour ce faire on positionne le flag FREEVERT. On positionne également le flag AUTOKNOB afin de laisser Intuition gérer la taille du curseur de déplacement.

Il nous faut maintenant indiquer à Intuition l'incrément du potentiomètre. Il est codé sur 16 bits, c'est-à-dire que la valeur d'incrément doit être ramenée sur 16 bits. Pour notre exemple, l'incrément est de 1/16, on précisera donc dans le champ "VertBody" OxFFFF/16 soit 4096. Comme le déplacement horizontal est interdit on indiquera -1 dans le champ "VertBody".

On initialisera le champ "HorizPot" à -1 (déplacement interdit suivant X) et "VertPot" à la valeur de la couleur, multipliée par 4096 (ex : si couleur vaut 2, alors "VertPot" vaut 2*4096 = 8192). Le reste des champs de la structure "PropInfo" est géré par Intuition. On les initialisera à 0 malgré tout.

GESTION DES GADGETS

Le mois dernier, pour attendre la sélection d'un gadget, nous avons eu recours à une boucle sans fin de type "while(1)". Cette méthode très simple de gérer les actions a pour grave inconvénient de faire tourner à vide le programme lorsqu'aucun gadget n'est sélectionné. Pour éviter cette boucle à vide qui consomme de toute façon du temps processeur, nous allons simplement utiliser la fonction Wait() d'exec.library, qui met la tâche en veille jusqu'à ce que l'argument transmis à la fonction passe à 1. Cet argument est tout simplement le champs fenetre->UserPort->mp_SigBit, qui vaut 1 lorsqu'une interaction a été effectuée par l'utilisateur, et 0 sinon. Ainsi on ne consomme plus inutilement du CPU.

Une fois qu'un signal a été envoyé à notre programme, il s'agit de déterminer quel type d'évènement est survenu. S'il s'agit d'un IntuiMessage, on le récupère avec la fonction GetMsg() et on étudie sa classe (IntuiMessage->Class). Trois cas nous intéressent alors :

- **GADGETDOWN :** dans ce cas, l'utilisateur a sélectionné un des gadgets, on exécute donc la fonction qui lui est associée, suivant la méthode décrite le mois dernier ;
- **CLOSEWINDOW :** l'utilisateur veut refermer la palette et terminer le programme. On exécute alors la fonction referme() qui, comme son nom l'indique, clos la fenêtre et les bibliothèques Intuition et Graphics. On positionne également la variable "fin" à 1, terminant ainsi l'éxecution du programme ;
- **MOUSEBUTTONS :** dans ce cas, l'utilisateur a cliqué avec le bouton gauche de la souris dans la fenêtre. Ce type d'évènement va nous permettre de savoir par une série de quatre tests, si l'utilisateur a sélectionné une couleur et laquelle, ou s'il a simplement cliqué au hasard dans la fenetre. Lors d'une sélection d'une couleur, on affecte le numéro de couleur à la variable "courant".

Toute autre évènement ne nous intéresse pas, donc on ne fait rien. Comme vous pouvez le voir tout cela est fort simple.

Dernier point sur la gestion des gadgets proportionnels. Lors de la sélection d'une couleur, il faut réinitialiser les potentiomètres des gadgets RVB pour tenir compte de chaque valeur des composantes de couleur. Pour ce faire, on initialise chaque champ "VertPot" des gadgets avec la valeur de la composante ROUGE, VERT, BLEU multipliée par 4096, ce qui revient à effectuer un décalage de 12 bits ver la gauche comme suit :

```
bleuinfo.VerPot = BLEU << 12;
```

Voici décrite la manière dont sont gérés les gadgets. Je vous conseille de vous reporter au programme d'exemple joint à l'article pour de plus amples renseignements.

LA GESTION DES COULEURS

Nous allons maintenant étudier comment sont gérés les couleurs par l'Amiga. Chaque écran ouvert sous Intuition possède sa propre palette de couleurs dont le nombre est lié, je vous le rappelle, au nombre de biplanes associés à l'écran. Notre palette sert à modifier les couleurs du Workbench, qui possède en standard 2 bitplanes, donc 4 couleurs. On accède à la table des couleurs par le ViewPort de l'écran, son adresse se trouvant dans le champ "ColorMap". La première étape consiste donc à récupérer l'adresse du ViewPort de l'écran à l'aide de la fonction ViewPortAdress :

```
viewport = (struct ViewPort *)ViewPortAdress(fenetre);
```

On récupère ensuite l'adresse de la table des couleurs :

```
colormap = viewport->ColorMap
```

Cette adresse obtenue, on peut modifier les valeurs de la table et ainsi les couleurs de l'écran. On lit la valeur d'une couleur à l'aide de la fonction GetRGB4() :

```
couleur = GetRGB4(colormap, numéro_de_la_couleur)
```

Une couleur contient trois composantes rouge, vert et bleu. Sur AMiga, la

couleur complète est codée sur 16 bits (UWORD). Chaque composante se voit associer 4 bits, soit 12 bits pour les trois. Les 4 bits de poids fort ne sont pas utilisés. Le schèma ci-dessous synthétise cette organisation.

```
15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
xx xx xx xx RR RR RR RR VV VV VV VV BB BB BB BB

avec :
où xx représente un bit inutilisé,
RR un bit associé à la composante ROUGE,
VV un bit associé à la composante VERTE,
BB un bit associé à la composante BLEU.
```

Pour extraire chaque composante de notre mot couleur, on utilisera les formules suivantes :

```
ROUGE = (couleur >> 8) & 0xF;
VERT  = (couleur >> 4) & 0xF;
bleu  = couleur & OxF;
```

Pour intialiser une couleur de la ColorMap, on utilise la fonction SetRGB4() :

```
SetRGB4(viewport, couleur, ROUGE, VERT, BLEU);
```

A noter qu'il est également possible, comme on peut le voir dans le programme d'exemple, de charger plusieurs couleurs à la fois dans la ColorMap avec LoadRGB4() :

```
LoadRGB4(viewport, couleurs, nombre_de_couleurs);
```

Les couleurs sont déclarées dans un tableau de mots de 16 bits :

```
UWORD tablecol[] = { 0x000, 0xfff, 0x888, OxBBB};
```

CONCLUSION

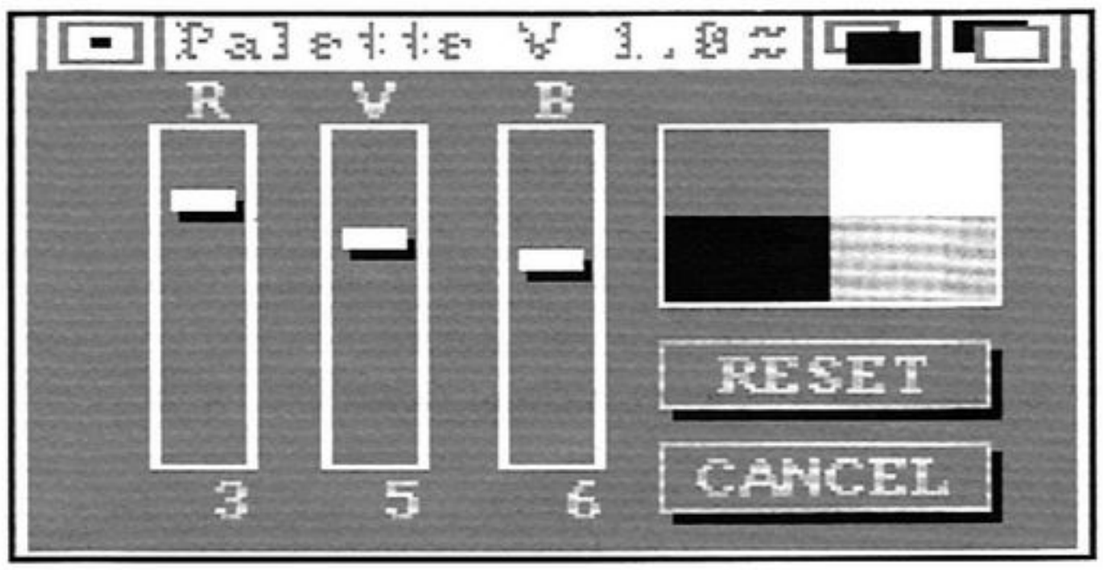
Notre article sur Intuition se termine. Il a certes été plus long que d'habitude, mais un minimum d'explications etait nécessaire pour ce programme. Si par hasard vous aviez des sujets que vous voudriez voir aborder dans cette rubrique, n'hésitez pas à me faire parvenir vos idées et suggestions à l'ANT, soit par courrier soit sur Minitel 3615 ANT, en laissant un message dans ma boîte aux lettres DOKTOR.

```
/*-------------------------------------------------------------*/
/*   Palette permettant de changer les couleurs du Workbench   */
/*                                                             */
/*   Exemple de programmation des gadgets sous intuition       */
/*         - Gadgets booléen                                   */
/*         - Gadgets proportionnels                            */
/*         - Gestion des couleurs                              */
/*                                  P. AMIABLE 1991            */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*   Quelques includes utiles.....                             */
/*-------------------------------------------------------------*/
#include <exec/types.h>
#include <intuition/intuition.h>
#include <stdio.h>

/*-------------------------------------------------------------*/
/*   Les defines maintenants.....                              */
/*-------------------------------------------------------------*/
#define INTUITION_REV 0L /* Version d'Intuition (la dernière)  */
#define GFX_REV       0L /* Version de Graphics (la dernière)  */
#define CSIZE         4  /* Nombre de couleurs à traiter       */

/* Pré-déclaration pour les gadgets */
void traite_reset(), traite_cancel();
void traite_rouge(), traite_vert(), traite_bleu();
void restorecol();

/* ------------------------------------------------------------*/
/*     Définition des variables externes                       */
/* ------------------------------------------------------------*/
SHORT cadre1[] = {
          0,0,
          61,0,
          61,12,
          0,12,
          0,0
};

struct Border bordure1 = {
          -1,-1,          /* position*/
          3,0,JAM1,       /* couleurs et mode de tracé */
          5,              /* nombre de vecteurs */
          cadre1,         /* tableau des vecteurs */
          NULL            /* bordure suivante dans la liste */
};

struct IntuiText texte1 = {
          3,0,JAM2,  /* couleurs et mode de tracé */
          10,2,      /* position */
          NULL,      /* fonte (NULL pour la police par défaut) */
          "RESET",   /* texte */
          NULL       /* structure IntuiText suivante dans la liste */
};

struct Gadget reset = {
          NULL,         /* adresse du gadget suivant  */
          124,69,       /* position */
          60,11,        /* taille de la boite de sélection */
          NULL,         /* Flags associés au gadget */
          RELVERIFY+GADGIMMEDIATE, /* Flags d'activation */
          BOOLGADGET,   /* type du gadget */
          (APTR)&bordure1, /* bordure ou image associée au gadget */
          NULL,         /* image alternée en cas de sélection */
          &texte1,      /* première structure IntuiText associée */
          NULL,         /* mot long d'exclusion mutuelle */
          NULL,         /* structure SpecialInfo */
          NULL,         /* identificateur du gadget */
          (APTR)traite_reset /* données utlisateur */
};

SHORT cadre2[] = {
          0,0,
          61,0,
          61,12,
          0,12,
          0,0
};
struct Border bordure2 = {
          -1,-1,
          3,0,JAM1,
          5,
          cadre2,
          NULL
};

struct IntuiText texte2 = {
          3,0,JAM2,
          6,2,
          NULL,
          "CANCEL",
          NULL
};

struct Gadget cancel = {
          &reset,
          125,85,
          60,11,
          NULL,
          RELVERIFY+GADGIMMEDIATE,
          BOOLGADGET,
          (APTR)&bordure2,
          NULL,
          &texte2,
          NULL,
          NULL,
          NULL,
          (APTR)traite_cancel
};

struct PropInfo bleuinfo = {
          AUTOKNOB+FREEVERT, /* flags liés à la structure PropInfo */
          -1,0,              /* valeurs du potentiomètre */
          -1,0x1000,         /* incrément */
};

struct Image Image1 = {
          0,12,          /* position */
          12,4,          /* largeur et hauteur de l'image */
          0,             /* nombre de bitplanes */
          NULL,          /* structure ImageData */
          0x0000,0x0000, /* PlanePick et PlaneOnOff */
          NULL           /* structure Image suivante */
};

struct IntuiText texte3 = {
          3,0,JAM2,
          7,-8,
          NULL,
          "B",
          NULL
};

struct Gadget bleu = {
          &cancel,
          90,20,
```

```
                    20,64,
                    NULL,
                    RELVERIFY|GADGIMMEDIATE,
                    PROPGADGET,
                    (APTR)&Image1,
                    NULL,
                    &texte3,
                    NULL,
                    (APTR)&bleuinfo,
                    NULL,
                    (APTR)traite_bleu
};

struct PropInfo vertinfo = {
                    AUTOKNOB+FREEVERT,
                    -1,0,
                    -1,0x1000,
};

struct Image Image2 = {
                    0,7,
                    12,4,
                    0,
                    NULL,
                    0x0000,0x0000,
                    NULL
};

struct IntuiText texte4 = {
                    3,0,JAM2,
                    6,-8,
                    NULL,
                    "V",
                    NULL
};

struct Gadget vert = {
                    &bleu,
                    57,20,
                    20,64,
                    NULL,
                    RELVERIFY|GADGIMMEDIATE,
                    PROPGADGET,
                    (APTR)&Image2,
                    NULL,
                    &texte4,
                    NULL,
                    (APTR)&vertinfo,
                    NULL,
                    (APTR)traite_vert
};

struct PropInfo rougeinfo = {
                    AUTOKNOB+FREEVERT,
                    -1,0,
                    -1,0x1000,
};

struct Image Image3 = {
                    0,7,
                    12,4,
                    0,
                    NULL,
                    0x0000,0x0000,
                    NULL
};

struct IntuiText texte5 = {
                    3,0,JAM2,
                    7,-8,
                    NULL,
                    "R",
                    NULL
};

struct Gadget rouge = {
                    &vert,
                    25,20,
                    20,64,
                    NULL,
                    RELVERIFY|GADGIMMEDIATE,
                    PROPGADGET,
                    (APTR)&Image3,
                    NULL,
                    &texte5,
                    NULL,
                    (APTR)&rougeinfo,
                    NULL,
                    (APTR)traite_rouge
};

struct IntuiText texte8 = {
                    3,0,JAM2,
                    94,86,
                    NULL,
                    "00",
                    NULL
};

struct IntuiText texte7 = {
                    3,0,JAM2,
                    60,86,
                    NULL,
                    "00",
                    &texte8
};

struct IntuiText texte6 = {
                    3,0,JAM2,
                    28,86,
                    NULL,
                    "00",
                    &texte7
};

struct NewWindow nouvfenetre = {
                    100,50,                         /* position */
                    200,100,                        /* largeur et hauteur */
                    0,1,                            /* couleurs */
                    MOUSEBUTTONS+GADGETDOWN+GADGETUP+
                       CLOSEWINDOW+MOUSEMOVE,       /* flags IDCMP */
                    WINDOWDRAG+WINDOWDEPTH+WINDOWCLOSE+
                       ACTIVATE+NOCAREREFRESH,      /* flags */
                    &rouge,                         /* premier gadget */
                    NULL,                           /* CHECKMARK personnelle */
                    "Palette V 1.0",                /* titre de la fenêtre */

                    NULL,                           /* pointeur sur un custom screen */
                    NULL,                           /* bitmap personnelle */
                    200,100,                        /* largeur et hauteur mini  */
                    -1,-1,                          /* largeur et hauteur maxi */
                    WBENCHSCREEN                    /* type d'écran associé */
};

char *bouton_gauche = (char *) 0xbfe001;
USHORT sauvecol[CSIZE];                             /* table de sauvegarde des couleurs */
struct IntuitionBase *IntuitionBase;
struct GfxBase  *GfxBase;
struct  Window  *fenetre = NULL;
struct  RastPort *rastport = NULL;
struct ViewPort *viewport;
int courant = 0;
UWORD ROUGE,VERT,BLEU;                              /* composante de chaque couleur
*/

/* ------------------------------------------------------------------*/
/*         Programme principal                                       */
/* ------------------------------------------------------------------*/
void _main()
{
struct IntuiMessage *message;
long GetMsg();
ULONG classemessage;
void init(), ouvrefenetre(), referme();
int fin=0, xmouse, ymouse;

 init();
 ouvrefenetre();
 while(!fin)
   {
    Wait(1L << fenetre->UserPort->mp_SigBit);
    while(message=(struct IntuiMessage *)GetMsg(fenetre->UserPort))
     {
      classemessage = message->Class; /* on recupere la classe */
      switch(classemessage) {

       case GADGETDOWN:
       /* On appelle la fonction liée au gadget activé */
        (*((void(*)())((struct Gadget *)message->IAddress)->UserData))();
         break;

       case CLOSEWINDOW: /* on a fermé la fenêtre */
        fin = 1;
        break;

       case MOUSEBUTTONS: /* bouton de la souris */
        xmouse = message->MouseX;
        ymouse = message->MouseY;
        if((xmouse > 120 && xmouse < 151) && (ymouse > 20 && ymouse < 36))
         courant = 0;
        if((xmouse > 150 && xmouse < 183) && (ymouse > 20 && ymouse < 36))
         courant = 1;
        if((xmouse > 120 && xmouse < 151) && (ymouse > 35 && ymouse < 53))
         courant = 2;
        if((xmouse > 150 && xmouse < 183) && (ymouse > 35 && ymouse < 53))
         courant = 3;

        restorecol();

        break;
```

```
        default: break;
       }
      ReplyMsg((struct Message *)message); /* libere le message */
    }
   }
  referme();                          /* au revoir */
}

/* ------------------------------------------------------------------*/
/*  Init() Ouvre la bibliothèque                                     */
/* ------------------------------------------------------------------*/
  void init()
   {
    char *OpenLibrary();

    IntuitionBase = (struct IntuitionBase *)
      OpenLibrary("intuition.library",INTUITION_REV);
    if(!IntuitionBase)
       {
        exit(FALSE);
       }
    GfxBase = (struct GfxBase *)
     OpenLibrary("graphics.library",GFX_REV);
    if(!GfxBase)
       {
        referme();
        exit(FALSE);
       }
 }


/* ------------------------------------------------------------------*/
/*  ouvrefenetre() ouvre la fenetre et les gadgets                   */
/* ------------------------------------------------------------------*/

void ouvrefenetre()
{
void referme();
int i;
UWORD cadre[] = {
     120,20,
     183,20,
     183,53,
     120,53,
     120,20 };

    if(!(fenetre=(struct Window*)OpenWindow(&nouvfenetre)))
      {
       referme();
       exit(FALSE);
      }
    rastport = fenetre->RPort; /* rastport nécessaire pour tracer */
    viewport = (struct ViewPort *)ViewPortAddress(fenetre);
     for(i=0 ; i<CSIZE ; i++)
         sauvecol[i] = GetRGB4(viewport->ColorMap,i);
     SetAPen(rastport,1);
     Move(rastport,120,20);
     PolyDraw(rastport,5,cadre);
     SetAPen(rastport,0);
     RectFill(rastport,121,21,151,36);
     SetAPen(rastport,1);
     RectFill(rastport,152,21,182,36);
     SetAPen(rastport,2);
     RectFill(rastport,121,37,151,52);
     SetAPen(rastport,3);
     RectFill(rastport,152,37,182,52);
     restorecol();
}

/* ------------------------------------------------------------------*/
/*  Cette fonction referme la fenetre et les bibliothèques           */
/* ------------------------------------------------------------------*/
void referme()
  {
   if(fenetre)          CloseWindow(fenetre);
   if(IntuitionBase)    CloseLibrary(IntuitionBase);
   if(GfxBase)          CloseLibrary(IntuitionBase);
}

/* ------------------------------------------------------------------*/
/*  fonction gérant le gadget RESET                                  */
/* ------------------------------------------------------------------*/
void  traite_reset() /* on remet la couleur sauvegardée associée */
{
     SetRGB4(viewport,courant,(sauvecol[courant]>> 8) & 0xF,
                              (sauvecol[courant]>> 4) & 0xF,
                               sauvecol[courant] & 0xF);
     restorecol();
}

/* ------------------------------------------------------------------*/
/*  fonction gérant le gadget CANCEL                                 */
/* ------------------------------------------------------------------*/
```

```
void  traite_cancel() /* remet les couleurs sauvegardées au départ */
{
     LoadRGB4(viewport,sauvecol,4);
     restorecol();
}

/* ------------------------------------------------------------------*/
/*  fonction gérant le gadget BLEU                                   */
/* ------------------------------------------------------------------*/
void traite_bleu()
{
          while((*bouton_gauche & 0x40)-64) {
                BLEU  = bleuinfo.VertPot  >> 12;

                SetRGB4(viewport,courant,ROUGE,VERT,BLEU);

                sprintf(texte8.IText,"%2d",BLEU);
                PrintIText(rastport,&texte8,0,0);
          }
}

/* ------------------------------------------------------------------*/
/*  fonction gérant le gadget VERT                                   */
/* ------------------------------------------------------------------*/
void  traite_vert()
{
          while((*bouton_gauche & 0x40)-64) {
                VERT  = vertinfo.VertPot  >> 12;

                SetRGB4(viewport,courant,ROUGE,VERT,BLEU);

                sprintf(texte7.IText,"%2d",VERT);
                PrintIText(rastport,&texte7,0,0);
          }
}

/* ------------------------------------------------------------------*/
/*  fonction gérant le gadget ROUGE                                  */
/* ------------------------------------------------------------------*/
void  traite_rouge()
{
          while((*bouton_gauche & 0x40)-64) {
                ROUGE  = rougeinfo.VertPot  >> 12;

                SetRGB4(viewport,courant,ROUGE,VERT,BLEU);

                sprintf(texte6.IText,"%2d",ROUGE);
                PrintIText(rastport,&texte6,0,0);
          }
}

/* ------------------------------------------------------------------*/
/*  fonction gérant le gadget ROUGE                                  */
/* ------------------------------------------------------------------*/
void restorecol()
{
     ROUGE = (GetRGB4(viewport->ColorMap,courant) >> 8) & 0xF ;
     VERT  = (GetRGB4(viewport->ColorMap,courant) >> 4) & 0xF ;
     BLEU  = GetRGB4(viewport->ColorMap,courant) & 0xF ;

     rougeinfo.VertPot = ROUGE << 12;
     vertinfo.VertPot = VERT << 12;
     bleuinfo.VertPot = BLEU << 12;
     RefreshGList(&rouge,fenetre,0,3);

     sprintf(texte6.IText,"%2d",ROUGE);
     sprintf(texte7.IText,"%2d",VERT);
     sprintf(texte8.IText,"%2d",BLEU);
     PrintIText(rastport,&texte6,0,0);
}
```



HARDWARE : L'ESPION DES

Nous entamons ce mois-ci une nouvelle série sur le hardware de l'Amiga. Depuis le Copper jusqu'au Blitter, en passant par les Sprites et le DMA disque et audio, nous verrons ensemble ces merveilleux composants électroniques qui font de l'Amiga le meilleur ordinateur du moment.

Vous avez déjà sans doute pu voir à droite et à gauche - surtout à gauche - des articles sur ce sujet, et vous êtes gavés des barres de Copper, des Bobs et autres tracés de droites... Bien sûr, nous n'y échapperons pas, mais l'approche sera différente. Plutôt que de nous contenter d'une description bête et méchante des puces, telle que l'on peut en trouver dans le RKM Hardware ou la Bible de l'Amiga, nous nous attacherons à mettre en oeuvre certaines particularités dudit hardware. Et pour commencer en beauté, nous allons voir de plus près l'utilisation et la programmation de ce fameux CIA 8520 dont l'Amiga est doublement équipé.

GENERALITES

Il en faut tout de même...
Le CIA est un Complex Interface Adaptor. En d'autres termes et plus simplement, c'est un circuit capable de gérer plusieurs choses en même temps. Ainsi, il est équipé de deux ports parallèles 8 bits, 1 port série bidirectionnel, 3 minuteries dont deux 16 bits et une 24 bits et une fonction d'alarme. Il est capable de libérer des interruptions processeur lorsqu'un évènement particulier intervient (port série plein ou vide, timer à 0). L'Amiga est équipé de deux CIAs, communément appelés CIA-A pour le premier, et CIA-B pour le second.

Le CIA est équipé de 16 registres 8 bits programmables, accessibles par le processeur par une simple instruction MOVE. Ces 16 registres contrôlent évidemment les fonctions du 8520. Nous verrons chacun de ces registres le moment venu, mais vous devez d'ores et déjà savoir qu'ils sont accessibles depuis les adresses $BFE001 (CIA-A) et $BFD000 (CIA-B), avec un espace de 256 octets entre chaque registre (par exemple, le registre 10 du CIA-A est à l'adresse $BFE001+(256*10)=$BFEA001 ; le registre 15 du CIA-B est à l'adresse $BFD000+(256*15)=$BFDF00).

LES PORTS PARALLELES

Comme mentionné plus haut, le CIA dispose de deux ports parallèles 8 bits programmables, accessibles par les registres 0 et 1, respectivement, et baptisés PRA et PRB. Les registres 2 et 3, appelés DDRA et DDRB, contrôlent la direction des données pour chacun des bits du registre PR correspondant. Si un bit de DDR est mis à 1, le bit correspondant de PR est une sortie, sinon, c'est une entrée (pour le CIA-A, évidemment, et non pour le processeur).

Ces quatre ports parallèles sont évidemment utilisés par l'Amiga, comme le montrent les tableaux suivants :

```
CIA-A PRA ($BFE001)
Bit     Nom     Fonction
7       FIR1    Bouton de feu du joystick (0=appuyé)
6       FIR0    Bouton gauche de la souris (0=appuyé)
5       RDY     Drive ready (0=prêt)
4       TK0     Tête de lecture en piste 0 (0=piste 0)
3       WPRO    Disquette protégée en écriture (0=protégée)
2       CHNG    Changement de disquette (0=disque changé)
1       LED     Contrôle de la PowerLed (filtre passe-bas)
0       OVL     Contrôle de recouvrement ROM/RAM (0=RAM)

DDRA est initialisé à 0x03 (LED et OVL en sortie, tous les autres bits
en entrée).

CIA-A PRB ($BFE101)
Les 8 bits sont utilisés pour les données de l'interface Parallèle.

DDRB peut être initialisé à 0 ou à 0xFF suivant le sens de
transmission.

CIA-B PRA ($BFD000)
Bit     Nom     Fonction
7       DTR     Interface sérielle, sortie DTR
6       RTS     Interface sérielle, sortie RTS
5       CD      Interface sérielle, sortie CD (détection de porteuse)
4       CTS     Interface sérielle, sortie CTS
3       DSR     Interface sérielle, sortie DSR
2       SEL     Contrôle interface parallèle
1       POUT    Paper Out (imprimante)
0       BUSY    Imprimante occupée

DDRA est initialisé à 0xFF (tous les bits en sortie).

CIA-B PRB ($BFD100)
Bit     Nom     Fonction
7       MTR     Contrôle du moteur du drive
6       SEL3    Sélection drive 3
5       SEL2    Sélection drive 2
4       SEL1    Sélection drive 1
3       SEL0    Sélection drive 0
2       SIDE    Sélection de la face de la disquette
1       DIR     Sélection de la direction du moteur du drive
0       STEP    Contrôle du pas du moteur du drive

DDRB est initialisé à 0xFF (tous les bits en sortie).
```

Le port A du CIA-A est multi-fonctionnel (bouton de feu du joystick, bouton gauche de la souris, contrôle disquette, power-led et recouvrement disquette) tandis que son port B est entièrement dédié aux données de l'interface parallèle.

Le port A du CIA-B contrôle les interfaces série (contenue dans Paula) et parallèle, tandis que son port B est entièrement dédié au contrôle des lecteurs de disquette (en supplément au port A du CIA-A).

LE PORT SERIE

Attention, ne confondez pas : le port série du CIA n'est PAS celui de la RS-232, qui est émulée par Paula.

Le port série du CIA est accessible par son registre numéro 12. Celui du CIA-A est connecté au processeur clavier pour la transmission des touches, tandis que celui du CIA-B est inutilisé. Nous reparlerons plus en détails du clavier plus tard.

LES MINUTERIES

C'est là le plus gros morceau du 8520. Il comporte deux minuteries 16 bits appelées TA et TB (Timer A et Timer B). Chacune se voit attribuer un registre de contrôle, baptisé CRA pour TA et CRB pour TB.

Le rôle des minuteries est de décompter à partir d'une valeur fixée à l'avance par le processeur jusqu'à 0. Dès que le timer dépasse le 0 (en d'autres termes, dès qu'il arrive à -1), il peut déclencher ou non une interruption processeur et, suivant le mode choisi, recommencer ou non le décompte. Le décompte commence dès que la valeur de départ a été inscrite dans le registre adéquat.

Chaque minuterie dispose en fait de deux registres 16 bits, l'un pour la lecture, l'autre pour l'écriture. Ces deux registres étant logés à la même adresse, il est impossible de savoir par lecture à quelle valeur la minuterie à été initialisée, puisqu'elle est décrémentée dès l'écriture (le CIA, lui, s'y retrouve en recopiant juste avant le démarrage de la minuterie, la valeur de début dans une mémoire interne). On peut par contre lire à tout moment la valeur en cours du compteur, en l'arrêtant provisoirement.

Le registre de contrôle de la minuterie détermine son mode de fonctionnement. Chacune des deux minuteries dispose de son propre registre de contrôle, car la minuterie B peut éventuellement fonctionner de manière légèrement différente de la minuterie A. Les registres CRA et CRB sont les registres numéros 14 et 15 (respectivement) du CIA.

```
Registre CRA (Control Register A)
Bit     Nom     Fonction
7       TOD IN  Contrôle de TOD (0=60Hz, 1=50Hz)
6       SPMODE  Contrôle port série (0=entrée, 1=sortie)
5       INMODE  Contrôle du signal de la minuterie
4       LOAD    Recharge la valeur de départ dans le compteur (strobe)
3       RUNMODE Contrôle du mode (0=continu, 1=une fois)
2       OUTMODE Sélection du type de sortie (0=pusle, 1=toggle)
1       PBON    Redirection de la minuterie vers le port B (1=on)
0       START   Contrôle de la minuterie (0=stop, 1=start)
```

```
Registre CRB (Control Register B)
Bit     Nom      Fonction
7       ALARM    Mode alarme (0=TOD, 1=alarme)
6-5     INMODE   Contrôle du signal (00=Horloge, 01=CNT, 10=Timer A,
                 11=CNT+Timer A)
4       LOAD     Recharge la valeur de départ dans le compteur (strobe)
3       RUNMODE  Contrôle du mode (0=continu, 1=une fois)
2       OUTMODE  Sélection du type de sortie (0=pusle, 1=toggle)
1       PBON     Redirection de la minuterie vers le port B (1=on)
0       START    Contrôle de la minuterie (0=stop, 1=start)
```

Les bits les plus importants sont LOAD, INMODE, RUNMODE et START.

LOAD force le CIA à recharger dans le compteur sa valeur initiale, qu'il avait sauvegardé dans une mémoire latch.

INMODE, quant il est mis à 0, indique que la valeur du compteur est décrémentée à chaque impulsion d'horloge (signal ECLK du processeur ou de FAT GARY sur l'A3000). La fréquence du CIA est donc d'1/10ième celle du processeur, soit 0,716 MHz. Nous verrons dans le prochain paragraphe comment déterminer un délai précis dans ce mode. Mis à 1, INMODE indique une décrémentation à chaque impulsion du le signal CNT du CIA.

RUNMODE indique si la minuterie doit être continue ou non. Une minuterie continue permet par exemple de déclencher une interruption à intervalles réguliers.

START permet d'interrompre la minuterie, pour la refaire démarrer plus tard.

CALCUL D'UN DELAI

Pour programmer un délai de manière précise en mode horloge (INMODE=0), il faut d'abord savoir sur quel système, NSTC ou PAL, l'on se trouve.

Sur un Amiga NTSC, la fréquence d'horloge du 68000 est de 7,15909 MHz. Celle des CIAs est donc de 0,715909 MHz. Une décrémentation du compteur survient donc toutes les 1/0,715909 = 1,3968255 microsecondes.

Sun un Amiga PAL, l'horloge du 68000 est de 7,09379 MHz, celle des CIAs à 0,709379. La décrémentation intervient donc toutes les 1/0,709379 = 1,4096836 microsecondes.

Lorsque l'on sait qu'une seconde contient un million de microsecondes, on calcule qu'attendre 1/100ième de seconde équivaut à attendre 10.000 microsecondes. Il faut donc initialiser le compteur avec la valeur 10.000/1,3968255 = 7159 en NTSC ou 10.000/1,4096839 = 7093 en PAL (on ne prend bien sûr que la partie entière du quotient). De même, attendre 3 millisecondes (1 milliseconde égale 1000 microsecondes) requiert d'initialiser le compteur à la valeur de 2148 en NTSC, ou 2128 en PAL.

Reste maintenant à savoir dans quel mode l'on se trouve, du NTSC ou du PAL... Le moyen le plus simple de le savoir est d'inspecter le champ DisplayFlags de la structure GfxBase. Ce mot contient la valeur 1 sur un système NTSC et la valeur 4 sur un système PAL. Les différentes valeurs possibles du compteur seront avantageusement calculées par avance et sauvegardées dans une table, parce que bon, c'est pas pour dire, mais les calculs en virgule flottante avec le 68000, hein...

Dernier point important : rappelez-vous que les minuteries sont 16 bits. Autrement dit, le délai maximum que l'on puisse programmer est de 65534 (65535 arrêtant la minuterie), soit environ 9/100ièmes de seconde. Pour des délais plus longs, il faudra utiliser la minuterie 24 bits, comme nous le verrons le mois prochain.

RECREATION

Vous avez bien gagné le droit de respirer un peu... Le petit programme que je vous propose maintenant est totalement inutile, si l'on fait abstraction de ses qualités didactives. Il fait clignoter la power-led à intervalles réguliers de 1/100ième de seconde (10,000 microsecondes). Il teste le système pour savoir si l'on se trouve en PAL ou en NTSC et s'adapte en conséquence. Le reste est suffisamment commenté.

```
;
; Exemple de programmation du timer du CIA.
; Adapté du RKM Hardware, page F-331.
; Fait clignoter la PowerLed tous les 1/100èmes de seconde.
;
; (c) 1991 Loïc Far pour ANT.
;

        incdir  "include:"
        include "graphics/gfxbase.i"
        include "hardware/cia.i"
        include "hardware/custom.i"

        include "exec/exec_lib.i"
        include "libraries/dos_lib.i"

        opt o+,ow-

ciaa    EQU $bfe001
ciab    EQU $bfd000
custom  EQU $dff000

N_DELAI EQU 7159    ; Délai pour 1/100ième de seconde NTSC
P_DELAI EQU 7093    ; Délai pour 1/100ième de seconde PAL

Run     lea dosname(pc),a1 ; Ouvre la dos.library
        moveq   #0,d0
        CALLEXEC OpenLibrary
        move.l  d0,_DOSBase
        beq Exit

        CALLDOS Output
        move.l  d0,stdout

        lea gfxname(pc),a1 ; Ouvre la graphics.library
        moveq   #0,d0
        CALLEXEC OpenLibrary
        tst.l   d0
        beq Ciao    ; Petit problème...
        move.l  d0,a1
        move.w  #P_DELAI,d7 ; Délai pour mode PAL
        cmpi.w  #4,gb_DisplayFlags(a1) ; Mode PAL ?
        beq.s   Ok
        move.w  #N_DELAI,d7 ; Délai pour mode NTSC
Ok      CALLEXEC CloseLibrary  ; Ferme la librarie

        lea ciab,a4 ; Adresse du CIA-B dans A4
        lea ciaa,a5 ; Adresse du CIA-A dans a5

        move.b  ciacra(a4),d0   ; Lit le registre CRA du CIA-B
        and.b   #%11000000,d0   ; Masque les bits inutilisés
        or.b    #%00001000,d0   ; Mode One-Shot
        move.b  d0,ciacra(a4)
        move.b  #%01111111,ciaicr(a4)  ; Pas d'interruption CIA-B

        move.l  stdout(pc),d1
        move.l  #message,d2
        moveq   #messlen,d3
        CALLDOS Write

        move.b  d7,ciatalo(a4) ; Programme la valeur du compteur
        ror.b   #8,d7   ; poids faible d'abord
        move.b  d7,ciatahi(a4) ; poids fort en dernier

busy_wait:
        btst    #0,ciaicr(a4)   ; Compteur à 0 ?
        beq.s   busy_wait   ; Pas encore
        bchg    #CIAB_LED,ciapra(a5)    ; Switche la LED
        btst    #CIAB_GAMEPORT0,ciapra(a5) ; Bouton souris ?
        beq.s   Ciao        ; Oui-> fin
        bset    #0,ciacra(a4)   ; Redémarre le timer
        bra.s   busy_wait   ; et boucle

Ciao    move.l  _DOSBase(pc),a1
        CALLEXEC CloseLibrary

Exit    moveq   #0,d0
        rts

; ************************************
_DOSBase    dc.l    0
stdout  dc.l    0

dosname dc.b    "dos.library",0
        even
gfxname dc.b    "graphics.library",0
        even

message dc.b    "Regardez la PowerLed...",$0a
        dc.b    "Pressez le bouton gauche "
        dc.b    "de la souris pour terminer.",$0a
messlen EQU *-message
        even

        END
```

REQUESTER V.2.3 free

Depuis que la rubrique Requester existe, nous avons répondu à pas mal de questions dans beaucoup de domaines... Mais l'ANT étant devenu "le premier magazine dédié aux programmeurs sur Amiga", nous vous saurions gré de ne plus nous parler des C64, PETs et autres PCs... Même si ce sont bien des machines Commodore, l'ANT ne se voue qu'à l'Amiga. D'avance, merci.

Bonjour à tous, et en particulier à Herr Doktor von GlutenStimmelImDorf. Lecteur d'ANT intéressé par la 3D, j'ai jeté un oeil sur le programme de 3D du "Coin des démo-makers" du mois d'avril. Je me permets de faire plusieurs remarques concernant la programmation. Ces remarques sont un peu techniques, mais bon, c'est ANTech, non ?
Commençons par les moins importantes.
Concernant le clipping de droites : les arguments donnés en faveur de la technique dichotomique et contre le calcul utilisant une division sont faux : le calcul par le coefficient directeur ne fait perdre aucune précision s'il est fait intelligemment. Il est sûr que faire le calcul (y2-y1)/(x2-x1) donne un résultat fractionnaire difficile à gérer, mais si on fait (x_max-x1)(y2-y1) PUIS l'on divise ce résultat par (x2-x1), on obtient une valeur parfaitement correcte de y_inter. Le seul argument en faveur de la dichotomie est qu'elle permet parfois de trouver l'intersection plus rapidement, mais par contre, dans d'autres cas, elle peut se révéler beaucoup plus longue.*
L'auteur de l'article nous dit qu'il n'a pas l'intention de faire la 3D la plus rapide, mais plutôt d'initier les débutants, alors qu'il utilise des techniques d'optimisation dignes d'un programmeur ST. Exemple : l'effacement blitter + 68000 qui, pour un débutant, n'est pas forcément facile à appréhender. Mais surtout, il commet une erreur impardonnable : le code auto-modifié ! Non seulement cette pratique est indigne du 68000 mais aussi, comme vous le dîtes si bien dans votre dernière page, elle a le gros inconvénient de ne pas fonctionner sur micros avec mémoire cache. De plus, en utilisant correctement le 68000, il aurait été aisé de faire quelque chose de plus rapide, par exemple en utilisant un pointeur sur une variable contenant les sinus et cosinus courants, et en faisant des "movem.w (a0)+,d4/d5". C'est plus rapide et sans danger !
Quelques mots sur le journal pour finir : le contenu du premier numéro était un peu décevant et donnait une sensation de "vide", mais les choses vont en s'améliorant. Par contre, j'attends toujours les interviews, notamment celle de Fred Fish.
En tout cas, bonne continuation.

Thomas Landspurg

Heeeyyy, vous avez vu qui nous écrit là ? Thomas Landspurg, Mister 3D himself ! Ceux qui ne connaîtraient pas ses démos, dont celle qui l'a rendu célèbre, la VectorBall Demo, sont priés de se ruer chez le petit copain les copier illico !
Celà dit, les critiques de Thomas sont parfaitement fondées. En ce qui concerne la 3D, je pense qu'on peut lui faire confiance quant aux arguments pour ou contre la recherche dichotomique ainsi que pour les questions d'optimisations diverses... Mais Jerôme Etienne, l'auteur de l'article en question, a pour but d'initier à la 3D en assembleur, non à l'assembleur lui-même. Accélérer l'effaçage de l'écran en utilisant conjointement le blitter et le processeur peut passer inaperçu, du moment que le lecteur aura compris le principe de la 3D elle-même. Ce n'est pas là le point le plus important de l'article. Quant à l'utilisation du code auto-modifié, il est vrai que celà jure un tantinet avec la rubrique CATS dans laquelle on déconseille fortement cette pratique. Jérôme Etienne en a d'ailleurs été puni, se retrouvant sans Amiga pour plusieurs semaines !
Pour terminer, merci à Thomas pour les critiques sur le journal en lui-même. Elles résument parfaitement le sentiment général, qui, il faut bien le dire, est également le nôtre. Nous pensons toutefois avoir largement remédié à cet état de fait depuis quelques numéros. Quoiqu'il en soit, n'oubliez jamais que l'ANT est et reste avant tout votre journal. Nous sommes prêts à accueillir toutes les idées, toutes les suggestions et propositions d'articles que vous jugerez bon de nous faire parvenir. Il est évident que si vous ne faites pas vivre l'ANT par votre courrier, si vous ne nous donnez pas de fil directeur, il ne pourra sans doute jamais répondre à toutes vos attentes. Le sondage du premier numéro, destiné à vous mobiliser un peu, nous a déjà permi de nous faire une idée de ce que vous vous attendiez à trouver dans l'ANT et de ce dont vous vous fichez éperdument... dont les interviews, même de Fred Fish!

Chers ANTiens, je vous félicite en tant qu'Amigaïste et que membre de l'association Corsaire Productions pour dire qu'enfin, un vrai journal technique a fait son apparition en France. En tant que technicien de l'association, j'ai essayé différents matériels sur mon Amiga et ma question sera : peut-on installer un lecteur 3"1/2 HD (haute densité) et si oui, comment ? Mes essais on été infructeux : 880 Ko, mais un de plus !
Une autre question tournera autour de l'utilisation du port parallèle de l'Amiga en assembleur. J'aimerais connaître les différentes adresses permettant de le paramétrer sans gêner le multitâche. Dans le même ordre d'idée, j'ai de gros problèmes de transmission avec le port série, qui ne fonctionne correctement qu'en Basic. Les liaisons entre deux Amigas par l'un ou l'autre restent difficiles à réaliser.
Vous désirez des questions, en voici quelques uns qui vous feront passer quelques heures de recherches, fructueuses j'espère. Merci d'avance, et continuez comme ça encore longtemps. Et longue vie à l'Amiga !

Bruno Lebresne

Connecter un lecteur haute-densité sur l'Amiga est évidemment possible, mais c'est moins facile que l'on ne pourrait le penser à priori... A l'inverse du PC, qui a au moins ça de bien, l'Amiga ne sait pas reconnaître le type des lecteurs connectés. Il ne différencie déjà pas le 5"1/4 du 3"1/2, alors lui demander de faire la différence entre un double-densité et un haute-densité, hein, franchement... Le problème ne se situe pas au niveau de la connectique, comme tu as déjà pu en faire l'expérience, mais bel et bien au niveau du logiciel. En effet, le trackdisk.device ne sait rien gérer d'autre que la double-densité.
Pour régler le problème, il faut donc y aller du compilateur C et de l'assembleur, et écrire un handler particulier pour le lecteur haute-densité capable de gérer un nombre accru de secteurs par piste et d'octets par secteur. Ce qui n'est pas une tâche aisée, j'en conviens.
Pour ce qui concerne le port parallèle, deux possibilités s'offrent à toi si tu tiens réellement à rester "amiga friendly" (ce dont nous ne pouvons que te féliciter) : utilise soit le parallel.device, soit la misc.resource (voir à ce sujet l'article de Max dans ce même numéro). Cette seconde solution est cependant plus difficile à mettre en oeuvre, car il faut alors attaquer directement le hardware.
Il en va de même pour le port série. Le serial.device et la misc.resource (encore elle) sont là pour te permettre de l'utiliser à ta guise. Si la transmission ne fonctionne par correctement, vérifie surtout que le port est paramétré de la même manière sur les deux terminaux (les Préférences sont là pour ça) et que tu utilise un protocole de communication correct (en fonction de la parité, du nombre de bits de données, du handshaking et que sais-je encore ?). Et merci pour ton enthousiasme qui nous va droit au coeur!

Bonjour à toute l'équipe de l'ANT. Je ne m'attarderai pas sur le journal que je trouve très bien comme il est, et passerai directement à une question qui me turlupine depuis quelques temps. Je programme en assembleur, et je recherche le moyen d'ajouter des menus personnels à ceux du Workbench. Je sais que plusieurs programmes du Domaine Public réalisent déjà celà, mais ceux que j'ai pu voir jusqu'ici ne me conviennent pas. Pourriez-vous me donner un fil conducteur ?

Philippe LAMY-JONC

Le principe est simple : il faut rechercher dans la liste des écrans et des fenêtres - liste maintenue par Intuition - celle du Workbench. Elle est facilement reconnaissable : le flag BACKDROP est positionné et son titre est "Workbench" (tout bêtement). Il faut alors extraire de cette structure les adresses du Menu et du MsgPort utilisés par le Workbench (tous deux se trouvent en RAM). Ensuite, ClearMenuStrip(), ajout d'une ou plusieurs structures Menu derrière le menu "Special", puis SetMenuStrip() suffisent à ajouter autant de menus que tu le désires. Pour les gérer, il faut remplacer le champ wd_UserPort de la fenêtre du Workbench par un MsgPort à toi et filtrer tous les messages qui y arrivent : tant qu'il ne s'agit pas d'un message du type MENUPICK, il faut le détourner avec PutMsg() sur le MsgPort original du Workbench (sinon, il ne reçoit plus rien !). S'il s'agit bien d'un MENUPICK, il faut évidemment vérifier que c'est bien l'un des nouveaux menus qui a été sélectionné, sinon PutMsg() à nouveau. Bref, voici une routine qui réalise tout celà (on suppose que toute la partie initialisation - ouverture des libraries, création d'un MsgPort avec CreatePort() ou une routine équivalente, etc. - est déjà réalisée).

```
; Recherche l'écran et la fenêtre du WB
        CALLEXEC    Forbid              ; IN-DIS-PEN-SABLE !

        lea wbscr(pc),a0
        suba.l  a1,a1
        move.l  #sc_SIZEOF,d0
        moveq   #WBENCHSCREEN,d1
        CALLINT GetScreenData           ; on a l'écran

        lea wbscr(pc),a2                ; Recherche la fenêtre
        move.l  sc_FirstWindow(a2),d0
FindWB  beq.s   FoundWB                 ; Dernière fenêtre de la liste ?

        move.l  d0,a2
        move.l  wd_Flags(a2),d0
        andi.l  #BACKDROP,d0            ; Si c'est une BACKDROP...
        bne.s   MayBeWB                 ; vérifie le titre
NotWB   move.l  wd_NextWindow(a2),d0
        bra.s   FindWB

MayBeWB move.l  wd_Title(a2),a0
        lea wbtitle(pc),a1
        bsr strcmp
        bne.s   NotWB

FoundWB move.l  a2,wbwin                ; Pointeur sur la fenêtre du WB

        move.l  wd_MenuStrip(a2),wbmenu ; Menu du WB
        move.l  wd_UserPort(a2),wbport  ; MsgPort du WB
        move.l  myport,wd_UserPort(a2)  ; Place le notre

; Il faut maintenant ajouter notre menu derrière celui
; du WorkBench. Pour ce faire, il faut d'abord appeler
; ClearMenuStrip() pour la fenêtre du WB, parcourir les
; ses structures Menu jusqu'à la dernière, lui "greffer"
; la notre, et enfin appeler SetMenuStrip().

        CALLEXEC    Permit              ; Multitâche autorisé

WaitMsg move.l  myport,a0               ; Attend un message normalement
        CALLEXEC    WaitPort            ; destiné au WorkBench...

AllMsgs move.l  myport,a0               ; Boucle normale pour lire tous
        CALLEXEC    GetMsg              ; les messages arrivés sur le
        tst.l   d0  ; MsgPort
        beq.s   WaitMsg

        move.l  d0,a1
        cmpi.l  #MENUPICK,im_Class(a1)  ; MENUPICK ?
        beq.s   GereMenu

NotMenu move.l  wbport,a0               ; Si c'est pas un MENUPICK, il faut
        CALLEXEC    PutMsg              ; transmettre le message au vrai
        bra.s   AllMsgs                 ; MsgPort du WorkBench

GereMenu:
        ...                             ; Vérifie que c'est bien l'un des
        ...                             ; menus supplémentaires qui a été
        ...                             ; sélectionné
        bne.s   NotMenu                 ; sinon, envoie le message au WB

GereMenuSupplementaire:
        CALLEXEC    ReplyMsg            ; Ne pas oublier !!
        ...                             ; On fait ce qu'on veut
        bra AllMsgs

; ************************************
strcmp  movem.l a0-a1,-(sp)
.loop   move.b  (a0)+,d0
        beq.s   .eos1
        cmp.b   (a1)+,d0
        bne.s   .ret
        bra.s   .loop
.eos1   tst.b   (a1)+
.ret    movem.l (sp)+,a0-a1
        rts
; ************************************
myport  dc.l    0                   ; pointeur sur un MsgPort personnel
wbscr   dcb.b   sc_SIZEOF,0         ; Buffer pour une structure Screen
wbwin   dc.l    0                   ; pointeur sur la fenêtre du WB
wbmenu  dc.l    0                   ; pointeur sur le Menu du WB
le MsgPort du WB
wbtitle dc.b    "Workbench",0       ; titre de la fenêtre du WB
```

Voilà. Il n'y a bien sûr que le squelette du programme, mais celà devrait te permettre d'avancer quand même un peu... Tu peux même, si celà t'amuse, envoyer de faux messages au Workbench, ou bien en filtrer certains, les modifier... Tout (ou presque) est permis !
Au fait, si tu réalises quelque chose d'amusant à partir cette routine, soit gentil de nous faire parvenir ton programme, juste comme ça, par curiosité...

Pour terminer, un petit appel au peuple... Si certains parmi vous ont des renseignements précis sur la manière dont communiquent la carte Passerelle (XT et/ou AT) et l'Amiga 2000 (adresses, protocole, etc.), merci de passer un petit coup de fil à la rédaction, Stéphane se fera une joie de noter tout celà...

SUITE DE LA PAGE 23

```
LastDevice:
        rts

; ***** Liste les devices de type DLT_DIRECTORY
ListDirectories:
        lea msg2(pc),a0
        bsr PrintBSTR

        move.l  devinfo(a5),a2
DirectoriesLoop:
        bsr.s   CheckBreak

        cmpi.l  #DLT_DIRECTORY,dn_Type(a2)
        bne.s   NextDirectory   ; Pas celui demandé

        movea.l dn_Name(a2),a0  ; Nom du device dans a0
        adda.l  a0,a0   ; Conversion BPCL
        adda.l  a0,a0   ;       ""      ""
        bsr.s   PrintBSTR   ; Affiche le nom

        lea crlf(pc),a0
        bsr.s   PrintBSTR

NextDirectory:
        move.l  dn_Next(a2),d0  ; Device suivant dans d0
        lsl.l   #2,d0
        movea.l d0,a2
        bne.s   DirectoriesLoop ; Boucle jusqu'au dernier

LastDirectory:
        rts
; ***** Liste les devices de type DLT_VOLUME
ListVolumes:
        lea msg3(pc),a0
        bsr.s   PrintBSTR

        move.l  devinfo(a5),a2
VolumesLoop:
        bsr.s   CheckBreak

        cmpi.l  #DLT_VOLUME,dn_Type(a2)
        bne.s   NextVolume  ; Pas celui demandé

        movea.l dn_Name(a2),a0  ; Nom du device dans a0
        adda.l  a0,a0   ; Conversion BPCL
        adda.l  a0,a0   ;       ""      ""
        bsr.s   PrintBSTR   ; Affiche le nom

        lea crlf(pc),a0 ; Passage à la ligne
        bsr.s   PrintBSTR

NextVolume:
        move.l  dn_Next(a2),d0  ; Device suivant dans d0
        lsl.l   #2,d0
        movea.l d0,a2
        bne.s   VolumesLoop ; Boucle jusqu'au dernier

LastVolume:
        rts
; ***** Vérifie si CTRL/C est pressé
CheckBreak:
        moveq   #0,d0
        move.l  d0,d1
        EXEC    SetSignal
        andi.l  #SIGBREAKF_CTRL_C,d0
        bne.s   Broken
        rts

Broken  addq.l  #8,sp   ; Corrige la pile (2 fois bsr)
        lea msg4(pc),a0 ; Affiche le message 'BREAK'
        bsr.s   PrintBSTR
        moveq   #RETURN_WARN,d7 ; Code d'erreur dans d7
        bra Exit    ; Saut à 'Exit'
; ***** Affiche une chaine BCPL pointée par a0
PrintBSTR:
        moveq   #0,d3
        move.b  (a0)+,d3
        move.l  a0,d2
        move.l  stdout(a5),d1
        DOS Write
        rts
; ***** Domaine des variables
VARS    dcb.b   VARSIZE,0

dosname DOSNAME

msg1    dc.b    m1end-m1str
m1str   dc.b    $9b,"1m",$9b,"4m"
        dc.b    "Devices 'physiques' (DLT_DEVICE) :"
        dc.b    $9b,"0m",10
m1end   even

msg2    dc.b    m2end-m2str
m2str   dc.b    10,$9b,"1m",$9b,"4m"
        dc.b    "Devices 'logiques' (DLT_DIRECTORY) :"
        dc.b    $9b,"0m",10
m2end   even

msg3    dc.b    m3end-m3str
m3str   dc.b    10,$9b,"1m",$9b,"4m"
        dc.b    "Devices 'volumes' (DLT_VOLUME) :"
        dc.b    $9b,"0m",10
m3end   even

msg4    dc.b    m4end-m4str
m4str   dc.b    10,"*** Break",10
m4end   even

tab     dc.b    1,9
crlf    dc.b    1,10
```

CATS : Question de style

Nous allons ce mois-ci nous éloigner un tout petit peu de la programmation pure et dure, pour aborder un sujet tout aussi important : le "look and feel" de vos applications.

Il est en effet important que chaque programme développé pour l'Amiga, suive quelques règles de présentation simples et peu contraignantes, qui ont l'avantage de proposer à l'utilisateur un environnement homogène. De cette façon, on lui assure un sentiment de "déjà vu" lorsqu'il aborde un nouveau logiciel, il ne se sent pas complètement perdu. Celà assure de plus qu'il continuera à utiliser votre programme, au lieu de le rejeter dès le premier abord.

STYLE DE MENU

Utilisez toujours OffMenu() lorsqu'une option de menu devient inutile ou superflue. Ne laissez jamais l'utilisateur choisir une option de menu sans que votre programme puisse y réagir correctement.
Informez, chaque fois que possible, l'utilisateur qu'un sous-menu est lié à une option particulière. Par exemple :

```
Césure
Couleur du texte »
Couleur du fond  »
```

Cette méthode utilise le caractère "»" du jeu de caractère ECMA Latin-1 (code ASCII 187). Ce caractère peut apparaître à gauche ou à droite de l'option de menu, suivant l'endoit où apparaîtra le sous-menu. Si c'est à gauche, utilisez plutôt le caractère "«" (code ASCII 171).

Informez également l'utilisateur qu'un requester va apparaître consécutivement au choix d'une option de menu. Là encore, le but est de lui permettre de prévoir ce qui va se passer. Par exemple :

```
Ouvrir...
Sauver
Sauver Sous...
```

On utilise ici les trois caractères "..." qui laissent à penser que quelque chose va se passer.

Les sous-menus devraient toujours être positionnés à droite de l'option de menu correspondante (si possible). Positionnez-les de manière à éviter tout effet de clignotement lorsqu'Intuition les dessine.

Tout programme devrait être capable de gérer correctement et efficacement les sélections multiples. Ceci est particulièrement appréciable pour les options de menus crochetées.

Les couleurs de stylos que vous définissez à l'ouverture de votre fenêtre sont utilisées pour la barre de menus et les options. Si vous ouvrez plusieurs fenêtres, vous pouvez utiliser des codes de couleurs pour chaque fenêtre, afin que l'utilisateur sache en permanence quel menu appartient à quelle fenêtre. Quoiqu'il en soit, utilisez toujours des couleurs atténuées, plus agréables à l'oeil que des couleurs vives.

Si votre programme propose à l'utilisateur de travailler des documents, créez un menu "Projet". Par convention, il est établi que le menu "Projet" doit apparaître totalement à gauche de la barre de menus. Il doit proposer les options suivantes, si possible dans le même ordre :

```
Nouveau            Création d'un nouveau document
Ouvrir...          Ouvrir un document depuis le disque
Sauver             Sauver le document en cours sur disque
Sauver Sous...     Sauver le document sous un autre nom
Imprimer           Imprimer le document entier
Imprimer comme...  Imprimer une partie du document ou définir
                   les options d'impression
A propos...        Afficher les informations sur le programme
Quitter            Quitter le programme (si le document est
                   modifié, proposer une sauvegarde)
```

Si votre application possède des fonctions d'édition, vous nous suggérons de créer un menu "Edition", immédiatement à droite du menu "Projet". Il doit proposer les options suivantes, si possible dans l'ordre listé ci-dessous :

```
Annuler (Undo)  Annuler la dernière opération (si possible,
                sinon rendez cette option indisponible !)
Couper  (Cut)   Efface la partie sélectionnée du document
                et la placer dans le Clipboard
Copier  (Copy)  Place une copie de la partie sélectionnée
                du document dans le Clipboard
Coller  (Paste) Place une copie du Clipboard dans le
                document
Effacer (Erase) Efface la partie sélectionnée du document
```

STYLE DE GADGETS

Lorsque vous créez une liste de gadgets pour un requester ou une fenêtre, assurez-vous de mettre en évidence (gras, couleur...) celui qui propose d'annuler l'opération en cours ("Annuler" ou "Cancel", en général).

Créer des gadgets qui se superposent n'est pas une bonne idée. A moins que vous n'y fassiez vraiment très attention, des superpositions de gadgets peuvent entrainer des effets visuels très désagréables.

Comme pour les menus, utilisez OffGadget() pour rendre non sélectionnable un gadget devenu inutile.

L'utilisateur devrait toujours apprendre rapidement à utiliser vos gadgets proportionnels. Utilisez les flags FOLLOWMOUSE et MOUSEMOVE pour actualiser ce qui peut l'être rapidement (par exemple, les couleurs d'une palette ou une liste de noms). Vous pouvez toujours éviter un rafraichissement inutile, en vérifiant que le gadget a été suffisamment bougé pour nécessiter de redessiner ce qu'il représente.

Quand vous utilisez des gadgets proportionnels pour controler une liste (par exemple dans un sélecteur de fichier), faîtes en sortes que les valeurs HorizBody et VertBody soient initialisées au nombre d'éléments visibles moins 1. De cette façon, si l'utilisateur se déplace dans la liste en cliquant dans le container du gadget, la dernière ligne de la sous-liste affichée deviendra la première de la nouvelle, laissant ainsi l'utilisateur savoir où il en est.

Voici sûrement la règle la plus importante concernant les requesters : prévoyez toujours un moyen simple et sûr d'annuler l'action en cours. Si, par exemple, l'utilisateur sélectionne l'option "Initialize" du menu "Disk" du WorkBench, le requester qui apparaît lui laisse l'opportunité d'annuler le formatage. Ceci est extrèmement important.

Si vous construisez un requester avec une BitMap particulière, vérifiez que les boîtes de sélection des gadgets soient en accord avec le dessin de cette BitMap.

Attachez un soin tout particulier aux requester contenant des gadgets de chaîne : celui-ci doît être activé lorsque le requester apparaît. Si votre requester contient plusieurs gadgets de chaîne, presser Return dans l'un doit activer le suivant.

Chaque fois que vous présentez à l'utilisateur un choix qui peut se résumer par oui ou non, l'option "Oui" devrait se trouver à gauche du requester, et l'option "Non", à sa droite. Par exemple, si votre requester contient "Ok" et "Annuler", "Ok" sera à gauche et "Annuler" à droite. Si c'est "Retry" et "Abort", "Retry" sera à gauche et "Abort" à droite. Les fonctions AutoRequest() et DisplayAlert() suivent cette convention. Si tous les programmes l'adoptent, l'utilisateur saura automatiquement que cliquer le gadget de droite interrompra l'action en cours, même s'il ne comprend pas son libellé ou le sens exact du dessin qu'il contient.

Ne créez jamais de requester avec deux boutons identiques ! AutoRequest() peut créer des requesters avec un seul gadget, à droite. "Continuer" est alors un bon exemple de texte à y placer.

A suivre...