

AMIGA

LA

BIBLE

TOUT SUR LE HARDWARE,
LA PROGRAMMATION SYSTEME
ET LES CIRCUITS SPECIALISES

EDITIONS MICRO APPLICATION



LIVRE DATA BECKER

Bleek, Dittrich, Gelfand
Jennrich, Schemmel, Schulz

La
BIBLE

Amiga

EDITIONS MICRO APPLICATION

Copyright © 1990 DATA Becker GmbH
Merowingerstraße, 30
4000 Düsseldorf

© 1990 Micro Application
58, rue du Faubourg Poissonnière
75010 Paris

Auteurs Bleek, Dittrich, Gelfand, Jennrich, Schemmel, Schulz.

Traducteur Marc Nichanian

'Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de MICRO APPLICATION est illicite (Loi du 11 Mars 1957, article 40, 1er alinéa).

Cette représentation ou reproduction illicite, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

La Loi du 11 Mars 1957 n'autorise, aux termes des alinéas 2 et 3 de l'article 41, que les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à l'utilisation collective d'une part, et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration'.

ISBN : 2-86899-345-1

*Collection dirigée par Philippe Olivier
Edition réalisée par Frédérique Beaudonnet*

Amiga et Commodore sont des marques déposées de Commodore Electronics Limited.
Macintosh et Apple sont des marques déposées de Apple Computer Inc.
IBM est une marque déposée de International Business Machines.
Tous les autres produits sont des marques déposées de leurs sociétés respectives.

Sommaire

1. Le Hardware de l'Amiga	41
1.1. Introduction	41
1.2. Les composantes de l'Amiga	41
1.2.1. Le processeur 68000	42
1.2.2. Le CIA 8520	48
1.2.3. Rôle des circuits spécialisés dans le hardware de l'Amiga	62
1.3. Les connecteurs de l'Amiga	81
1.3.1. Connecteur Audio/Video	82
1.3.2. Connecteur RGB	84
1.3.3. Le connecteur Centronics	86
1.3.4. Le connecteur série	88
1.3.5. Connecteur disquette externe	91
1.3.6. Le connecteur souris-joystick	97
1.3.7. Connecteur d'extension	100
1.3.8. Alimentation des connecteurs	103
1.4. Le clavier	104
1.4.1. Schéma électronique du clavier	106
1.4.2. Transfert des données	107
1.5. La programmation du hardware	110
1.5.1. Organisation de la mémoire	111
1.5.2. Eléments de base	123
1.5.3. Les interruptions	135
1.5.4. Le coprocesseur COPPER	137
1.5.5. Playfields	146
1.5.6. Sprites	174
1.5.7. Le blitter	194
1.5.8. La sortie du son	230
1.5.9. Souris, joysticks et paddles	257
1.5.10. Le connecteur série	264
1.5.11. Le contrôleur de disquette	268
1.6. L'architecture d'extension	273
1.6.1. Le hardware	273
1.6.2. La partie logicielle	279
1.7. La conception Janus - Amiga et PC	281
1.8. La structure de la carte-passerelle (bridgeboard) PC	282

2. Exec	287
2.1. Les listes chaînées	287
2.2. Fonctions Exec pour la gestion des listes	295
2.3. Macros en assembleur pour la gestion des listes	300
2.4. Programmation professionnelle en assembleur	305
2.4.1. Indications sur l'utilisation des assembleurs	306
2.4.2. L'utilisation de macros	308
2.4.3. Utilisation des fichiers Include	313
2.4.4. Remarques pour une programmation "élégante"	316
2.4.5. Utilisation des bibliothèques de liens	320
2.5. L'utilisation des tables de fonctions	327
2.5.1. Ouverture et fermeture d'une bibliothèque	330
2.5.2. Autres fonctions sur les bibliothèques	335
2.6. Le fonctionnement Multi-tâches	336
2.6.1. La structure Task	337
2.6.2. Fonctions pour les tâches	346
2.6.3. Communication entre tâches	350
2.7. Fonction de la mémoire de l'Amiga	370
2.7.1. Les fonctions AllocMem() et FreeMem()	372
2.7.2. La structure Memory-List	374
2.7.3. L'affectation de mémoire et les tâches	377
2.7.4. La gestion interne de la mémoire	377
2.7.5. Les fonctions Allocate, Deallocate et AddMem	380
2.7.6. Description des autres fonctions	382
2.8. La structure interne des librairies	383
2.8.1. Modifier une librairie	385
2.8.2. La création d'une librairie	386
2.9. Structure interne d'un device Exec	397
2.9.1. L'intégration du device dans le système	397
2.9.2. Division interne du device	398
2.9.3. Les commandes Device	400
2.9.4. L'initialisation des devices	402
2.9.5. Un exemple de device	402
2.10. Manipulations des interruptions	416
2.10.1. Utilisation des interruptions	419
2.10.2. Exemples de programmes	427
2.11. Sémaphores	431
2.11.1. Les structures de sémaphores	433
2.11.2. Les fonctions de sémaphores-signaux	435
2.11.3. Utilisation des ports de sémaphores	440
2.11.4. Exemple d'utilisation des sémaphores-signaux	442
2.11.5. Exemple d'utilisation des ports de sémaphores	445
2.12. La librairie RAM	449
2.13. La structure ExecBase	451

2.14.	Programmes résistants au reset et secteurs de données	460
2.14.1.	Intégration dans un reset par les vecteurs Capture	461
2.14.2.	Les modules résidents	464
3.	L'AmigaDOS	477
3.1.	Du CLI au hardware : la hiérarchie du DOS	477
3.1.1.	Le premier contact : Le CLI	477
3.1.2.	La bibliothèque DOS	478
3.1.3.	Les handlers	478
3.1.4.	Les systèmes de fichiers	480
3.1.5.	Les devices	480
3.2.	La bibliothèque DOS	480
3.2.1.	Chargement de Dos.Library	481
3.2.2.	Appels d'une fonction et transmission de paramètres	482
3.2.3.	Les fonctions DOS	482
3.2.4.	Messages d'erreur du DOS	489
3.3.	Entrée/Sortie	491
3.3.1.	Clavier et écran	495
3.3.2.	Fichiers sur disquettes	502
3.3.3.	Interface sérielle	503
3.3.4.	Interface parallèle	504
3.4.	Programmes	504
3.4.1.	Lancement d'un programme et paramètres	504
3.4.2.	Structures d'un fichier programme	513
3.4.3.	Un exemple de handler DOS	525
3.5.	Les disquettes	530
3.5.1.	Le bootage	530
3.5.2.	Distribution des données sur la disquette	532
4.	Entrée et sortie par les devices	543
4.1.	Qu'est-ce qu'un device ?	543
4.2.	Communiquer avec les devices	552
4.3.	Le device parallèle	555
4.3.1.	L'ouverture du device	555
4.3.2.	Ecrire des données	556
4.3.3.	Lire des données	557
4.3.4.	Obtenir l'état de l'interface	558
4.3.5.	Messages d'erreur du device parallèle	560
4.3.6.	Les affectations du port Centronics	560
4.4.	Le device série	561
4.4.1.	L'ouverture du device	562
4.4.2.	Lire et écrire par l'intermédiaire de l'interface série	563
4.4.3.	Les différents paramètres du device série	564
4.4.4.	L'état de l'interface	566

4.4.5.	Messages d'erreur	567
4.5.	Le device d'imprimante	571
4.5.1.	Imprimer des textes non traités	572
4.5.2.	La substitution des séquences Escape	573
4.5.3.	Les commandes d'imprimante	576
4.5.4.	Copies d'écran	576
4.5.5.	Messages d'erreur du device d'imprimante	582
4.5.6.	Le device d'imprimante sous Kickstart V1.3	582
4.6.	Le device de clavier (Keyboard-device)	583
4.6.1.	L'ouverture du device de clavier	584
4.6.2.	La lecture du device	584
4.6.3.	Resets par l'intermédiaire du device de clavier	585
4.6.4.	Un exemple de programme	586
4.7.	Le device du port de jeu (gameport)	587
4.8.	Le device Input	594
4.8.1.	Les fonctionnalités du device d'input	594
4.8.2.	Le premier pas : l'ouverture	595
4.8.3.	Utilisation du device d'input	595
4.8.4.	Le device d'input, souris et clavier	608
4.9.	Le device de console	612
4.9.1.	Keymapping	625
4.9.2.	Eléments internes à la console	638
4.10.	Le device Clipboard	641
4.11.	L'Amiga comme synthétiseur de sons - le device Audio	645
4.11.1.	Définir les canaux Audio	646
4.11.2.	Définir le volume	650
4.11.3.	Enfin, on entend quelque chose	652
4.11.4.	Autres fonctionnalités du device Audio	654
4.11.5.	Son dans le code Interrupt	666
4.12.	...et comme synthétiseur de parole	667
4.13.	Le device Timer	677
4.14.	Le device TrackDisk (TrackDisk-device)	687
4.14.1.	Lire et écrire des secteurs	688
4.14.2.	Ecriture et lecture des tracks non traités	692
4.14.3.	Formatage d'une disquette	693
4.14.4.	Commandes d'état	694
4.14.5.	Disk-Interrupts	696
4.14.6.	Traitement des erreurs de disque	699
4.14.7.	L'éditeur de disquette	700
5.	Le format d'échange standard	713
5.1.	Les formats IFF de Electronic Arts	713
5.1.1.	Le format graphique IFF-ILBM	714
5.1.2.	Le format texte IFF-FTXT	727
5.1.3.	Le format musical IFF-SMUS	727

5.1.4. Le format IFF-8SVX-Sample	731
5.2. Le format ANIM	732
6. Les librairies de l'Amiga	743
6.1. La librairie Exec	744
6.2. La DOS-Librairie	799
6.3. La librairie Intuition	820
6.4. La librairie Layers	877
6.5. La librairie Icon	892
6.6. La librairie Graphics	903
6.7. La librairie Diskfont	
6.8. Les librairies Mathématiques	981
6.8.1. La librairie Mathfp	981
6.8.2. La librairie MathTrans	986
6.8.3. La Mathl ^{eeee} DoubBas	994
6.8.4. La librairie Mathl ^{eeee} DoubTrans	999
6.9. La librairie Expansion	1005
7. L'Amiga 3000	1021
7.1. Caractéristiques techniques	1021
7.2. La nouvelle interface utilisateur	1022
7.2.1. La fenêtre Workbench	1022
7.2.2. La barre des menus	1024
7.3. Le Shell et l'Amiga OS 2.0	1029
7.3.1. Le Shell	1029
7.3.2. Les nouvelles commandes Amiga OS 2.0	1031
7.4. Les nouvelles "Preferences"	1035
Annexe	1043
Index	1087

Introduction

Bible, voilà un bien grand mot ! Que se cache-t-il derrière ? Avec "La bible Amiga", vous savez exactement ce qui vous attend : comment fonctionnent les routines système, quelles tâches accomplissent-elles, que peut et que fait le hardware ? Mais le livre que vous tenez entre les mains fait bien plus que répondre à ces questions. En premier lieu, il contient une partie centrale, le chapitre 6.

Dans ce chapitre, vous trouverez toutes les fonctions des bibliothèques Amiga, avec les explications adéquates. Pour chaque bibliothèque (Library), on a une documentation permettant une application immédiate des fonctions, aussi bien en Assembleur qu'en C. L'explication contient le format général et les paramètres transmis, mais aussi pour chaque valeur une définition avec les variables en C, ce qui permet à l'utilisateur débutant de retrouver facilement d'éventuelles erreurs dans un programme. En outre, les renvois permettent de retrouver facilement des fonctions en relation avec le même thème.

Exemple

Vous cherchez la fonction qui sert à fermer une fenêtre (Window), mais vous ne connaissez pas son nom. La seule chose que vous sachiez est qu'elle se trouve dans la bibliothèque Intuition, car c'est Intuition qui gère les fenêtres. Voici donc comment procéder :

- ① Si vous avez déjà marqué les différentes sections du chapitre sur les bibliothèques à l'aide de petits auto-collants, vous retrouverez facilement la bibliothèque Intuition. Sinon, consultez la table des matières.
- ② Examinez maintenant le groupe des fonctions Windows. Vous trouverez ici CloseWindow() et le numéro de la page contenant les explications sur cette fonction.
- ③ Ouvrez le livre à cette page ! Voici toutes les informations dont vous avez besoin. Au début du chapitre sur les bibliothèques, vous pourrez lire des indications générales sur la présentation de ces informations.

Pour ceux d'entre vous qui connaissent déjà le nom de la fonction, il existe un moyen plus rapide pour parvenir à ces informations. Prenez dans l'annexe la liste alphabétique des fonctions. Vous trouverez ici pour chacune des fonctions l'indication de la page où figurent les explications qui la concernent.

L'autre section importante de ce livre est le chapitre sur les devices de l'Amiga. Chaque échange de données avec les périphériques externes ou les périphériques simulés internes s'effectue par l'intermédiaire d'un device. Pour chaque device, vous trouverez un programme d'exemple, et des explications sur les différents modes et les diverses commandes de ce device. Vous pourrez ainsi enfin adresser les périphériques d'entrée et de sortie de l'Amiga de manière correcte et exhaustive, sans être arrêté par les déficiences de la documentation.

Au début de ce livre, vous pourrez lire une introduction concernant les conventions pour le style de programmation. Ce chapitre d'introduction contient des explications détaillées sur la division, l'organisation et la distribution des différentes parties dans un programme, et constitue ainsi une base utile pour une programmation clairement structurée. Vous pouvez voir ici comment on intègre un numéro de version, comment on appelle les bibliothèques, et comment on introduit des commentaires dans le programme, pour être en mesure de le comprendre ultérieurement.

Vous trouverez ensuite des informations sur la transmission des paramètres au début du programme. Il ne s'agit pas seulement de la façon de rassembler les valeurs entrées avec le CLI. La transmission des paramètres par l'intermédiaire du Workbench, utilisée par tous les programmeurs professionnels, est beaucoup plus importante, et malheureusement bien moins connue. Vous saurez ici enfin quelles fonctions utilisent les structures .info, et comment on définit une routine correspondante au début du programme, pour permettre les deux modes de lancement, par le CLI et par le Workbench.

Tout le monde veut être compatible. Mais comment y parvenir si l'on ne connaît pas le format IFF ? Vous trouverez des informations sur ce point au chapitre 5. Ce chapitre contient une description du format ILBM, relativement simple. Mais il contient également des listes et des tableaux concernant les formats musique et texte, pour tous les "chunks" connus. Ce chapitre est conclu par la description du format Anim de Aegis, qui constitue une liaison complexe entre les chunks ILBM et ses chunks propres.

Il reste le dernier chapitre, septième du nom, dans lequel nous nous occupons des structures de base du système d'exploitation de l'Amiga. Elles sont souvent insuffisamment connues, alors qu'elles interviennent dans tous les programmes. Pouvez-vous travailler sans les "keymaps" si vous interrogez le clavier ? N'avez-vous pas toujours besoin des réglages de Preferences pour votre programme ? Comment ferez-vous pour mettre en forme vos sorties, si vous ne disposez pas des fontes Amiga ? Ici encore, vous trouverez un tas d'informations complémentaires sous la forme de programmes d'exemple, de documentations sur les structures, et d'explications détaillées sur tous les paramètres.

Conventions pour le style de programmation

La communication avec l'ordinateur ne peut se faire qu'en respectant certaines conventions, comme c'est d'ailleurs le cas dans toute relation entre les hommes. Il existe des conventions générales, mais aussi d'autres plus subtiles, souvent oubliées, et pourtant dignes d'être prises en compte. Il s'agit des divisions et des présentations graphiques ou

optiques, qui sont normées sur l'Amiga, comme sur tous les autres ordinateurs. C'est une façon de simplifier la tâche des utilisateurs qui doivent passer d'un programme à l'autre, lorsque ces programmes sont les œuvres de programmeurs différents, ce qui est très souvent le cas.

En outre, il existe évidemment des conventions qui concernent la programmation en elle-même. Nous allons commencer par ces conventions et ces définitions devant entrer dans le texte des programmes, car la seconde partie, dans laquelle il sera question aussi de la forme extérieure des programmes, repose là-dessus.

Mettre en forme correctement le texte du programme

Dans le domaine de la programmation professionnelle, le texte d'un programme joue un rôle important. Prenez l'exemple d'une société qui produit plusieurs logiciels pour l'Amiga. Quelques mois après, Commodore sort le nouveau système d'exploitation, et les logiciels existants doivent être corrigés ou réécrits, à cause des modifications introduites. Mais la société a oublié d'insérer des commentaires à l'intérieur des programmes, et les programmeurs ont changé. Plus personne ne sait à quoi correspond telle ou telle partie du programme.

La même chose peut vous arriver personnellement, si vous écrivez un programme d'une certaine longueur, et si vous n'insérez pas de commentaires. Il suffit de quelques semaines pour perdre de vue ce qu'on a fait dans telle ou telle partie du programme. Il faut alors se mettre à analyser celui-ci, ce qui est un travail très pénible, et qui demande beaucoup de temps. Pour s'épargner ce travail autant que possible, il est essentiel de structurer les programmes, et de les accompagner de commentaires pour qu'ils demeurent lisibles ultérieurement. Voici les éléments qu'il faut utiliser pour cela :

L'en-tête du programme

Dans l'en-tête d'un programme, on place dans tous les cas le nom du programme et la tâche qu'elle exécute. On y trouve également la date de création, le nom de l'auteur, et d'autres données. Considérez l'exemple suivant:

```
*****  
*  
* Programme: Windows définition locale *  
* _____ *  
* Auteur: Date: Commentaire: *  
* ----- ----- ----- *  
* Wgb 16.10.1988 Premier test *  
* Window *  
******/
```

Comme l'indiquent les premières lignes, il s'agit de l'en-tête d'un programme. Ce programme en C définit localement une fenêtre (Window). Cela veut dire que les données de la fenêtre ne sont pas conservées globalement, mais ne sont accessibles qu'à une fonction particulière. Ensuite vient un petit tableau, dans lequel sont mentionnés l'auteur, la date et un commentaire. La première mention, comme on le voit, concerne

l'auteur du programme. On y trouve une abréviation servant à l'identifier, puis vient la date de création, et enfin une remarque quelconque. Ce tableau peut ensuite être complété par toutes les personnes qui modifient quelque chose au programme. Ces dernières inscrivent leur nom, et indiquent sous la ligne de commentaire ce qui a été corrigé ou modifié.

De cette façon, chaque nouveau programmeur peut voir facilement quelles ont été les modifications, les améliorations ou les suppressions entreprises à l'intérieur du programme. L'en-tête de programme qui précède ne contient pas d'autres indications importantes. Mais on peut ajouter de telles indications. Voici un exemple :

```
*****  
*  
* Ouvrir une fenêtre avec Intuition    *  
* Auteur:          W.G Bleek           *  
* Date:           22 mai 1988          *  
* Salutations:    Denis "Angle"      *  
* Version:        1.1                 *  
* Système d'exploitation: V1.2 & V1.3 *  
*  
*****
```

Outre les indications déjà mentionnées, on trouve ici des informations sur la version du programme et sur le système d'exploitation. Celles-ci sont essentielles. En effet, sans la version du système d'exploitation, il est impossible de savoir si le programme doit être compilé, et comment il doit l'être, si on l'a écrit par exemple en C. Pour les programmes en BASIC, ces valeurs sont tout aussi importantes, car il faut encore charger des bibliothèques, et celles-ci diffèrent selon le numéro de version.

Les "salutations" ne sont pas indispensables, mais chacun peut apporter une touche personnelle au texte de son programme. Un programmeur n'est pas obligé non plus de se prendre terriblement au sérieux. Si l'on veut en outre dédier son oeuvre à quelqu'un, il faut bien mentionner son nom !

Le numéro de version du programme doit se trouver d'une part dans le texte du programme, pour qu'il soit possible de différencier entre deux versions d'un même programme à l'aide de ce numéro ; mais, pour la même raison, il faut aussi qu'il apparaisse dans le programme lui-même. Cela veut dire qu'il doit se trouver dans le code du programme, pour rendre possible l'identification, mais aussi dans la barre des titres ou à un emplacement analogue, pour que l'utilisateur puisse savoir toujours exactement avec quelle version il travaille. En effet, il arrive souvent qu'il y ait des différences considérables entre les versions.

Le système d'exploitation doit être mentionné sans faute, lorsque l'ordinateur est sujet à des modifications du système. C'est la seule façon de garantir une utilisation sans défaillances. Cette caractérisation permet également de savoir si le programme utilise des fonctionnalités implémentées dans des versions récentes. Ainsi, un programme compatible à partir du Kickstart 1.1 ne soutiendra certainement pas le bootage automatique à partir du disque dur, car il n'était encore connu de personne à cette époque. D'un autre côté, le programme fonctionnera sur n'importe quel Amiga s'il appelle les fonctions du système d'exploitation dans les règles.

Le numéro de version

Pour finir, disons quelques mots concernant le numéro de version. Il doit obéir à certaines règles, que vous ne devez pas ignorer ! C'est la seule façon de savoir jusqu'où vont les différences avec l'ancienne version. Le numéro de version comprend deux nombres, séparés par un point:

Version X.Y

On commence la numérotation par 0. Le numéro de gauche représente ce qu'on appelle la version principale. Elle est désignée par 0 tant qu'on se trouve dans la phase des tests. La première version du logiciel utilisable et complète est désignée ensuite par le chiffre 1. Toutes les versions intermédiaires peuvent être définies avec exactitude par Y. Ce nombre commence lui aussi à 0 et il peut aller jusqu'à 9. On peut encore diviser cette unité en dixièmes et en centièmes, de sorte qu'il est possible d'effectuer 100 modifications dans un logiciel avant de changer la version principale.

C'est au programmeur de juger du moment où il doit modifier le numéro de version, et jusqu'où il ira. Je commence ainsi par la version 0.1, lorsque la première version de test est prête, et que l'on peut s'en servir. Tout ce qui précède n'a pas de numéro. Si je n'ajoute pas de composantes essentielles dans ce programme, en me contentant de corriger les erreurs, je numérote de 0.11 à 0.19, et j'utilise au besoin les millièmes. Ce n'est qu'au moment où j'introduis une nouvelle fonction dans le logiciel que je change le numéro de version, pour l'appeler 0.2 !

Supposons que j'en sois à 0.7, et que le logiciel soit entièrement prêt, sans avoir besoin d'autre extension. Je lui donne alors le numéro 1.0. Prenons pour exemple les numéros de version du système d'exploitation de l'Amiga. Les premiers appareils étaient équipés du système d'exploitation 1.0. C'était la première version en état de marche. Il a fallu la modifier très rapidement, car elle contenait encore de nombreuses erreurs. C'est ainsi que la version 1.1 est apparue sur le marché ! Mais lorsque la diffusion en Europe a commencé, on a ajouté d'autres améliorations, et on en était déjà à la version 1.2.

Vous vous demanderez peut-être où sont restées toutes les versions intermédiaires. Je répondrai à cela qu'un numéro de version doit demeurer compréhensible. Il serait ainsi inhumain et très peu pratique d'appeler une version principale par exemple 1.2759. Personne ne pourrait s'y retrouver, en lisant sur un emballage de logiciel: "Ce logiciel fonctionne avec toutes les versions à partir de 1.2623, mais pas sur les versions intermédiaires 1.2758 et 1.296". C'est pourquoi on s'est décidé à introduire une "sur-version", qui permettent à tous les utilisateurs de connaître en gros les différences entre les versions.

Si l'on veut en savoir plus, on sélectionne l'option "Version" dans le menu Workbench, et l'on obtient l'indication précise: elle fournit en effet la version Kickstart et Workbench actuelle. Avec 1.2, Kickstart se trouve par exemple à 33.192, ce qui s'explique par les bibliothèques: 33 est en effet le numéro de version des bibliothèques. Nous y reviendrons plus loin dans ce chapitre, lorsque nous parlerons des bibliothèques en particulier. Une

version très répandue du Workbench est le numéro 33.57, qui a été cependant élevé à 34.28 à cause de Kickstart 1.3. C'est tout pour le moment ce qui concerne les versions, les numéros de version, les versions principales, les versions intermédiaires et les versions finales...

L'en-tête des fonctions

Chaque fonction (chaque sous-programme) devrait de même porter un en-tête. Ici, les informations exigées sont cependant différentes. Pour la programmation, on a besoin d'avoir des renseignements sur les paramètres transmis ou retournés, sur les variables influencées par le sous-programme, et sur les conditions qui doivent être satisfaites. Considérez l'exemple suivant:

```
*****  
*  
* Fonction servant à supprimer une *  
* structure dans le lexique: *  
* FreeLexique() *  
*  
* Extensions: *  
* - soutien des entrées *  
* - recherche si sauvegardé *  
*  
* Paramètres d'entrée: *  
* Lexique - pointeur sur la structure *  
* existante *  
*  
* Paramètres de sortie: *  
* Aucun *  
*  
* Date: 03 avril 1988 *  
* Auteur: W.G. Bleek *  
* Salutations: Christian *  
******/
```

Dans cet en-tête de fonction, nous trouvons signalée tout à fait au début la tâche du sous-programme, pour qu'il puisse être facilement localisé. On indique ensuite la fonction C, pour que l'on ne soit pas obligé de la chercher dans le texte du programme. C'est également utile pour le cas où l'on recherche un nom de fonction par exemple avec un traitement de texte. Celui cherchera d'abord l'en-tête où se trouvent toutes les informations essentielles.

La ligne suivante est introduite par "Extensions", ce que je trouve très utile en particulier dans la phase de développement. En effet, il arrive souvent que l'on planifie des modifications pendant le travail, sans pouvoir les réaliser tout de suite. C'est là que je reporte ces modifications à effectuer, de façon à ne pas les oublier dans la suite du travail.

Puis viennent les paramètres d'entrée. Ce sont toutes les valeurs qui seront transmises à la fonction ou au sous-programme. Il est vrai qu'en C, AmigaBASIC ou dans d'autres langages de programmation, on peut choisir des noms assez longs, mais il est néanmoins conseillé de porter ici une explication plus détaillée sur la variable. On peut dire la même

chose sur les paramètres de retour, souvent réduits à un seul, dont la signification est cependant essentielle. Il représente en effet le résultat de tout ce qui est accompli par cette routine. En fin de compte viennent les indications connues, concernant la date, l'auteur et les salutations. Celles-ci peuvent évidemment être complétées à volonté, comme nous l'avons vu plus haut.

Travailler dans un canal de communication Multi-tâches

Dans le système d'exploitation Multi-tâches, il existe un certain nombre de règles qui doivent préside à la mise en forme du programme pour qu'il n'entre pas en conflit avec d'autres programmes. L'aspect le plus important à cet égard est celui de l'ouverture et la fermeture des canaux de communication. On entend par là aussi bien les canaux destinés à la communication avec l'utilisateur et les périphériques que ceux qui concernent la communication avec les routines du système. Tous ces canaux ne sont pas adressés directement, mais par l'intermédiaire de systèmes de gestion.

On doit se représenter l'ouverture d'un canal de données comme l'ouverture d'une porte. Si elle est ouverte, on peut parler avec les gens qui se trouvent dans l'autre espace et échanger des informations. Dès qu'elle est fermée au contraire, la conversation prend fin. Mais une autre règle veut qu'une porte soit toujours fermée si on n'a pas besoin de passer par là, et si personne ne doit être dérangé. C'est vrai également pour les ordinateurs. Tous les canaux de données doivent être fermés dès que l'on n'en a plus besoin.

On peut également effectuer une comparaison avec le téléphone. On essaie d'abord d'obtenir la communication. En général, cela se fait en composant un numéro sur un cadran. Si l'on obtient la communication, le téléphone réagit par un signal sonore ; il peut aussi arriver que l'on ne l'obtienne pas, et l'on est prévenu alors par le signal "occupé" ; dans ce cas, on raccroche ! De même, l'Amiga prévient l'utilisateur, lorsqu'il n'est pas possible provisoirement d'ouvrir une bibliothèque ou d'effectuer une action analogue. Cela veut dire qu'il n'y a pas assez de mémoire disponible, ou que le programme n'a pas trouvé de "correspondant à ce numéro", auquel cas il renonce.

Comment faut-il donc s'y prendre pour ouvrir ou fermer des bibliothèques, des devices ou des canaux de données ? Considérons pour cela l'exemple d'une bibliothèque. Pour l'ouverture, il suffit d'avoir le nom et le numéro de version. On peut alors lancer un essai. On obtient en retour l'adresse de base ou un zéro pour signaler une erreur. Le fait d'une erreur n'est jamais à exclure, il faut pouvoir la contrôler, et ne pas être pris au dépourvu. On le fait par exemple de la façon suivante :

```
Libray = OpenLibrary("LibName", Version);
IF (Library = 0)
    Annuler();
```

Pour la version, il faut indiquer le numéro de version de la bibliothèque dont le programme a besoin. Ce numéro s'obtient avec la nouvelle commande "Version" du CLI. Celle-ci peut également fournir les numéros de version des bibliothèques. Mais il

est également possible d'entrer ici 0. Le programme travaille alors avec toutes les bibliothèques, même avec celles qui ne sont pas étendues. Une méthode couramment utilisée rassemble l'ouverture et l'interrogation dans une ligne de commande unique:

```
IF (!(Library = OpenLibrary("LibName", Version)))
    exit(FALSE);
```

La fermeture est une opération très simple. La seule chose dont il faut tenir compte est qu'une bibliothèque qui n'a pas pu être ouverte ne doit pas non plus être fermée. Sinon, le système s'effondrerait immédiatement. Testons donc d'abord si le pointeur s'est vu attribuer une valeur quelconque:

```
IF(Library) CloseLibrary(Library);
```

De cette façon, on ne court aucun risque ! L'opération de fermeture est légèrement plus compliquée lorsque l'on annonce plusieurs accès à l'intérieur d'un programme. Dans ce cas, toutes les bibliothèques ouvertes doivent être fermées à nouveau, ce qui paraît à première vue simple et facile à réaliser. Mais un problème se pose aux endroits où tout n'a pas pu être ouvert, et où le programme a dû se terminer prématurément. Dans ce cas, une routine CleanUp ne doit fermer que ce qui est réellement ouvert.

Voici à nouveau un exemple pour expliquer ce qui vient d'être dit: votre programme a besoin de deux bibliothèques système, d'une fenêtre et de plusieurs blocs de mémoire. En essayant d'ouvrir la fenêtre, vous obtenez un message d'erreur. Un mauvais programmeur interromprait le programme en cet endroit, ce qui poserait de nombreux problèmes au système Multi-tâches. Ainsi, les deux bibliothèques restent à la disposition du programme, alors que celui-ci a déjà pris fin, et elles occupent donc inutilement de la place en mémoire. Elles gênent d'autres tâches qui s'accomplissent simultanément, d'une part en ne restituant pas la place qu'elles occupent, d'autre part en faisant croire que l'accès à cette bibliothèque reste actif.

Pour économiser de la mémoire, et garder ouvert l'accès aux bibliothèques, il existe une méthode très simple. Nous exécuterons tous les accès aux dispositions du système d'exploitation avec une fonction `Open_All()`, et de manière équivalente toutes les opérations de fermeture avec une fonction `Close_All()`. S'il se produit une erreur lors de l'ouverture d'un secteur de mémoire ou d'une bibliothèque, on passe tout de suite à la routine de fermeture, et l'on met fin de cette façon au programme. La routine de fermeture a cependant un autre avantage: elle sait exactement ce qui a été ouvert, et ce qui ne l'a pas été. Elle peut donc, lors d'une interruption, fermer uniquement ce qui avait été ouvert. Voici maintenant un exemple concret:

```
*****  
* Fonction: Ouvrir Librarys et Window *  
* _____ *  
* *  
* Auteur: Date: Commentaire: *  
* ----- ----- ----- *  
* Wgb 15.06.1988 Mémoire aussi *  
* *  
*****  
Open_All()  
{
```

```
void      *OpenLibrary();
struct Window *OpenWindow();

if (!(IntuitionBase = (struct IntuitionBase *)
      OpenLibrary("intuition.library", 0L)))
{
  printf("Pas de Intuition - Library trouvée!\n");
  Close_All();
  exit(FALSE);
}
if (!(GfxBase = (struct GfxBase *)
      OpenLibrary("graphics.library", 0L)))
{
  printf("Pas de Graphics - Library trouvée!\n");
  Close_All();
  exit(FALSE);
}
if (!(FirstWindow = (struct Window *)
      OpenWindow(&FirstNewWindow)))
{
  printf("La fenêtre ne veut pas s'ouvrir!\n");
  Close_All();
  exit(FALSE);
}

UndoBuffer = AllocMem(512L, MemoryType);
if (!UndoBuffer)
{
  printf("Il y a des problèmes avec le tampon Undo!\n");
  Close_All();
  exit(FALSE);
}

FileBuffer = AllocMem(30L, MemoryType);
if (!FileBuffer)
{
  printf("Il y a des problèmes avec le FileBuffer!\n");
  Close_All();
  exit(FALSE);
}
} ****
* Fonction: Fermer tout ce qui est ouvert *
* _____ *
*          *
*          *
*          *
*          *
* Auteur: Date:    Commentaire:   *
* ----- ----- -----   *
* Wgb     15.06.1988  Intuition, Window   *
*          Graphics & Mem   *
****/
Close_All()
{
  if (FirstWindow)    CloseWindow(FirstWindow);
  if (IntuitionBase) CloseLibrary(IntuitionBase);
  if (GfxBase)        CloseLibrary(GfxBase);
```

```
if (UndoBuffer)      FreeMem(UndoBuffer, 512L);
if (FileBuffer)     FreeMem(FileBuffer, 30L);
}
```

La routine Close_All satisfait toutes les exigences que nous avons formulées: elle peut être appelée à partir de n'importe quel endroit du programme, et elle ferme tout ce qui a été ouvert. Tout ce qui n'a pas été ouvert n'est pas fermé, ce qui permet d'éviter en partie les erreurs fatales.

On a choisi ce moyen, car il représente le procédé de manière particulièrement bien structurée. Mais on pourrait aussi bien déclencher la routine Open_All() à partir de sa fonction, et la placer au début du programme, ce qui permet d'économiser quelques octets dans les programmes ultérieurs. Tant que l'on est sûr de ce qu'on fait, il n'y rien à y redire.

En principe, on pourrait aussi placer Close_All à la fin de la fonction Main(), mais cela peut parfois exiger que l'on saute à cet endroit, lorsqu'une erreur survient. Cela ne fait pas bonne impression, surtout en langage C. Mais c'est légitime, et c'est réalisable sans problème. En outre, on économise ici aussi quelques octets de mémoire.

Je voudrais simplement déconseiller ici le lecteur de remplacer chaque IF par une référence de saut, comme on le voit trop souvent dans les logiciels du domaine public. On aboutit en effet rapidement de cette façon à un mic-mac invraisemblable pour les lecteurs du programme. En outre, cela provoque très rapidement des erreurs lorsque l'ordre n'est pas correct, et que l'on ferme ainsi des choses qui n'avaient pas été ouvertes le moins du monde. En conclusion: restez-en à notre bonne vieille méthode. Elle a le mérite d'être sûre, et d'avoir fait ses preuves !

Directives spéciales pour la programmation en assembleur

Ceux qui programment en assembleur peuvent prendre leurs aises, bien plus que les programmeurs en C. Ils peuvent en effet exécuter directement toutes les tâches, et n'ont pas besoin d'un compilateur et d'un éditeur de liens. Il existe toutefois certaines conditions observées par le compilateur de notre langage de programmation, conditions qui doivent être respectées également par les programmeurs en assembleur. Je veux parler des registres.

Registres de données

Le processeur de notre Amiga possède 8 registres de données (A0 à D7) et 8 registres d'adresse (A0 à A7), plus le compteur d'adresses (Program Counter PC), deux pointeurs de pile (USP et SSP) et le registre d'état (SR). Tous ces registres peuvent être utilisés par le programme. Il y a cependant certaines directives que l'on ne doit pas ignorer, si l'on veut s'y retrouver dans le système d'exploitation de l'Amiga. En effet, toute intervention des fonctions de bibliothèque suppose que l'on sache quels sont les registres utilisés et ceux qui ne le sont pas.

Registre A7 ("Registre interdit")

Le premier registre interdit est le registre d'adresse A7. Celui-ci est normalement utilisé comme pointeur de pile (Stack Pointer SP), et il ne peut donc pas être chargé avec les données de l'utilisateur. La même chose vaut pour le pointeur de pile User (USP), et le pointeur de pile Supervisor (SSP). L'utilisation de ces trois registres suppose une connaissance très approfondie du système, et il vaut mieux ne pas en faire usage. On risque en effet d'empêcher le fonctionnement Multi-tâches, ce qui n'était pas dans les intentions de l'inventeur de notre Amiga.

Registres en relation avec les bibliothèques

Lorsqu'on utilise des routines système dans le programme (Librarys), il faut respecter certaines conventions, sans quoi on risque de rencontrer ici encore quelques difficultés. Il faut parler en premier lieu du registre d'adresse de base. Lorsqu'on appelle une routine, l'adresse de base de la bibliothèque se trouve toujours en A6. Cette adresse de base pourrait théoriquement être déposée dans un autre registre, mais les routines des bibliothèques utilisent aussi des variables internes, adressées par l'intermédiaire de ce registre. Chaque fonction suppose donc que son adresse de base se trouve en A6. Ce registre n'est modifié dans aucune bibliothèque. A chaque appel, on peut donc partir du fait que la valeur de base se trouve bien à cet endroit.

Les deux premiers registres d'adresse et de données (A0, A1 & D0, D1) se comportent différemment. Ils sont utilisés par presque toutes les fonctions, et ne sont pas sauvegardés ou restaurés. Lors de la programmation, il faut simplement veiller à ce que ces registres ne soient en aucun cas utilisés par le programme entier. Pour de petits transferts de données ou des calculs secondaires, on peut toujours en faire usage.

La Dos.Library représente une exception par rapport aux modifications dont il vient d'être question. Elle utilise en effet pour son travail les registres D2 et D3 également, car elle a besoin de beaucoup de mémoire intermédiaire. Faites donc attention aussi à ces registres lorsque vous travaillez avec la Dos.Library. Tous les autres registres sont sauvegardés par la fonction elle-même, si celle-ci les utilise.

Nous allons encore faire quelques remarques sur la sauvegarde des registres. Lorsque vous écrivez dans vos programmes des fonctions qui vous sont propres, il est fortement conseillé de noter dans l'en-tête de la fonction, outre les indications mentionnées plus haut, également les registres sauvegardés et non sauvegardés. Cela facilite grandement la tâche de programmation ultérieure. Le procédé le plus simple est évidemment de s'en tenir aux conventions des bibliothèques. De cette façon, aucune considération individuelle n'intervient.

Il faut noter que l'habitude veut que l'on sauvegarde les registres à l'intérieur d'une fonction et non pas avant l'appel de la fonction. Ici encore, ne variez pas du procédé que vous avez choisi, et ne mélangez en aucun cas les façons de faire !

Utiliser correctement les répertoires système

Beaucoup de programmes sur l'Amiga ne fonctionnent pas sans être soutenus. Ils reçoivent ce soutien des données existantes comme les fontes, les pilotes d'imprimante, les bibliothèques, etc. Mais où trouveront-ils ces données, si nous les utilisons nous aussi ? Ici encore, une convention s'est imposée. Toutes les données et tous les programmes accessibles à tous doivent se trouver dans les répertoires système. Ces répertoires sont accessibles sous un nom général, pour que le programme n'ait pas à les chercher longtemps. On obtient une liste de ces répertoires par l'intermédiaire de la fonction ASSIGN du CLI:

```
Directories:  
ENV      RAM DISK:Env  
T         RAM DISK:T  
FONTS    Workbench 1.3 :fonts  
S         Workbench 1.3 :S  
L         Workbench 1.3 :L  
C         Workbench 1.3 :c  
DEVS     Workbench 1.3 :devs  
LIBS     Workbench 1.3 :libs  
SYS      Workbench 1.3 :
```

On trouve ici la liste des répertoires que l'on peut toujours adresser sous un nom défini et connu universellement. Ce nom général conduit alors, par l'intermédiaire de la liste des répertoires, au périphérique proprement dit et à son répertoire. Cela permet d'assurer un fonctionnement sans heurt. Voici un exemple: votre programme permet de sortir un texte quelconque sur l'imprimante. L'utilisateur peut choisir en outre lui-même une police de caractères Amiga. Le programme va chercher cette police de caractères sur la disquette Workbench qui se trouve dans le lecteur DF0. On pourrait donc écrire "DF0:fonts/name". Mais cette écriture ne fait que générer des problèmes:

- ✓ Que feront les possesseurs de deux lecteurs de disquettes, s'ils ont placé leur disquette Workbench dans le lecteur externe ?
- ✓ Comment ferai-je savoir au programme que les fontes se trouvent sur le disque dur ?
- ✓ Lorsqu'on procède ainsi, il n'est plus possible de donner un autre nom au répertoire "fonts" !
- ✓ On ne peut plus utiliser de disquette de fontes supplémentaire, puisque toutes les polices doivent se trouver sur la disquette Workbench !

Vous voyez donc bien que ce n'était pas la solution idéale. Quel est le chemin emprunté par le système d'exploitation de l'Amiga ? Il accède au répertoire de contenu souhaité par l'intermédiaire du nom que vous indiquez. On peut donc à volonté affecter un autre répertoire au même nom avec Assign, et l'accès fonctionnera quand même, car le programme adresse toujours le périphérique FONT:. De la sorte, il n'y a plus d'inconvénient à accumuler les fontes sur le disque dur ou sur le lecteur externe. Tous les emplacements sont adressables et autorisés.

En procédant ainsi, on permet à chaque programme d'exister dans l'environnement Amiga, qui peut néanmoins être défini de toutes les façons possibles. Pour le programmeur, il est alors intéressant de savoir ce qu'il trouvera dans les différents répertoires, et ce qu'il peut y déposer. Il existe là-dessus des prescriptions qu'il faut absolument respecter !

FONTS*Workbench 1.3 :fonts*

Dans le répertoire FONTS:, on trouve toutes les polices de caractères. Ce répertoire est surtout adressé par la Diskfont.Library. Le plus raisonnable est de l'utiliser à partir du CLI, ce qui permet de modifier la définition du répertoire, et donc d'utiliser des disquettes supplémentaires de fontes. Il est toujours possible de compléter ce répertoire, ou au contraire de supprimer des fontes existantes.

S*Workbench 1.3 :S*

Le répertoire S contient des fichiers séquence. Jusqu'à la version 1.2 du système d'exploitation, on n'y trouvait que la Startup-sequence. A partir de la version 1.3, on y trouve également une Startup-sequence pour le CLI et une autre pour le shell. Il existe en outre une version Démo de la Startup-sequence du disque dur. C'est un répertoire très utile car la commande Execute du CLI examine ce répertoire sans qu'on indique de chemin. De cette façon, on économise le temps qu'il faut pour taper ce chemin, et tous les fichiers batch sont dans un même répertoire. Il faut aussi savoir que le répertoire S: est examiné en priorité lors du boot.

L*Workbench 1.3 :L*

Dans ce device sont déposés les handlers pour toutes les bibliothèques non résidentes. Si vous créez donc une nouvelle bibliothèque, vous pouvez placer ici ce qu'on appelle les segments OverlayCode.

C*Workbench 1.3 :c*

C: abrite toutes les commandes CLI. Dès qu'on a tapé une commande sous le CLI ou le shell, le CLI va voir dans ce répertoire si la commande s'y trouve. On peut évidemment compléter cette liste par l'intermédiaire d'un Path.

DEVS*Workbench 1.3 :devs*

Tous les devices de l'Amiga ne sont pas implantés dans le système d'exploitation, car certains d'entre eux ne sont utilisés que très rarement. Après un OpenDevice(), le système d'exploitation va voir ici si le device ne se trouve pas déjà en mémoire. Si vous en avez envie, vous pouvez aussi écrire vos devices propres, et les déposer ici.

LIBS**Workbench 1.3 :libs**

Ce que nous avons dit pour les devices vaut aussi pour les bibliothèques. Dans ce répertoire se trouvent les bibliothèques qui n'ont pas été déposés en mémoire de manière résidente. Elles peuvent cependant être chargées à tout moment, et on peut également compléter cette liste soi-même.

SYS**Workbench 1.3 :sys**

On désigne par SYS: la disquette système à partir de laquelle on effectue le bootage. Elle peut donc être adressée sans indication du lecteur. Ce fait a une importance particulière si l'on est habitué à placer le Workbench dans le lecteur DF0 et si on l'a enlevé. Lorsqu'on adresse la disquette par SYS:, l'Amiga corrige lui-même l'accès.

Il existe encore le répertoire :T adressé par beaucoup de programmes, et dans lequel on place les "Temporary Files". Il contient les backups ou les fichiers de données utilisés provisoirement. A partir de la version 1.3, le device a changé de nom, et s'appelle maintenant T:, pour profiter de tous les avantages mentionnés. Il ne faut en aucun cas placer ce répertoire dans la RAM ; il est vrai que l'accès y est plus rapide, mais la fonction de backup perdrat alors toute sa signification. En effet, les backups seraient perdus après un reset.

Eléments d'utilisation de Intuition

Nous voici parvenus à la seconde section du présent chapitre. Nous allons quitter les opérations internes, propres à la mise en forme des programmes. Ici, le centre de gravité va se trouver du côté de la communication avec l'utilisateur. Nous allons donc parler de la transmission des messages. La question n'est plus le moyen de transmission (car seul Intuition offre un véritable choix, qui diffère de toutes les autres méthodes), mais la manière.

Menus

Nous avons des menus, par lesquels on peut sélectionner des fonctions dans le programme. Ce qui nous intéresse ici n'est pas la programmation, mais la distribution des fonctions dans les menus. En effet, le grand confort que représente l'utilisation des menus risque d'être réduit à rien par une mauvaise distribution des commandes. C'est pourquoi il faut observer, ici encore, certaines règles au moment de programmer.

Il est recommander de dresser tout d'abord une liste de toutes les fonctions que l'on veut recevoir dans le programme. On peut ensuite distribuer ces fonctions en 10 domaines au plus, qui représentent les menus principaux, et qui apparaîtront plus tard dans la barre des menus, lorsque l'utilisateur actionnera le bouton droit de la souris. Les titres des menus principaux doivent être choisis de manière à ce que l'on puisse les retrouver facilement, sans être obligé de tâtonner pour voir ce qui se cache derrière.

Cette distribution initiale doit être suivie d'un classement plus poussé. En effet, l'habitude veut que l'on place dans le coin gauche le titre du menu le plus important, et de repousser vers la droite les menus moins importants. La raison de cette convention se situe dans le fait que l'on a tendance, en tant qu'utilisateur, à se déplacer avec la souris presque toujours automatiquement vers le coin supérieur gauche pour ouvrir les menus. Cela fait que le menu recherché est dans la plupart des cas tout de suite sous le pointeur. Le menu placé tout à fait à gauche est donc le plus souvent le menu Fichier ou File, dans lequel se trouvent les opérations sur les disquettes.

Les menus principaux créés de cette façon contiennent toutes les fonctions proposées par le programme. Pour soutenir les applications les plus courantes du logiciel, il faut penser à définir une séquence "shortcut", c'est-à-dire un raccourci clavier. Notez bien que chaque lettre du clavier peut recevoir une double affectation, en différenciant entre majuscule et minuscule. Les caractères spéciaux sont également autorisés.

Mais attention, les fonctions placées à l'intérieur d'un menu doivent elles-mêmes être rangées en bon ordre ! Ici encore, les yeux sont habitués à chercher du haut en bas, et c'est dans cet ordre que le programmeur doit placer les options du menu. Dans le menu Fichier, cela veut dire qu'on mettra d'abord les fonctions de chargement et de sauvegarde, et ensuite seulement les fonctions de suppression de fichiers existants, ou d'autres du même genre. Si plusieurs options sont possibles pour un certain type de fonction (par exemple Sauvegarder), on peut créer une interrogation adressée à l'utilisateur par l'intermédiaire d'un requester, ou alors créer un sous-menu. Dans tous les cas, le style de programmation n'est pas de la meilleure veine lorsqu'on place dans le menu principal quatre options d'importance égale par exemple pour la sauvegarde.

Si le logiciel offre un choix de formats dans lesquels on peut sauvegarder (ASCII, IFF, format privé), il faut simplement créer dans l'option Sauvegarder les sous-options ASCII, IFF et PRIVE. De cette façon, on économise un requester, et l'on s'évite un travail de programmation inutile. Il est vrai que l'établissement des raccourcis devient alors plus difficile, car si l'on doit de plus programmer la même chose pour l'opération de chargement, les touches deviennent insuffisantes. C'est au programmeur de peser dans ce cas le pour et le contre.

Mais il ne faut en aucun cas renoncer aux requesters, si l'on veut attirer l'attention sur une situation importante. Il se peut par exemple que la suppression d'un texte occasionne des dommages irréparables, qu'il faut prévenir à l'aide d'un requester. C'est en effet la dernière chance pour l'utilisateur d'interrompre le processus. Si le requester n'intervient pas, il peut se faire que l'on reprenne le tout au début.

Requester

Le second élément essentiel de Intuition, après les menus, sont donc constitués par les requesters. Ceux-ci attendent parfois de l'utilisateur qu'il choisisse entre plusieurs paramètres, ou qu'il réponde à des questions importantes. Ici encore, comme pour les menus, il faut observer une distribution rigoureuse, pour que l'utilisateur ne s'y perde pas, et qu'il ne fasse pas un choix erroné à cause d'un malentendu. Si nous prenons par exemple un requester fonctionnant comme garde-fou pour une fonction importante

(supprimer toutes les données), il faut prévoir un choix entre deux réponses possibles au moins. Le texte doit être facilement compréhensible, et on doit le faire suivre de deux gadgets, "Oui" et "Non". Souvent, "OK" et "Annuler" sont plus satisfaisants.

Dans les cas où l'on introduit des données, il faut emprunter une autre voie. Ici encore, il y a évidemment un gadget pour le Oui (confirmation) et un autre pour le Non (annulation). Mais le programmeur doit également laisser une porte ouverte permettant de rétablir les définitions initiales ou encore d'annuler la dernière modification. De cette façon, vous laissez à l'utilisateur toutes ses chances pour qu'il puisse effectuer un test, ou encore ne pas pâtir de son incertitude.

Il faut insister sur la nécessité de pouvoir interrompre une fonction avant qu'elle n'ait agi. Il n'y a aucun sens à élaborer un requester qui ne permette pas d'annuler l'ensemble de l'opération, et qui n'autorise que la confirmation. Cette annulation doit annihiler éventuellement toutes les spécifications, et rétablir les valeurs initiales. Il faut y veiller en particulier lorsqu'il n'est pas prévu de ramener toutes les valeurs à leur état originel.

Une autre fonction qu'il est recommandé de fournir à l'utilisateur est la possibilité d'actionner les gadgets par l'intermédiaire du clavier, qui n'est pas soutenu comme pour les menus de Intuition. Il devra s'agir d'un gadget du requester que l'on signale de manière particulière. Si l'on prend par exemple BECKERTEXT, ces gadgets y sont signalés par une bordure rouge supplémentaire. Si donc l'utilisateur actionne la touche Return dans un requester actif, la routine d'évaluation interprète cette action comme une sélection du gadget mis en évidence de cette façon. C'est le plus souvent le gadget que l'on a besoin de sélectionner. Mais il faut bien veiller à ne pas affecter cette possibilité de sélection accélérée à des gadgets qui risqueraient de provoquer des dégâts. Un gadget à sélectionner par Retum lorsqu'il s'agit d'une fonction de suppression, ce n'est pas une très bonne idée !

Transmission des données

Le contact avec le logiciel commence avant même qu'il soit entré en fonction. En effet, le lancement débute par la transmission de paramètres, qui effectuent des réglages à des endroits clés. Si l'on indique par exemple pour un éditeur le nom du texte tout de suite après l'appel du programme, le texte en question sera chargé tout de suite après le lancement. On économise ainsi l'appel d'une fonction, avec laquelle on devra définir le texte par l'intermédiaire d'une fenêtre de sélection, ou d'un moyen analogue. La même chose est prévue par l'intermédiaire du Workbench.

Avec celui-ci, c'est encore plus simple ! Il suffit de cliquer sur l'icône du fichier - en supposant que le texte en possède une -, et le logiciel est chargé tout de suite avec le texte disponible à l'écran. Ces deux moyens sont à la disposition du programmeur, s'il veut recevoir de l'utilisateur des données nécessaires pour l'exécution du programme. Il faut distinguer ici deux choses. Il y a d'une part les programmes, comme les commandes CLI, qui ne peuvent en aucune manière fonctionner sans paramètres. Mais d'autre part, la transmission initiale des données est envisagée comme un complément.

En effet, le traitement de texte dont il a été question plus haut fonctionne parfaitement aussi sans l'indication préalable d'un fichier texte ; dans ce cas, il présente au départ une mémoire de texte vide.

Nous voyons par là que la transmission des données au lancement du logiciel présente deux aspects: un aspect de soutien, et un aspect primordial. Elle est importante dans les deux cas, et c'est pourquoi nous allons l'envisager ici de manière un peu plus approfondie. Nous allons diviser nos réflexions en deux parties. La première s'occupera des paramètres que l'on envoie au logiciel par l'intermédiaire du CLI. La programmation en C est ici très simple, car le compilateur fournit aussitôt un traitement de ces données. La seconde partie s'occupera de la transmission des données par le Workbench, plus complexe et plus prometteur sous maints aspects. Nous reviendrons plus tard sur ce dernier point.

Transmission des données par l'intermédiaire de paramètres CLI

La manière la plus simple et la plus connue de transmettre des paramètres passe par le CLI. On ajoute ici derrière les noms des programmes les valeurs qui doivent être transmises. De cette façon, le programme a la possibilité de les interpréter.

Quelles sont les indications reçues par le programme ?

La fonction main() du programme a deux arguments, en général peu utilisés, et pourtant fort utiles. Le premier de ces arguments est le nombre des paramètres transmis. Le second représente un pointeur sur un secteur chaîne, dans lequel tous les paramètres sont sauvegardés. Le travail que cela suppose est accompli par ce qu'on appelle une fonction startup. On voit ici plus précisément la partie qui distribue la ligne de commande dans un secteur à partir duquel on peut ultérieurement rassembler les entrées une à une.

```
*****
*
* Sous-routine: Exécute la ligne de commande *
*
* _____
*
* Auteur: Date: Commentaire: *
* ----- -----
* Wgb Juil 1988 Routine Aztecs *
*
*****
#include <Librarrys/dosextens.h>
extern int _argc, _arg_len;
extern char **_argv, *_arg_lin;
_cli_parse(pp, alen, aptr)
struct Process *pp;
long alen;
register char *aptr;
{
    register char *cp; /* Character Pointer */
    register struct CommandLineInterface *cli;
    register int c; /* Position du pointeur */
```

```
void *_AllocMem();
cli = (struct CommandLineInterface *) ((long)pp->pr_CLI << 2);
cp = (char *)((long)cli->cli_CommandName << 2);
_arg_len = cp[0]+alen+2; /* Longueur+ PrgName + Null-Bytes */
if ((_arg_lin = _AllocMem((long)_arg_len, 0L)) == 0)
    return;
strncpy(_arg_lin, cp+1, cp[0]);      /* Nom du programme*/
strcpy(_arg_lin+cp[0], " ");        /* Espacements*/
strncat(_arg_lin, aptr, (int)alen); /* Paramètres */
for (_argc=0,aptr=cp=_arg_lin;:_argc++)
{
    while (((c=*cp) == ' ' || c == '\t' || c == '\f' ||
            c == '\r' || c == '\n'))
        cp++;
    if (*cp < ' ')
        break;
    if (*cp == '\"')
    {
        cp++;
        while (c = *cp++)
        {
            *aptr++ = c;
            if (c == '\"')
            {
                if (*cp == '\"')
                    cp++;
                else
                {
                    aptr[-1] = 0;
                    break;
                }
            }
        }
    }
    else
    {
        while ((c-*cp++) && c != ' ' && c != '\t' && c != '\f' &&
               c != '\r' && c != '\n')
            *aptr++ = c;
        *aptr++ = 0;
    }
    if (c == 0)
        --cp;
}
*aptr = 0;
if ((_argv = _AllocMem((long)(_argc+1)*sizeof(*_argv), 0L)) == 0)
{
    _argc = 0;
    return;
}
for (c=0,cp=_arg_lin;c<_argc;c++)
{
    _argv[c] = cp;
    cp += strlen(cp) + 1;
}
_argv[c] = 0;
```

Description du programme

Une fois que le programme est allé s'enquérir de la longueur de la ligne de commande, il crée une mémoire pour le tableau des paramètres. Si ce tableau n'existe pas, le programme s'interrompt, ce qui signifie simplement que les paramètres ne sont pas transmis. La mémoire destinée aux paramètres reçoit ensuite le nom du programme, l'espacement, et les paramètres. C'est alors que vient la boucle suivante:

Elle parcourt toute la liste et recherche des caractères de séparation. Ceux-ci sont l'espacement, la tabulation, le saut de page (form feed), le retour chariot (carriage return), et le début d'une nouvelle ligne (new line). Lorsqu'un de ces caractères est rencontré, il s'ensuit plusieurs comparaisons, pour savoir quel est le caractère rencontré, ou s'il s'agit de guillemets, ce qui supprime évidemment l'effet des caractères de séparation jusqu'aux guillemets suivants.

A la fin de cet examen, la chaîne de caractères a été parcourue aussi longtemps qu'un caractère de séparation n'a pas été rencontré. La boucle place alors un octet nul derrière le texte, pour mettre fin à celui-ci. Cette opération se répète autant de fois qu'il le faut, jusqu'à ce que l'ensemble du texte ait été examiné. A la suite de quoi une nouvelle fonction de sauvegarde assure l'enregistrement de la table des pointeurs qu'on utilisera ultérieurement dans le programme. Dans ce secteur de mémoire, tous les pointeurs sur les différents textes sont reportés une fois de plus.

Pour finir, la dernière entrée dans le tableau des pointeurs est fixée à 0, pour caractériser la fin, même en l'absence de variable de comptage. Comme premier exemple d'application de cette routine, voici un programme qui affiche le nombre de paramètres transmis, et qui dresse la liste de toutes les valeurs dans un tableau.

```
*****
*
* Programme: Affichage des paramètres CLI
* -----
*
* Auteur: Date: Commentaire:
* ----- -----
* Wgb 20.06.1988 Liste seule
*
*****
main(ArgC, ArgV)
int ArgC;
char *ArgV[];
{
    int i;

    printf("Nombre des arguments: %d\n", ArgC);

    for (i=0; i<ArgC; i++)
        printf("CLIArg %d: >%s<\n", i, ArgV[i]);
}
```

Description du programme

La fonction main() transmet les deux paramètres ArgC et ArgV. Le premier représente le compteur des arguments, une variable de comptage qui contient le nombre de tous les éléments transmis. Ici, le nom du programme est considéré lui aussi comme un paramètre ! C'est une particularité que l'on peut expliquer facilement par le fait que l'on a affaire à la ligne de commande en son entier, reçu comme tableau de texte. En outre, on peut en tirer profit, en donnant le nom de chemin à la commande et en l'utilisant, ou encore pour constater si le programme a été lancé ou non à partir du Workbench, car on reçoit toujours le nom du programme comme paramètre. Nous allons y revenir dans un instant.

ArgV est le pointeur du tableau de texte (Argument Vector) dont il a été question plus haut. Il s'agit ici d'un tableau à une dimension, fait d'éléments char (strings). La routine qui élabore ce tableau à partir de la saisie sait qu'un nouveau paramètre commence grâce aux espacements entre deux textes, comme nous l'avons expliqué précédemment. Si l'on veut empêcher cette reconnaissance, par exemple parce que le texte doit contenir lui-même un espace (nom de fichier), il faut placer le texte entier contenant des espacements entre guillemets, ce qui ne présente aucun problème, et que l'on connaît déjà pour l'avoir vu dans le CLI.

Quel intérêt présente ce programme ? Il est conçu uniquement comme programme de démonstration, pour que vous puissiez discerner le comportement des routines de paramètre. Tapez-le et enregistrez-le pour cette raison sous le nom "Parametres". Il sera ensuite compilé et linké tout à fait normalement (Option -lm -lc). Nous allons faire maintenant quelques tests pour analyser son comportement. Entrez pour cela tout d'abord uniquement le nom du programme et confirmez:

Parametres

Le résultat est le suivant:

```
Nombre d'arguments: 1
Argument 0: Parametres
```

Ensuite, nous ajouterons quelques énoncés derrière le nom du programme:

Parametres 1. DFO: Hello Allo Woody

sur quoi nous recevons le résultat suivant:

```
Nombre d'arguments: 6
Argument 0: Parametres
Argument 1: 1.
Argument 2: DFO:
Argument 3: Hello
Argument 4: Allo
Argument 5: Woody
```

Faisons un troisième essai pour vérifier à nouveau ce qui a été dit sur les guillemets. Nous allons entrer un nom de fichier contenant quelques espacements, et nous allons le "recomposer". Est-ce que les guillemets seront transmis eux aussi ?

Parametres "DF1:1. Test"

et voici la réponse du programme:

```
Nombre d'arguments: 2
Argument 0: Parametres
Argument 1: DF1:1. Test
```

Une fois ces éléments rassemblés sur le fonctionnement du programme, envisageons la programmation d'une interrogation des paramètres. Notre programme devra offrir la possibilité d'obtenir aussitôt le format général lorsqu'on indique un point d'interrogation comme premier paramètre. C'est ainsi pour presque toutes les commandes CLI, et cela représente une aide pour l'utilisateur dans maintes situations. Notre programme doit donc tester si un point d'interrogation a été introduit, ou encore si le nombre de paramètres ne coïncide pas avec les paramètres utilisés. Dans ce dernier cas, on peut éventuellement obtenir un message d'erreur, et la communication du format général.

Le programme qui suit exécute cette tâche, et présente l'en-tête de la fonction main(), qui doit toujours être configuré de cette façon dans les programmes que vous aurez à écrire:

```
*****
* Programm: Examen des paramètres CLI *
* _____ *
*           *
* Auteur: Date:      Commentaire:   *
* ----- ----- ----- *
* Wgb     20.06.1988 Option "?" aussi*
*           *
*****
```

```
#include <exec/types.h>
main(ArgC, ArgV)
int ArgC;
UBYTE *ArgV[];
{
    int i;
    if (ArgC == 1)
        printf("Pas de paramètre venant du CLI!\n");
    else
        printf("%d Paramètres qui ont été interprétés\n", ArgC-1);
    if ((ArgC == 2) && (*ArgV[1] == '?')) && (ArgV [1+1]+1)==0)
        printf("Format: %s [...] [...]\n", ArgV[0]);
}
```

Description du programme

Le programme recherche d'abord le nombre de paramètres transmis. S'il n'y en a aucun, ce fait est conservé en mémoire. Sinon, il affiche le nombre de valeurs existantes. L'interrogation qui vient ensuite est importante. Elle indique si le nom du programme est suivi d'un seul point d'interrogation. Elle demande ensuite à l'utilisateur de fournir le format général, et elle l'affiche aussitôt après.

Ce format ne tient pas forcément en une ligne, c'est évident. Il est possible qu'il y ait des explications supplémentaires sur les différents paramètres, ou encore d'autres commentaires. Il faut simplement tenir compte d'une particularité: Notre affichage ne prend pas le nom connu du programme, mais sort le premier texte qui se trouve dans la table des paramètres, ce qui revient au même. La différence tient cependant dans le fait que l'on peut à tout moment pourvoir le programme d'un nouveau nom. L'indication du format suit la même règle. On peut également indiquer le chemin exact qui mène à la commande. Ce chemin est enregistré lui aussi dans le format. On est sûr de cette façon que l'utilisateur ne sera pas irrité en rencontrant des modifications.

Pour finir, une remarque annexe: cet exemple ne reflète évidemment pas toutes les possibilités qui peuvent se présenter, et pour lesquelles on peut avoir besoin de l'affichage du format. On peut par exemple limiter son programme à signaler une erreur éventuelle - trop peu de paramètres, domaine erroné, trop de paramètres. Mais il est souvent utile d'indiquer également le format général pour permettre une analyse de l'erreur. Jouez donc la sécurité, pour ne pas laisser l'utilisateur se débrouiller seul avec le programme. Le seul inconvénient de la fonction "?" tient dans le fait que le programme doit d'abord être chargé en son entier avant que l'on puisse recevoir des informations. Il peut alors arriver que le message ne soit pas affiché parce qu'il n'y a plus assez de place en mémoire pour charger tout le programme.

Travailler avec les entrées dans le fichier .info

La transmission des paramètres en CLI est facile à programmer, et est assez confortable. Cependant, il faut toujours laisser au programmeur et à l'utilisateur du logiciel la possibilité de pénétrer à l'intérieur du système. Le vrai chemin pour cela passe par le Workbench. Cette interface utilisateur a été conçue spécialement pour rendre son usage aussi simple que possible, et pour garantir un emploi aisément aux non-spécialistes.

Quelles sont donc les possibilités réservées à l'utilisateur pour la transmission des paramètres, et comment peut-on interpréter les paramètres ? Examinons de plus près le processus qui a lieu au moment de lancer le programme. Celui-ci reçoit un message startup, qui est interprété par la fonction main(). Cette fonction crée alors d'une part la table des paramètres à partir de la ligne de texte, et transmet d'autre part à l'utilisateur certaines valeurs importantes du Workbench. Ce sont ces valeurs que nous allons passer en revue.

La première d'entre elles est une liste des "files" et des "locks" dont nous disposons pour ce programme. Vous allez dire: d'où viennent ces files et ces locks ? Les files sont ceux sur lesquels on a cliqué avec le programme, et les locks sont les pointeurs dirigés sur les répertoires, puisque nous ne recevons jamais que le nom du fichier et lui seul. Avant d'analyser plus avant ce processus, voyons le programme suivant:

```
*****
*                                         *
* Programme: Liste WBMessag          *
* _____                                *
*                                         *
* Auteur: Date: Commentaire:         *
```

```
*
* ----- -----
* Wgb      23.06.1988 seulement Locks et*
*                      noms de fichiers   *
*
*****/
```

```
#include <exec/types.h>
#include <workbench/startup.h>
#include <stdio.h>
extern struct WBStartup *WBenchMsg;
main()
{
    int i;
    struct WBArg *Arg;

    for (i=0, Arg=WBenchMsg->sm_ArgList; i<WBenchMsg->sm_NumArgs;
         i++, Arg++)
    {
        printf("WBArg %d: Lock=0x%lx Name = %s\n",
               i, Arg->wa_Lock, Arg->wa_Name);
    }
    printf("PRESS <Return> TO EXIT\n");
    Delay(5*60L);
}
```

Description du programme

Le programme est réellement très bref ! Cela vient du fait qu'il n'a pas été lancé à partir du CLI. Sur la possibilité d'un lancement à partir du CLI ou du Workbench, vous trouverez un exemple plus loin. Nous obtenons le message WBenchMsg. Il s'agit du message dont nous avons parlé, contenant tous les noms de fichiers et les locks correspondants. Dans la boucle for(), ceux-ci sont disposés sous forme de tableau.

Une fois le programme compilé et linké, et une icône convenable (tool-icon) trouvée pour ce programme, nous allons pouvoir passer à notre premier essai. Pour cela, sélectionnez la fonction Workbench INFO, pour raccourcir la liste des ToolTypes. On peut l'avoir en une seule ligne, qui devrait se présenter ainsi:

WINDOW-CON : 0/0/640/80/TestWindow

La fenêtre ici définie est ouverte automatiquement par la routine startup mentionnée plus haut selon les valeurs indiquées. Nous avons besoin de cette fenêtre pour l'affichage de notre tableau, car il n'y a plus de fenêtre AmigaDOS, si nous démarrons à partir du Workbench ! La routine qui exécute cette tâche est très courte. La voici:

```
*****  
* Programm: Examen des ToolTypes pour Window*  
* Auteur: Date: Commentaire:  
* ----- -----  
* Wgb Juli 1988 Routine Aztecs  
*  
*****
```

```

#include <Librarys/dosextens.h>
#include <workbench/workbench.h>
#include <workbench/startup.h>
#include <workbench/icon.h>
void *IconBase = 0;
_wb_parse(pp, wbm)
register struct Process *pp;
struct WBStartup *wbm;
{
    register char *cp;
    register struct DiskObject *dop;
    register struct FileHandle *fhp;
    register long wind;
    void *_OpenLibrary();
    long _Open();
    if ((IconBase = _OpenLibrary("icon.library", OL)) == 0)
        return;
    if ((dop = GetDiskObject(wbm->sm_ArgList->wa_Name)) == 0)
        goto closeit;
    if (cp = FindToolType(dop->do_ToolTypes, "WINDOW"))
    {
        if (wind = _Open(cp, MODE_OLDFILE))
        {
            fhp = (struct FileHandle *) (wind << 2);
            pp->pr_ConsoleTask = (APTR) fhp->fh_Type;
            pp->pr_CIS = (BPTR) wind;
            pp->pr_COS = (BPTR) _Open("*.MODE_OLDFILE");
        }
    }
    FreeDiskObject(dop);
closeit:
    CloseLibrary(IconBase);
    IconBase = 0;
}

```

Description du programme

Cette routine essaie d'abord d'ouvrir la bibliothèque des icônes, Icon.Library, pour pouvoir constater si un ToolType WINDOW a été reporté dans le fichier programme. Elle va chercher ensuite la structure .info au moyen de GetDiskObject(), et l'examine avec FindToolType. S'il existe une entrée portant le nom WINDOW, la routine se met à ouvrir une fenêtre d'affichage avec la définition donnée derrière cette entrée. Si elle réussit à le faire, elle rapporte le canal d'entrée dans la structure Process, et elle ouvre la même fenêtre pour l'affichage. Elle conserve celle-ci en mémoire dans la structure.

Sa tâche est ainsi achevée, et le DiskObject est libéré, ainsi que la bibliothèque. Cliquons maintenant sur l'icône du programme, et lançons le programme. La fenêtre apparaît à l'endroit défini, et l'entrée prise dans le tableau est également affichée. Nous obtenons ainsi le lock pointé sur le répertoire dans lequel se trouve le programme, et le nom de ce dernier.

Cela ne représente pas quelque chose d'extraordinaire, direz-vous: après tout, on peut s'attendre à ce que le programme sache dans quel répertoire il se trouve et quel est son nom. Mais cliquons maintenant sur une autre icône, et cliquons ensuite deux fois sur

notre programme en conservant la touche Shift pressée. Nous obtenons une autre entrée de la liste. Il existe donc un moyen de transmettre au programme par exemple des noms de fichier à l'aide d'une activation multiple, ces noms de fichiers pouvant être ensuite utilisés pour autre chose. Essayons maintenant une autre voie. Prenez un texte du Notepad, et copiez-le sur la disquette de travail sur laquelle se trouve également Echo. Ecrivez alors dans DefaultTool votre propre programme

DF0:WBTest

au lieu de

SYS:Utilities/Notepad

ou placez à la place de "DF0:" le chemin d'accès de votre programme. Cliquez alors deux fois sur le texte Notepad. Vous voyez de nouveau apparaître la fenêtre avec le tableau. Mais cette fois, nous n'avons aucune valeur sous l'entrée Lock de notre programme, et celui-ci est indiqué avec son nom de chemin complet, tel que nous l'avions conservé dans le fichier .info du texte Notepad. Sous l'entrée du texte, nous retrouvons le lock et le nom correspondant. Il est ainsi possible de vérifier si un programme a été lancé directement ou par l'intermédiaire d'une icône Projet. Complétez pour cela l'interrogation suivante:

```
printf("WBArg %d: Lock=0x%lx Name = %s\n",
      i, Arg->wa_Lock, Arg->wa_Name);
if ((i == 0) && (Arg->wa_Lock == 0))
{
    printf("Lancé sans programme. Celui-ci a été chargé ensuite!\n");
}
```

Si vous avez envie de travailler encore un peu, créez un double du texte, et cliquez sur les deux textes avant de sélectionner le programme. On obtient une liste encore plus longue, avec encore plus de locks et de noms de fichiers. Que signifient tous ces noms de fichiers ? Chacun d'eux reconduit, au prix d'un petit détour, aux données spécifiées dans le fichier .info. On peut rassembler ces données, les examiner et les traiter à volonté. Il faut pour cela accéder au fichier .info. Dans la bibliothèque des icônes (Icon, library), il existe une fonction qui porte le nom de GetDiskObject(), et qui est prévue précisément pour remplir cette tâche. Définissons donc à l'aide du lock un nouveau répertoire actuel, et lisons ensuite le fichier .info:

```
*****
*
* Programme: Liste des ToolTypes
* -----
*
* Auteur: Date: Commentaire:
* ----- -
* Wgb 24.06.1988 accès uniquement
* fichier .info
*
*****
#include <exec/types.h>
#include <workbench/workbench.h>
#include <workbench/startup.h>
#include <workbench/icon.h>
```

```

#include <stdio.h>
extern struct WBStartup *WBenchMsg;
extern struct IconBase *IconBase;
void           *OpenLibrary();
main()
{
    int i;
    char **ToolArray, *Value;
    LONG OldDir;
    struct DiskObject *Lock;
    struct WBArg *Arg;
    if (!(IconBase = (struct IconBase *)
          OpenLibrary("icon.library", OL)))
    {
        printf("Library introuvable!\n");
        exit(FALSE);
    }
    for (i=0, Arg=WBenchMsg->sm_ArgList; i<WBenchMsg->sm_NumArgs;
         i++, Arg++)
    {
        printf("WBAArg %d: Lock=0x%lx Name = %s\n",
               i, Arg->wa_Lock, Arg->wa_Name);
        if ((i == 0) && (Arg->wa_Lock == 0))
        {
            printf("Lancé sans programme. Celui-ci a été chargé
ensuite!\n");
        }
        else
        {
            OldDir = CurrentDir(Arg->wa_Lock);
            Lock = GetDiskObject(Arg->wa_Name);
            if (Lock != NULL)
            {
                FreeDiskObject(Lock);
            }
            CurrentDir(OldDir);
        }
    }
    printf("\nWAIT A MOMENT!\n");
    CloseLibrary(IconBase);
    Delay(5*60L);
}
}

```

Dans un premier temps, cette routine ne fait rien d'autre que de libérer le DiskObject, et de rétablir ensuite le répertoire actuel. Ce n'est qu'au moment où l'on accède au champ de texte contenu dans ToolTypes que l'on voit quelles sont les valeurs obtenues. Il faut pour cela la routine d'affichage, que nous avons laissée de côté:

```

ToolArray = Lock->do_ToolTypes;
j = 0;
do
{
    printf("%d. Eintrag: %s\n", j, ToolArray[j]);
    j++;
}
while(ToolArray[j] != Null);

```

Pour faire des essais plus prégnants, prenez à nouveau les fichiers texte mentionnés plus haut, ou le fichier programme, et entrez plusieurs ToolTypes sous INFO dans le menu Workbench. Ces ToolTypes ont le format général suivant:

TYP = FLAGS

TYP désigne toujours un mot clé écrit en majuscules, qui peut être ensuite appelé par l'intermédiaire de certaines fonctions. On affecte à ce mot clé une valeur ou des flags. Nous avons par exemple fourni une valeur, lorsque nous avons affecté au mot clé WINDOW la valeur CON:0/0/640/80/TestWindow. Mais on peut aussi introduire des nombres ou des textes.

Les flags sont utilisés en général pour indiquer ou non l'existence de certains états. Le Notepad utilise ainsi quelques flags qui disent si le texte fait usage d'une seule fonte ou de plusieurs. Vous trouverez là-dessus d'autres informations dans la suite du chapitre. Si l'on veut poser plusieurs flags, on les sépare par le caractère OU ("").

Demandons-nous maintenant comment nous allons interpréter les données reçues, et travailler ensuite avec elles. C'était une tâche relativement simple que de séparer les paramètres du CLI, mais les possibilités du Workbench présentent un éventail bien plus large, et nous devons travailler plus prudemment. Examinons d'abord s'il y a des entrées dans le fichier .info du programme. Celles-ci doivent être considérées comme des valeurs par défaut, et être envisagées en premier lieu. Il faut donc examiner si une autre icône représentant un fichier de données a aussi été cliquée. On lit et on interprète aussi les entrées dans le fichier .info. Celles-ci ont évidemment la priorité sur celles du programme.

L'examen se déroule comme suit: on va voir d'abord le ToolTypes. Si l'on trouve par exemple FILETYPE, l'examen se poursuit par la recherche des types connus traités également par notre programme. Nous devrons donc nous demander s'il s'agit d'un fichier ASCII, qui pourra alors être chargé par la routine correspondante. Si ce n'est pas un fichier ASCII, le programme doit se demander s'il est capable de traiter cet autre format.

Le programme qui suit examine le FILETYPE, et affiche un message en conséquence. Vous devez donc préparer également un fichier .info pour ce programme, possédant une entrée WINDOW. A la place des textes, vous pouvez introduire dans votre propre programme des embranchements vers les sous-routines souhaitées, ou encore vous contentez d'affecter des valeurs à des flags, et sauter à ces routines lorsque l'examen de tous les paramètres sera terminée. Cette façon de faire est fortement recommandée lorsqu'on attend d'autres paramètres que le seul type de fichier.

```
*****
*                                         *
* Programme: Examen des ToolTypes      *
* _____                                *
*                                         *
* Auteur: Date: Commentaire:          *
* ----- ----- -----                   *
* Wgb   25.06.1988 teste FILETYPE     *
*                                         *
```

```
*
*****
#include <exec/types.h>
#include <workbench/workbench.h>
#include <workbench/startup.h>
#include <workbench/icon.h>
#include <stdio.h>
extern struct WBStartup *WBenchMsg;
extern struct IconBase *IconBase;
void           *OpenLibrary();
main()
{
    int i, Test;
    char **ToolArray, *Value;
    LONG OldDir;
    struct DiskObject *Lock;
    struct WBArg *Arg;
    if (!(IconBase = (struct IconBase * )
          OpenLibrary("icon.library", OL)))
    {
        printf("Library introuvable!\n");
        exit(FALSE);
    }
    for (i=0, Arg=WBenchMsg->sm_ArgList; i<WBenchMsg->sm_NumArgs;
         i++, Arg++)
    {
        printf("WBArg %d: Lock=0x%lx Name = %s\n",
               i, Arg->wa_Lock, Arg->wa_Name);
        if ((i == 0) && (Arg->wa_Lock == 0))
        {
            printf("Chargé sans programme. Celui-ci a été chargé
ensuite!\n");
        }
        else
        {
            OldDir = CurrentDir(Arg->wa_Lock);
            Lock = GetDiskObject(Arg->wa_Name);
            if (Lock != NULL)
            {
                ToolArray = Lock->do_ToolTypes;
                Value = FindToolType(ToolArray, (char *)"FILETYPE");
                if (Value)
                {
                    printf("ToolType FILETYPE existe avec %s!\n", Value);
                    Test = MatchToolValue(Value, (char *)"TOOLTEST");
                    printf("Résultat du test %d\n", Test);
                }
                else
                {
                    printf("ToolType FILETYPE n'existe pas!\n");
                }
                FreeDiskObject(Lock);
            }
            CurrentDir(OldDir);
        }
    }
    printf("\nWAIT A MOMENT!\n");
```

```
CloseLibrary(IconBase);
Delay(5*60L);
}
```

Ce programme ne répond pas à toutes les exigences formulées plus haut. Ce serait d'ailleurs exagéré, car il est destiné uniquement à servir d'exemple, et non pas à effectuer une tâche concrète. Mais si vous créez vous-même un programme, n'oubliez pas que toutes ces fonctions et ces schémas doivent être respectés, sans quoi le travail sur l'Amiga ne présenterait pas d'intérêt. Le dernier exemple de programme va servir à préciser le lien entre l'interrogation du CLI et la lecture du fichier .info. Il offre à l'utilisateur la possibilité de définir trois flags: a(dd), p(rint) et i(nsert). On peut le faire soit en faisant précéder les lettres par un tiret (-a, -p, -i), soit encore par l'intermédiaire du fichier .info. Les entrées ADD, PRINT et INSERT sont possibles sous le point FLAGS. Voici d'abord le listing final:

```
*****
* Programm: Spécification des valeurs CLI & WB
*****  
*  
* Auteur: Date: Commentaire:  
* -----  
* Wgb 02.07.1988 -a -p -i  
* FLAGS=ADD|PRINT|  
* INSERT  
*  
*****  
#include <exec/types.h>  
#include <workbench/workbench.h>  
#include <workbench/startup.h>  
#include <workbench/icon.h>  
#include <stdio.h>  
extern struct WBStartup *WBenchMsg;  
extern struct IconBase *IconBase;  
void *OpenLibrary();  
main(ArgC, ArgV)  
int ArgC;  
UBYTE *ArgV[];  
{  
    int i;  
    int TestA = 0, TestP = 0, TestI = 0;  
    char **ToolArray, *Value;  
    LONG OldDir;  
    struct DiskObject *Lock;  
    struct WBArg *Arg;  
*****  
*  
* Programme partiel: Examen CLI  
*  
*  
* Auteur: Date: Commentaire:  
* -----  
* Wgb 02.07.1988 -a -p -i  
*  
*****  
if (ArgC > 0)  
{
```

```
for (i=0; i<ArgC; i++)
{
    if (*ArgV[i] == (UBYTE)'-')
    {
        if ((*ArgV[i]+1) == 'a') TestA = TRUE;
        if ((*ArgV[i]+1) == 'p') TestP = TRUE;
        if ((*ArgV[i]+1) == 'i') TestI = TRUE;
    }
}
else
/*********************************************
*                                     *
* Programme partiel: Examen WB      *
* _____*
*                                     *
* Auteur: Date:      Commentaire: *
* ----- ----- ----- *
* Wgb     03.07.1988 ADD|PRINT| *
*           INSERT      *
*                                     *
*****************************************/
{
    if (!(IconBase = (struct IconBase *)
        OpenLibrary("icon.library", OL)))
    {
        printf("Library introuvable!\n");
        exit(FALSE);
    }
    for (i=0, Arg=WBenchMsg->sm_ArgList; i<WBenchMsg->sm_NumArgs;
        i++, Arg++)
    {
        printf("WBArg %d: Lock=0x%lx Name = %s\n",
               i, Arg->wa_Lock, Arg->wa_Name);
        if ((i == 0) && (Arg->wa_Lock == 0))
        {
            printf("Lancé sans programme. Celui-ci a été lancé
ensuite!\n");
        }
        else
        {
            OldDir = CurrentDir(Arg->wa_Lock);
            Lock = GetDiskObject(Arg->wa_Name);
            if (Lock != NULL)
            {
                ToolArray = Lock->do_ToolTypes;
                Value = FindToolType(ToolArray, "FLAGS");
                if (Value)
                {
                    TestA = MatchToolValue(Value, "ADD");
                    TestP = MatchToolValue(Value, "PRINT");
                    TestI = MatchToolValue(Value, "INSERT");
                }
                else
                {
                    printf("ToolType FLAGS n'existe pas");
                }
            }
            FreeDiskObject(Lock);
        }
    }
}
```

```

        CurrentDir(OldDir);
    }
}
printf("\nWAIT A MOMENT!\n");
CloseLibrary(IconBase);
/* Delay(5*60L); */
}
if (TestA) printf("Flag ADD posé!\n");
if (TestP) printf("Flag PRINT posé!\n");
if (TestI) printf("Flag INSERT posé!\n");
Delay(5*60L);
}

```

Description du programme

Le programme se divise en deux parties principales. La première section examine si des paramètres ont été transmis par le CLI. Si c'est le cas, il s'ensuit un test qui permet de savoir de quel type ils relèvent. Les flags sont ensuite posés en conséquence, et ils seront interrogés à la fin du programme pour mener à la tâche voulue. Il n'y a pas de test pour savoir si l'on a aussi indiqué des flags non soutenus. Il faudrait pour cela afficher un message d'erreur. Il manque aussi l'affichage du format général, décrit au début de cette section du chapitre. Vous n'aurez aucune peine à les intégrer. On peut non seulement définir des flags, mais aussi transmettre des valeurs par leur intermédiaire. Voici comment:

Programme -w=20

Mais cette interrogation ne pose à son tour aucun problème, si l'on utilise la fonction atoi(), qui transforme la valeur ASCII "20" en la valeur entière 20. La seconde partie du programme est exécutée si le programme n'a pas été lancé à partir du CLI, mais du Workbench. Dans ce cas, les ToolTypes sont examinés, et les flags posés en conséquence. Cette interrogation n'est pas complète non plus. Elle examine seulement tous les ToolTypes, en se demandant s'il correspondent à nos flags. On peut compléter par d'autres comparaisons. Le programme se termine de cette façon. Pour conclure ce chapitre, voici encore quelques tableaux et règles concernant les fichiers .info:

① Suite des messages Workbench

1. toujours l'info du programme
2. d'abord le fichier de données sur lequel on a cliqué
3. le second fichier de données sur lequel on a cliqué

Il n'existe pas d'info sur le programme, si l'on n'a pas cliqué dessus ! On peut le voir au fait que Lock = 0.

② La suite des ToolTypes

1. Les ToolTypes sont fournis comme blocs dans l'ordre indiqué ci-dessus.
2. Les différents blocs sont transmis dans le même ordre que la disposition des entrées dans INFO.
3. C'est toujours le DefaultTool du premier fichier sur lequel on a cliqué qui est chargé ; les autres ne sont pas pris en compte !

③ Entrées ToolTypes dans le Notepad

Nom	Exemple	Signification
FILETYPE	notepad	indique un texte du Notepad
FONT	topaz.8	indique la fonte globale
WINDOW	0,0,50,50	indique les coordonnées de la fenêtre
FLAGS	NOGLOBAL	contient les flags posés

④ Les flags du Notepad

NOGLOBAL	désactive la fonction Global-Font
GLOBAL	active la fonction Global-Font
NOWRAP	désactive le wordwrapping
WRAP	active le wordwrapping
NOFONTS	pas de tableau de fontes chargé
FORMFEED	active un formfeed
DRAFT	active l'impression normale

1. Le Hardware de l'Amiga

1.1. Introduction

L'AMIGA de COMMODORE propose des possibilités que personne n'aurait seulement imaginées, il y a quelques années, pour un ordinateur familial.

Pour rendre ce résultat possible, on trouve réunis sur l'AMIGA, d'une part un système d'exploitation performant et d'autre part une architecture évoluée.

L'un des buts des concepteurs de cet ordinateur était d'aboutir à une grande facilité d'utilisation. On a donc intégré dans le système d'exploitation une routine pour pratiquement chacune des tâches concevables, ce qui rend apparemment inutile la programmation du hardware.

Cette simplicité n'est pourtant qu'apparente. Malgré toutes ces confortables routines, on ne peut pas se passer de la programmation directe, car la vitesse des routines système est bien plus faible que ce à quoi on aurait pu s'attendre. Il faut en chercher l'explication dans le fait que le système d'exploitation est écrit en grande partie avec le langage de programmation C. Si l'on veut donc écrire des programmes rapides et performants, ou tout simplement mieux connaître le fonctionnement interne de l'Amiga, il faut acquérir des notions sur le hardware. Le chapitre présent se propose en conséquence de fournir une description de la structure interne et de la programmation des différents chips.

1.2. Les composantes de l'Amiga

Le hardware des différentes versions de l'AMIGA, A500, 1000, A2000, B2000 reste essentiellement identique :

- ✓ microprocesseur 68000 de Motorola,
- ✓ deux adaptateurs d'interface de commande, type 8520,
- ✓ trois circuits spécialisés, AGNUS, DENISE, PAULA.

En écartant la RAM (mémoire vive) et toute la construction logique, ce sont les six circuits décrits plus haut qui assurent l'essentiel du travail de l'AMIGA.

Une des particularités de cet ordinateur est aussi la présence de nombreux connecteurs :

- ✓ port parallèle (centronics),

- ✓ connecteur série RS-232,
- ✓ connecteur vidéo RGB,
- ✓ connecteur modulateur TV,
- ✓ sortie audio stéréo,
- ✓ sortie vidéo composite (absente sur la version A2000 mais disponible sur la version B2000),
- ✓ connecteur clavier,
- ✓ connecteur disquette externe,
- ✓ 2 connecteurs identiques pour joystick, souris ou manette,
- ✓ connecteur d'extension mémoire de 256 Koctets (il ne sera fait aucune description de ce connecteur, étant donné qu'il ne concerne que l'AMIGA 1000, qui n'était doté que de 256 Koctets de RAM au départ ; l'adjonction d'une extension mémoire lui permet de passer à un total de 512 Koctets de RAM),
- ✓ connecteur d'extension système (ce connecteur se trouve sur les versions 500 et 1000, celui du 2000 se divisant en plusieurs connecteurs d'extension).

Afin de mieux comprendre le travail d'ensemble de tous ces éléments, nous devons tout d'abord examiner chaque composante à part.

1.2.1. Le processeur 68000

Le Motorola 68000 est indiscutablement l'un des plus remarquables processeurs 16 bits. Malgré sa présence sur le marché depuis 1979, on ne le trouve que depuis peu sur des ordinateurs de la classe de l'AMIGA.

Vous ne trouverez pas dans ce volume une description détaillée sur le 68000. Ceux qui veulent en savoir plus sur la programmation peuvent se référer à la littérature spécialisée que l'on peut trouver sur ce sujet. A la place, nous allons examiner exclusivement le brochage du circuit ainsi que les groupes de signaux isolés. Une connaissance fondamentale des signaux du 68000 est essentielle à la compréhension du Hardware de l'AMIGA.

Brochage du 68000

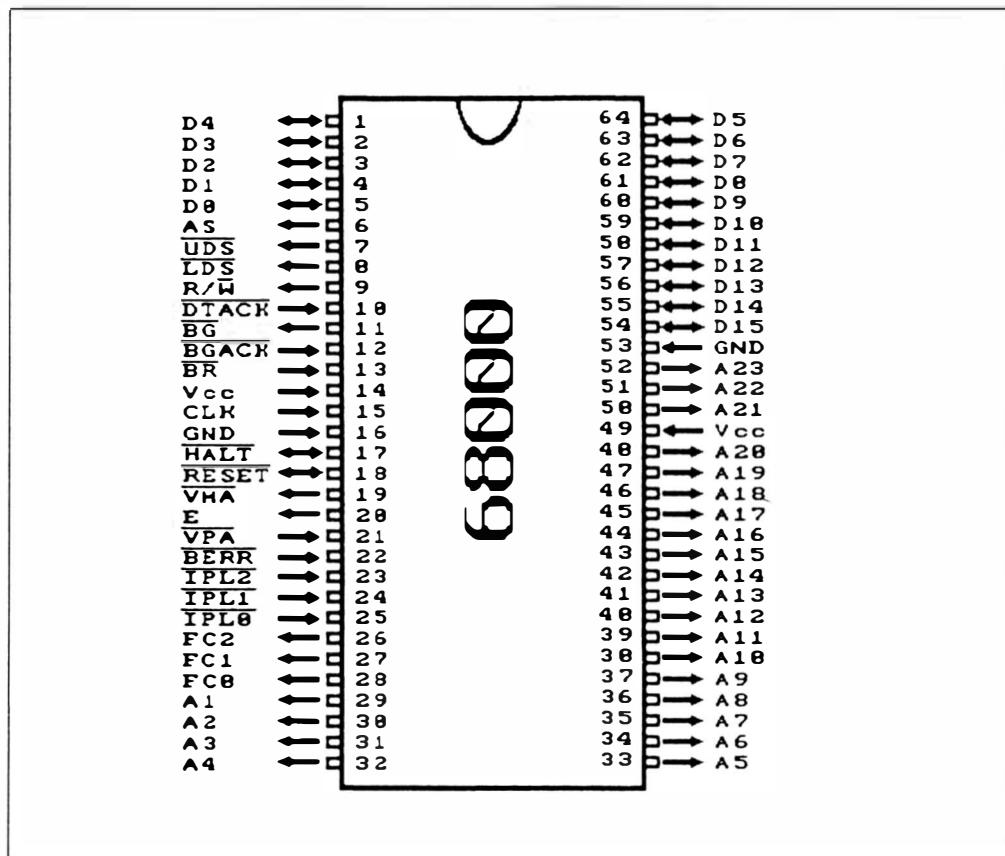


Figure 1 - 1

Remarques :

- ✓ les flèches indiquent le sens du signal,
- ✓ le trait au-dessus du nom du signal indique le double état que peut prendre ce signal (0 correspondant à actif).

On peut distinguer ces broches suivant plusieurs groupes :

► **L'alimentation VCC et GND**

Le processeur Motorola 68000 demande une alimentation de 5 volts. Les deux broches sont dédoublées et placées au centre du circuit, afin que les chutes de tension soient minimales.

► **Broche d'horloge : CLK**

La fréquence de l'horloge du processeur 68000 dépend de la version du processeur. Sur l'AMIGA, la fréquence est de 7.16 MHZ (millions de cycles par secondes).

► **Bus de données D0-D15**

Le bus de données est de type 16 bits et peut ainsi véhiculer un mot (16 bits) à la fois. Le processeur permet aussi le transfert d'octets (8 bits), soit supérieurs (D8-D15), soit inférieurs (D0-D7).

► **Bus d'adresse A1-A23**

Le bus d'adresses peut adresser avec ses 23 lignes unidirectionnelles huit mega mots ou 16 mega octets.

► **Bus de contrôle asynchrone : AS, R/W, UDS, LDS, DTACK**

Une des particularités du 68000 est son fonctionnement asynchrone. En mode asynchrone, le processeur signale avec AS (ADDRESS STROBE/ adresse valide) qu'une adresse valide se trouve dans le bus d'adresses. Simultanément, on détermine à l'aide de R/W (READ-WRITE/lecture-écriture) le sens du transfert (1 pour la lecture et 0 pour l'écriture). Le 68000 adresse sa mémoire avec des mots de 16 bits. Les deux signaux de contrôle UDS (UPPER DATA STROBE) et LDS (LOWER DATA STROBE) permettent de faire la distinction entre les deux octets d'un même mot lors du transfert d'un octet (8 bits).

- ✓ Si UDS est à 1 et LDS à 0 : il s'agit de l'octet de poids faible (inférieur),
- ✓ Si UDS est à 0 et LDS à 1 : il s'agit de l'octet de poids fort (supérieur),
- ✓ Si UDS et LDS sont à 0 : il s'agit alors d'un mot entier.

Le signal DTACK (Data Transfer Acknowledge/Reconnaissance de transfert de données) indique que les données sont prêtes à être transférées lorsqu'il est positionné à 0.

En mode asynchrone, le processeur s'aligne donc toujours sur la vitesse de la mémoire. Les mots et octets isolés s'organiseront de la manière suivante en mémoire :

Organisation des bus de données			
Adresse		D8-D15	D0-D7
0	mot 0	octet 0	octet 1
2	mot 1	octet 2	octet 3
4	mot 2	octet 4	octet 5
6	mot 3	octet 6	octet 7

► *Bus de contrôle en mode synchrone : E, VPA, VMA*

Au moment où le Motorola 68000 a été commercialisé, il n'y avait pas encore de circuits périphériques disponibles. Les circuits existant alors avaient été conçus pour la série précédente (série 6800, d'où descendait d'ailleurs le 6502) et n'avaient pas la possibilité de s'interfacer avec le bus de contrôle asynchrone du 68000. Ainsi, ce dernier s'est vu attribuer des bus synchrones, afin de faciliter les échanges entre ce microprocesseur et ceux de la famille 6800.

Sur la broche E, on applique un signal d'horloge correspondant au dixième du signal CLK (Clock) (7,16 MHZ) qui peut servir à tous les circuits périphériques (il sera relié à l'entrée Phi-2 d'un circuit périphérique).

Le passage du mode synchrone en asynchrone se produit sur l'entrée VPA (VALID PERIPHAD ADDRESS/adresse périphérique valide).

Cette entrée doit être mise à 0 par un décodeur externe d'adresse ; dès que ce dernier reconnaît une adresse de circuit périphérique, le processeur répond aussitôt en mettant également à 0 le signal VMA (VALID MEMORY ADDRESS/adresse mémoire valide).

Le circuit périphérique concerné prend alors en charge les données (et par conséquent les met à disposition) pendant un cycle d'horloge de E. Après cela, le processeur 68000 abandonne le mode synchrone automatiquement jusqu'à ce que le signal VPA soit à nouveau actif.

Ceci signifie donc qu'un circuit périphérique est contraint de prendre en charge ou de mettre à disposition, des données en mode synchrone.

► *Commandes système : RESET, HALT, BERR*

Pour initialiser le système, il faut mettre à 0 le signal HALT et le signal RESET. Dès que ces signaux sont remis à 1, le 68000 commence à exécuter le programme qui a été mis au préalable dans l'adresse mémoire 4. Le signal RESET peut aussi être remis à 0 à partir du 68000, afin d'initialiser le système, sans modifier l'état du processeur.

Avec le signal BERR (BUS ERROR), un contrôle externe peut indiquer au processeur qu'une information erronée circule sur un bus, comme par exemple : "Hardware défectueux" ou "Accès à une adresse inexistante".

Si un signal BERR se déclenche, le processeur 68000 renvoie à une routine spéciale du système d'exploitation qui prend en charge le traitement de l'erreur (salutations et méditations du Gourou !). Si pendant ce traitement de l'erreur, un nouveau signal BERR se déclenche, le 68000 stoppe toute l'exécution en cours et met le signal HALT à 0. Cette double erreur est le seul cas où le processeur, pardonnez l'expression, se plante. Pour d'autres types d'erreurs, le processeur accède à des vecteurs spéciaux dans les routines de programmation, qui peuvent effectuer le traitement de l'erreur et permettre ainsi au système la poursuite du travail (on remarque l'abondance des Guru-méditations qui se comportent chez l'AMIGA suivant la loi de Murphy : un ordinateur se plante toujours lorsque l'on travaille sur des données importantes, de préférence lorsqu'elles ne sont pas encore sauvegardées).

Si le processeur arrête l'exécution d'un programme en raison d'une double erreur du bus, il ne pourra redémarrer qu'au moyen d'un RESET (HALT et RESET à 0).

Une fonction du signal HALT est aussi l'arrêt des processeurs. En mettant HALT à 0, le 68000 achève ses accès mémoire en cours, puis attend alors jusqu'à ce que le signal HALT soit remis à 1.

► *Etat de fonctionnement du processeur : FC0, FC1, FC2*

Le signaux FC0-FC2 traduisent l'état de marche du processeur.

Voici les différents états possibles :

FC2	FC1	FC0	ETAT
0	0	1	Accès aux données utilisateur
0	1	0	Accès au programme utilisateur
1	0	1	Accès aux données superviseur
1	1	0	Accès au programme superviseur
1	1	1	Reconnaissance d'interruption

Le processeur peut avoir deux modes différents de travail : le mode utilisateur et le mode superviseur. Un programme qui tourne en mode superviseur accède à la totalité du registre d'état (SR = Status Register) du processeur. Le système d'exploitation travaille toujours en mode superviseur.

En mode utilisateur, la partie superviseur de SR est verrouillée. Pour plus d'informations, reportez-vous à la littérature sur le 68000.

Les trois signaux FCx permettent ainsi au système de connaître le mode de fonctionnement actuel du processeur, et selon le cas, de réagir sur ce mode. En mode utilisateur, par exemple, lors de l'accès à une zone mémoire réservée au système d'exploitation, une erreur (BERR=0) peut être engendrée.

► *Broches d'interruption : IPL0, IPL1, IPL2*

Les signaux des trois broches d'interruption (IPL = INTERRUPT PRIORITY LEVEL) permettent au processeur 68000 de différencier 8 signaux d'interruption (2^3) parmi lesquels 0 correspond à une absence d'interruption et 7 au plus haut degré d'interruption. Tous les niveaux se voient affecter un vecteur d'interruption, renfermant une adresse de routine d'interruption.

Si une interruption, conforme aux niveaux autorisés se déclare, le processeur met tous ses signaux FCx à 1, signalisant ainsi qu'il l'a reconnue et qu'il attend une confirmation de la part de l'auteur de cette interruption. La réponse pourra intervenir par le biais des signaux UPA et DTACK. Pour une confirmation par un signal UPA, il résulte une auto-interruption, le processeur sautant à une adresse se trouvant dans le vecteur correspondant à l'interruption. On peut donc accéder à 7 niveaux d'interruption ; le niveau 0 correspondant à l'état "pas d'interruption".

A chaque niveau d'interruption correspond une source d'interruption et le processeur saute alors au programme conforme.

L'AMIGA utilise seulement ces vecteurs "auto-programmés" en cas d'interruption.

► *Signaux d'attribution du bus : BR, BG, BGACK*

Ces trois signaux autorisent l'allocation du bus par un autre processeur. Ceci peut être le cas du contrôleur d'un disque dur, par exemple, qui écrit les données du disque directement dans la mémoire (DMA, DIRECT MEMORY ACCESS/mémoire accès direct).

1.2.2. Le CIA 8520

Brochage du 8520

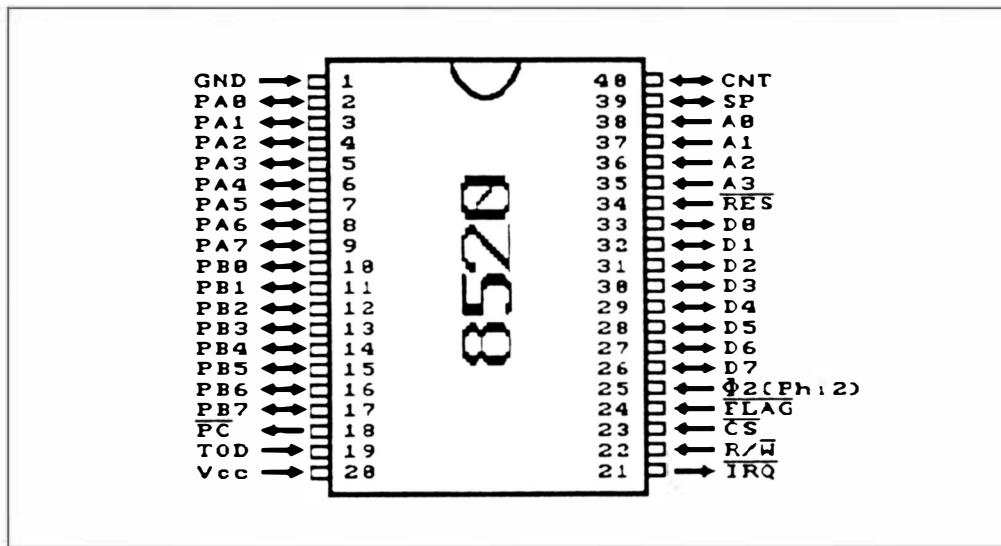


Figure 1 - 2

Remarques :

- ✓ les flèches indiquent le sens des signaux,
- ✓ le trait au-dessus des noms des signaux, traduit le double état que peut prendre le signal (0-actif).

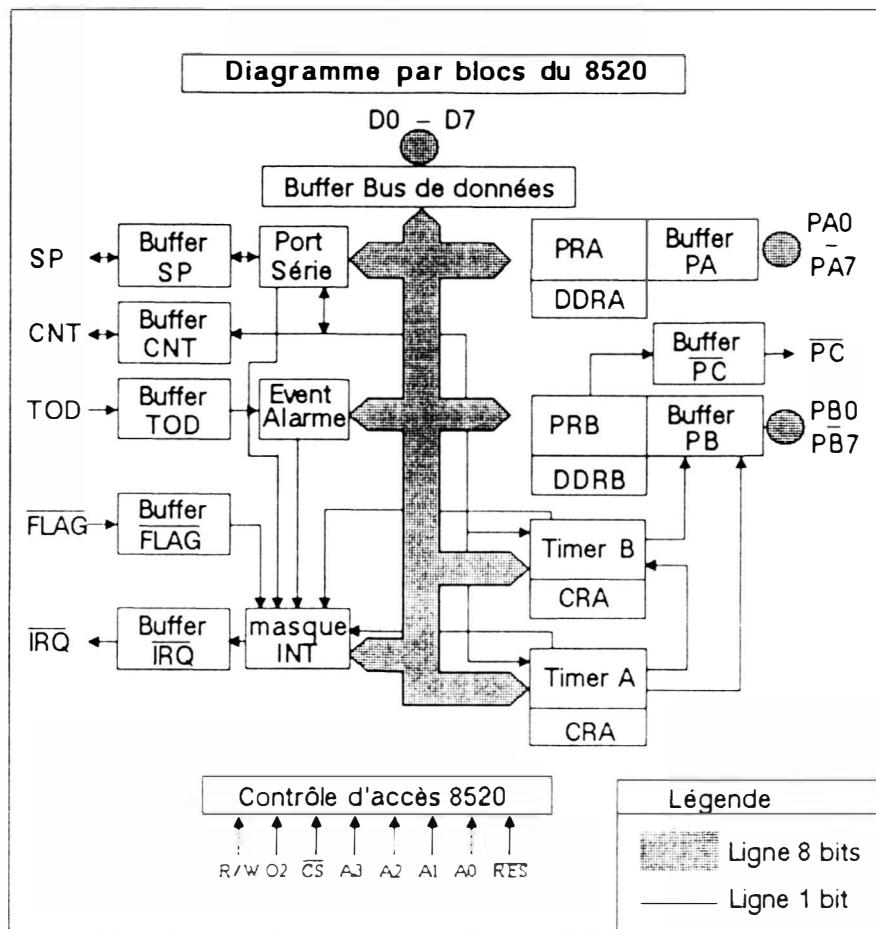


Figure 1 - 3

Le 8520 est un circuit périphérique du type Complex Interface Adapters (CIA), correspondant à un circuit à connecteurs multiples.

Ce sont en fait les concepteurs du 8520 qui ont cherché à introduire le maximum de fonctions au sein d'un même circuit. En examinant bien le 8520, on remarque la similitude entre ce circuit et un autre bien connu, le 6526 du C64.

Seuls les registres 8 à 11 (\$8 à \$B) sont différents, cela rassurera sûrement les programmeurs qui connaissent le 6526.

Le 8520 dispose des possibilités suivantes : deux ports parallèles 8 bits programmables (PA et PB), 2 minuteries 16 bits (Timer A et Timer B), un port série bidirectionnel (SDR)

et un compteur 24 bits (Event counter) avec une fonction alarme lorsqu'il atteint une valeur fixée.

Certaines fonctions sont en mesure de libérer des interruptions.

Les fonctions du 8520 sont réparties en 16 registres. Pour le processeur, ils apparaissent comme des registres mémoires normaux, les circuits périphériques d'un système 68000 étant considérés comme faisant partie intégrante de la mémoire, ceux-ci supportant toutes les opérations de lecture et d'écriture du 68000.

Le 8520 a été développé pour le processeur 8 bits de la série 65xx, et ne peut donc communiquer avec le 68000 qu'en mode synchrone.

L'horloge E du 68000 est reliée à l'entrée Phi 2 du 8520. Les 16 registres internes sont accessibles au moyen de quatre entrées A0-A3 du 8520. Des détails plus précis sur les liens entre le système de l'AMIGA et le CIA se trouvent à la fin de ce chapitre.

Voici le détail des 16 registres du circuit (le registre 11 (\$B) étant inutilisé).

► Détail des registres du 8520

Registre	Nom	Description
0 0	PRA	Réglage données port A
1 1	PRB	Réglage données port B
2 2	DDRA	Réglage direction des données port A
3 3	DDRB	Réglage direction des données port B
4 4	TALO	Minuterie A (octet bas)
5 5	TAHI	Minuterie A (octet haut)
6 6	TBLO	Minuterie B (octet bas)
7 7	TBHI	Minuterie B (octet haut)
8 8	EVENT LO	Compteur (valeur événement octet bas) bits 0-7
9 9	E. 8-15	Compteur bits 8-15 (octet moyen)
10 A	EVENT HI	Compteur bits 16-23 (octet haut)
11 B	--	inutilisé
12 C	SP	Données série
13 D	ICR	Contrôle Interruption
14 E	CRA	Réglage contrôle port A
15 F	CRB	Réglage contrôle port B

► *Les ports parallèles*

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
0	PRA	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
1	PRB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
2	DDRA	DPA7	DPA6	DPA5	DPA4	DPA3	DPA2	DPA1	DPA0
3	DDRB	DPB7	DPB6	DPB5	DPB4	DPB3	DPB2	DPB1	DPB0

Le 8520 dispose de 2 ports 8 bits parallèles PA et PB, chacun relié à un registre de données PRA et PRB. En toute conformité, le circuit dispose de 16 signaux de port, PA0-PA7 et PB0-PB7.

Chaque port peut être utilisé soit en tant que signal d'entrée, soit en tant que signal de sortie. Ceci sera indiqué par le sens des données. Le 8520 autorise la modification du sens des données de chaque signal. Ainsi à chaque port correspond un registre direction de données, DDRA et DDRB (DATA DIRECTION REGISTER).

Chacun de ces registres contient 8 bits. Si l'un des bits des registres DDR est placé à 0, le bit correspondant du port (PR) est une entrée. Si ce bit est mis à 1, le bit correspondant dans PR est une sortie.

La lecture des bus de données (registre PR) donne l'état des broches d'E/S (entrée/sortie), quel que soit le sens du transfert.

Les transferts de données via le port parallèle peuvent être contrôlés au moyen des signaux PC et FLAG. Ces signaux indiquent un début de transfert de données (Handshaking). La broche PC se positionne à l'état bas au troisième cycle après tout accès au port B. Elle indique donc que des données ont été émises de ce port. La broche FLAG réagit quand le signal à son entrée passe de 1 à 0. Dans ce cas, le bit FLAG du registre de contrôle des interruptions sera positionné.

On peut réaliser un contrôle efficace en connectant la broche PC d'un 8520 sur la broche FLAG du deuxième.

Le circuit transmetteur écrit ses données sur le port du registre et doit attendre un signal FLAG pour pouvoir écrire l'octet suivant. Puisque ce signal peut libérer une interruption, le transmetteur a la possibilité, pendant une attente, de se consacrer à une autre occupation.

► *Les minuteries (Timer en anglais)*

Accès lecture

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
4	TALO	TAL7	TAL6	TAL5	TAL4	TAL3	TAL2	TAL1	TAL0
5	TAHI	TAH7	TAH6	TAH5	TAH4	TAH3	TAH2	TAH1	TAH0
6	TBLO	TBL7	TBL6	TBL5	TBL4	TBL3	TBL2	TBL1	TBL0
7	TBHI	TBH7	TBH6	TBH5	TBH4	TBH3	TBH2	TBH1	TBH0

Accès écriture

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
4	PALO	PAL7	PAL6	PAL5	PAL4	PAL3	PAL2	PAL1	PAL0
5	PAHI	PAH7	PAH6	PAH5	PAH4	PAH3	PAH2	PAH1	PAH0
6	PBLO	PBL7	PBL6	PBL5	PBL4	PBL3	PBL2	PBL1	PBL0
7	PBHI	PBH7	PBH6	PBH5	PBH4	PBH3	PBH2	PBH1	PBH0

Le 8520 dispose de deux minuteries 16 bits. Elles sont en mesure de décompter à partir d'une valeur fixée au préalable jusqu'à 0. De plus, il existe un grand nombre de modes possibles qui sont sélectionnés avec un registre de contrôle (CONTROL REGISTER) spécifique à chaque minuterie (CRA et CRB).

Chaque minuterie est composée de 2 registres 16 bits (un pour la lecture et le deuxième pour l'écriture). Le registre d'écriture et celui de lecture ont la même adresse. Pratiquement, cela signifie que l'on ne peut pas savoir à quelle valeur la minuterie a été initialisée.

Lorsqu'un des registres de la minuterie est accédé en écriture, sa valeur est verrouillée en mémoire, chargée dans le registre du compteur puis décrémentée jusqu'à ce que le compteur atteigne une valeur négative.

La valeur initiale sera alors de nouveau chargée dans le registre du compteur.

Pour obtenir une valeur correcte de l'état de la minuterie, il est nécessaire d'arrêter le compteur. Voici un exemple de ce qui peut se produire (cas typique d'un effet de bord) :

Etat du compteur à un moment donné : \$0100.

Un accès lecture sur le registre nous donne d'abord l'octet haut de l'état actuel, soit \$01.

Avant que l'octet bas ne puisse être lu, une impulsion peut être transmise au compteur, ce dernier étant alors décrémenté d'un pas. La valeur du compteur sera donc \$00FF.

La lecture de l'octet bas donnera : \$FF

On obtiendra donc pour le compteur : \$01FF !

Au lieu d'arrêter le compteur, on peut employer la méthode suivante, beaucoup plus élégante : on lit l'octet haut, puis l'octet bas et à nouveau l'octet haut. Si la valeur de l'octet haut n'a pas changé, c'est que la valeur est correcte. Sinon, le processus doit être recommencé.

Le signal déterminant la décrémentation du compteur correspond, d'un part au bit 5 pour la minuterie A, et d'autre part aux bits 5 et 6 pour la minuterie B des registres de contrôle correspondants.

Il n'y a que deux sources possibles de signaux pour la minuterie A :

- ① La minuterie A sera décrémentée à chaque cycle d'horloge (le CIA de l'AMIGA étant relié au signal horloge E du processeur, sa fréquence est donc de 716 KHz) (INMODE=0).
- ② Chaque impulsion sur le signal CNT décrémente le compteur (INMODE=1).

Il existe pour la minuterie B, quatre modes d'entrée (reproduction binaire, le premier chiffre étant pour le bit 6, le deuxième pour le bit 5) :

- ① Cycle d'horloge (INMODE = 00)
- ② Impulsion CNT (INMODE = 01)
- ③ Combinaison avec la minuterie A (les deux minuteries correspondant à une seule minuterie 32 bits) (INMODE =10).
- ④ Combinaison avec la minuterie A lorsque le signal CNT est "haut" (on peut de toute façon mesurer la longueur d'une impulsion sur le signal CNT) (INMODE = 11).

Le passage à zéro d'un compteur sera indiqué dans le registre de contrôle d'interruption (ICR). Pour la minuterie A, ce sera le bit TA (bit 0) et pour la minuterie B le bit TB (bit 1). Ces bits restent actifs jusqu'à la lecture du registre ICR.

Toutefois, il reste la possibilité de diriger les minuteries vers le port B. Il faut pour cela activer le bit PB du registre de contrôle d'un des compteurs (CRA ou CRB). La minuterie A fonctionne avec le port PB6 et la minuterie B avec PB7.

Avec le bit outmode, on peut choisir entre deux types de sorties :

outmode = 0 Pulse-mode

Chaque décrémentation sera émise comme impulsion positive d'une durée d'un cycle d'horloge sur le port correspondant.

outmode = 1 Toggle-mode

A chaque décrémentation, le signal du port correspondant changera d'une valeur basse en valeur haute et vice-versa. A chaque départ de la minuterie, la sortie correspondra à une valeur haute.

La minuterie sera démarrée ou stoppée à partir du bit START du registre de contrôle (0 = ARRET, 1 = DEMARRAGE).

Avec le bit RUNMODE, on peut choisir entre le one-shot-mode et le continuous mode. Le premier mode arrête la minuterie après chaque décompte et remet le bit START à 0. Avec le continuous-mode, le compteur recommence à la valeur de départ.

Comme cela a été précisé, l'écriture d'une valeur dans un registre minuterie ne sera pas directement exécutée, mais auparavant sauvegardée et verrouillée en mémoire (aussi appelée Prescaler ; le nombre de décrémentations par seconde est égal à la fréquence du compteur divisée par la valeur se trouvant dans le Prescaler).

Il existe plusieurs possibilités de transfert des valeurs verrouillées en mémoire vers le compteur :

- ① Activer le bit LOAD du registre de contrôle. Ceci entraîne automatiquement un chargement (Force- Load). Indépendamment de l'état du compteur, la valeur verrouillée en mémoire y sera placée. Le bit LOAD est un bit STROBE, ce qui signifie que le bit ne sera pas mis en mémoire, mais permettra l'exécution d'un processus. Pour déclencher à nouveau un transfert prioritaire (Force Load), on doit remettre à 1 le bit LOAD.
- ② A chaque fin de décrémentation des minuteries, la valeur verrouillée en mémoire est transférée dans le compteur.
- ③ Après un accès en écriture dans le registre minuterie octet haut, alors que le compteur est arrêté (STOP = 0), ce dernier sera chargé automatiquement avec la valeur verrouillée en mémoire. C'est pour cette raison que l'on doit retenir la suite dans l'ordre :
 - premièrement : l'octet haut (octet de poids fort)
 - deuxièmement : l'octet bas (octet de poids faible).

Détail des bits du registre de contrôle A

Registre N 14 / \$E Nom : CRA

D7	D6	D5	D4	D3	D2	D1	D0
TOD IN	SPMODE	INMODE	LOAD	RUNMODE	OUTMODE	PBON	START
0-60Hz	0-entrée	0-Hor1.	1-FORCE	0-cont.	0-pulse	0-PB60FF	0-stop
1-50Hz	1-sortie	1-CNT	LOAD (strobe)	1-one-shot	1-toggle	1-PB60N	1-start

Détail des bits du registre de contrôle B

Registre N 15 / \$F Nom : CRB

D7	D6+D5	D4	D3	D2	D1	D0
ALARM	INMODE	LOAD	RUNMODE	OUTMODE	PBON	START
0-TOD	00-Hor1.	1-FORCE	0-cont.	0-pulse	0-PB70FF	0-stop
1-Alarm	01-CNT 10-Timer A 11-CNT+ Timer A	LOAD (strobe)	1-one-shot	1-toggle	1-PB70N	1-start

Le compteur (Event-counter)

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
8 \$8	LSB Event	E7	E6	E5	E4	E3	E2	E1	E0
9 \$9	Event 8-15	E15	E14	E13	E12	E11	E10	E9	E8
10 \$A	MSB Event	E23	E22	E21	E20	E19	E18	E17	E16

Comme mentionné précédemment, le 8520 se différencie du 6526 par quelques modifications seulement, les seules différences se trouvant au niveau des fonctions des registres 8-11. Sur le 6526 on trouve une horloge temps réel (time of day, TOD), l'heure du jour étant sauvegardée sous forme d'heures, minutes et secondes dans des registres

isolés. Dans le circuit 8520 on trouve cette horloge dans un compteur 24 bits, qui se nomme EVENT COUNTER.

On pourrait être induit en erreur, par le fait que COMMODORE utilise en partie, sur le 8520, les anciennes instructions TOD.

Le rôle de l'EVENT COUNTERS est simple : il reproduit un compteur 24 bits. Ceci signifie qu'il peut prendre les valeurs de 0 à 16 777 215 (\$FFFFFF). A chaque impulsion positive (variable de la valeur basse vers la valeur haute), le signal TOD élève de 1 la valeur du compteur. Lorsque le compteur atteint \$FFFFFF, il se réinitialise à 0 à la prochaine impulsion. Le compteur peut être fixé si l'on écrit la valeur désirée dans son registre.

Le registre 8 qui contient les bits 0-7 du compteur, est appelé LSB (Lowest Significant Byte = octet de plus faible poids), le registre 9 renfermant les bits 8-15, et le registre 10(\$A) les bits 16-23, le MSB (Most Significant- Byte = octet de plus fort poids).

A chaque accès en écriture, le compteur s'arrête, afin qu'aucune erreur ne se produise pendant les transferts de registres.

Après que la valeur ait été chargée dans le LSB, le compteur continue son travail. Dans l'ordre normal, le registre 10 (MSB), puis le registre 9 et enfin le registre 8 sont alors pris en compte.

S'il n'apparaît aucune erreur de transfert lors de la lecture de la valeur du compteur, cette dernière sera sauvegardée et verrouillée en mémoire pendant la lecture du MSB (registre 10). Chaque accès au registre du compteur fournit la valeur verrouillée pendant que le compteur interne tourne, la lecture pouvant se faire en toute tranquillité. La valeur sera à nouveau verrouillée lorsqu'on tentera de lire le LSB. Si l'on veut lire le compteur en entier, il faudra, comme en écriture, lire en premier le MSB, puis le registre 9 et enfin le LSB.

En outre, une fonction alarme est intégrée ; si l'on met dans le registre de contrôle B, le bit alarme (n°7) à 1, on peut écrire dans les registres 8-10, une valeur alarme.

Aussitôt que la valeur du compteur correspond à celle de l'alarme, le bit alarme du registre de contrôle des interruptions sera activé. La valeur de l'alarme ne pourra qu'être prise en compte. Un accès lecture sur les adresses 8-10 ne donne que l'état actuel du compteur, et ceci que le bit alarme du registre de contrôle soit fixé ou non.

Le port série

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
12 \$C	SDR	S7	S6	S5	S4	S3	S2	S1	S0

Le port série se compose d'un registre de données séries (SDR, Serial Data Register) et d'un registre à décalage, sur lequel on n'a pas d'accès direct. Avec le bit SPMODE du registre de contrôle A, on peut se configurer soit en mode entrée (SPMODE = 0), soit en mode sortie (SPMODE = 1).

Dans le premier mode, les données séries sur le signal SP seront décalées dans le registre de même nom à chaque impulsion positive donnée par le signal CNT. Après 8 impulsions, le registre à décalage est plein et son contenu est transféré dans un registre de données. En même temps le bit SP du registre contrôle des interruptions est activé. Si une nouvelle impulsion CNT se présente, les données se décalent à nouveau dans le registre jusqu'à ce que celui-ci soit totalement renouvelé. Si entre temps, l'utilisateur a lu le registre de données séries (SDR), la nouvelle valeur sera copiée dans SDR, et les transferts se dérouleront continuellement suivant le même schéma.

Pour pouvoir utiliser le port série en tant que sortie, on met le bit SPMODE à 1.

La fréquence de la minuterie A détermine le débit (en bauds ou bits par seconde). Les données seront toujours extraites du registre à décalage suivant la demi-fréquence de la minuterie, le taux de sortie maximum équivalant au quart de la fréquence d'horloge du 8520.

Le transfert commence après que le premier octet des données ait été inscrit dans SDR. Le CIA transfère l'octet dans le registre à décalage. Les bits des données apparaissent sur le signal SP suivant la fréquence réduite de moitié de la minuterie A, le signal d'horloge de cette minuterie s'appliquant sur le signal CNT.

Le transfert commence avec le MSB des octets des données. Lorsque les huit bits sont émis, CNT reste sur l'état actif et le signal SP, quant à lui, reste au niveau du dernier bit émis. De plus, le bit SP sera mis dans le registre de contrôle, afin de montrer que le registre de décalage peut être remplacé par de nouvelles données. Si l'octet suivant est inscrit dans le registre de données avant l'émission du dernier bit, l'émission des données se poursuivra sans interruption.

Si on veut un transfert continu, on doit alimenter opportunément le registre de données série en nouvelles données. Les signaux SP et CNT sont utilisés comme des collecteurs entrée/ sortie. Ces signaux permettent aussi la direction de plusieurs circuits de type 8520.

Le registre de contrôle d'interruption (ICR)

Accès lecture (READ) = registre de données

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
13 \$D	ICR	IR	0	0	FLAG	SP	Alarm	TB	TA

Accès écriture (WRITE) = registre masque

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
13 \$D	ICR	S/C	x	x	FLAG	SP	Alarm	TB	TA

L'ICR se compose d'un registre de données et d'un registre masque. Chacune des 5 sources d'interruption trouve un bit correspondant dans le registre des données.

Voici un rappel des cinq sources d'interruptions possibles :

- ① passage à zéro de la minuterie A (TA, Bit 0).
- ② passage à zéro de la minuterie B (TB, Bit 1).
- ③ concordance de la valeur de l'Event-Counter avec celle de l'alarme (alarm, bit 2).
- ④ registre à décalage du port série plein (entrée) ou vide (sortie) (SP, bit 3).
- ⑤ niveau négatif de l'entrée FLAG (FLAG, bit 4).

Lorsqu'on lit le registre ICR, on obtient toujours la valeur du registre de données, lequel sera effacé (tous les bits actifs IR inclus, sont remplacés). Si on a encore besoin de cette valeur, on devra la sauvegarder en mémoire (RAM) après la lecture.

Le registre masque ne supporte que le mode écriture. Sa valeur détermine si un bit dans le registre de données, libère ou non une interruption. Afin qu'une de ces dernières soit possible, le bit correspondant du registre masque doit être mis à 1. Le 8520 met le signal IRQ à 0 dès qu'un bit, dans un des registres, est activé, et met le bit 7, IR, dans le registre de données, afin qu'il signale une interruption au niveau du software.

Lorsque le registre ICR sera lu et que le registre des données sera effacé, le signal IRQ se remettra à nouveau à 1.

On peut écrire dans le registre masque de la même façon que dans un autre emplacement mémoire. Afin d'activer un bit du registre masque, on doit aussi activer le bit S/C (SET/CLEAR, bit n7), les autres bits restant non influents. Pour effacer un bit, on doit activer le bit correspondant, le bit S/C devant être, lui, mis à 0. Le bit S/C détermine donc si les bits activés du masque effacent (S/C= 0) ou activent (S/C= 1) les bits correspondants du registre masque. Tous les bits effacés dans le masque n'ont aucune conséquence sur les bits du registre masque.

Exemple :

La valeur actuelle du registre masque est 0000 0011. On désire autoriser l'interruption à l'aide du signal FLAG.

On écrit pour cela dans le registre masque la valeur 1001 0000 binaire (S/C = 1, FLAG = 1).

Le contenu du registre masque est maintenant : 0001 0011.

Si l'on veut interdire les deux interruptions minuterie, on écrit la valeur suivante : 0000 0011 (S/C = 0, TA = 1, TB = 1). Les bits TA et TB seront supprimés du registre masque.

Le registre masque renferme 0001 0000, et ainsi seule l'interruption FLAG est autorisée.

Le rôle du CIA dans le système AMIGA

Comme les entrées le précisent, l'AMIGA possède deux CIA du type 8520. L'adresse de base du premier 8520 (appelé 8520 A) est BFE001. Les écarts entre les registres sur l'espace adresse sont de 256 octets.

Ainsi, tous les registres du 8520 A se trouvent à des adresses impaires, ce dernier étant relié aux bus de données du processeur par les 8 signaux inférieurs (D0-7).

Les tableaux suivants donnent la liste des adresses des registres avec leur utilisation pour l'AMIGA.

CIA - A : adresses des registres

Adresse	Nom	D7	D6	D5	D4	D3	D2	D1	D0
\$BFED01	ICR								
		Registre de contrôle d'interruption							
\$BFE001	CRA								
		Registre de contrôle A							
\$BFEF01	CRB								
		Registre de contrôle B							

Le CIA-B a son adresse de base à \$BFD000. Ses registres se trouvent à des adresses paires. Les bus de données du CIA-B sont reliés par les signaux supérieurs aux bus de données du processeur.

CIA - B : adresses des registres

Adresse	Nom	Adresses des registres							
\$BFD000	PRA	DTR	/RST	/CD	/CTS	/DSR	/SEL	/POUT	/BUSY
\$BFD100	PRB	MTR	/SEL3	/SEL2	/SEL1	/SELO	/SIDE	/DIR	/STEP
\$BFD200	DDRA	1	1	0	0	0	0	0	0
\$BFD300	DDRB	1	1	1	1	1	1	1	1
\$BFD400	TAL0	La minuterie A n'est utilisée que pour le							
\$BFD500	TAHI	transfert de données séries							
\$BFD600	TBLO	La minuterie B est utilisée par le blitter en							
\$BFD700	TBHI	mode synchrone pour le transfert d'images							
\$BFD800	E.LSB	L'évent counter du CIA-B compte les impulsions							
\$BFD900	E.8.15	synchrone horizontales ; en temps normal elles							
\$BFDA00	E.MSB	ont une fréquence de 15625 par seconde							
\$BFDB00		inutilisé							
\$BFDC00	SP	registre de données séries							
\$BFDD00	ICR	registre de contrôle d'interruption							
\$BFDE00	CRA	registre de contrôle A							
\$BFDFO0	CRB	registre de contrôle B							

Les deux adresses \$BFD000 du CIA-B et \$BFE001 du CIA-A sont données par COMMODORE comme étant les adresses de base du CIA. En examinant le schéma de montage, vous pouvez remarquer que les deux CIA sont entièrement adressés de A0XXXX à BFXXXX. Le choix entre ces deux circuits est déterminé par les signaux d'adresses A12 et A13. CIA-A sera sélectionné lorsque A12 sera égal à 0 et CIA-B, lorsque A13 sera égal à 0. Les adresses quant à elles seront toujours prédéfinies et s'étaleront entre l'adresse A0XXXX et BFXXXX.

Du fait des liaisons des bus de données du CIA-A et du CIA-B avec les bus de données, respectivement, D0-7 et D8-15, on a la possibilité de transférer des mots (16 bits) lorsque A12 et A13 sont à 0.

MOVE.W \$BF0000, D0 charge le registre PA des deux CIA dans D0, les 8 bits de poids faibles de D0, renfermant le contenu du registre PA du CIA-A, les bits 9-15, renfermant le contenu du registre CIA-B.

On peut remarquer la façon d'adresser le CIA :

Le CIA-A sera sélectionné par une adresse binaire de type :

101X XXXX XX01 RRRR XXXX XXX0

Le CIA-B :

101X XXXX XX10 RRRR XXXX XXX1

Les 4 bits décrits par R permettent la sélection d'un des 16 registres du CIA.

Ceci ne fonctionne qu'avec l'AMIGA 1000. Il est possible que vous ayez à le modifier sur les nouveaux modèles, et à utiliser les adresses recommandées par COMMODORE (CIA-A \$BFE001 et CIA-B \$BFD000).

La liste suivante donne les références des différents signaux des CIA de l'AMIGA.

CIA-A

IRQ	INT2 entrée de PAULA
RES	broche de reset system
D0-D7	bus de données du processeur bits 0-7
A0-A3	bus d'adresse du processeur bits 8-11
Phi 2	horloge E du processeur
R/W	processeur R/W
PA 7	port manette 1/broche 6 (bouton de tir)
PA 6	port manette 0/broche 6 (bouton de tir)
PA 5	/RDY "disk ready", disque prêt
PA 4	/TK0 "disk track 0", disque piste 0
PA 3	/WPRO "write protect", protection en écriture
PA 2	/CHNG "disk change", disque changé
PA 1	/LED état de la diode LED (0= allumé)
PA 0	/OVL "memory overlay bit" recouvrement mémoire
SP	KDAT données série du clavier
CNT	KCLK horloge clavier
PB0-PB7	signaux de données pour port parallèle (centronic)
PC	/DRDY données prêtes pour port parallèle
FLAG	/ACK acquittement pour port parallèle

CIA-B

/IRQ	/INT 6-entrée de PAULA
/RES	signaux de reset system
D0-D7	bus de données du processeur bits 8-15
A0-A3	bus d'adresse du processeur bits 8-11
Phi 2	horloge E du processeur
R/W	processeur R/W
PA 7	/DTR connecteur série sortie DTR
PA 6	/RTS connecteur série sortie RTS
PA 5	/CD connecteur série sortie CD (carrier detect)
PA 4	/CTS connecteur série sortie CTS
PA 3	/DSR connecteur série sortie DSR
PA 2	SEL "SELECT" contrôle port parallèle
PA 1	POUT "paper out", papier absent (imprimante sur port centronic)
PA 0	BUSY "busy", imprimante occupée sur port parallèle
SP	BUSY imprimante occupée port A bit 0
CNT	POUT imprimante occupée port A bit 1
PB 7	/MTR "motor" moteur lecteur de disquette
PB 6	/SEL 3 "drive select" sélection lecteur disquette n°3 (DF3:)
PB 5	/SEL 2 "drive select" sélection lecteur disquette n°2 (DF2:)
PB 4	/SEL 1 "drive select" sélection lecteur disquette n°1 (DF1:)
PB 3	/SEL 0 "drive select" sélection lecteur disquette interne (DF0:)
PB 2	/SIDE "side select" sélection de la face de la disquette
PB1	DIR "direction" direction d'avancement du moteur de lecteur de disquette
PB 0	STEP "step", avance d'un pas du moteur du lecteur de disquette
FLAG	/INDEX "index", début du cylindre (disquette)
PC	non utilisé.

1.2.3. Rôle des circuits spécialisés dans le hardware de l'Amiga

Les processeurs dont nous avons parlé jusqu'à présent, sont assez banals. Même le 68000 n'est qu'un circuit standard, qu'on peut trouver pour quelques centaines de francs dans n'importe quel magasin d'électronique.

Tout de même, les fans et les utilisateurs de l'Amiga ont d'autres aspects en vue.

Que l'ordinateur soit capable de calculer d'importantes quantités de feuilles de salaire par secondes, ou qu'il soit plus rapide qu'un vieux calculateur, ne sont pas les critères prépondérants pour l'achat d'un AMIGA.

Le fait d'avoir la possibilité d'afficher et de traiter des images d'une qualité télévisuelle, tout en écoutant la neuvième de Beethoven, avec une telle sonorité, que l'observateur cherche en vain une platine CD, cela en revanche peut attirer l'attention.

Les concepteurs de l'AMIGA l'ont pourvu de capacités graphiques et sonores qui n'ont jamais été égalées par un ordinateur de cet ordre de prix.

Le but de ce chapitre est de présenter le HARDWARE de l'AMIGA qui permet ces fantastiques possibilités graphiques et sonores, afin de donner au lecteur une base de choix pour une meilleure programmation.

Les éléments de base permettant les possibilités citées plus haut, se réduisent à 3 circuits.

Ils ont été conçus pour l'AMIGA et portent le nom de circuits spécialisés (on appelle circuit spécialisé un circuit intégré, conçu par une firme spécialisée dans les semi-conducteurs, pour une machine déterminée, dans un but précis).

Leurs dénominations sont respectivement 8361, 8362, 8364.

Cette désignation étant trop lourde pour les concepteurs, ceux-ci les baptisèrent AGNUS, DENISE, PAULA (l'AMIGA français comporte un circuit à la norme télévisuelle PAL, qui est en fait une version ultérieure de AGNUS, sa dénomination étant 8367).

Ces puces prennent en charge la création des sons, la reproduction d'images et les accès disquette indépendamment du processeur, plus d'autres fonctions diverses. Ces différentes tâches ne sont pas séparées, de telle façon que l'un des circuits s'occupe du son, l'autre du graphisme et le troisième des opérations de lecture sur disquette.

Au contraire, les tâches se répartissent sur plusieurs circuits à la fois; ainsi, par exemple, la reproduction du graphisme est assurée par deux des trois circuits intégrés.

Pour une telle mise en commun des tâches, on aurait pu réunir les 3 circuits au sein d'un seul circuit intégré. Mais la conception d'un tel circuit spécialisé serait revenue trop chère par rapport à la conception des trois séparés.

Avant de voir plus en détail les fonctions de AGNUS, DENISE et PAULA, voici une courte introduction à la structure de l'AMIGA.

1.2.3.1. L'architecture de l'Amiga

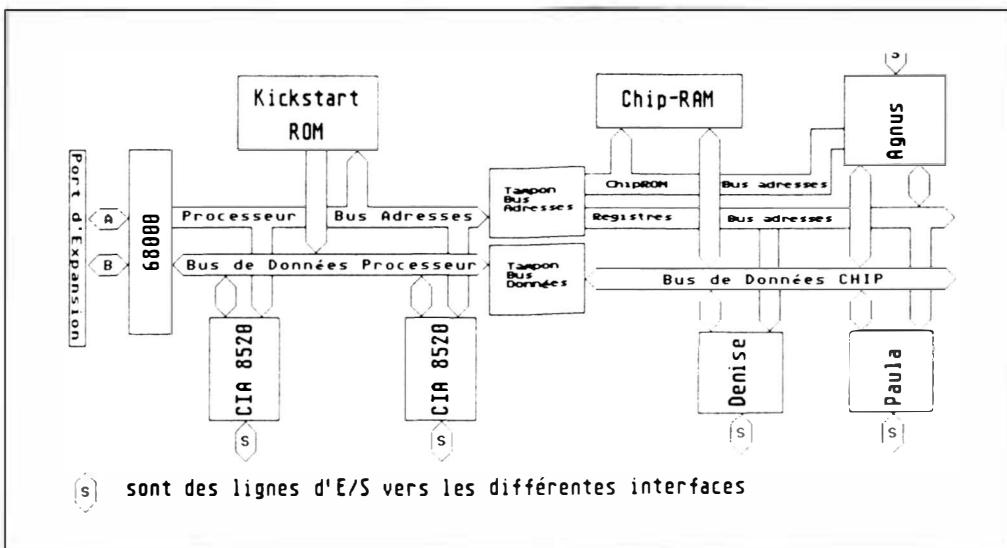


Figure 1 - 4

La structure d'un ordinateur se compose simplement d'un processeur, d'une ROM (mémoire morte) contenant le système d'exploitation, d'une quantité de mémoire vive (RAM) connue, et d'au moins une structure périphérique assurant la gestion des entrées/sorties.

Il ne faut pas oublier les bus d'adresses et les bus de données. En effet, le processeur contrôle le système en envoyant ou en recevant les messages des composantes du système par le biais de ces bus.

Il contrôle aussi les signaux de direction du bus, comme par exemple, les signaux READ/WRITE.

Chaque système d'ordinateur comporte en outre des circuits de contrôle, comme un décodeur d'adresse, qui, pour une valeur déterminée sur le bus d'adresse, active un circuit affecté à cette adresse.

Comme on peut le remarquer sur le schéma 1-4, la structure de l'AMIGA diffère un peu de cette courte description.

On s'aperçoit que de nombreuses broches, présentes sur l'ancien circuit, n'apparaissent plus sur le schéma. Elles étaient en fait reliées à des circuits qui font maintenant partie intégrante de FAT-AGNUS, ce qui explique la présence de nouveaux signaux.

Sur la gauche on voit le processeur 68000 relié directement aux 2 CIA et à la ROM, par le biais des signaux de données et d'adresses. Cette partie de l'AMIGA est en fait conventionnelle, seul le processeur ayant accès à la ROM et aux deux CIA.

Sur la droite du schéma, on trouve les trois circuits spécialisés AGNUS, DENISE et PAULA, ainsi que la mémoire interne RAM, qui sont reliés directement entre eux, via le bus de données et séparés du processeur par un buffer (tampon). Le 68000 peut être relié ou séparé, suivant son choix, par le circuit bus de données.

Les trois circuits spécialisés peuvent être liés par le registre bus d'adresse, le processeur pouvant, selon son choix, être relié par le bus d'adresses processeur.

Etant donné que la CHIP-RAM possède une grande zone adressable et qu'elle nécessite un adressage multiplexé, ses accès transitent par un bus d'adresses particulier (bus CHIP-RAM).

Voici une petite explication de ce qu'est l'adressage multiplexé :

La CHIP-RAM de l'Amiga (A1000) est composée de 216 adresses (65536). On a donc besoin de 16 signaux d'adresses pour pouvoir accéder à la totalité de la CHIP-RAM. Le boîtier ne possède que 8 signaux d'adresse pour des raisons de proportions. C'est pour cela que l'on a introduit l'adressage multiplexé : les 8 bits de poids fort seront d'abord véhiculés puis les 8 bits de poids faible. L'octet de poids fort sera stocké en mémoire en attendant que l'octet le moins significatif soit obtenu.

La raison de la séparation du bus d'adresses processeur et du bus CHIP-RAM est que les différents signaux d'E/S nécessitent un approvisionnement continu. Par exemple, les données se rapportant à un point de l'écran doivent être lues dans la RAM cinquante fois par seconde (norme PAL).

Un graphique haute résolution peut utiliser jusqu'à 64 Ko de mémoire écran. A chaque seconde, 50×64 Ko doivent être transférés de la mémoire vers l'écran, ce qui correspond à environ 1,5 millions d'accès mémoire par seconde. Le 68000 ne pouvant traiter autant de données par seconde, un tel travail serait au dessus de ses capacités.

L'AMIGA peut, en plus du graphisme, sortir des sons digitalisés tout en gérant des accès au lecteur de disquettes et ceci sans utiliser le 68000.

Il existe en fait un deuxième processeur, qui gère tout seul ces accès mémoire. Il s'agit du DMA-CONTROLLER, inclus dans le circuit AGNUS. Pour cette raison, AGNUS est aussi reliée au bus d'adresses CHIP-RAM.

Les deux autres circuits spécialisés, DENISE et PAULA, ainsi que le reste d'AGNUS, sont structurés comme des circuits périphériques. Ils possèdent une grande quantité de registres qui peuvent être lus ou écrits par le processeur (ou le contrôleur DMA). Les registres sont sélectionnés sur le bus d'adresses registres.

Il possède 8 signaux et peut donc prendre 256 états différents. Il n'y a pas de sélection spéciale des circuits. Si le bus d'adresses véhicule la valeur 255 (\$FF), les signaux sont

tous activés (mis à 1) et aucun n'est sélectionné. La sélection est effectuée par le décodeur d'adresse registre.

Ainsi, puisque la sélection d'un registre ne dépend que de son adresse registre et non du circuit dans lequel il se trouve, il est possible de mettre la même valeur dans les registres de deux circuits différents, lorsqu'ils ont les mêmes adresses registres. Cette possibilité est utilisée par certains registres qui contiennent des données nécessaires à plusieurs circuits.

Tous les registres d'un circuit peuvent aussi bien être accédés en lecture qu'en écriture. La commutation entre lire et écrire se fait au moyen d'un signal R/W spécial (ceci, par exemple, n'existe pas dans le 8520).

L'adresse registre détermine le mode d'accès (lecture ou écriture). Les registres qui acceptent ces deux modes sont réalisés de telle façon, que les accès lecture et écriture ont lieu sur des adresses registres différentes.

Etant donné qu'AGNUS renferme le contrôleur DMA, il peut avoir lui-même accès à tous les registres des circuits spécialisés.

Si deux circuits délivrent en même temps des données sur le même bus, il en résultera une collision des données suivie d'une rupture du système. Les circuits doivent donc se partager le bus de façon alternée, ce qui a été réalisé sur AMIGA d'une élégante manière :

Premièrement, le bus d'adresses et le bus de données de l'AMIGA sont séparés en deux. La première partie relie tous les circuits qui peuvent dialoguer avec le processeur. Les liaisons entre les bus d'adresses et de données du processeur et ceux des circuits se faisant par l'intermédiaire des deux buffers (tampons).

Ainsi, le processeur et AGNUS peuvent avoir un accès protégé au système d'exploitation ou à une extension mémoire RAM.

Cette dernière est souvent appelée FAST RAM, étant donné que le processeur peut y accéder sans perte de vitesse.

Deuxièmement, les accès au bus par le processeur et par AGNUS sont encastrés l'un dans l'autre, afin que les accès à la mémoire ou aux registres du 68000 ne soient pas freinés. Les tampons relient les deux systèmes de bus lors d'un tel accès.

Troisièmement, pour accéder aux bus, le processeur doit attendre qu'AGNUS les ait libérés. Ce cas de figure ne se présente que lorsque la haute résolution graphique est utilisée ou quand le blitter est mis à contribution.

L'AMIGA possède en fait trois types de RAM. La mémoire vidéo, que l'on appelle CHIP-RAM, la mémoire rafraîchie qui sert également à l'affichage vidéo et la FAST-RAM. Sur le modèle A1000, les 256 Ko de base sont de la CHIP-RAM et toute extension mémoire sera de la FAST-RAM. Les modèles A500 et B2000 possèdent 512 Ko de CHIP-RAM, les 512 Ko suivants forment la mémoire rafraîchie et au-dessus de 1 Mo on trouve de la FAST-RAM.

1.2.3.2. L'architecture d'AGNUS

Brochage du 8361

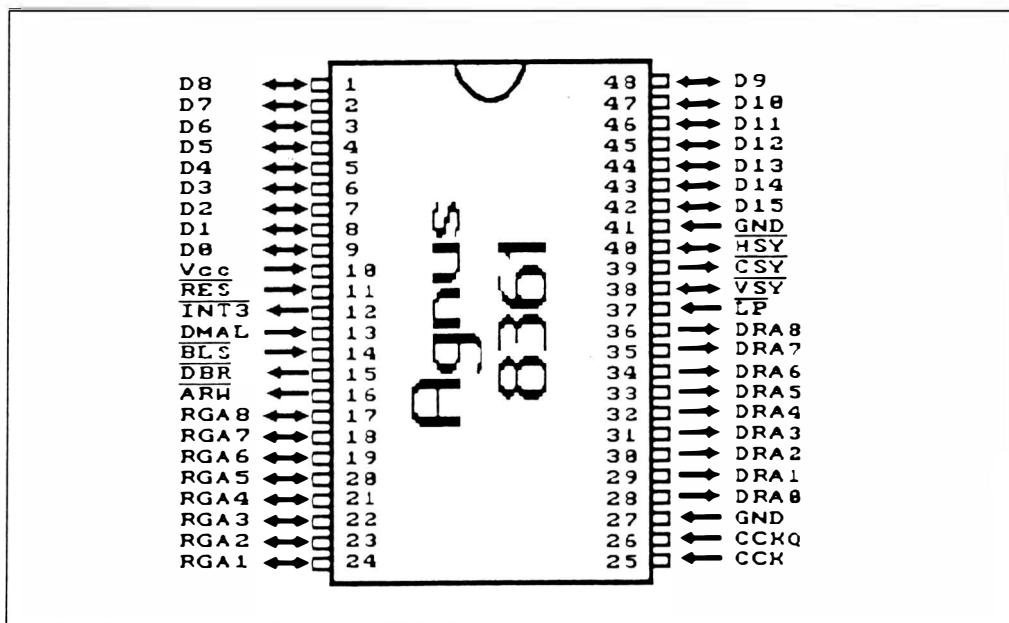


Figure 1 - 5

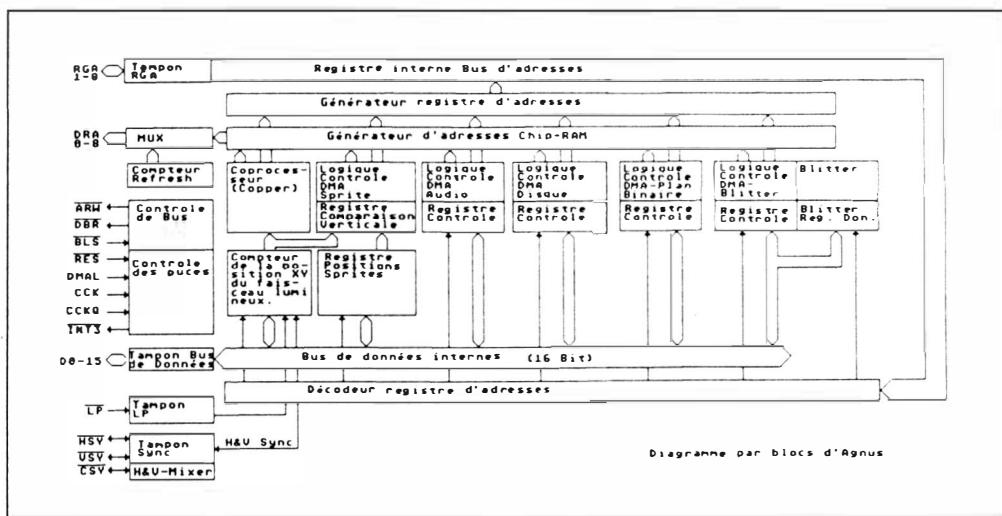


Figure 1 - 6

Comme nous l'avons mentionné plus haut, AGNUS renferme les canaux d'accès direct à la mémoire (DMA). Il existe pour chaque source DMA possible, un contrôle logique. Ces derniers sont reliés au générateur d'adresse mémoire, ainsi qu'au générateur d'adresse registre. Ces générateurs d'adresses produisent les adresses RAM des emplacements mémoire souhaités et les adresses registres des registres cibles. C'est pour cette raison que les canaux DMA ravitaillent les registres correspondants en données issues de la RAM ou écrivent le contenu de registres déterminés dans la RAM.

Il existe aussi un compteur dynamique lié au générateur d'adresse mémoire, qui produit un rafraîchissement du signal, permettant ainsi l'exploitation de la mémoire dynamique.

AGNUS dirige aussi le déroulement des accès DMA, par le biais d'un élément de base : une ligne écran. A chaque affichage d'une ligne écran, il y a 225 accès mémoire qui sont répartis par AGNUS entre les canaux DMA et le 68000. Comme AGNUS a besoin de connaître la position actuelle de la ligne et du point affiché, il mémorise aussi la trame désignée de l'écran et un compteur de colonnes.

Ce compteur de coordonnées est à l'origine des signaux de synchronisation horizontale et verticale, qui signalent à l'écran raccordé, le début d'une nouvelle ligne (H- SYNC) et d'une nouvelle image (V- SYNC).

Les signaux horizontaux et verticaux peuvent provenir d'une autre source que le circuit AGNUS, ce qui permet à l'image de l'AMIGA, d'être synchronisée avec une image extérieure, par exemple celle d'un magnétoscope. Ce système, aussi appelé GENLOCK, peut être facilement réalisé sur AMIGA.

Le Blitter et le coprocesseur Copper sont deux autres éléments importants du circuit AGNUS. Le Blitter est un circuit spécialisé, dont le rôle est de manipuler et de décaler les zones mémoire. Il peut ainsi soulager le 68000 de ces tâches et permettre un traitement plus rapide des données. Son rôle est aussi de tracer des lignes et de remplir des surfaces.

Le Copper est un coprocesseur, dont les programmes, aussi appelés COPPER-LIST, sont composés de trois instructions différentes (MOVE, WAIT et SKIP). Son rôle est de gérer les différents registres du circuit spécialisé. Des informations plus complètes se trouvent au chapitre : Programmation des circuits intégrés.

Voici une description détaillée des broches d'AGNUS :

Bus de données D0-15

Les 16 signaux de données sont directement reliés au circuit bus de données RAM, par le biais d'un tampon (buffer) interne.

Bus d'adresse registre RGA 1-8.

Les bus d'adresses registre sont bidirectionnels. Lors d'un accès DMA, le générateur d'adresses registre lâche l'adresse registre sur le bus. Lorsque le processeur intervient, ces signaux fonctionnent en mode entrée, et les adresses registre choisies par le processeur sont placées sur le décodeur d'adresses registre à l'intérieur d'AGNUS.

Les signaux d'adresses de la mémoire dynamique : DRA1-DRA8.

Ces signaux d'adresses sont reliés au bus d'adresses circuit RAM. Ce sont des signaux de sortie qui seront activés par AGNUS lors des accès DMA sur la CHIP-RAM. Les adresses, sur ces broches, sont multiplexées et peuvent relier directement 32 bits de mémoire dynamique avec les signaux d'adresses. C'est le cas pour les AMIGA 500 et 2000, le modèle précédent A 1000 ne possédant que 8 signaux d'adresses. Le signal DRA8 d'AGNUS sera démultiplexé et employé pour la sélection entre les différentes banques RAM.

Les signaux d'horloge d'AGNUS CCK et CCKQ.

Ce sont les seuls signaux d'horloge de l'AMIGA. La fréquence des deux signaux s'élève à 3,58 MHz, soit la demi fréquence du processeur. Le signal CCKQ est ralenti d'un quart de cycle d'horloge par rapport au signal CCK. Le timing de l'AGNUS est régi par ces deux signaux.

Les deux signaux sont reliés à la direction logique de l'AMIGA. Avec le signal DBR (Data Bus Request/demande de bus), AGNUS communique à cette direction logique qu'il prendra le bus en charge au prochain cycle de bus. Ce signal est toujours prioritaire pour une demande de bus du processeur. Si AGNUS a besoin du bus pendant plusieurs cycles, le 68000 devra patienter.

Le signal ARW (écriture mémoire par AGNUS) indique à la direction logique qu'AGNUS veut avoir un accès écriture sur la carte mémoire.

Le signal BLS (Blitter Slow Down/Ralentir Blitter) signale à AGNUS que le processeur attend pour un accès depuis trois cycles. Suivant l'état interne, AGNUS peut céder le bus au processeur pendant un cycle.

Les signaux de direction : RES, INT3, DMAL

Le signal RES (RESET) est relié directement au signal RESET du processeur et réinitialise AGNUS.

Le signal INT3 (Interruption #3) est une entrée qui est reliée au signal de même nom du circuit PAULA. AGNUS signale à la logique d'interruption de PAULA, qu'une composante d'AGNUS a libéré une interruption.

Le signal DMAL (DMA Request Line) relie, dans tous les cas, PAULA à AGNUS. PAULA signale ainsi qu'AGNUS doit entamer un transfert DMA.

Les signaux : HSY, VSY, CSY et LP.

Les signaux de synchronisation du moniteur proviennent des signaux HSY (Horizontal SYnc/synchronisation horizontale) et VSY (Vertical SYnc/synchronisation verticale). Le signal CSY (Composite SYnc) est la somme des signaux HSY et VSY. Il sert à la connexion d'un moniteur, qui demande un signal synchronisé mélangé et est utilisé par le vidéomixer, qui génère le signal vidéo.

Le signal LP (Light Pen) permet la connexion d'un stylo lumineux ou photostyle. Un signal négatif à cette broche entraînera l'acquisition des coordonnées en mémoire.

L'AMIGA peut recevoir un signal vidéo. La synchronisation externe d'AGNUS (GENLOCK) utilise les signaux HSY et VSY.

1.2.3.3. L'architecture de DENISE

Brochage du 8362

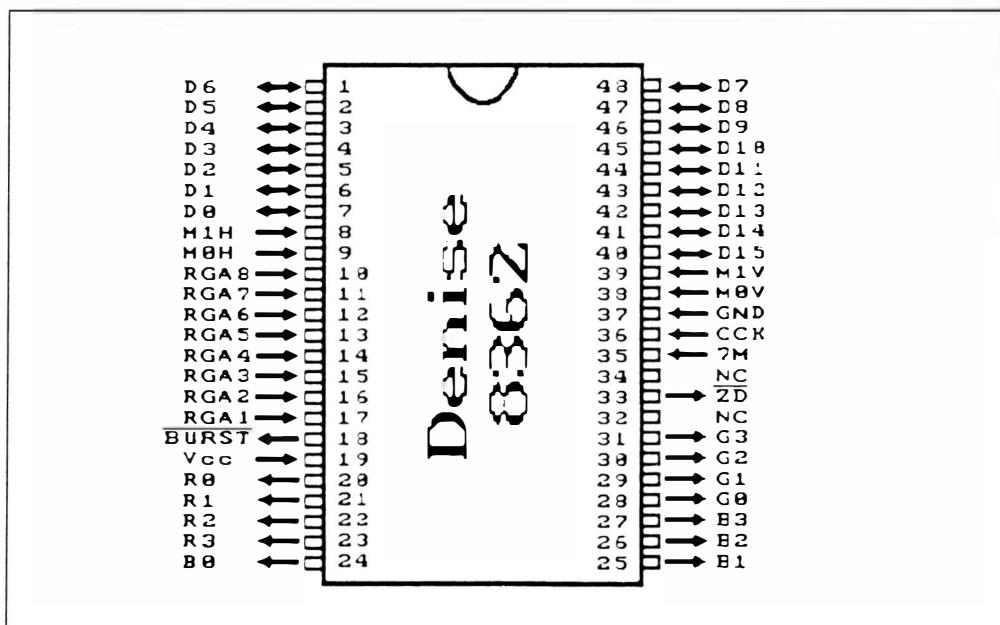


Figure 1 - 7

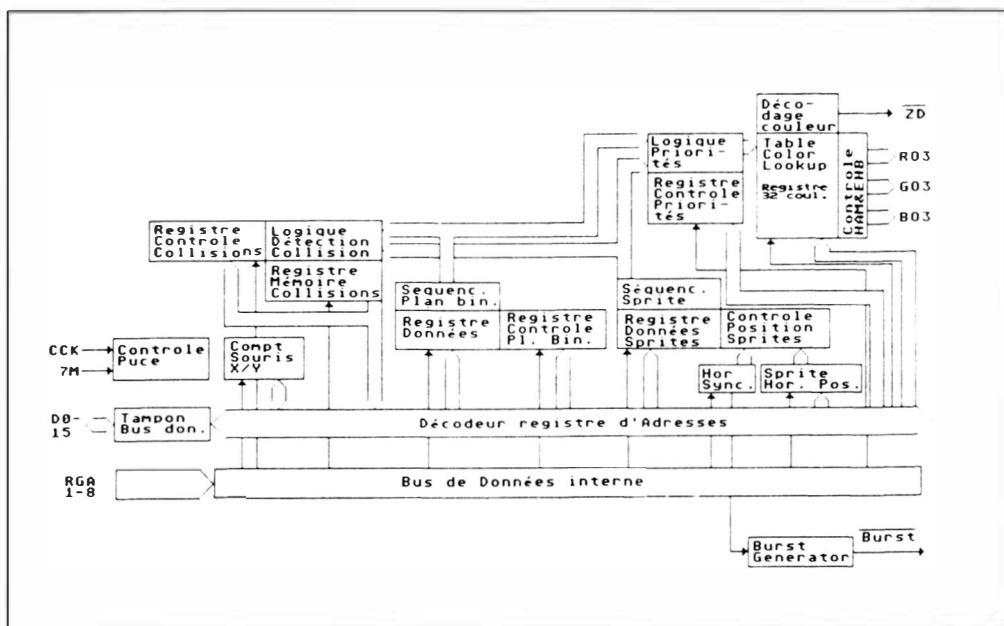


Figure 1 - 8

DENISE a pour rôle de générer les images. Les données graphiques en provenance de la CHIP-RAM sont transférées par AGNUS vers les registres BitMap (littéralement : Plan binaire) de DENISE. Le procédé est le même pour les sprites (lutins). La taille des registres de données de DENISE est de 16 bits. Chaque registre contiendra donc 16 points (un point par bit). DENISE va maintenant transformer ces données pour la visualisation RGB.

Un séquenceur va tout d'abord transformer les 16 points contenus dans le registre en signaux électriques séries. A chacun des 6 BitMaps correspond une transformation. Les données séries électriques en provenance du séquenceur BitMap formeront donc un signal d'une longueur de 6 bits.

Ensuite, le contrôle logique prioritaire, sur la base des priorités mises en place, va sélectionner les données valides du point parmi celles du séquenceur BitMap et celles du séquenceur Sprite.

A partir de ces données, le décodeur couleur va choisir un des 32 registres couleur.

La valeur de ce registre sera traduite sous la forme d'un signal digital RGB (Red, Green, Blue/Rouge, Vert, Bleu). Si l'un des modes HAM (Hold and Modify) ou EHB (Extra Half Brite) est sélectionné, les données des registres couleur seront modifiées avant que le signal quitte le circuit. Les données du séquenceur passent aussi par le contrôle logique

des collisions qui indiquera si un sprite occupe la même position sur l'écran qu'une image.

La dernière fonction de DENISE n'a rien à voir avec la représentation à l'écran. En effet ce circuit renferme également le compteur de la souris, ou autrement dit, les coordonnées x-y de la souris.

Voici la description des fonctions de chaque broche :

Bus de données : D0-D15

Les signaux bus de données sont reliés, comme AGNUS, au circuit bus de données.

Bus adresse registre : RGA1-RGA8

Le bus adresse registre se comporte seulement comme une entrée. Avec l'aide de la valeur du bus adresse registre, le décodeur d'adresse registre sélectionne le registre interne correspondant.

Les entrées Horloge : CCK et 7M

Le timing de DENISE s'aligne sur le signal CCK. La broche CCK est reliée à celle du circuit AGNUS. L'impulsion de l'Horloge du signal 7M a une fréquence de 7,15 MHertz. Le circuit nécessite une telle fréquence d'horloge, pour le traitement des points images. Un point en basse résolution (320 points/ligne) a exactement la durée d'un cycle d'horloge 7M.

En haute résolution (640 points/ligne), 2 points sont émis à chaque impulsion d'horloge 7M. Cette dernière correspond aussi à celle du processeur 68000, la broche 7M sera reliée avec son entrée CLK (entrée horloge).

Les signaux de sortie : R0-3, G0-3, B0-3, ZD et BURST.

Les signaux R0-3, G0-3 et B0-3 reproduisent le signal de sortie RGB de DENISE, ce dernier émettant la valeur sous forme digitale. Les trois composantes couleur seront reproduites à l'aide de 4 bits. Ceci fait en tout 16 valeurs par composante et totalise donc $16 \times 16 \times 16$ (4 096) couleurs possibles. Les trois signaux couleur, après que DENISE les ait lâchés, passeront par un tampon (buffer) et seront transformés en signal analogique RGB au moyen de trois transformateurs digitaux/ analogiques, afin qu'ils puissent se tenir à disposition sur le port RGB.

Un mélangeur vidéo transforme alors ce signal analogique en signal vidéo pour le connecteur vidéo. Il nécessite pour cela le signal BURST de DENISE, qui est en fait une oscillation d'une fréquence de CCK, c'est-à-dire de 3,58 MHertz.

Pour plus de précision sur ce signal, reportez-vous à un manuel traitant de la technique de la télévision.

Le dernier signal de DENISE est le signal ZD (zero detect ou back-ground indicator/indication de la couleur de fond). Il est toujours désactivé, lorsqu'un point de la couleur du fond est reproduit, sa couleur dérivant du registre couleur numéro 0.

Ce signal sera utilisé pour un adaptateur GENLOCK et servira dans ce cas de commutateur entre le signal vidéo externe (ZD=0) et le signal vidéo de l'amiga (ZD=1), le signal ZD s'applique sur le port RGB.

Les entrées souris-joystick : M0H, M1H, M0V, M1V.

Ces quatre signaux correspondent directement aux entrées souris ou joystick des deux connecteurs.

Etant donné que l'AMIGA possède deux connecteurs, il aurait fallu 8 entrées libres, alors que DENISE n'en possède que 4. Ce problème a été résolu par COMMODORE de la façon suivante :

Les huit signaux d'entrée des deux connecteurs accèdent à un commutateur, dont le rôle est de répartir les 4 signaux de chaque connecteur sur les 4 entrées de DENISE.

Ce commutateur fonctionne de façon synchrone avec l'horloge de DENISE, ceci permettant aux quatre signaux internes d'être partagés en deux registres, un pour chaque connecteur.

1.2.3.4. L'architecture de PAULA

Brochage du 8364

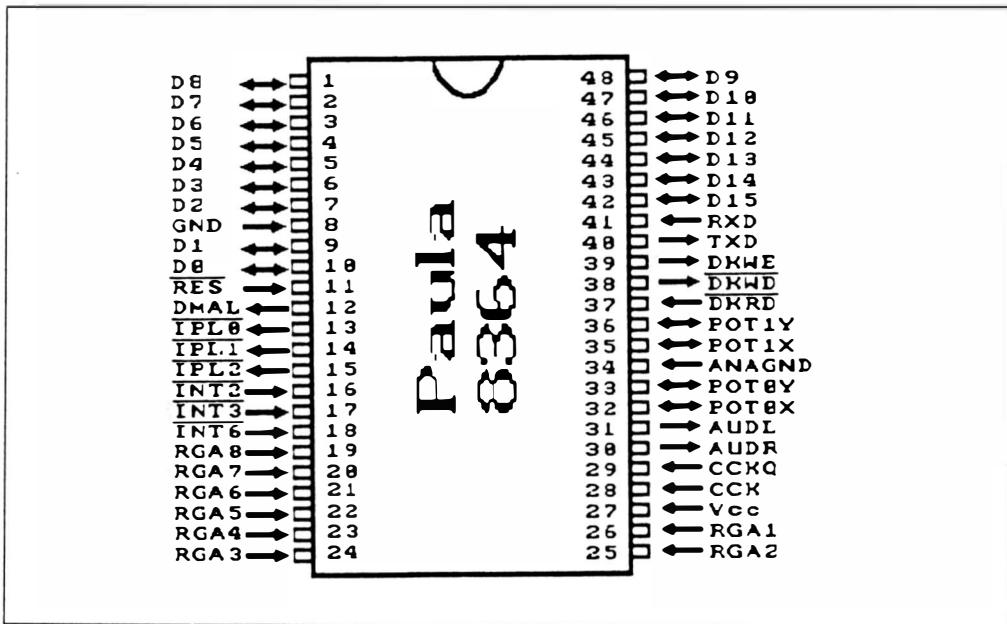


Figure 1 - 9

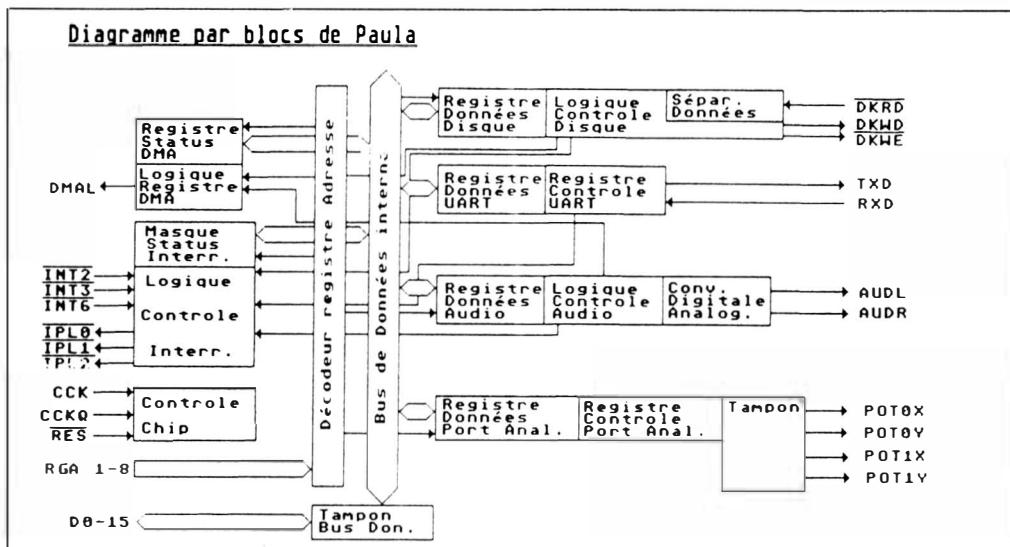


Figure 1 - 10

Le processeur spécialisé PAULA gère les traitements des entrées/ sorties (Input/output - I/O), disquettes, séries, sons et l'entrée analogique.

De plus, Paula sert de support à la gestion des interruptions. Toutes les interruptions présentes dans le système passent par lui. A partir de 14 sources d'interruptions possibles, PAULA engendre les signaux d'interruption pour le 68000. Les signaux d'interruption des plans binaires seront générés sur les signaux IPL du 68000, par exemple. Paula offre au programmeur la possibilité d'autoriser ou d'interdire chacune des 14 sources d'interruption.

Le transfert de données à partir de la disquette et la sortie son se déroulent également via les canaux DMA. On ne peut pas toujours prévoir par les données d'une disquette, le moment où le mot sera prêt pour un transfert DMA d'AGNUS ; la raison essentielle étant l'inévitable variation du nombre de tours du moteur du lecteur de disquette.

De même pour la sortie son, AGNUS ne peut prévoir quand les données seront nécessaires.

Afin de rendre possible un transfert DMA sans accroc, PAULA utilise le signal DMAL qui indique à AGNUS le moment où un accès DMA est nécessaire.

Les communications séries sont prises en charge par un boîtier USART au sein de PAULA. USART signifie Universal Asynchronous Receive Transmit ou plus simplement : Emetteur/Récepteur Asynchrone Universel.

Les fonctions de l'UART seront décrites, comme les 4 canaux audio et le port analogique, dans le chapitre Programmation des circuits intégrés.

Description des broches

Bus de données : D0-15

Comme sur les autres circuits, ces broches sont reliées au circuit bus de données.

Register Adress Bus : RGA 1-8

Idem DENISE

Le signal d'horloge et Reset : CCK, CCKQ et RES

Paula renferme les mêmes signaux d'horloge qu'AGNUS. Le signal RES remet le circuit en état d'allumage.

DMA Request : DMAL

Paula signale à AGNUS, au moyen de ce signal, qu'il nécessite un transfert DMA.

Sorties Audio : AUDL et AUDR

Les sorties AUDL (LEFT AUDIO) et AUDR (RIGHT AUDIO) sont analogues et produisent les signaux sons, le signal AUDL correspond aux canaux audio internes 0 et 3 ; le signal AUDR correspond aux canaux 1 et 2.

Les signaux du connecteur série : TXD et RXD

RXD (RECEIVE DATA/Donnée série reçue) est l'entrée série de l'UART, TXD (TRANSMIT DATA/Donnée série envoyée) étant la sortie série. Ces signaux ont un niveau TTL, ce qui signifie, que leur tension d'entrée/sortie varie de 0 à 5 volts.

De plus un transformateur de niveau engendre des tensions de +12/-5 volts, nécessaires au connecteur série RS232 de l'AMIGA.

Les entrées analogiques : POT0X, POT0Y, POT1X, POT1Y

Les entrées POT0X et POT0Y sont reliées aux signaux correspondant du connecteur souris 0, POT1X et POT1Y étant reliées au connecteur 1. A ces entrées peuvent être raccordés, soit des manettes (Paddles), soit des joysticks. Ces appareils de commande contiennent des résistances variables, dénommées potentiomètres et branchées entre le +5V et l'entrée POT. PAULA peut mesurer la valeur de cette résistance et la mettre

dans des registres internes. Les entrées POT peuvent être commutées en mode sortie par le biais du Software.

Les signaux d'accès disquette : DKRD, DRWD, DKWE

PAULA obtient la lecture des données d'une disquette par le signal DKRD (DISK READ/lecture données disquette). Le signal DKWD (DISK WRITE) permet l'écriture des données sur disquette. Le signal DKWE, quant à lui, sert de commutateur au lecteur de disquette pour passer du mode lecture en mode écriture (DISK WRITE ENABLE/autorisation d'écriture disque).

Les signaux d'interruption INT2, INT3, INT6, IPL0, IPL1, IPL2

Les trois signaux IPL permettent la création d'une interruption d'un niveau correspondant.

Le signal INT2 est relié au boîtier 8520 (CIA-A). On retrouve ce signal sur le port d'expansion et sur le connecteur série. Lorsqu'il est en position LOW, PAULA engendre une interruption du niveau 2, à condition qu'une interruption de ce niveau soit autorisée. Le signal INT3 est relié à la sortie correspondante d'AGNUS. Le signal INT6 est relié au CIA-B et au port d'expansion. Toutes les autres interruptions se déclenchent suivant les composantes I/O de PAULA.

Les signaux IPL0-IPL2 (ligne d'interruption du 68000) sont reliés directement aux signaux correspondants du processeur. PAULA engendre sur ces signaux, une interruption dont le niveau dépend du processeur.

1.2.3.5. Particularités de l'A500

La description du HARDWARE, dans ce chapitre concerne essentiellement l'AMIGA 1000. Elle est en grande partie valable pour l'A500. Son architecture principale n'est pas modifiée, les concepteurs n'ayant cherché qu'à présenter une version moins chère. Les plus grandes différences entre les deux modèles se trouvent dans la séparation des différents éléments du HARDWARE sur le même circuit.

On remarque, sur les circuits intégrés du modèle 1000, la présence d'un interrupteur logique de circuit, qui permet la création du signal d'horloge, la gestion des bus et le décodage d'adresse.

Ces fonctions logiques sont toutes réunies au sein d'un même circuit. On a ainsi rajouté au circuit AGNUS, une nouvelle partie du HARDWARE. Ce nouveau circuit portant le nom de FAT-AGNUS (modèle 8370).

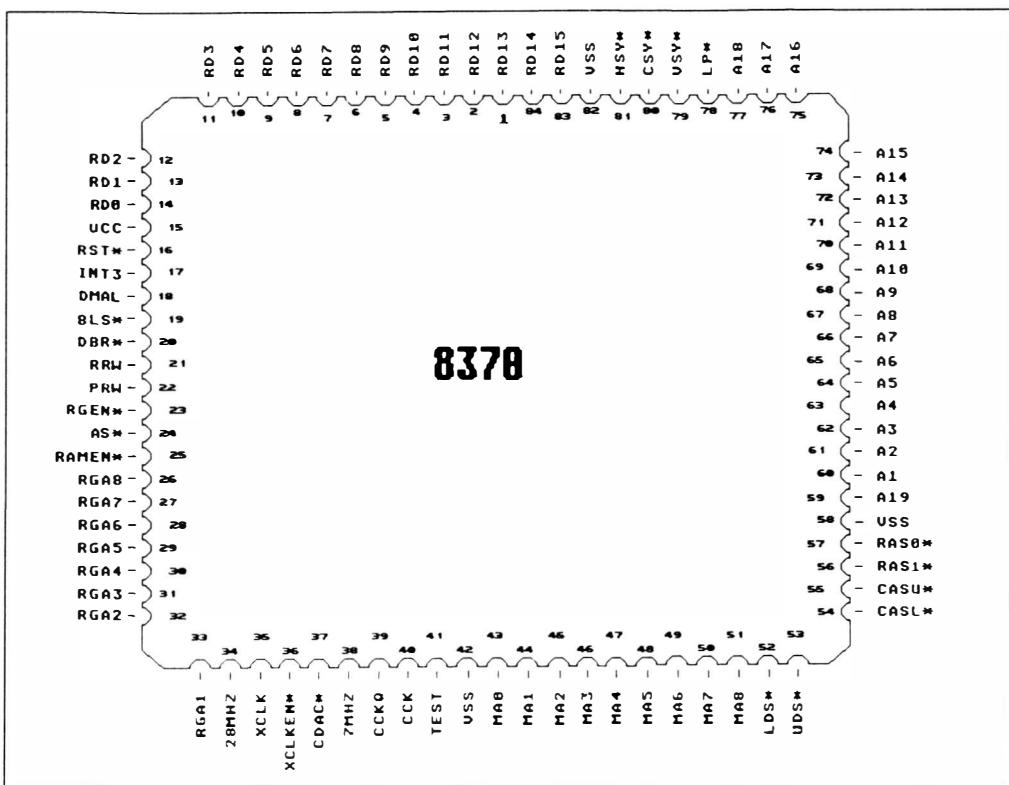


Figure 1 - 11

Le générateur d'horloge

Le générateur d'horloge du système de l'AMIGA est intégré dans AGNUS, seul le signal haute impulsion 28 MHERTZ doit être raccordé. Les signaux de cette fonction sont :

28MHERTZ, XCLK, XCLKEN, 7MHERTZ, CCKQ, CCK, CDAC.

Le tampon bus d'adresse

Sur le schéma de l'architecture de l'AMIGA, on remarque un tampon reliant le bus d'adresse de l'AMIGA au bus d'adresse CHIP-RAM et au bus d'adresse registre.

De plus, il multiplexe les adresses processeur correspondantes. Ce tampon est complètement intégré dans AGNUS ; le bus d'adresse processeur peut être raccordé directement aux signaux A1-18 de FAT- AGNUS. Le décodeur d'adresses signale que le processeur veut avoir accès à la RAM ou aux registres mémoire, au moyen des deux signaux RAMEN (RAM ENABLE) et RGEN (REGISTER ENABLE).

Toutefois, AGNUS est maintenant relié aux signaux UDS, LDS et PR/W du processeur.

La gestion de la Chip-Ram

La gestion de la chip-ram est complètement assurée par AGNUS. Celui- ci engendre les signaux nécessaires RAS et CAS avec les adresses RAM multiplexées. Ainsi AGNUS est capable de gérer 512 Koctets RAM de plus, soit un total de 1 mega. Les deux banques mémoires seront gérées au moyen des signaux de gestion de la RAM :

- ✓ RAS0 et CAS0 pour la chip-ram
- ✓ RAS1 et CAS1 pour l'extension mémoire.

Toutes les autres fonctions d'AGNUS décrites au début du chapitre ne sont pas modifiées.

En plus de FAT-AGNUS, un quatrième circuit spécialisé du nom de GARY a été réalisé. Il comprend les fonctions de décodeur d'adresses et de contrôleur de bus. Il crée les signaux de sélection des circuits spécialisés, tel UPA et DTACK du processeur. GARY renferme, enfin, la logique RESET et le FLIP-FLOP moteur pour le lecteur de disquette.

1.3. Les connecteurs de l'Amiga

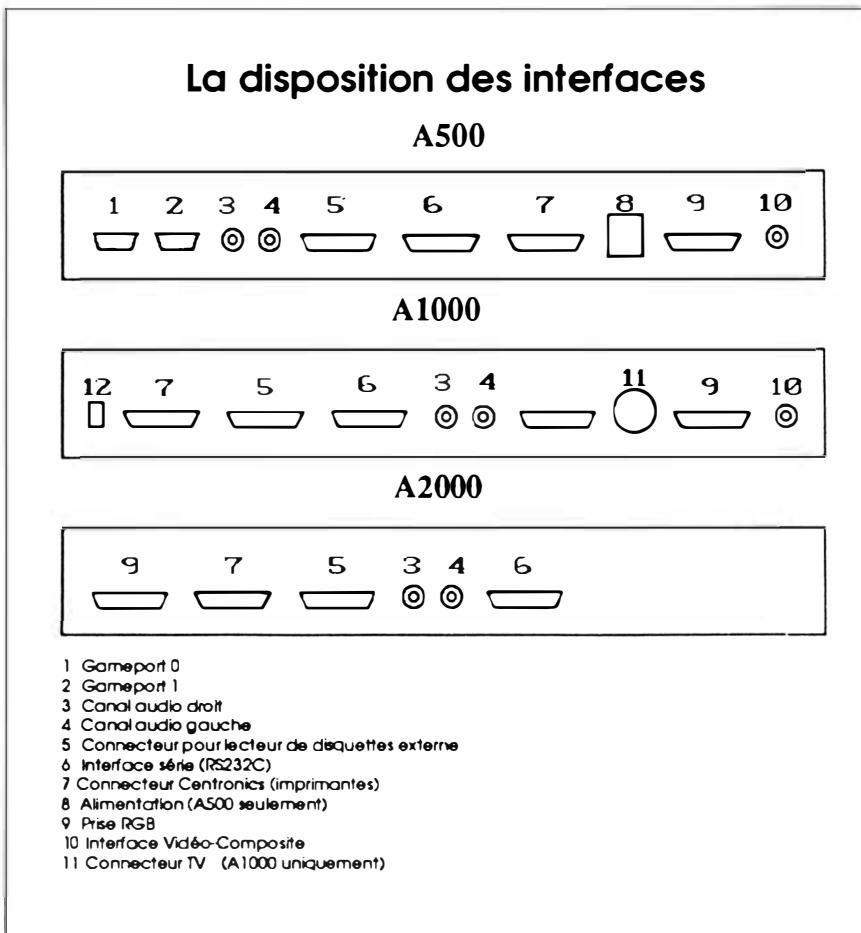


Figure 1 - 12

1.3.1. Connecteur Audio/Video

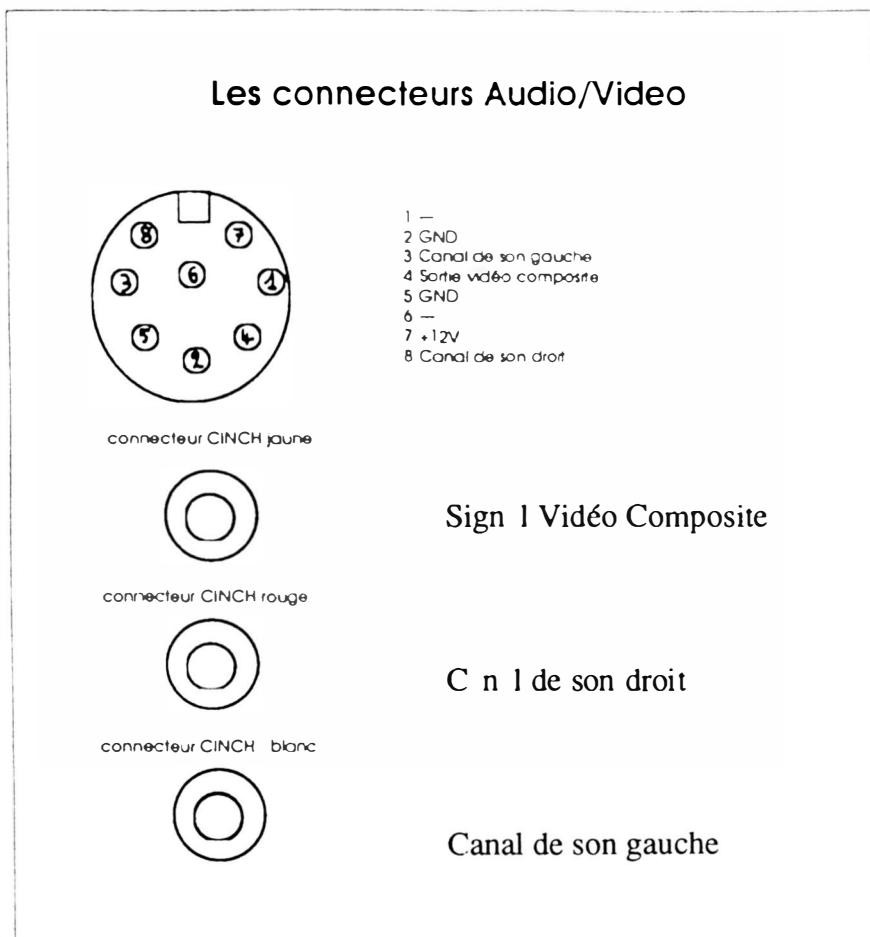


Figure 1 - 13

La description du connecteur vidéo diffère selon le type de l'Amiga. Le plus défavorisé est l'Amiga A2000 qui ne dispose d'aucune sortie vidéo, juste un connecteur permettant l'adaptation d'un modulateur vidéo ou d'une interface GENLOCK. Seuls les AMIGA A500, A1000 et B2000 disposent d'une sortie vidéo sous la forme d'un connecteur CINCH. Le signal vidéo, sur le connecteur de l'A1000, est de type FBAS, permettant le branchement avec la plupart des moniteurs du marché. Sur l'A500, par souci d'économie, il ne livre qu'un signal BAS (signal vidéo noir et blanc). Un autre problème est que l'ancien modèle A1000 n'a pas été fourni avec un clavier français. Sur ces modèles, la sortie vidéo livre un signal NTSC (signal vidéo suivant la norme américaine).

Sur moniteur couleur à la norme PAL (comme le moniteur de l'AMIGA), l'image apparaît en noir et blanc avec des raies perpendiculaires. Une image couleur nette, à la norme PAL, ne peut être obtenue que sur les nouveaux modèles A1000, disponibles avec un clavier français.

Sur tous les modèles, le signal vidéo est véhiculé sur un tampon transistorisé avec une résistance de sortie de 75 ohms. Il est ainsi protégé contre les court-circuits de façon permanente.

Le signal audio est transmit par 2 connecteurs CINCH sur tous les modèles de l'AMIGA. Le canal stéréo droit correspond au connecteur CINCH rouge, celui de gauche au CINCH blanc. A l'aide de câbles stéréo CINCH, on peut relier ces connecteurs à un amplificateur (entrée AUX ou CD) stéréo. La résistance de sortie s'élève à 1 Kohm (1000 ohm). Les sorties sont protégées contre les court-circuits et possèdent de façon interne une résistance de charge de 36 ohms.

L'AMIGA 1000 dispose d'un autre connecteur audio-vidéo. Ce connecteur modulateur TV a été prévu pour le raccord avec un modulateur HF, permettant l'utilisation d'une télévision comme moniteur. Ce modulateur HF n'est plus compris dans la panoplie des connecteurs. Il véhicule aussi bien le signal vidéo que les signaux audio. On y trouve aussi une prise 12 volts, qui a été prévue pour l'alimentation du modulateur. La sortie vidéo dispose d'un tampon transistorisé, celui-ci n'étant donc pas seulement relié avec le connecteur CINCH vidéo. Les deux broches audio ont une résistance de sortie de 1 Kohm.

Etant donné qu'il n'a pas été prévu de résistance interne de charge, leurs broches se trouvent dans un état non chargé, quatre fois aussi important que celui des connecteurs audio-CINCH.

Le connecteur modulateur TV est de type DIN à 8 broches. Les fiches convenant à ce type de connecteur sont difficiles à obtenir. Toutefois, il est bon de savoir que le connecteur modulateur TV est identique à celui du C64.

1.3.2. Connecteur RGB

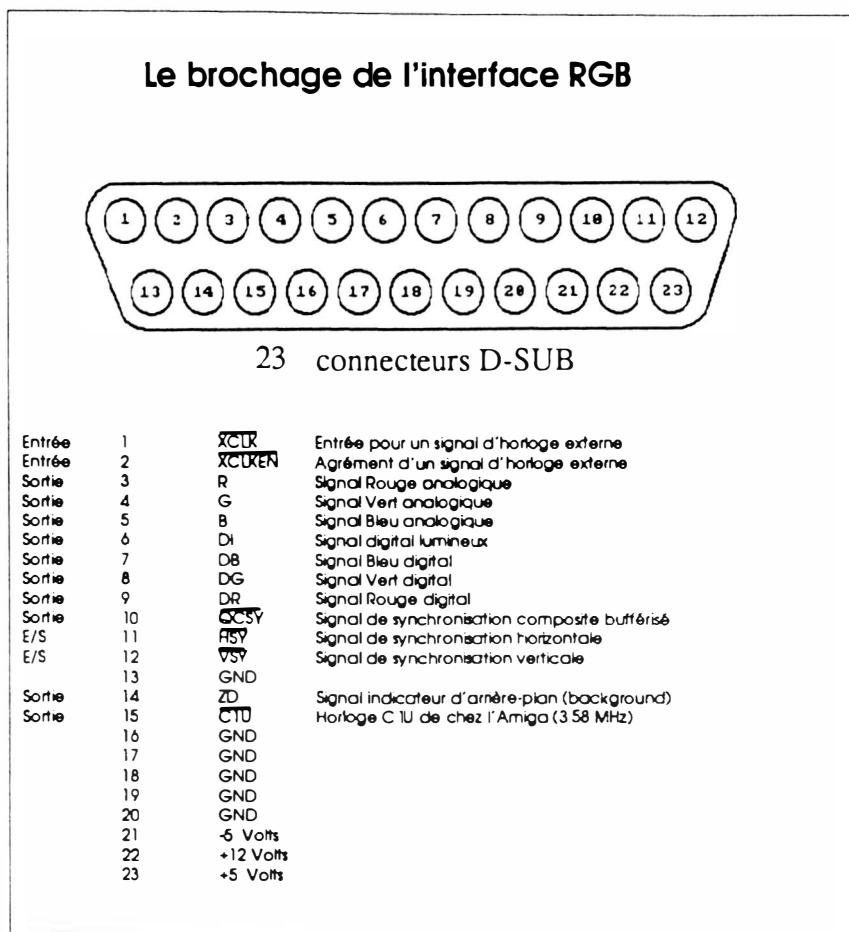


Figure 1 - 14

Le connecteur RGB est identique sur les trois modèles d'AMIGA. Il rend possible le branchement de différents moniteurs RGB, mais aussi d'extensions spéciales, telles que l'adaptateur Genlock. Lors d'un raccord avec un moniteur RGB analogique, tel le moniteur de l'AMIGA, les trois sorties RGB analogiques et la sortie composite Sync. sont utilisées. La transformation des signaux RGB digitaux issus de DENISE en signaux analogiques correspondants se fait au moyen d'un transformateur digital/analogique à 4 bits. Le signal de synchronisation composite provenant d'AGNUS est affiché après le mixage des signaux de synchronisation verticale et horizontale. Ces quatre fiches sont pourvues d'une résistance de sortie de 75 ohms et d'un tampon transistorisé, assurant une protection contre les court-circuits.

Les signaux DI, DB, DG et DR sont prévus pour les connexions des moniteurs RGB numériques. La source des signaux RGB numériques provient de la sortie digitale RGB de DENISE.

Les trois signaux de couleurs sont reliés avec ceux de DENISE (DB relié avec B3 par exemple). Toutefois on remarque un tampon de type 74HC 244 entre DENISE et les sorties. Les quatre signaux ont une résistance de sortie de 47 ohm et comme ils proviennent du tampon 74 HC 244, un niveau TTL.

On remarque la présence des broches HSY et VSY, celles-ci peuvent être nécessaires à certains moniteurs. Il faut cependant être attentif au fait que ces signaux ont une résistance de 47 ohms et sont directement reliés aux broches HSY et VSY d'AGNUS. Ils ont également un niveau TLL.

Si le bit Genlock d'AGNUS est activé, ces deux signaux fonctionnent en tant qu'entrée. L'AMIGA synchronise alors son signal vidéo d'après les signaux de synchronisation sur HSY et VSY. Aussi bien en mode entrée que sortie, ces signaux s'alignent au niveau TTL. Les signaux de synchronisation sont à usage Low-active, c'est-à-dire qu'à l'état normal, les signaux sont à 5 volts. Les signaux sont seulement à 0 volts lorsque les impulsions de synchronisation sont actives.

Les nouveaux Kickstart 1.2 (et 1.3) reconnaissent automatiquement à chaque reset si des signaux ont été envoyés aux deux conduits SYNC. Il suffit donc de créer ses impulsions synchrones, et l'Amiga passe en synchronisation externe.

Lorsque l'adaptateur Genlock est branché, le signal ZD est activé(zero detect). L'AMIGA met ce signal sur LOW, lorsque le point affiché reproduit un point de l'arrière plan de l'écran, autrement dit, lorsque cette couleur est contenue dans le registre de couleur 0.

Pendant le temps mort vertical (VSY=0), la fonction du signal ZD se modifie. On y trouve la valeur GAUD-bits (Genlock audio Enable) provenant du registre \$100 d'AGNUS (BPLON0). Ce signal est utilisé comme interrupteur du signal son par l'interface Genlock.

Pour l'utilisateur normal, le signal ZD n'est pas intéressant puisqu'il n'est nécessaire qu'avec l'interface Genlock. Le signal ZD de DENISE fonctionne également par l'intermédiaire d'un pilote 74HC244.

Les autres signaux du connecteur RGB n'ont plus aucun rapport avec le signal RGB.

Le signal C1U est un signal d'horloge à 3,58 MHZ et correspond au signal CLK des circuits spécialisés. Les signaux XCLK (External Clock) et XCLKEN (External Clock Enable) permettent d'alimenter l'AMIGA avec une fréquence d'horloge externe. Tous les signaux d'horloge de l'AMIGA dérivent d'une horloge de 28MHZ. Cette dernière peut être remplacée par une autre fréquence d'horloge par l'entrée XCLV, lorsque l'on met ce signal à 0. L'AMIGA peut donc être beaucoup plus rapide si on installe une horloge de 32MHZ à cette entrée, mais attention, seul le HARDWARE de l'AMIGA peut répondre de la fiabilité d'une telle expérience. Lors de l'utilisation des signaux

XCLK et XCLKEN, on utilise la broche de terre 13. Elle est directement reliée au signal de masse de l'horloge.

1.3.3. Le connecteur Centronics

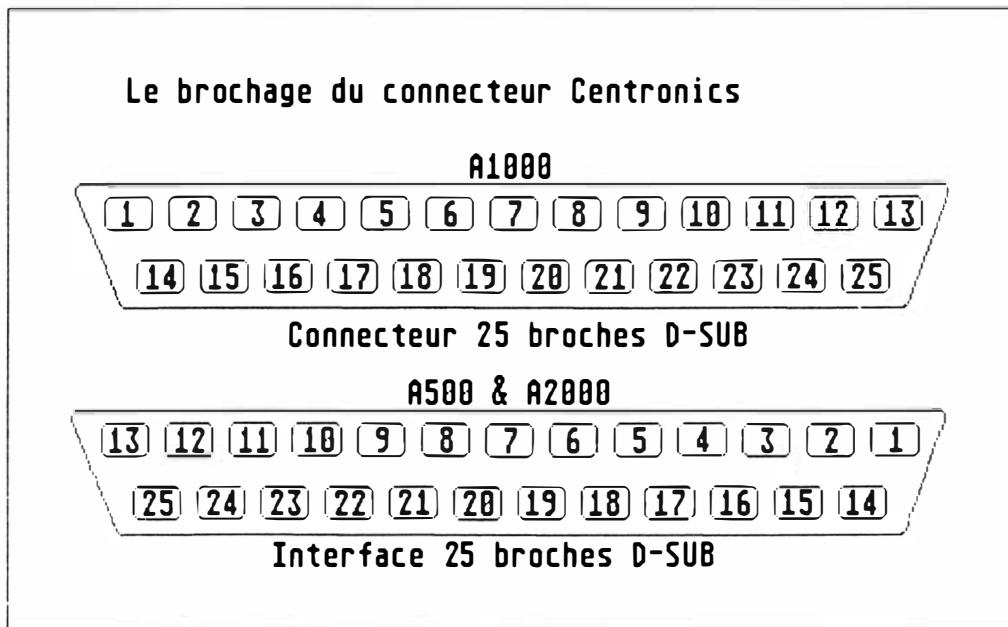


Figure 1 - 15

sortie	1	donnée valide
entrée/sortie	2	bit de donnée 0
entrée/sortie	3	bit de donnée 1
entrée/sortie	4	bit de donnée 2
entrée/sortie	5	bit de donnée 3
entrée/sortie	6	bit de donnée 4
entrée/sortie	7	bit de donnée 5
entrée/sortie	8	bit de donnée 6
entrée/sortie	9	bit de donnée 7
entrée	10	/acknowledge - acquittement
entrée/sortie	11	busy : imprimante occupée
entrée/sortie	12	paper out : papier manquant
entrée/sortie	13	on line : imprimante sélectionnée
	14	+ 5 volts
	15	inutilisé

sortie	16	reset
	17- 25	GND masse de référence

Sur l'AMIGA 1000, certaines broches sont disposées d'une autre façon :

14- 22	GND
23	+ 5 volts
24	inutilisé
25	reset

Le connecteur centronics de l'AMIGA réjouira les passionnés. Il est compatible PC, et les imprimantes à cette norme peuvent lui être directement raccordées. Ceci ne correspond en fait qu'aux modèles A500 et A2000, le port centronics de l'AMIGA 1000 n'obéissant pas à ce standard. En effet, à la place du connecteur DSUB, on trouve une prise mâle et la broche 23 est soumise à une tension de 5 volts. Ce signal serait relié à la masse (GND) si on branchait le câble d'une imprimante, et entraînerait automatiquement un court-circuit. Il est cependant possible de contourner ce problème en fabriquant un câble approprié.

Les broches du port Centronics sont tous reliés aux signaux de port du CIA (et ce, jusqu'aux signaux RESET et + 5 volts). Les attributions correctes sont les suivantes :

Centronics N° de broche	Fonction	CIA	Broche	Descriptif
1	Donnée valide	A	18	PC
2	bit de donnée 0	A	10	PB0
3	bit de donnée 1	A	11	PB1
4	bit de donnée 2	A	12	PB2
5	bit de donnée 3	A	13	PB3
6	bit de donnée 4	A	14	PB4
7	bit de donnée 5	A	15	PB5
8	bit de donnée 6	A	16	PB6
9	bit de donnée 7	A	17	PB7
10	Acknowledge	A	24	Flag
11	busy	B	2	PA0
			et 39	SP
12	paper out	B	3	PA1
			et 40	CNT
13	select	B	4	PA2

Le connecteur centronics est un connecteur parallèle. L'octet donnée se trouve sur les 8 signaux de donnée. Si un octet valide arrive sur le port, le signal STROBE est mis à 0 pendant 1,4 micro seconde, signalant ainsi à l'imprimante qu'un octet valide est prêt à être transmis. L'imprimante signale alors qu'elle accepte les données en mettant le signal Acknowledge pendant 1 micro seconde à 0. Puis l'ordinateur remet le prochain octet sur le bus.

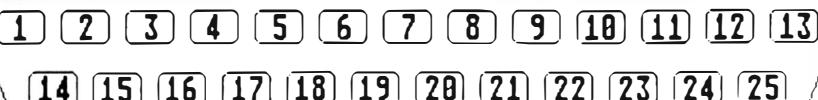
L'imprimante signale au moyen de busy qu'elle est occupée et qu'elle ne peut prendre en charge les données suivantes. Ceci peut arriver, lorsque, par exemple, le buffer de l'imprimante est plein. L'ordinateur attend alors, avant de reprendre le transfert, que le signal busy soit à nouveau actif. Avec le signal paper out, l'imprimante signale qu'elle n'est plus approvisionnée en papier. Le signal de sélection est activé à partir de l'imprimante. Il signale si l'imprimante est sélectionnée (on line) ou non (off line).

Le port centronics se prête parfaitement aux branchements d'extensions comme, par exemple, un digitaliseur de sons, les signaux se programmant facilement aussi bien en mode entrée qu'en mode sortie.

1.3.4. Le connecteur série

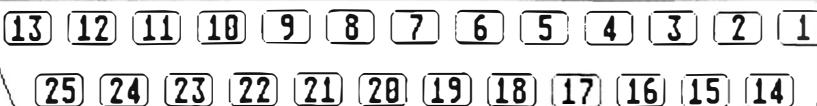
Le brochage de l'interface RS232

A500 & A2000



Connecteur 25 broches D-SUB

A1000



Interface 25 broches D-SUB

Figure 1 - 16

	1	GND	(FRAME GROUND) masse de protection
sortie	2	TXD	(TRANSMIT DATA) donnée transmise
entrée	3	RXD	(RECEIVE DATA) donnée reçue
sortie	4	RTS	(REQUEST TO SEND) demande d'émission
entrée	5	CTS	(CLEAR TO SEND) prêt à émettre
entrée	6	DSR	(DATA SET READY) donnée prête à envoyer
	7	GND	masse de référence
entrée	8	CD	(CARRIER DETECT) modem détecté
	9	+ 12 volts	
	10	- 12 volts	
sortie	11	AUDOUT	sortie audio à recevoir
entrée	22	RI	(RING INDICATOR)
	23	inutilisé	
	24	inutilisé	
	25	inutilisé	

Certains signaux sont disposés d'une autre façon sur l'AMIGA 1000 :

	9	inutilisé	
	10	inutilisé	
	11	inutilisé	
	12	inutilisé	
	13	inutilisé	
	14	- 5 volts	
sortie	15	AUDOUT	sortie audio
entrée	16	AUDIN	entrée audio
sortie	17	EB	horloge tampon (716 KHZ)
entrée	18	/INT2	entrée d'interruption de niveau 2
	19	inutilisé	
sortie	20	DTR	(Data Terminal Ready) terminal prêt à recevoir
	21	+ 5 volts	
	22	inutilisé	
	23	+ 12 volts	
sortie	24	MCLK	horloge de transmission à 3,58 MHZ
sortie	25	/MERS	RESET

Le connecteur série possède tous les signaux RS232. De plus, sur ce connecteur, on trouve plusieurs signaux qui n'ont aucun rapport avec la communication série.

Seuls les signaux TXD, RXD, DSR, CTS, DTR, RTS et CD assurent la communication RS232. TXD et RXD sont les signaux de données séries, TXD étant la sortie série, RXD l'entrée série. Ils sont reliés aux signaux correspondant de PAULA. Le signal DTR indique aux périphériques raccordés, que le connecteur série de l'AMIGA est en activité. Le signal DSR, au contraire, signale aux périphériques de l'AMIGA, que le connecteur série est prêt à être activé. Le signal RTS indique aux périphériques que l'AMIGA veut envoyer des données séries sur le RS232 et qu'il est prêt à les transmettre (signal CTS). Le signal CD est seulement utilisé par un modem, celui-ci indique qu'il emploie une

certaine fréquence de transfert. Les cinq signaux de gestion RS232 sont reliés avec le CIA-B(PA3- PA7) ; DSR-PA3 ; CTS-PA4 ; CD-PA5 ; RTS-PA6 ; DTR-PA7.

Le signal RI est relié au signal SEL du connecteur centronics, par l'intermédiaire d'un transistor.

Les signaux RS232 ne sont pas reliés directement aux circuits spécialisés, mais dirigés sur un pilote RS232. On pourra ainsi relier ce connecteur, par l'intermédiaire d'un câble approprié à des terminaux et modems. Le transformateur de niveau RS232 de type 1488 sera utilisé comme conducteur de sortie, avec une tension de +12 à -5 volts. Les circuits de type 1489 A seront utilisés comme tampon d'entrée, les entrées acceptant alors des tensions entre -12 et +0,5 volts (état Low) et des tensions de 3 à 25 volts (état High).

Les conventions retenues pour le connecteur RS232, sont que les signaux de gestion soient actifs à l'état high, alors qu'au contraire les signaux RXD et TXD sont mis à 1 suivant un niveau négatif. Etant donné que le conducteur de sortie est interverti, les bits de port correspondants du CIA-B seront low-active, c'est-à-dire qu'un bit du CIA-B d'une valeur 0 met le signal de gestion RS232 correspondant sur high. Cela se produit aussi en mode entrée.

Le signal audout est relié au canal audio gauche et pourvu d'une résistance de sortie de 1 Kohm. Le signal audin possède une résistance de 47 ohm et est directement relié au signal audr de PAULA.

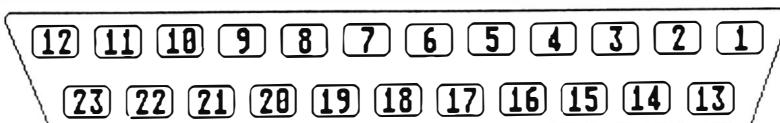
Le signal audio, qui passe dans l'AMIGA via la broche audin, accède, par le filtre passe bas et le canal audio droit de PAULA, à la sortie audio droite.

Le signal INT2 est directement relié à l'entrée INT2 de PAULA et peut libérer une interruption du processeur de niveau 2, lorsque le bit masque correspondant est activé. Le signal E est relié à l'horloge E du processeur via un tampon.

Une fréquence de 3,58 MHERTZ est appliquée au signal MCLK, ce signal d'horloge étant identique et en phase avec l'horloge du connecteur RGB et les deux fréquences des circuits spécialisés.

1.3.5. Connecteur disquette externe

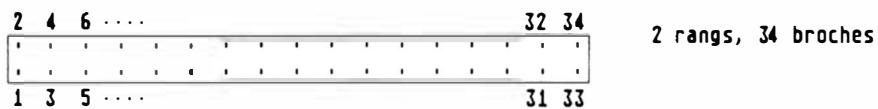
Le brochage de l'interface des lecteurs de disquettes externes.



Interface 23 broches D-SUB

Figure 1 - 17

Entrée entrée	1	/RDY	disque prêt
	2	/DKRD	lecture données disquette
	3	GND	
	4	GND	
	5	GND	
	6	GND	
	7	GND	
sortie	8	/MTRX	moteur on/off
sortie	9	/SEL	sélection disque2 off
sortie	10	/DRES	floppyreset (arrêt moteur)
entrée	11	/CHNG	disque changé
	12	+5 volts	
sortie	13	/SIDE	sélection de la face de la disquette
entrée	14	/WPRO	protection en écriture
entrée	15	/TK0	piste 0
sortie	16	/DKWE	autorisation écriture
sortie	17	/DKWD	écriture donnée
sortie	18	/STEP	déplacement de la tête pas à pas
sortie	19	/DIR	direction déplacement de la tête (0 = intérieur)
sortie	20	/SEL3	sélection disque DF3
sortie	21	/SEL1	sélection disque DF1
entrée	22	/INDEX	index début de cylindre
	23	+ 12 volts	

Le brochage du connecteur interne Floppy*Figure 1 - 18*

Toutes les broches impaires sont de type GND.

2	/CHNG	4	/INUSE
6	inutilisé	8	INDEX
10	SEL0	12	inutilisé
14	inutilisé	16	/MTR0
18	DIR	20	/STEP
22	/DKWD	24	/DKWE
26	/TK0	28	/WPRD
30	/DKRD	32	/SIDE
34	/RDY		

Boîtier d'alimentation pour lecteur de disquette interne :

- 1 + 5 volts
- 2 GND
- 3 GND
- 4 + 12 volts

Le branchement du lecteur de disquettes de l'AMIGA est compatible avec le bus SHUGART. Il rend possible la connexion de 4 lecteurs de disquettes de ce type. La sélection du lecteur se fait au moyen des signaux SEL x,x désignant le numéro de l'unité. Etant donné que l'AMIGA possède déjà son propre lecteur, il n'autorise que le branchement de 3 unités externes sélectionnées grâce à SEL1, SEL2 et SEL3.

Le signal SEL0 est relié au connecteur interne du lecteur de disquette.

Voici les différentes fonctions des signaux du bus SHUGART de l'AMIGA :

SEL X

Avec ce signal on sélectionne une des quatre unités de disquette possibles. Lorsque ce signal est activé, tous les autres le sont aussi, à part MTRX et DRES.

MTRX

Ce signal active les moteurs de toutes les unités raccordées. La gestion de 4 unités n'est pas simple. C'est pourquoi on a prévu, pour chaque lecteur de l'AMIGA, un Flip-Flop (un Flip-Flop est un montage électronique, dont le rôle est de mettre en mémoire un bit de donnée). Lorsque le signal SEL du lecteur concerné se met à Low, le Flip-Flop de ce lecteur prend en charge la valeur du signal MTRX. La sortie du montage Flip-Flop est reliée au signal MTR du lecteur. Ainsi les lecteurs de disquettes peuvent être mis en marche ou arrêtés, indépendamment les uns des autres. Si on met, par exemple, le signal SEL0 sur Low pendant que le signal MTRX se trouve à 0, le moteur du lecteur interne s'enclenchera. Ce Flip-Flop se trouve sur le montage du lecteur interne. Pour chaque lecteur externe, il est nécessaire d'un rajouter un. Sur le deuxième lecteur on l'a installé sur un petit adaptateur.

RDY

Lorsque le signal MTR du lecteur correspondant est à 0, le signal RDY indique à l'AMIGA, que le moteur du lecteur a atteint son nombre de tour nominal, et que l'unité est prête pour les accès lecture comme pour les accès écriture. Si le signal MTR est à 1, le moteur du lecteur étant arrêté, il y a un mode d'identification spécial.

DRES

Le signal DRES (DRIVE RESET) est relié au signal RESET de l'AMIGA et remet dans sa position d'origine le Flip-Flop moteur, c'est-à-dire que tous les moteurs seront arrêtés.

DKRD

Les données disquette du lecteur sélectionné (SELX) accèdent à la broche DKRD (DISK READ DATA). Cette dernière est reliée à la broche DKRD de PAULA.

DKWD (DISK WRITE DATA)

Les données de PAULA sont transmises au lecteur de disquette, afin d'y être écrites par le biais du signal DKWD.

DKWE

Le signal DKWE (Disk Write Enable) permet de passer du mode lecture au mode écriture. Si le signal est High, les données seront lues sur la disquette, alors que si le signal est Low, les données seront écrites.

SIDE

Le signal SIDE permet la sélection de la face de disquette qui sera lue ou écrite.

S'il est high, ce sera la face 0, c'est-à-dire la tête de lecture sous le disque qui sera activée. A l'inverse, si le signal est low, c'est la face supérieure qui sera sélectionnée.

WPRO

Le signal WPRO (Write protect) indique à l'AMIGA si la disquette insérée est protégée (WPRO=0) ou non en écriture.

STEP

Un état positif du signal STEP (celui-ci variant de l'état high vers low), déplace la tête de lecture/écriture du lecteur d'une piste à l'autre, suivant l'état du signal DIR. Avant que le signal SELX du lecteur activé ne soit remis sur high, le signal STEP doit dans tous les cas être mis à 1, sinon il risque d'y avoir des problèmes, dans la reconnaissance de la disquette, lors d'un changement.

DIR

Le signal DIR (Direction) détermine la direction de la tête, qui sera déplacée après une impulsion du signal STEP. Low signifie qu'il y aura déplacement vers le centre de la disquette, alors que high engendre un déplacement vers le bord de la disquette, la piste 0 étant la première piste d'une disquette.

TK0

Le signal TK0 (TRACK 0) est sur Low lorsque la tête de lecture/écriture se trouve sur la piste 0. Il y a alors la possibilité de placer la tête à un endroit déterminé.

INDEX

Le signal INDEX est une impulsion courte qui est fournie par le lecteur à chaque début de cylindre (chaque "tour" de disquette) et toujours entre le début et la fin d'une piste.

CHWG

Le lecteur de disquette indique, par le biais de ce signal, un changement de disquette. Aussitôt que la disquette sera sortie du lecteur, le signal CHWG sera mis à 0. Ce dernier restera en cet état tant que l'ordinateur n'aura pas libéré une impulsion STEP. Si à ce moment on introduit une disquette, le signal CHWG se remettra à 1. Sinon il restera sur 0 et l'ordinateur sera obligé de libérer des impulsions périodiquement afin de savoir si une disquette se trouve ou non dans le lecteur. Ces impulsions périodiques sont à l'origine du bruit désagréable que l'on peut entendre lorsque l'unité est vide.

INUSE

Ce signal n'existe que pour le lecteur interne. Lorsqu'on le met à 0, le lecteur allume une diode lumineuse. En temps normal, ce signal est relié au signal MTR.

Afin de savoir si le lecteur est connecté au bus, il existe un mode spécial d'identification. Le lecteur lira un long mot de données série de 32 bits.

Afin de démarrer cette identification, le signal MTR du lecteur concerné devra être rapidement activé, puis désactivé. De cette façon, le registre à décalage du lecteur sera remis dans son état de base. On pourra après cela lire chaque bit de donnée, tout en mettant le signal SELX sur Low, la valeur du signal RDY étant lue comme bit de données, puis remettre le signal SELX sur high. On répétera ce processus 32 fois. Le premier bit sera le MSB (Most Significant Bit/bit de poids fort) du mot de donnée. Etant donné que le signal est Low-Active, les bits de données doivent être intervertis.

Voici les définitions établies pour le lecteur de disquettes externe :

\$0000 0000	pas de lecteur raccordé (00)
\$FFFF FFFF	lecteur standard format 3 pouces 1/2 (11)
\$5555 5555	lecteur format 5 pouces 1/4, 2*40 pistes (01)

Comme on peut le remarquer, il n'y a que peu d'identifications différentes et il suffit de lire les deux premiers bits. Les valeurs entre parenthèses donnent la combinaison des deux bits.

Comme cela a été mentionné plus haut, tous les signaux sauf DRES n'auraient qu'une influence sur le lecteur sélectionné par SELX. Initialement le signal MTRX œuvrait indépendamment du signal SELX, mais les concepteurs de l'AMIGA l'ont modifié en rajoutant le Flip-Flop moteur.

Tous les signaux du bus SHUGART sont Low-actives, étant donné que les sorties de l'AMIGA, comme le lecteur de disquette, sont pourvues de conducteurs OPEN-COLLECTOR de type 7407.

Les quatre entrées CHWG, WPRO, TK0 et RDY sont reliées directement aux signaux PA4-PA7 du CIA-A. Les huit sorties STEP, DIR, SIDE, SEL0, SEL1, SEL2, SEL3, MTR proviennent du CIA-B, PB0-7 et sont reliées aux connecteurs disquette interne et externe sur le conducteur mentionné plus haut. Etant donné que ce dernier n'intervertis rien, les bits du CIA sont toujours intervertis. Les signaux DKRD, DKWD et DKWE proviennent de PAULA. Les branchements entre le lecteur interne et le lecteur externe sont identiques, la seule différence concerne les signaux MTRX et SELX.

Le lecteur interne est relié à SEL0. Son signal MTR passe par le Flip-Flop moteur.

Sur le connecteur externe on trouve le signal MTRX directement relié au CIA et aux 3 signaux SEL.

Branchements d'un lecteur externe à l'AMIGA

L'utilisation d'un seul lecteur est possible pendant un bon moment. Mais il vient un temps où une deuxième unité devient indispensable.

Le lecteur A1010 double face, au format 3 1/2 pouces, utilisé par l'AMIGA, nous est proposé à un tel prix, qu'il est préférable, actuellement, d'en "bidouiller" un soi-même. Que faut-il faire ?

La fiche de branchements d'un lecteur standard 3 1/2 pouces, comme par exemple le NEC FD 1035 ou FD 1036, est identique à celle du lecteur interne à 34 broches, y compris les fiches d'alimentation. Afin de raccorder un lecteur du type FD1035, il suffit de rajouter un Flip-Flop moteur. La figure suivante reproduit le schéma de montage.

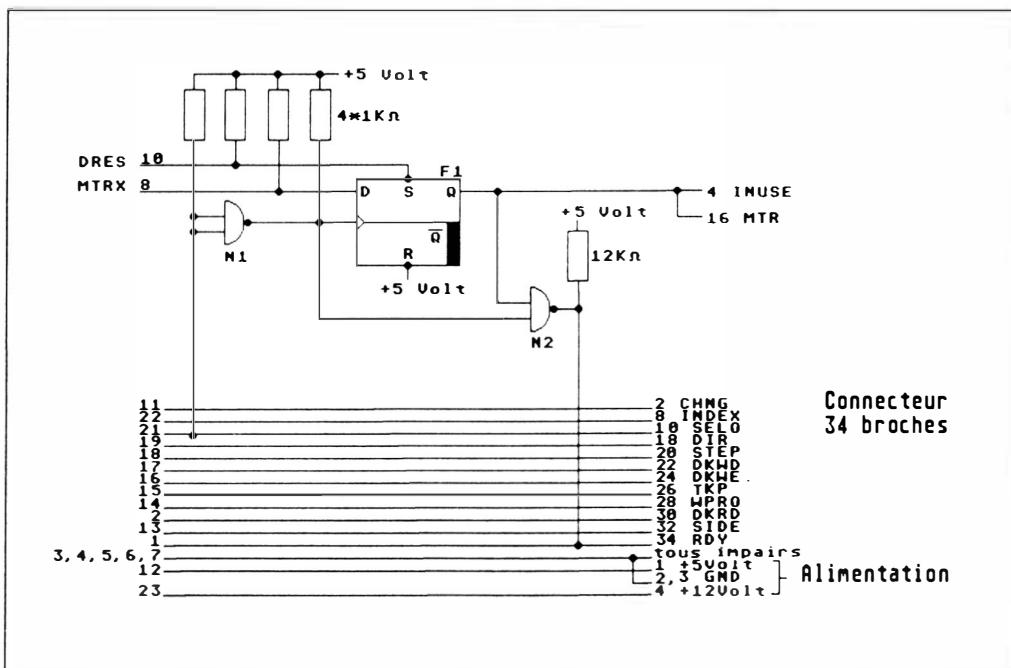


Figure 1 - 19

On remarque que le Flip-Flop caractéristique met en mémoire le signal sur MTRX au moyen de F1, lorsque le signal passe de High à Low. Etant donné que le Flip-Flop met en mémoire la valeur à son entrée donnée qu'avec le niveau positif de l'impulsion d'horloge, on doit intervertir SEL1. C'est le rôle de la porte NAND N1. La sortie Q est reliée directement à l'entrée MTR du deuxième lecteur de disquette.

Le deuxième port NAND n'a aucun rapport avec la gestion du moteur. Il sert au mode d'identification, la majorité des lecteurs du commerce n'ayant besoin d'aucune aide.

Lorsque le moteur est coupé, le signal SEL est actif, c'est-à-dire à 0, et le port met le signal RDY sur Low. L'AMIGA reconnaîtra ainsi la deuxième unité avec le terme DF1.

Etant donné que seul la moitié des deux circuits est nécessaire, cela suffit pour un lecteur supplémentaire. Les entrées de N1 doivent être reliées à SEL2 (broche 9 du lecteur externe).

Afin que le signal CHWG puisse fonctionner sans problème, certains lecteurs doivent être munis d'un JUMPER. Par exemple le lecteur FD 1035 de NEC doit être court-circuité par un JUMPER J1 caractéristique. Pour en savoir davantage, référez-vous au manuel du lecteur de disquette.

1.3.6. Le connecteur souris-joystick

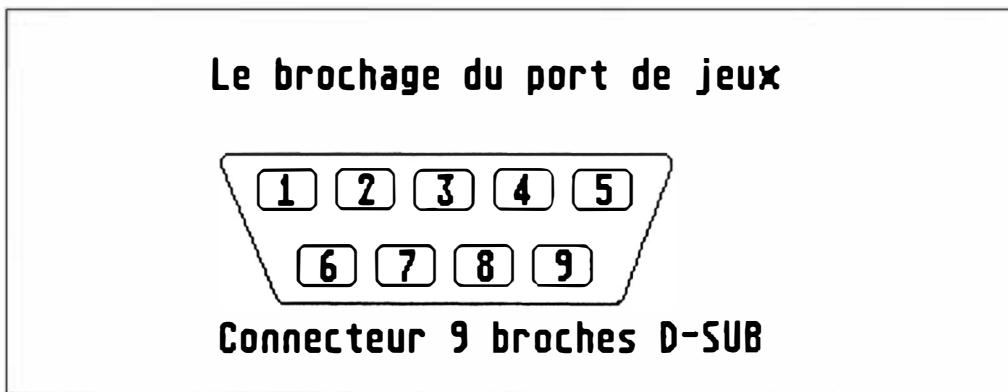


Figure 1 - 20

Utilisation

	Souris	Joystick 1	Joystick 2	Light Pen
entrée 1	impulsion V	avant	inutilisé	inutilisé
entrée 2	impulsion H	arrière	inutilisé	inutilisé
entrée 3	impulsion VQ	gauche	bouton gauche	inutilisé
entrée 4	impulsion HQ	droite	bouton droit	inutilisé
E/S 5	bouton 2	inutilisé	potentiomètre droit	crayon touche l'écran
E/S 6	bouton 1	bouton feu	inutilisé	signal LP

	Souris	Joystick 1	Joystick 2	Light Pen
7	+ 5 volts	+ 5 volts	+ 5 volts	+ 5 volts
8	GND	GND	GND	GND
E/S	9 bouton 3	inutilisé	potentiomètre vertical	inutilisé

Ces connecteurs sont des entrées d'outils de commande, tels la souris, le joystick, le trackball, la manette ou le stylo lumineux. On en trouve deux :

Le connecteur gauche (gameport 0) et le connecteur droit (gameport 1). La structure des deux connecteurs est identique, la seule différence réside dans l'utilisation du signal LP par le gameport 0. Ces connecteurs sont reliés, de façon interne, au CIA, à AGNUS, à DENISE et à PAULA.

Voici la répartition des liaisons entre les connecteurs et les circuits spécialisés.

GAMEPORT 0

Broche N°	Circuit	Broche
1	Denise	M0V
2	Denise	M0H
3	Denise	M1V
4	Denise	M1H
5	Paula	POY
6	CIA-A	PA-6
9	Paula	POX

GAMEPORT 1

Broche N°	Circuit	Broche
1	Denise	M0V
2	Denise	M0H
3	Denise	M1V
4	Denise	M1H
5	Paula	P1Y

Broche N°	Circuit	Broche
6	CIA-A	PA7
9	Paula	P1X

La structure des différents outils de commande a été choisie de telle façon que tous les joysticks, souris, stylo lumineux du marché puissent convenir. Il est ainsi possible d'employer, par exemple, le stylo lumineux utilisé sur le C64. Le signal LP (light pen) est engendré par le stylo, lorsque sa pointe est au contact de l'écran.

Tous les signaux caractérisés par bouton, ainsi que ceux des quatre directions du joystick, sont Low-actives. Dans les différents outils de commande, se trouvent des interrupteurs qui sont reliés aux entrées correspondantes et à la masse (GND). Un signal High sur l'entrée signifie interrupteur ouvert, le signal Low correspondant à l'interrupteur fermé.

Les entrées analogiques raccordées à P0X, P0Y, P1X et P1Y acceptent des résistances variables (potentiomètre) d'une valeur de 470 Kohm. Elles sont installées entre le +5 volts et l'entrée correspondante.

Les deux boutons de tir, qui sont reliés au CIA-A, peuvent être programmés en mode sortie. Il faudra toutefois être attentif lors d'un accès écriture sur le registre port, à ne pas surcharger le dernier bit de port, afin de ne pas "planter" le système (PA0 : OVL).

La question des signaux des gameports sera vue plus en détail dans le chapitre Programmation des circuits spécialisés.

L'alimentation 5 volts des deux connecteurs n'est pas reliée directement à la tension de service de l'AMIGA. On a instauré un interrupteur frontière d'alimentation entre les deux.

Il sépare en fait le courant permanent de court-circuit de 400 mA des pointes de tension de 700 mA. Cette entrée est donc complètement protégée des court-circuits. Comme la tension aux deux broches +5 volts ne doit pas tomber brusquement, la consommation de courant ne doit pas dépasser 250 mA à l'ensemble des deux connecteurs.

Malheureusement, on a supprimé ces interrupteurs de protection sur les modèles A 500 et A 2000.

1.3.7. Connecteur d'extension

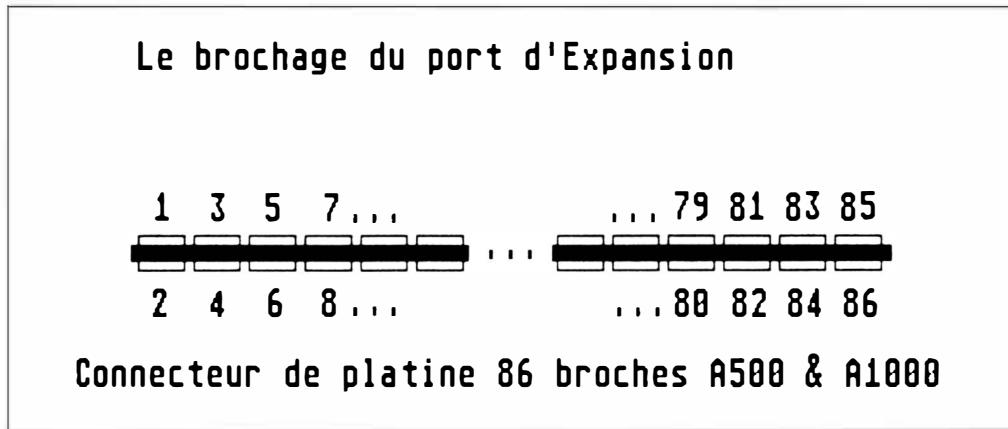


Figure 1 - 21

1	GND	2	GND
3	GND	4	GND
5	+5 Volts	6	+5 Volts
7	extension	8	-5 Volts
9	extension (28 MHz)	10	+12 Volts
11	extension	12	Config
13	GND	14	/C3
15	CDAC	16	/C1
17	/OVR	18	XRDY
19	/INT2	20	/PALOPE
21	A5	22	/INT6
23	A6	24	A4
25	GND	26	A3
27	A2	28	A7
29	A1	30	A8
31	FC0	32	A9
33	FC1	34	A10
35	FC2	36	A11
37	GND	38	A12
39	A13	40	/IPL0
41	A14	42	/IPL1
43	A15	44	/IPL2
45	A16	46	/BERR
47	A17	48	/VPA
49	GND	50	E
51	/VMA	52	A18
53	/RES	54	A19
55	/HLT	56	A20
57	A22	58	A21
59	A23	60	/BR
61	GND	62	/BGACK
63	PD15	64	/BG

65	PD14	66	/DTACK
67	PD13	68	/PRW
69	PD12	70	/LDS
71	PD11	72	/UDS
73	GND	74	/AS
75	PDO	76	PD10
77	PD1	78	PD9
79	PD2	80	PD8
81	PD3	82	PD7
83	PD4	84	PD6
85	GND	86	PD5

On reconnaît tous les signaux de gestion importants et de bus du système de l'AMIGA sur le connecteur d'extension. On peut y raccorder des extensions mémoires, de nouveaux processeurs et autres. Ce connecteur, pour le modèle 1000, est placé du côté des deux gameports et caché sous un capot en plastique. En ce qui concerne l'AMIGA 500, ce connecteur est placé dans un boîtier sous l'appareil. Il se présente sous la forme d'un montage de 86 broches. L'écart entre les broches est de 0,1 pouce, ce qui correspond à 2,54 mm. Une fiche pouvant s'y raccorder peut se trouver assez facilement dans le commerce.

Dans l'AMIGA 2000, on observe 2 branchements de bus différents, composés d'un connecteur MMU, correspondant à ce qui a été vu plus haut, et de 5 connecteurs à 100 broches, aussi appelés Bus Zorro. Ces six emplacements se trouvent sur le montage général, dans le boîtier du 2000. Ces connecteurs de 86 et 100 broches sont conçus pour des cartes d'extension spéciales.

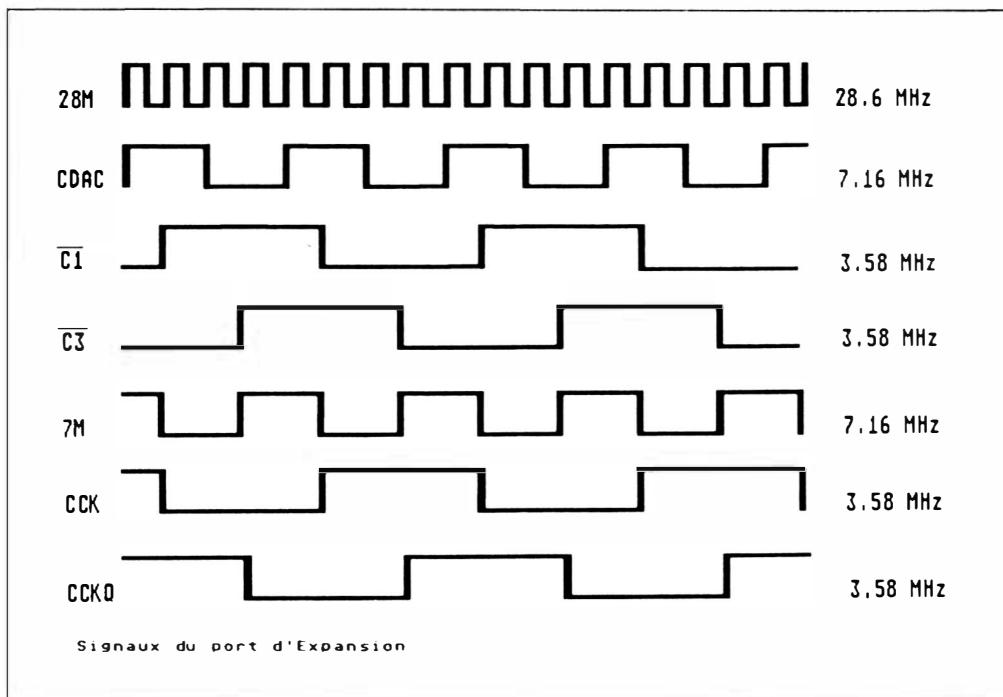
La majorité des signaux du connecteur d'extension sont reliés directement aux signaux correspondants du 68000. Les fonctions des signaux suivants sont décrites au chapitre 1.2.1.

A0-A23	: bus d'adresse
PD0-PD15	: bus de données processeur
IPL0-IPL2	: signaux d'interruption processeur
FC0-FC2	: signaux des codes fonction du 68000
AS, UDS, LDS, PRW, DYACK, VMA, VPA	: signaux de gestion des bus
RES, HLT, BERR, BG, BGACK, BR, E	: signaux de gestion du 68000

Les autres signaux ont les fonctions suivantes :

INT2 et INT 6

Ces deux signaux sont reliés aux broches du même nom de PAULA. Par leur entremise, une interruption de niveau 2 ou de niveau 6 peut être appelée.

CDAC, C1, C3 et 28M (A2000)**Figure 1 - 22**

La fréquence et la position de phase des différents signaux d'horloge se déduisent facilement du schéma. Sur le modèle A2000, la haute impulsion d'horloge, d'une fréquence de 28,64 MHZ, s'applique aussi au connecteur d'extension. Les signaux 7M, CCK et CCKQ ne se trouvent évidemment pas sur le connecteur, 7M étant l'horloge du 68000, CCK et CCKQ étant reliés aux circuits spécialisés.

XRDY, OVR et PALOPE

Ces signaux servent à la configuration automatique des cartes d'extension. Leur fonction réelle n'a pu, jusqu'ici, être expérimentée. Les signaux caractérisés par "extension" n'ont jamais été référencés. Les futures extensions de l'AMIGA sont tous droits réservés. Avec l'AMIGA 2000, ils ont déjà été, en partie, utilisés (cf. signal d'horloge 28 MHZ).

1.3.8. Alimentation des connecteurs

A chaque connecteur de l'AMIGA s'applique une ou plusieurs des trois tensions de service. Il est aussi possible d'alimenter plusieurs périphériques au travers des connecteurs correspondants.

On devra toutefois être attentif à la charge maximale admise par le branchement.

Voici le tableau des charges maximales recommandées par Commodore pour l'AMIGA 1000.

Connecteur	+ 5 volts	+ 12 volts	- 5 volts
Modulateur TV	-	60 mA	-
RGB	300 mA	175 mA	50 mA
RS232	100mA	50 mA	50 mA
Disk ext.	270 mA	160 mA	-
Centronics	100 mA	-	-
Extension	1 000 mA	50 mA	50 mA
Gameport 0	125 mA	-	-
Gameport 1	125 mA	-	-

Ce tableau n'est à prendre en compte que dans ses grandes lignes. Ces valeurs ne sont valables que lorsque tous les connecteurs sont en charge. Si, par exemple, le connecteur d'extension est inoccupé, les 1 000 mA du + 5 volts sont disponibles pour un autre connecteur. Si certains connecteurs sont libres dans une configuration système déterminée, les autres sorties se verront attribuer une charge maximale plus importante. Evidemment on peut aussi employer la méthode de forcer, tant qu'une carte d'extension est connectée, jusqu'à ce que le bloc d'alimentation saute.

Cette façon de court-circuiter le système ne cause pas trop de dommages, mais les expériences sont à mener avec précaution. Lors d'un court-circuit, il peut s'écouler un courant de plus de 8 ampères.

Ce tableau n'est valable que pour l'AMIGA 1000. On ne peut pas s'y reporter pour l'AMIGA 500 ou 2000, étant donné que l'alimentation de ces ordinateurs est dimensionnée d'une autre façon. Pour l'A500, la charge admise de l'alimentation est plus faible. On est obligé d'utiliser un bloc d'alimentation supplémentaire pour toutes extensions un peu gourmandes en électricité.

L'alimentation de l'A2000 est plus importante que celle de l'AMIGA 1000. En effet cet ordinateur doit être capable de supporter plusieurs cartes d'extension, dont celles d'émulation IBM.

Attention :

L'AMIGA 500 diffère aussi du modèle 1000 par sa tension négative, celle-ci étant de -12 volts sur l'AMIGA 500 et de -5 volts sur le modèle 1000.

1.4. Le clavier

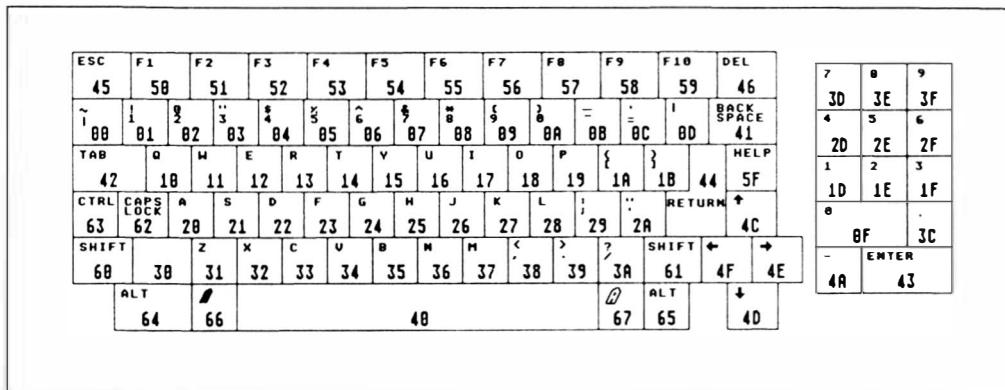


Figure 1 - 23 Clavier ASCII Américain

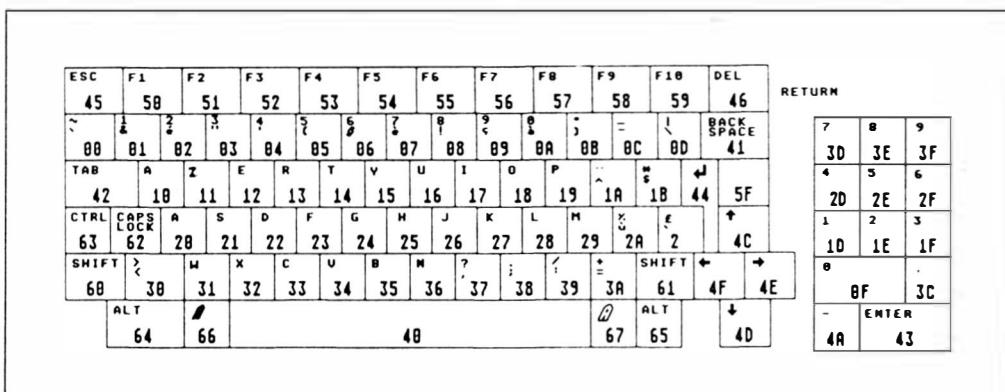


Figure 1 - 24 Clavier français AZERTY

Le clavier de l'AMIGA est dit intelligent, du fait de la présence d'un microprocesseur qui prend en charge le traitement des touches, ne délivrant à l'AMIGA que les codes claviers effectifs. Il existe plusieurs réalisations et versions de clavier, mais elles ne

diffèrent entre elles que par le rajout de certaines touches, donc de certains codes. Les schémas montrent la version américaine et la version française, avec leurs codes touches correspondants. Comme on peut le remarquer, les codes ne correspondent pas au standard ASCII. Le clavier délivre exclusivement des codes clés RAW, qui seront traduits en codes ASCII, par le système d'exploitation, à l'aide d'une table de transcription.

On remarque toutefois une organisation des codes clés RAW :

- \$00-\$3F** ce sont les codes des lettres de l'alphabet, des nombres et autres touches à caractères. Leur répartition correspond à l'ordre des touches du clavier.
- \$40-\$4F** codes des touches RETURN, TAB, BACKSPACE.
- \$50-\$5F** codes des touches de fonctions et HELP.
- \$60-\$67** codes des touches d'option des différents niveaux du clavier (Shift, Amiga, Alternate et Control).

Le processeur clavier peut aussi différencier l'état touche enfoncée de l'état touche relâchée. Les codes clavier s'étaisent sur 7 bits (valeur de \$00 à \$7F), le huitième bit étant le drapeau KEY up/down. C'est ce bit qui sera utilisé par l'ordinateur pour connaître l'état de la touche. Si le bit 8 vaut 0, c'est que la touche est enfoncée (Key down). Si ce même bit est à 1, c'est que la touche a été relâchée (Key up). On peut aussi utiliser le clavier en appuyant sur plusieurs touches en même temps, certains logiciels de musique notamment, exploitent cette possibilité afin de réaliser des accords.

La touche Caps Lock est une exception. En effet le clavier simule un interrupteur. Si on appuie sur cette touche, elle reste activée et la diode s'allume. Lorsqu'on appuie une deuxième fois, une fois la touche relâchée, la diode s'éteint. Cette touche retenue porte l'état du drapeau Key up/down.

Si on active CapsLock, la LED s'allume et le code touche (8 bits) est envoyé à l'ordinateur, afin de signaler que la touche est enfoncée. Si cette touche est laissée tranquille, aucun code Key up ne sera envoyé à l'ordinateur. Lorsqu'on enfoncera la touche à nouveau, le code Key up sera envoyé (8ème bit activé) et la LED s'éteindra.

1.4.1. Schéma électronique du clavier

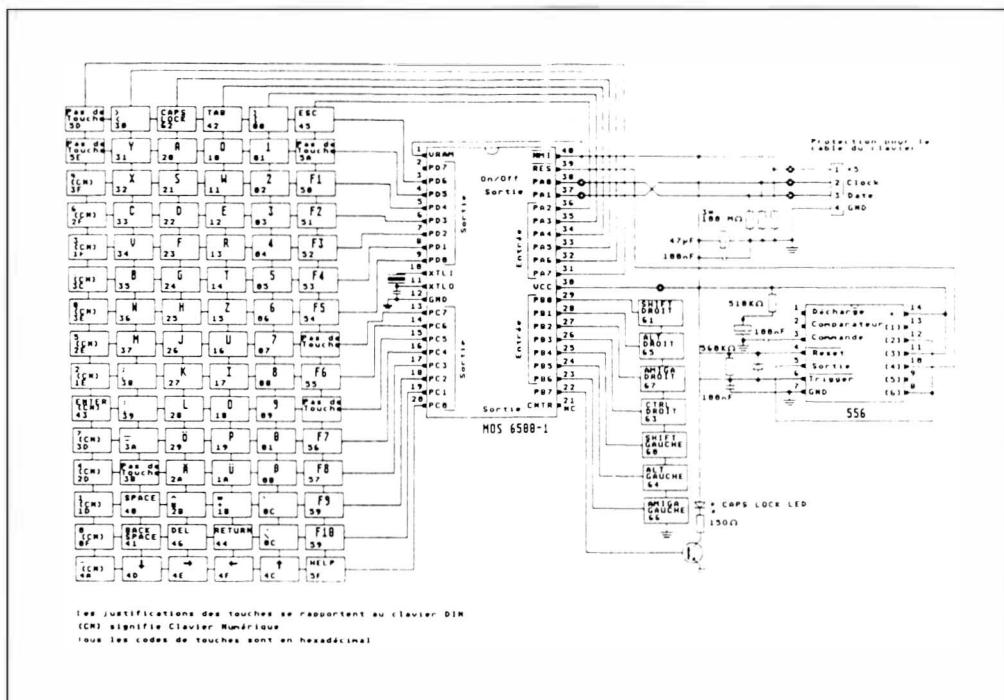


Figure 1 - 25

Le microprocesseur 6500/1 est le maître du montage clavier. Le 6500/1 est en fait un micro-ordinateur à lui tout seul et possède toutes les composantes nécessaires au travail d'un ordinateur. Le cœur du 6500/1 est en fait le microprocesseur de type 6502 (aha!). De plus, il possède 2 KO de ROM, contenant le programme directeur, 64 octets de RAM statique, 4 ports 8 bits directionnels, un compteur 16 bits avec une entrée directrice et, enfin, un générateur d'horloge.

Pour fonctionner, le 6500/1 n'a besoin que d'une alimentation de 5 volts et d'un quartz pour le signal d'horloge. Le clavier de l'AMIGA exploite le 6500/1 avec un quartz à 3 MHZ. Etant donné que la fréquence interne est divisée par 2, la fréquence d'horloge est de 1,5 MHZ.

Le deuxième circuit du montage clavier est une minuterie de précision du type 556. En fait, cette minuterie en comporte deux dans le même boîtier. Ce circuit engendre le signal RESET du 6500/1.

Les touches sont séparées en deux groupes. Les sept touches, Shift (gauche et droite), Alt (gauche et droite), Amiga (gauche et droite) et Control sont reliées directement aux premiers signaux de port PB.

Les autres touches sont ordonnées suivant une matrice de six colonnes sur 15 lignes. Les colonnes sont reliées du signal PA2 au signal PA7 du port A. Ces six signaux de port sont commutés en mode entrée. Les 15 lignes sont dirigées vers le Port C et D. La broche PD7, correspondant à une 16ème ligne, n'est plus connectée dans les dernières versions.

Lorsque le 6500/1 interroge le clavier, il met la rangée de chaque ligne à 0. Etant donné que les sorties des ports C et D sont de type OPEN COLLECTOR, sans résistance PULLUP intégrée, les sorties mises à 1 sont retenues comme étant inactives. Lorsque le processeur a mis une ligne à 0, il teste les 6 colonnes. Les six entrées colonnes sont pourvues de résistances PULLUP internes, afin que les touches enfoncées ne soient pas interprétées en tant que signal high.

Chaque touche enfoncée relie une colonne à une ligne. Si la touche, dans la rangée activée par le processeur, est enfoncée, les entrées de colonnes correspondantes seront mises à 0. Une fois que toutes les rangées seront activées, et que la colonne correspondante sera lue, le processeur connaîtra l'état de toutes les touches. Si celui-ci s'est modifié depuis le dernier test, il enverra les codes claviers correspondant à l'ordinateur.

1.4.2. Transfert des données

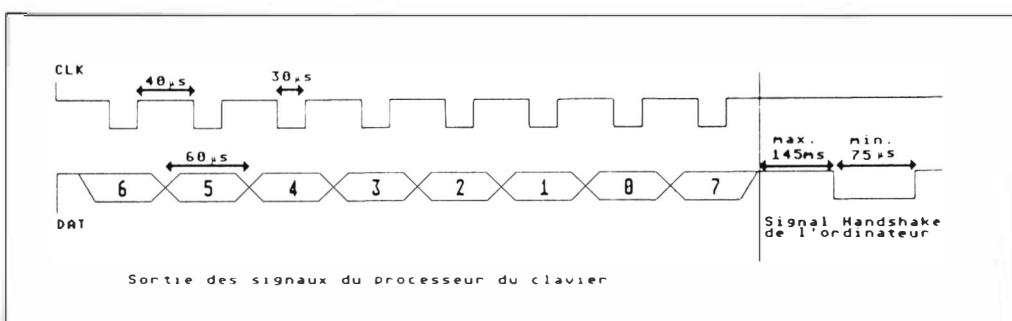


Figure 1 - 26

Le clavier est relié à l'Amiga avec un câble en spirale à 4 lignes. Deux des lignes servent simplement à alimenter l'électronique du clavier avec les 5 volts nécessaires. Le transfert des données s'effectue à l'aide des deux autres lignes. La première de ces lignes sert au signal des données KDAT, l'autre au signal d'horloge KCLK. Dans l'Amiga, les signaux KDAT et KCLK sont reliés respectivement au signal d'entrée SP et à la broche CNT du CIA-A.

Le transfert des données est unidirectionnel, du clavier vers l'Amiga. Le 6500/1 lâche les bits de données sur le signal KDAT, et les accompagne d'une longue impulsion LOW d'horloge (KCLK), d'une durée de 20 microsecondes. La durée entre chaque impulsion d'horloge est de 40 microsecondes. Le temps de transfert est donc de 60 microsecondes par caractère (40+20). Ceci correspond à 1 octet (8 bits) toutes les 480 microsecondes ou un débit de 1666 BAUD (bits/seconde).

Après l'émission du dernier bit, le clavier attend une impulsion HANDSHAKE de l'ordinateur. Dans ce but, l'Amiga met le signal KDAT sur LOW, pendant environ 75 microsecondes.

Le schéma explique bien le déroulement du transfert. On remarque que les données ne sont pas envoyées suivant l'ordre décroissant 7-6-5-4-3-2-1-0, mais après rotation vers la gauche d'une position de bit : 6-5-4-3-2-1-0-7. Par exemple, le code clavier de la lettre 'J', 00100110, sera envoyé, après rotation, sous la forme : 01001100. Le drapeau Key up/down sera toujours transmis en dernier.

Les signaux de données sont de type LOW-ACTIVE, c'est-à-dire que 0 reproduit l'état LOW, 1 reproduisant l'état HIGH.

Le registre à décalage du CIA de l'Amiga, prend en charge à chaque impulsion d'horloge, le bit se trouvant sur le signal SP. Après huit impulsions, le CIA possède un octet complet de données. Le CIA engendre alors une interruption de niveau 2, permettant au système d'exploitation de réaliser les opérations suivantes :

- ✓ lecture des registres de données séries,
- ✓ inversion et rotation des octets vers la droite, afin d'obtenir le code clavier valide,
- ✓ émission de l'impulsion HANDSHAKE,
- ✓ tâche interne suivant les codes reçus.

Synchronisation

Afin qu'un transfert de données puisse aboutir sans problème et sans erreur, le timing du transmetteur et celui du récepteur doivent concorder. La position des bits doit être la même, lors du transfert. Si une erreur se déclare, le clavier peut très bien envoyer une série de huit bits, alors que le port série du CIA n'est qu'au milieu du traitement de l'octet précédent. Une perte de synchronisation peut être consécutive à un arrêt de l'Amiga, ou à un branchement intempestif du clavier alors que l'Amiga est allumé. L'ordinateur n'a aucune possibilité de reconnaître une erreur de synchronisation. Cette tâche incombe, en fait, au clavier.

Après chaque transfert d'octets, le clavier attend au maximum 145 millisecondes le signal HANDSHAKE. S'il ne se passe rien durant ce cours moment, le processeur clavier décide qu'il y a erreur de transfert et introduit un mode spécial, avec lequel il cherchera à retrouver la synchronisation perdue. Cela consiste à émettre le signal KDAT à 1 en même temps qu'une impulsion d'horloge et d'attendre à nouveau pendant 145 millisecondes, le signal de synchronisation. Ceci sera répété jusqu'à ce que le signal HANDSHAKE soit reçu, signifiant que la synchronisation est à nouveau effective.

Lorsque l'octet donnée reçu par l'Amiga est incorrect, l'état des sept premiers bits est ignoré. Seul le dernier bit est reconnu comme étant à 1, le processeur clavier n'émettant que des 1 comme cela a été décrit plus haut. Etant donné que le dernier bit est un drapeau Key up/down, le code clavier est toujours un code Key up erroné, c'est-à-dire, un code touche relâchée. Si l'émission du bit incorrect avait été du type Key down, les possibilités de trouble de programmation auraient été plus fréquentes. C'est pourquoi il y a, avant chaque transfert, rotation d'un bit vers la gauche, afin que le drapeau Key up/down soit toujours envoyé en dernier.

Les codes spéciaux

Il existe encore quelques indications spéciales, qui, pendant le transfert, sont communiquées par le clavier de l'Amiga au moyen de codes spéciaux. En voici la liste :

Code	Signification
\$F9	le dernier code clavier était incorrect
\$FA	tampon clavier saturé
\$FC	test propre du clavier était incorrect
\$FD	début des touches enfoncées après mise en route
\$FE	fin des touches enfoncées après mise en route

\$F9

Le code \$F9 sera toujours envoyé par le clavier lors d'une perte de synchronisation, et lorsque celle-ci est rétablie. L'Amiga reconnaît alors qu'il y a eu émission incorrecte. Après ce code spécial, le clavier transmettra le code précédemment perdu.

\$FA

Le clavier dispose d'un tampon de 10 caractères. Lorsque ce dernier est saturé, il envoie le code \$FA, afin de signaler à l'ordinateur que des codes clavier risquent d'être perdus s'il ne le vide pas.

\$FC

Après la mise en route de l'ordinateur, le processeur clavier commence une série de tests propres. On le remarque à la courte activation de la LED de la touche CAPS LOCK. Si ces tests mettent à jour une erreur, le clavier, tout en se bloquant, envoie un code \$FC à l'Amiga, la LED restant allumée.

\$FD et \$FE

Lorsque le test de mise en route est terminé, le clavier commence le transfert de tous les codes des touches qui ont été activées depuis la mise en route de l'ordinateur. Avant le début de l'émission, le clavier place le code \$FD, le code \$FE étant placé en fin de transfert.

Si aucune touche n'a été activée, \$FD et \$FE seront envoyés l'un derrière l'autre.

Reset par le clavier

Le clavier de l'Amiga a la possibilité de libérer un signal RESET. Si on appuie simultanément sur les 2 touches AMIGA, ainsi que sur la touche CONTROL, le processeur clavier met le signal KCLK sur LOW, pendant environ 0,5 secondes. Ceci provoque un signal RESET, libérant une remise à zéro du processeur. Ce dernier se déclenche dès qu'une des trois touches est relâchée. On le remarque aussi au voyant clignotant de la touche CAPS LOCK (un fait intéressant est que le signal KCLK, qui libère le RESET, est relié au signal CNT du CIA. On pourra donc, avec la programmation appropriée, déclencher une remise à zéro).

1.5. La programmation du hardware

Dans le chapitre précédent, c'est la structure du hardware de l'Amiga qui vous a été présentée. Maintenant que les profondeurs de votre ordinateur préféré n'ont plus aucun secret pour vous, nous allons découvrir les charmes de la programmation, qui permet, notamment, la création des sons et des graphismes.

Les bases d'une programmation sans problème, à tous les niveaux de la machine, passe par une bonne connaissance, d'une part, de l'organisation de la mémoire, et d'autre part, des registres de chaque circuit spécialisé.

1.5.1. Organisation de la mémoire

Configuration de la mémoire

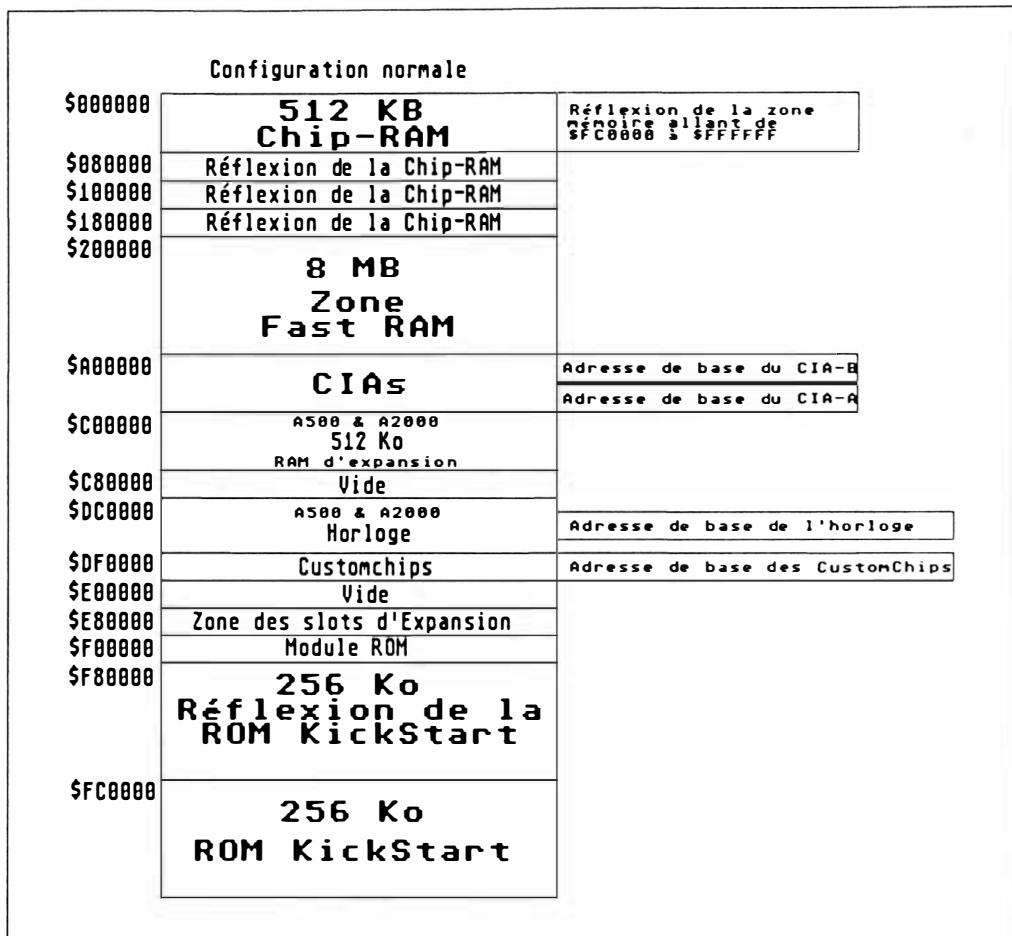


Figure 1 - 27

Ce graphique nous montre la configuration mémoire, telle qu'elle se présente au programmeur au démarrage. La zone adressable du 68000 se monte à 16 megaoctets (adresse 0 à \$FFFFFF). La raison de cette taille n'est pas une surprise, étant donné qu'une grande zone est inexploitée et que certains circuits apparaissent à différentes adresses. Mais il n'y a pas de raison d'être avare avec la mémoire utilisable, l'époque où les switchings entre les zones étaient nécessaires étant, avec le 68000, heureusement dépassée.

La RAM

La zone mémoire représentée par CHIP-RAM correspond à la mémoire accessible normalement sur Amiga. Si l'extension mémoire de l'A1000 est absente, elle n'atteindra que \$3FFFF. Etant donné que c'est la seule zone mémoire accessible par les trois circuits spécialisés, ces 512 Ko sont appelés CHIP-RAM.

Il est possible que le processeur soit ralenti par l'activité des trois circuits spécialisés, lors d'un accès mémoire à la CHIP-RAM. Si on veut l'éviter, on peut étendre la mémoire par adjonction de FAST-RAM. Celle-ci se placera à partir de l'adresse \$200000, et pourra atteindre 8 mégaoctets. Comme cette zone est exclusivement réservée au 68000, ce dernier pourra y accéder avec toute sa vitesse, d'où le nom FAST. Dans sa version de base, l'Amiga ne comprend pas de FAST-RAM.

La carte d'extension des modèles 500 et 2000 a comme zone d'adresse \$C00000 à \$C7FFFF. Elle peut se comporter aussi bien en tant que CHIP-RAM, qu'en tant que FAST-RAM. En effet, l'accès peut être aussi bien interdit aux circuits spécialisés pendant un moment, que les accès processeurs peuvent y être ralentis par ces mêmes circuits. Cet état n'est pas dû à la malice d'un des concepteurs, mais à la simplicité de la conception qui est en fait une extension très bon marché.

Le CIA

Les différents registres du CIA apparaissent à l'adresse \$A00000 et se terminent à l'adresse \$BFFFFF. Les précisions sur l'adressage du CIA se trouvent au chapitre 1.2.

Voici à nouveau le détail des registres et leur adresse respective :

CIA-A	CIA-B	NOM	FONCTION
\$BFE001	\$BFD000	PA	données port A
\$BFE101	\$BFD100	PB	données port B
\$BFE201	\$BFD200	DDRA	direction port A
\$BFE301	\$BFD300	DDRB	direction port B
\$BFE401	\$BFD400	TALO	minuterie A (octet bas)
\$BFE501	\$BFD500	TAHI	minuterie A (octet haut)
\$BFE601	\$BFD600	TBLO	minuterie B (octet bas)
\$BFE701	\$BFD700	TBHI	minuterie B (octet haut)
\$BFE801	\$BFD800	E.LSB	eventcounter (bits 0-7)

CIA-A	CIA-B	NOM	FONCTION
\$BFE901	\$BFD900	E.MID	eventcounter (bits 8-15)
\$BFEA01	\$BFDA00	E.MSB	eventcounter (bits 16-24)
\$BFE801	\$BFDB00	---	inutilisé
\$BFEC01	\$BFDC00	SP	données séries
\$BFED01	\$BFDD00	ICR	registre contrôle interruption
\$BFEE01	\$BFDE00	CRA	registre contrôle port A
\$BFFEF01	\$BFDFO0	CRB	registre contrôle port B

Les circuits spécialisés

Les différents registres des circuits spécialisés se trouvent dans une zone de 512 octets. Chaque registre ayant une largeur d'un mot, on les trouve donc à toutes les adresses paires.

L'adresses de base des zones registres se trouve à \$DFF000. L'adresse effective d'un registre se tient donc à \$DFF000 + adresse registre. La liste qui suit présente les noms et fonctions de chaque registre des circuits. Il est clair que la plupart des descriptions de registres ne sont pas connues, étant donné qu'il n'a jamais rien été dit sur la plupart des registres. Cette liste procure une vue d'ensemble et peut servir de compléments aux adresses registres.

Il existe 4 types de registres différents.

R (read)

Ces registres ne sont accessibles qu'en mode lecture.

W (write)

Ces registres ne sont accessibles qu'en mode écriture.

S (strobe)

Un accès à un tel registre libère une routine particulière du circuit correspondant. Ainsi, la valeur (le mot) se trouvant sur le bus de données et qui sera écrite dans un de ces registres n'est pas importante. Ce type de registre n'intéresse qu'AGNUS.

ER (early read)

Un registre signalé par **EARLY READ** est un registre d'émission DMA. Il contient des données qui seront inscrites dans la CHIP-RAM via les canaux DMA. Il n'y a que deux types de registres de ce genre (DSKDATR et BLTDDAT - registres d'émission du blitter et lecteur de disquette). L'écriture dans la CHIP-RAM se fera sous surveillance du contrôleur DMA d'AGNUS. Le processeur ne peut pas avoir accès à de tels fichiers.

A.D.P

Ces trois caractères correspondent aux trois circuits différents AGNUS, DENISE et PAULA. Ils permettent d'indiquer l'origine des registres correspondants. Il est possible qu'un registre soit issue de plusieurs circuits à la fois. Lors d'un accès écriture, la valeur pourra être mise en mémoire dans les deux ou même les trois circuits, en même temps.

Pour le programmeur, il n'est pas fondamental de savoir dans quel circuit un registre particulier se trouve. On peut considérer la zone des circuits spécialisés comme n'en faisant qu'une, et ne prendre en compte que l'adresse et la fonction du registre désiré.

P.d.

Un petit d signifie que le registre est assisté par le contrôleur DMA. La lettre P signifie que le registre n'est utilisé que par le processeur ou le COPPER. La combinaison des deux est possible, signifiant que le registre est accessible via DMA, mais aussi par le processeur.

Nombre de registres : 197

Nombre de registres accessibles via les canaux DMA : 54

Adresse de base de la zone registre : \$DFF000

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
BLTDDAT	000	A	er	d	destination BLITTER
DMACONR	002	AP	r	p	registre de contrôle DMA
VPOSR	004	A	r	p	position verticale ; bit de poids fort
VHPOSR	006	A	r	p	position horizontale et verticale
DSDATR	008	P	er	d	lecture données disque
JOYODAT	00A	D	r	p	position souris/joystick gameport 0
JOY1DAT	00C	D	r	p	position souris/joystick gameport 1

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
CLXDAT	00E	D	r	p	registre des collisions
ADKCONR	010	P	r	p	contrôle AUDIO/DISQUE
POTODAT	012	P	r	p	potentiomètre gameport 0
POT1DAT	014	P	r	p	potentiomètre gameport 1
POTGOR	016	P	r	p	lecture données sur POT
SERDATR	018	P	r	p	status du port série
DSKBYTR	01A	P	r	p	status du port disque
INTENAR	01C	P	r	p	autorisation interrup.
INTREQR	01E	P	r	p	demande interruption
DSKPTH	020	A	w	p	adresse DMA disque bits 16-18
DSKPTL	022	A	w	p	adresse DMA disque bits 1-15
DSKLEN	024	P	w	p	longueur données disque
DSKDAT	026	P	w	d	données disque
REFPTR	028	A	w	d	compteur rafraîchis.
VPOSW	02A	A	w	p	MSB de la position vert.
VHPOSW	02C	A	w	p	position verticale et horizontale
COPCON	02E	A	w	p	registre de contrôle COPPER
SERDAT	030	P	w	p	données série et bit de stop
SERPER	032	P	w	p	période et contrôle du port série
POTGOT	034	P	w	p	démarrage compteur PDT
JOYTEST	036	D	w	p	écriture dans 2 compteurs souris/joystick
STREQU	038	D	s	d	synchronisation vert.
STRVBL	03A	D	s	d	synchronisation horiz.
STRHOR	03C	DP	s	d	signal synchronisation horizontale
STRLONG	03E	D	s	d	identification longueur horizontale

Seuls les registres suivants peuvent être accessibles au COPPER lorsque, COPCON = 1.

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
BLTCON0	040	A	w	p	contrôle Blitter
BLTCON1	042	A	w	p	contrôle Blitter
BLTAFWM	044	A	w	p	masque premier mot
BLTALWM	046	A	w	p	masque dernier mot
BLTCPTH	048	A	w	p	source C (bits 16-18)
BLTCPTL	04A	A	w	p	source C (bits 1-15)
BLTBPTH	04C	A	w	p	source B (bits 16-18)
BLTBPTL	04E	A	w	p	source B (bits 1-15)
BLTAPTH	050	A	w	p	source A (bits 16-18)
BLTAPTL	052	A	w	p	source A (bits 1-15)
BLTDPTH	054	A	w	p	destination D (bits 16-18)
BLTDPTL	056	A	w	p	destination D (bits 1-15)
BLTSIZE	058	A	w	p	démarrage Blitter et dimensionnement taille fenêtre
	05A				inutilisé
	05C				inutilisé
	05E				inutilisé
BLTCMOD	060	A	w	p	modulo source C
BLTBMOD	062	A	w	p	modulo source B
BLTAMOD	064	A	w	p	modulo source A
BLTDMOD	066	A	w	p	modulo destination D
	068				inutilisé
	06A				inutilisé
	06C				inutilisé
	06E				inutilisé
BLTCDAT	070	A	w	d	données source C
BLTBDAT	072	A	w	d	données source B
BLTADAT	074	A	w	d	données source A
	076				inutilisé
	078				inutilisé
	07A				inutilisé
	07C				inutilisé
DSKSYNC	07E	P	w	p	remplissage sync. disque

Seuls les registres suivants peuvent être, dans tous les cas, accessibles en mode écriture par le COPPER.

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
COP1LCH	080	A	w	p	1ère adresse COPPER (bits 16-18)
COP1LCL	082	A	w	p	1ère adresse COPPER (bits 1-15)
COP2LCH	084	A	w	p	2ème adresse COPPER (bits 16-18)
COP2LCL	086	A	w	p	2ème adresse COPPER (bits 1-15)
COPJMP1	088	A	s	p	redémarrage COPPER 1ère adresse
COPJMP2	08A	A	s	p	redémarrage COPPER 2ème adresse
COPINS	08C	A	w	d	identification instruc.
DIWSTRT	08E	A	w	p	coin sup. gauche fenêtre
DIWSTOP	090	A	w	p	coin inf. droit fenêtre
DDFSTRT	092	A	w	p	début bitplane (pos.hor)
DDFSTOP	094	A	w	p	fin bitplane (pos.hor)
DMACON	096	ADP	w	p	registre contrôle DMA
CLXCON	098	D	w	p	contrôle collision
INTENA	09A	P	w	p	autorisation interrup.
INTREQ	09C	P	w	p	demande d'interruption
ADKCON	09E	P	w	p	contrôle audio,disk,UART
AUDOLCH	0A0	A	w	p	canal audio 0 (bits 16-18)
AUDOLCL	0A2	A	w	p	canal audio 0 (bits 1-15)
AUDOLEN	0A4	P	w	p	longueur données audio
AUDOPER	0A6	P	w	p	période canal audio 0
AUDOVOL	0A8	P	w	p	volume canal audio 0
AUDODAT	0AA	P	w	d	canal audio 0 données
	OAC				inutilisé
	OAE				inutilisé
AUD1LCH	0B0	A	w	p	canal audio (1 bits 16-18)
AUD1LCL	0B2	A	w	p	canal audio 1 (bits 1-15)
AUD1LEN	0B4	P	w	p	longueur données audio
AUD1PER	0B6	P	w	p	période canal audio 1
AUD1VOL	0B8	P	w	p	volume canal audio 1
AUD1DAT	0BA	P	w	d	canal audio 1 données

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
	OBC				inutilisé
	OBE				inutilisé
AUD2LCH	OC0	A	w	p	canal audio 2 (bits 16-18)
AUD2LCL	OC2	A	w	p	canal audio 2 (bits 1-15)
AUD2LEN	OC4	P	w	p	longueur données audio
AUD2PER	OC6	P	w	p	période canal audio 2
AUD2VOL	OC8	P	w	p	volume canal audio 2
AUD2DAT	OCA	P	w	d	canal audio 2 données
	OCC				inutilisé
	OCE				inutilisé
AUD3LCH	OD0	A	w	p	canal audio 3 (bits 16-18)
AUD3LCL	OD2	A	w	p	canal audio 3 (bits 1-15)
AUD3LEN	OD4	P	w	p	longueur données audio
AUD3PER	OD6	P	w	p	période canal audio 3
AUD3VOL	OD8	P	w	p	volume canal audio 3
AUD3DAT	ODA	P	w	d	canal audio 0 données
	ODC				inutilisé
	ODE				inutilisé
BPL1PTH	OE0	A	w	p	bitplane 1 (bits 16-18)
BPL1PTL	OE2	A	w	p	bitplane 1 (bits 1-15)
BPL2PTH	OE4	A	w	p	bitplane 2 (bits 16-18)
BPL2PTL	OE6	A	w	p	bitplane 2 (bits 1-15)
BPL3PTH	OE8	A	w	p	bitplane 3 (bits 16-18)
BPL3PTL	OEA	A	w	p	bitplane 3 (bits 1-15)
BPL4PTH	OEC	A	w	p	bitplane 4 (bits 16-18)
BPL4PTL	OEE	A	w	p	bitplane 4 (bits 1-15)
BPL5PTH	OF0	A	w	p	bitplane 5 (bits 16-18)
BPL5PTL	OF2	A	w	p	bitplane 5 (bits 1-15)
BPL6PTH	OF4	A	w	p	bitplane 6 (bits 16-18)
BPL6PTL	OF6	A	w	p	bitplane 6 (bits 1-15)
	OF8				inutilisé
	OFA				inutilisé

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
	OFC				inutilisé
	OFE				inutilisé
BPLCON0	100	AD	w	p	reg. contrôle 0 bitplane
BPLCON1	102	D	w	p	reg. contrôle 1 bitplane
BPLCON2	104	D	w	p	reg. contrôle 2 bitplane
	106				inutilisé
BPL1MOD	108	A	w	p	contr. bitplane modulo plan impair
BPL2MOD	10A	A	w	p	contr. bitplane modulo plan pair
	10C				inutilisé
	10E				inutilisé
BPL1DAT	110	D	w	d	données bitplane 1 (RGB)
BPL2DAT	112	D	w	d	données bitplane 2 (RGB)
BPL3DAT	114	D	w	d	données bitplane 3 (RGB)
BPL4DAT	116	D	w	d	données bitplane 4 (RGB)
BPL5DAT	118	D	w	d	données bitplane 5 (RGB)
BPL6DAT	11A	D	w	d	données bitplane 6 (RGB)
	11C				inutilisé
	11E				inutilisé
SPROPTH	120	A	w	p	données sprite 0 (bits 16-18)
SPROPTL	122	A	w	p	données sprite 0 (bits 1-15)
SPR1PTH	124	A	w	p	données sprite 1 (bits 16-18)
SPR1PTL	126	A	w	p	données sprite 1 (bits 1-15)
SPR2PTH	128	A	w	p	données sprite 2 (bits 16-18)
SPR2PTL	12A	A	w	p	données sprite 2 (bits 1-15)
SPR3PTH	12C	A	w	p	données sprite 3 (bits 16-18)
SPR3PTL	12E	A	w	p	données sprite 3 (bits 1-15)
SPR4PTH	130	A	w	p	données sprite 4 (bits 16-18)
SPR4PTL	132	A	w	p	données sprite 4 (bits 1-15)
SPR5PTH	134	A	w	p	données sprite 5 (bits 16-18)
SPR5PTL	136	A	w	p	données sprite 5 (bits 1-15)
SPR6PTH	138	A	w	p	données sprite 6 (bits 16-18)
SPR6PTL	13A	A	w	p	données sprite 6 (bits 1-15)

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
SPR7PTH	13C	A	w	p	données sprite 7 (bits 16-18)
SPR7PTL	13E	A	w	p	données sprite 7 (bits 1-15)
SPROPOS	140	AD	w	dp	position départ sprite 0
SPROCTL	142	AD	w	dp	contrôle sprite 0
SPRODATA	144	D	w	dp	données A sprite 0 (RGB)
SPRODATB	146	D	w	dp	données B sprite 0 (RGB)
SPR1POS	148	AD	w	dp	position départ sprite 1
SPR1CTL	14A	AD	w	dp	contrôle sprite 1
SPR1DATA	14C	D	w	dp	données A sprite 1 (RGB)
SPR1DATB	14E	D	w	dp	données B sprite 1 (RGB)
SPR2POS	150	AD	w	dp	position départ sprite 2
SPR2CTL	152	AD	w	dp	contrôle sprite 2
SPR2DATA	154	D	w	dp	données A sprite 2 (RGB)
SPR2DATB	156	D	w	dp	données B sprite 2 (RGB)
SPR3POS	158	AD	w	dp	position départ sprite 3
SPR3CTL	15A	AD	w	dp	contrôle sprite 3
SPR3DATA	15C	D	w	dp	données A sprite 3 (RGB)
SPR3DATB	15E	D	w	dp	données B sprite 3 (RGB)
SPR4POS	160	AD	w	dp	position départ sprite 4
SPR4CTL	162	AD	w	dp	contrôle sprite 4
SPR4DATA	164	D	w	dp	données A sprite 4 (RGB)
SPR4DATB	166	D	w	dp	données B sprite 4 (RGB)
SPR5POS	168	AD	w	dp	position départ sprite 5
SPR5CTL	16A	AD	w	dp	contrôle sprite 5
SPR5DATA	16C	D	w	dp	données A sprite 5 (RGB)
SPR5DATB	16E	D	w	dp	données B sprite 5 (RGB)
SPR6POS	170	AD	w	dp	position départ sprite 6
SPR6CTL	172	AD	w	dp	contrôle sprite 6
SPR6DATA	174	D	w	dp	données A sprite 6 (RGB)
SPR6DATB	176	D	w	dp	données B sprite 6 (RGB)
SPR7POS	178	AD	w	dp	position départ sprite 7
SPR7CTL	17A	AD	w	dp	contrôle sprite 7

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
SPR7DATA	17C	D	w	dp	données A sprite 7 (RGB)
SPR7DATB	17E	D	w	dp	données B sprite 7 (RGB)
COLOR00	180	D	w	p	valeur de la couleur 0
COLOR01	182	D	w	p	valeur de la couleur 1
COLOR02	184	D	w	p	valeur de la couleur 2
COLOR03	186	D	w	p	valeur de la couleur 3
COLOR04	188	D	w	p	valeur de la couleur 4
COLOR05	18A	D	w	p	valeur de la couleur 5
COLOR06	18C	D	w	p	valeur de la couleur 6
COLOR07	18E	D	w	p	valeur de la couleur 7
COLOR08	190	D	w	p	valeur de la couleur 8
COLOR09	192	D	w	p	valeur de la couleur 9
COLOR10	194	D	w	p	valeur de la couleur 10
COLOR11	196	D	w	p	valeur de la couleur 11
COLOR12	198	D	w	p	valeur de la couleur 12
COLOR13	19A	D	w	p	valeur de la couleur 13
COLOR14	19C	D	w	p	valeur de la couleur 14
COLOR15	19E	D	w	p	valeur de la couleur 15
COLOR16	1A0	D	w	p	valeur de la couleur 16
COLOR17	1A2	D	w	p	valeur de la couleur 17
COLOR18	1A4	D	w	p	valeur de la couleur 18
COLOR19	1A6	D	w	p	valeur de la couleur 19
COLOR20	1A8	D	w	p	valeur de la couleur 20
COLOR21	1AA	D	w	p	valeur de la couleur 21
COLOR22	1AC	D	w	p	valeur de la couleur 22
COLOR23	1AE	D	w	p	valeur de la couleur 23
COLOR24	1B0	D	w	p	valeur de la couleur 24
COLOR25	1B2	D	w	p	valeur de la couleur 25
COLOR26	1B4	D	w	p	valeur de la couleur 26
COLOR27	1B6	D	w	p	valeur de la couleur 27
COLOR28	1B8	D	w	p	valeur de la couleur 28
COLOR29	1BA	D	w	p	valeur de la couleur 29

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
COLOR30	1BC	D	w	p	valeur de la couleur 30
COLOR31	1BE	D	w	p	valeur de la couleur 31

Les registres de 1C0 à 1FC sont inutilisés.

Un accès à l'adresse registre 1FE ne déclenche aucune fonction. Les circuits n'y ont d'ailleurs pas accès.

La ROM

La zone ROM se tient après la routine de démarrage. A partir de \$FC0000, les 256 Ko de ROM renferment le kickstart de l'Amiga.

La zone mémoire de \$F80000 à \$FBFFFF est identique à celle se tenant entre \$FC0000 et \$FFFFFF. On y retrouve encore une fois la ROM KICKSTART. Evidemment, cette configuration peut être modifiée. Après un RESET, le 68000 cherche l'adresse du premier ordre à exécuter à l'emplacement mémoire 4, que l'on appelle aussi vecteur RESET. Si la configuration mémoire n'est pas modifiée, le 68000 recherche le vecteur RESET dans la CHIP-RAM, qui se tient à l'adresse 4. Après le démarrage, ce contenu n'étant pas déterminé, le processeur sautera à une adresse arbitraire et le système se plantera dès le démarrage. La solution est la suivante : la CHIP-RAM, qui est si importante pour la configuration mémoire, a une entrée reliée au signal de port PA0 du CIA-A. Cette liaison, aussi appelée signal OVL (Memory Overlay), reste à l'état normal à 0, et la configuration mémoire correspond au schéma. Après un RESET, le signal de port se met automatiquement à 1. La zone mémoire, s'étalant de \$F80000 à \$FFFFFF, sera alors masquée par la zone d'adresse de 0 à \$7FFF (l'adresse 4 correspondra ainsi à \$F80004). Le 68000 trouvera donc une adresse RESET valide, qui lui laissera l'accès au KICKSTART. Lors du déroulement de la routine RESET, le signal OVL sera remis à 0, ramenant ainsi la mémoire à un état normal.

On doit être très attentif lorsqu'on veut expérimenter ces signaux. Le fait que le programme, qui doit mettre à 1 le signal OVL, tourne dans la CHIP-RAM, peut avoir pour fâcheuse conséquence que ce programme s'élimine tout seul de la mémoire, et que le processeur aborde n'importe où à l'intérieur du KICKSTART, celui-ci se trouvant à la place de la CHIP-RAM après la commutation.

La WOM de L'A1000

On trouve d'autres particularités sur le modèle 1000. Les possesseurs de ce type d'Amiga se sont déjà rendus compte qu'on parle continuellement d'une ROM KICKSTART alors qu'au départ, ce dernier était chargé à partir d'une disquette. En fait la situation de l'A1000 était la suivante : le hardware était terminé, l'ordinateur prêt à être vendu, mais le software, c'est-à-dire le système d'exploitation KICKSTART, était

incomplet et truffé d'erreurs. Pour résoudre ce problème, Amiga se vit inclure une RAM spéciale, où le système d'exploitation était chargé après le démarrage, l'accès à cette RAM étant verrouillé par la suite. Celle-ci se comporte en fait comme une ROM de 256 Ko et a été appelée par Commodore : WOM (write once memory). Ce modèle a été commercialisé avec la version 1.0 du KICKSTART, les nouvelles versions corrigées étant à présent disponibles.

La mémoire morte des A500 et A2000 est de type ROM, cette dernière étant bien moins chère que sa précédente, la WOM. De plus la version 1.2 du KICKSTART était achevée.

La présence de cette WOM a tout de même soulevé des problèmes :

- ✓ Après le démarrage, où se trouve le programme qui permet le chargement du KICKSTART ?
- ✓ Comment modifier ce KICKSTART s'il se trouve en RAM ?

La zone du système d'exploitation de l'A1000 est identique à celle des modèles ultérieurs. En effet, on la retrouve aux adresses \$FC0000 à \$FFFFFF, même la zone \$F80000 est présente. Il ne se passe rien si on cherche à écrire dans le KICKSTART. Il n'y a pas d'accès écriture possible. La routine de chargement (BOOT ROM) n'est pas non plus masquée.

En fait, l'ensemble est dirigé par le signal RESET. La configuration mémoire est modifiée après un signal RESET, celui-ci pouvant être d'origine différente (démarrage, touches Amiga & Control, ordre RESET du 68000).

Ainsi la BOOT-ROM se tient à partir de \$F80000 (comme un reset a pour conséquence l'activation du signal OVL, le vecteur RESET est aussi issu de la BOOT-ROM) et il est alors possible d'écrire dans le KICKSTART et le modifier à son gré. Cet état ne dure que jusqu'au moment où l'on cherche à écrire dans la zone de la BOOT-ROM (\$F80000 à \$FBFFFF). La BOOT-ROM sera alors prise en charge et l'accès écriture sera interdit.

En d'autres termes :

- ✓ RESET permet l'écriture dans le KICKSTART et masque la BOOT-ROM.
- ✓ Un accès écriture à une adresse se trouvant entre \$F80000 et \$FBFFFF, interdit l'écriture et déconnecte la BOOT-ROM.

1.5.2. Eléments de base

Comme cela a été vu dans le chapitre précédent, il existe des registres qui peuvent être accessibles à partir du processeur, et d'autres qui peuvent être lus et écrits via les canaux DMA. Nous allons tout d'abord examiner le premier cas.

Programmation des registres processeurs

Les registres processeurs peuvent être adressés directement. Par exemple : la valeur du registre couleur arrière-plan doit être modifiée. Le registre a pour nom COLOR00. Lorsqu'on se reporte au tableau du chapitre 1.5.1, on trouve une adresse registre \$180. A ceci, on rajoute l'adresse de base de la zone registre, c'est-à-dire l'adresse du premier registre du 68000. Celle-ci correspond à \$DFF000. La somme des deux adresses donne \$DFF180. L'instruction MOVE.W suffit à initialiser le registre :

```
MOVE.W #valeur,$DFF180      ; valeur dans COLOR00
```

Si on désire accéder à plusieurs registres, il suffit de mettre l'adresse de base dans un registre adresse et d'utiliser l'adressage indirect avec déplacement (la valeur de déplacement correspondant à l'offset) :

```
LEA $DFF000,A5      : mise en mémoire de l'adresse de base dans A5
MOVE.W #valeur,$180(A5)  : valeur 1 dans COLOR00
MOVE.W #valeur,$182(A5)  : valeur 2 dans COLOR01
```

En temps normal l'accès à un registre processeur se fait de cette manière. On peut aussi y accéder avec un mot long, et dans ce cas, 2 registres correspondront à un seul. Ceci a un sens dans le cas d'un registre adresse. Celui-ci se trouve dans une paire de registres contenant une adresse 19 bits, avec laquelle toute la zone CHIP-RAM de 512 Ko est accessible. Tout ce qui, en tant que données, a un rapport avec les circuits spécialisés doit être mis dans la CHIP-RAM. Le registre adresse ne montre que des adresses paires. Comme un registre processeur ne peut contenir qu'un mot, deux registres à adresse contiguë servent d'accueil à l'adresse mémoire 19 bits. Ainsi le premier contient les trois bits de plus fort poids (bits 16-18), le deuxième contenant les 16 bits inférieurs (bits 0-15). Il est ainsi possible d'initialiser deux registres lors d'un accès mot long.

Par exemple : le pointeur du premier bitplane doit être mis à l'adresse \$40000. BPL1PTH est le nom du premier registre (bits 16-18) et BPL1PTL (bits 0-15) celui du deuxième. Les adresses registres sont respectivement \$0E0 et \$0E2 et A5 contient l'adresse de base \$DFF000.

```
MOVE.L #$40000,$0E0(A5)      ;initialise les registres BPL1PTL et
                                ;BPL1PTH avec la valeur correcte.
```

Il faut bien insister sur le fait qu'on ne peut écrire et lire un registre sur une seule et unique adresse registre. La plupart des registres ne sont d'ailleurs accessibles qu'en écriture et ne peuvent donc pas être lus. C'est le cas des registres utilisés plus haut. D'autres ne peuvent qu'être lus. Il est assez rare que les deux modes soient possibles. Ces registres possèdent alors deux adresses différentes, l'une permettant la lecture, l'autre l'écriture. On peut prendre comme exemple le registre de contrôle DMA, sur lequel on reviendra par la suite, où l'écriture est possible à l'adresse \$096(DMACON) et où la lecture est possible à l'adresse \$002 DMACONR.

Les accès DMA

On entend par accès DMA, comme cela a déjà été énoncé au chapitre 1.2.3, l'accès direct d'un circuit périphérique, le contrôleur DMA, à la mémoire système. Dans le cas de l'Amiga, le contrôleur DMA est inclus dans Agnus.

Il reproduit la liaison des différentes unités d'entrée/sortie (input/output ou I/O) du circuit spécialisé avec la CHIP-RAM. Qu'il s'agisse de données audio, écran ou disquette, le processus DMA se déroule toujours de la même manière. N'importe quelle unité, par exemple le contrôleur disque, nécessite de nouvelles données de la mémoire ou possède de nouvelles données prêtes pour la mémoire. Le contrôleur DMA attend alors le moment où le canal DMA sera libre, et transfère les données de ou vers la mémoire.

Pour simplifier les choses, il n'y a pas de transfert spécial des données d'unité d'entrée/sortie vers le contrôleur DMA. Cela se déroule normalement par registre. Chacune de ces unités d'entrée/sortie possède deux types de registres différents. Le premier est un registre normal, qui est accessible à partir du processeur et dans lequel les différents paramètres d'une tâche sont mis en mémoire. Le deuxième est un registre de données qui contient celles du contrôleur DMA. Celui-ci accède simultanément, lors d'un transfert DMA, au registre de données correspondant ainsi qu'à son emplacement mémoire RAM. Suivant le sens du transfert DMA, ceci correspond soit à un registre à lire et un accès écriture CHIP-RAM, soit à un registre à écrire et un accès lecture CHIP-RAM. Comme ces deux sont reliés sur le bus de données, ces dernières cheminent automatiquement du registre de données vers la RAM ou l'inverse. Les données ne sont pas mises en mémoire dans n'importe quel registre interne.

Au travers du transfert DMA se rajoute un troisième type de registre : le registre adresse DMA qui, suivant les besoins de l'unité d'entrée/sortie, contient l'adresse (ou les adresses) des données dans la RAM.

De plus, il existe des registres de contrôle centraux qui ne sont pas attribués à une unité spéciale d'entrée/sortie, mais préposés à des fonctions de gestion. C'est à cette catégorie qu'appartient le registre DMACON, vu plus haut.

Les registres de données sont accessibles en mode écriture par le processeur, mais ceci n'a que peu de sens, étant donné que le contrôleur DMA réalise cette opération d'une manière plus élégante et plus rapide.

Quelques unités d'I/O ne possèdent pas de canaux DMA propres. Le 68000 sera alors obligé de lire et écrire les données lui-même. Ce sont des exceptions à qui, par nature, il n'échoit que peu de données et où un DMA n'est pas nécessaire, comme par exemple l'entrée joystick/ souris.

Voici les canaux DMA existants.

DMA bitplane : Par ces canaux DMA, les données image peuvent être lues dans la mémoire et écrites dans les registres de données de chaque bitplane, d'où ils atteignent les séquenceurs bitplane dont le rôle est la sortie de l'image sur l'écran.

- DMA sprite :** Transfert des données sprite de la RAM vers le registre de données sprite.
- DMA disque :** Transfert des données disquette vers la RAM ou inverse.
- DMA audio :** Lit les données sons digitalisés de la RAM et les écrit dans le registre de données audio correspondant.
- DMA copper :** Par lui, le coprocesseur copper obtient son mot d'ordre.
- DMA blitter :** Transfert des données du et vers le blitter.

Il y a donc six canaux DMA qui ont tous accès à la mémoire, en plus du processeur qui, naturellement, en a aussi la possibilité. Pour résoudre les difficultés qui en résultent, on a conçu un temporisateur compliqué, dans lequel on attribue une position définie à chaque canal. Comme ce temporisateur s'oriente suivant l'image écran, nous devons d'abord étudier la structure de l'écran. Cet intermède sera aussi peu technique que possible car ce chapitre est consacré à la programmation des circuits spécialisés et non au hardware.

La structure de l'écran

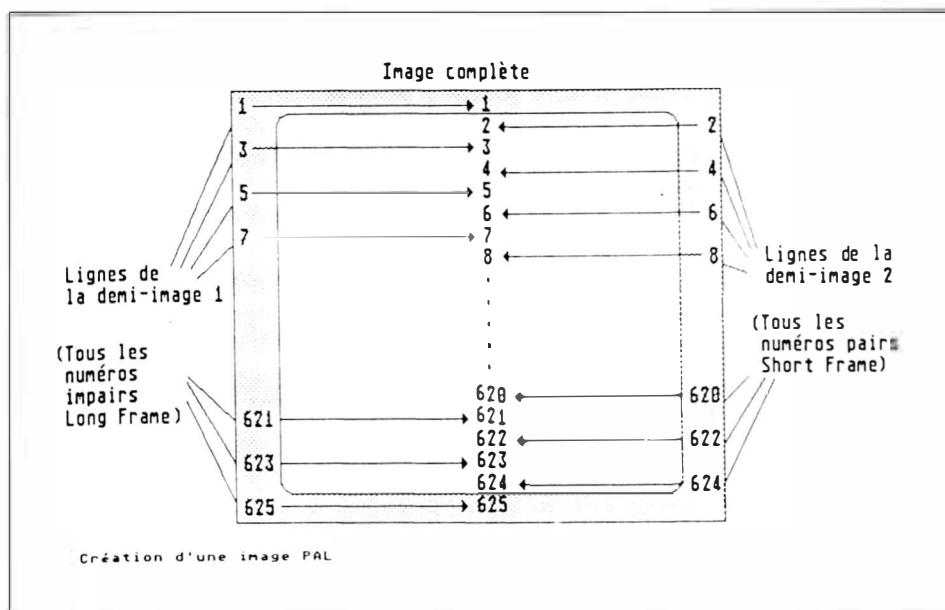


Figure 1 - 28

La fonction de transfert d'une sortie vidéo d'un Amiga français correspond exactement à la norme PAL. Une telle image est composée de 625 lignes horizontales. Ces dernières sont affichées de gauche à droite. Chaque fin de ligne est suivi d'une pause, afin de permettre au raster de revenir à gauche. Le raster est le mouvement continu des électrons qui permettent l'affichage. Pendant cette phase, le rayonnement électronique n'est pas lumineux, permettant au retour du raster, de ne pas engendrer de scintillements parasites. Puis le processus est à nouveau répété pour l'affichage d'une nouvelle ligne.

Afin que l'image soit exempte de tout scintillement, on doit réafficher une nouvelle image assez rapidement. Comme notre oeil n'observe plus un changement d'image à une certaine fréquence, on a fixé un nombre d'images affiché par seconde supérieure à cette limite. Pour la norme PAL ceci correspond à 50 images par secondes. Evidemment il existe un fait qui complique la chose.

Si on veut afficher 625 lignes 50 fois par secondes, on arrive à un total de 31250 lignes par seconde. Lorsque l'élément de base de ce système vidéo a été mis au point, il n'aurait pas été possible de fabriquer un moniteur abordable et bon marché avec une telle fréquence d'affichage de ligne. Il a fallu trouver un arrangement. D'une part, le nombre d'images par seconde ne devrait jamais tomber en dessous de 50, pour raison de forts scintillements, et d'autre part, le nombre de lignes de l'écran ne pouvait être diminué. La solution fut la suivante : répartition des 625 lignes en deux images. La première image est alors affichée avec les lignes impaires (1, 3, 5, ..., 625) et la deuxième avec les lignes paires (2, 4, 6, ..., 624). Ainsi a-t-on à la suite 50 demi-images qui renferment toujours la moitié des lignes. Deux images d'un tel type correspondent à une image complète, renfermant 625 lignes. Le nombre d'images complètes par seconde se réduit à 25. La fréquence des lignes se monte à 15625 Hz (25*625 ou 50*312.5).

Malgré la haute résolution de 625 lignes, il apparaît un scintillement lorsqu'un contour est circonscrit par une ligne. Cette dernière n'est reproduite qu'une seule fois tous les 25ème de seconde, ceci étant visible à l'oeil nu. Cet effet peut être observé sur les bords horizontaux des surfaces, l'origine résidant dans l'affichage de la ligne horizontale.

Le terme anglais correspondant à cette technique d'affichage des lignes paires et impaires est interlace. Les deux termes suivants servent à différencier les deux types de 'demi-image'. Avec LONG FRAME on caractérise celle reproduite à l'aide des lignes paires et l'autre avec SHORT FRAME. Cette dénomination est conséquente à la différence du nombre de lignes entre les 2 demi-images, 313 pour les lignes impaires, 312 pour les lignes paires. L'image LONG FRAME sera donc affichée plus longtemps.

Chaque affichage d'une demi-image est suivi d'une pause. Cette phase noire entre deux images correspond au retour vertical du faisceau d'électrons.

L'image générée sur l'Amiga obéit aux règles énoncées plus haut, mais avec quelques particularités. Normalement, la deuxième demi-image (short frame) s'affiche légèrement différée, afin que les lignes paires puissent s'afficher exactement entre les lignes impaires pour former une image complète.

Sur l'Amiga, les deux demi-images sont identiques et la fréquence reste à 50 hertz. A la suite de quoi le nombre de lignes a été défini à 313. On reconnaît aussi la séparation

entre deux lignes sur le moniteur, étant donné que les demi-images ne sont plus affichées, déplacées l'une vers l'autre.

Afin d'augmenter le nombre de lignes, l'Amiga a la possibilité de générer une image en mode interlace, 625 lignes étant alors possibles.

La structure de la sortie vidéo de l'Amiga

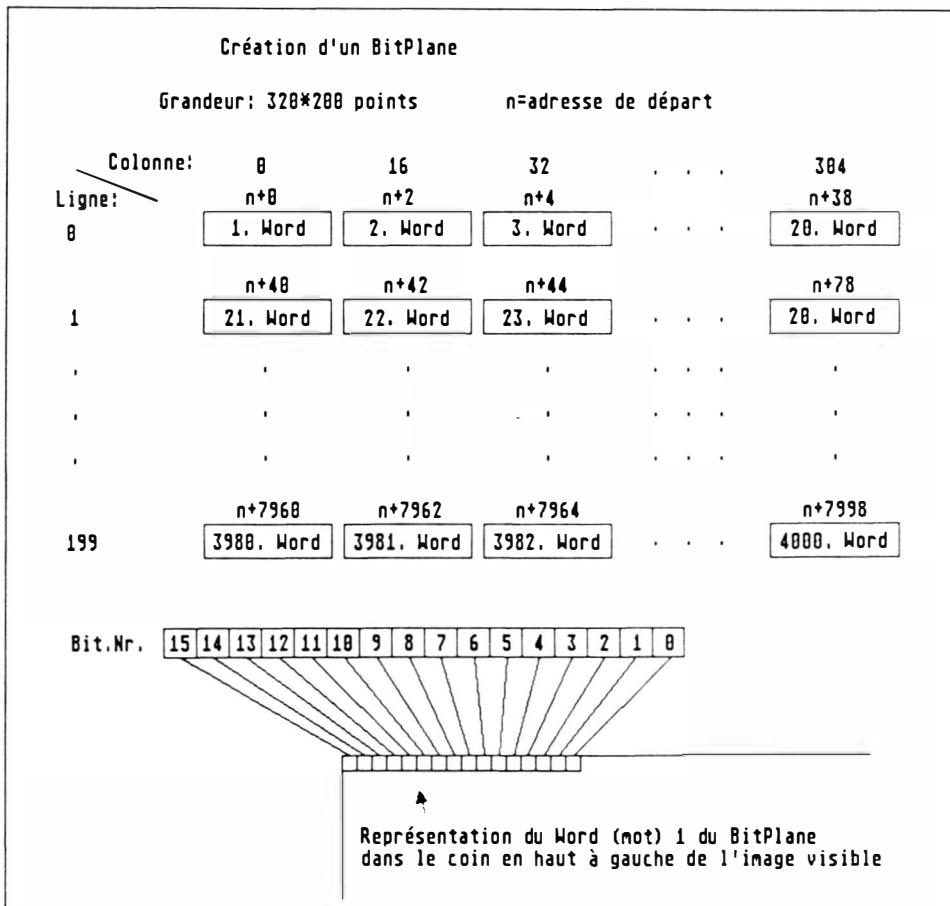


Figure 1 - 29

Bitplanes

L'Amiga reproduit toujours son image dans une sorte de mode graphique, c'est-à-dire que chaque point de l'image possède un correspondant en mémoire. Plus simplement,

un bit activé de la RAM correspond à un point sur le moniteur. Cette simple structure d'images en mémoire se nomme sur l'Amiga : bitplane.

C'est l'élément de base de toute reproduction d'image sur l'Amiga et il se trouve dans une zone mémoire cohérente. Un nombre fixé de mots d'une ligne d'écran donne la largeur d'une image. Un mot correspond à 16 points, étant donné que chaque bit représente un point. Une image de 320 points par ligne nécessite $320/16=20$ mots par ligne. Etant donné qu'un bitplane ne peut différencier que deux états (point allumé/point éteint), plusieurs bitplanes peuvent être combinés. Dans ce cas, les bits auront toujours la même position dans tous les plans. Le premier point de l'image devient, par combinaison des plans, le bit de plus fort poids du premier mot. La valeur résultante de ces bits détermine la couleur du point à l'écran. L'obtention des couleurs par combinaison des bits d'un point sera plus détaillée dans le chapitre 1.5.5.

Les différentes résolutions graphiques

L'Amiga connaît deux résolutions horizontales différentes. Le mode haute résolution possède 640 points par lignes, la basse résolution étant de 320. Il est préférable de définir les deux différentes résolutions suivant le temps d'affichage d'un point d'écran. Un pixel (point d'écran) en mode haute résolution est affiché durant 70 nanosecondes, celui en mode basse résolution étant affiché durant 140 nanosecondes. Dans cette dernière résolution, le point étant deux fois plus gros, le faisceau électronique accomplira le double de trajet.

Le plus important pour le programmeur est de savoir qu'en mode HR, seul 4 bitplanes sont accessibles, alors qu'en mode BR, 6 bitplanes sont disponibles.

Structure d'une ligne Raster horizontale

Par la notion de raster, on entend une ligne complète horizontale, engendrée par le mouvement des électrons. Le raster sert de mesure du temps pour tous les processus DMA. Afin de mieux comprendre le partage du raster, on doit savoir, comment se répartissent les accès mémoire sur la CHIP-RAM et sur les registres des circuits spécialisés entre le contrôleur DMA et le processeur. Les accès à ces deux zones mémoire s'alignent sur les cycles de bus de même nom. Les cycles de bus déterminent le timing de la CHIP-RAM. A chaque cycle, un accès mémoire peut se déclarer, mais il n'est pas évident que les données soient lues ou écrites. Si le processeur veut, par exemple, accéder à un bus, on le lui attribue pendant un cycle. Le contrôleur DMA peut alors à nouveau accéder à la RAM au cycle suivant. Un cycle de bus dure environ 280 nanosecondes, 4 accès mémoire étant donc possibles lors d'une microseconde.

Mais le 68000 ne peut accéder à la CHIP-RAM aussi facilement, il n'est pas assez rapide. Suivant la fréquence d'horloge qui est exploitée dans l'Amiga, il ne peut réaliser un accès que tous les 560 nanosecondes. Dans le même temps se déroulent deux cycles de bus. Le 68000 ne peut donc s'imposer que tous les deux cycles, ou cycles mémoire pairs (even cycles). Les autres cycles, impairs, (Odd cycles) sont réservés au contrôleur DMA,

ceci correspond à 227.5 cycles de bus par ligne, qui couvrent les 225 premiers du contrôleur DMA.

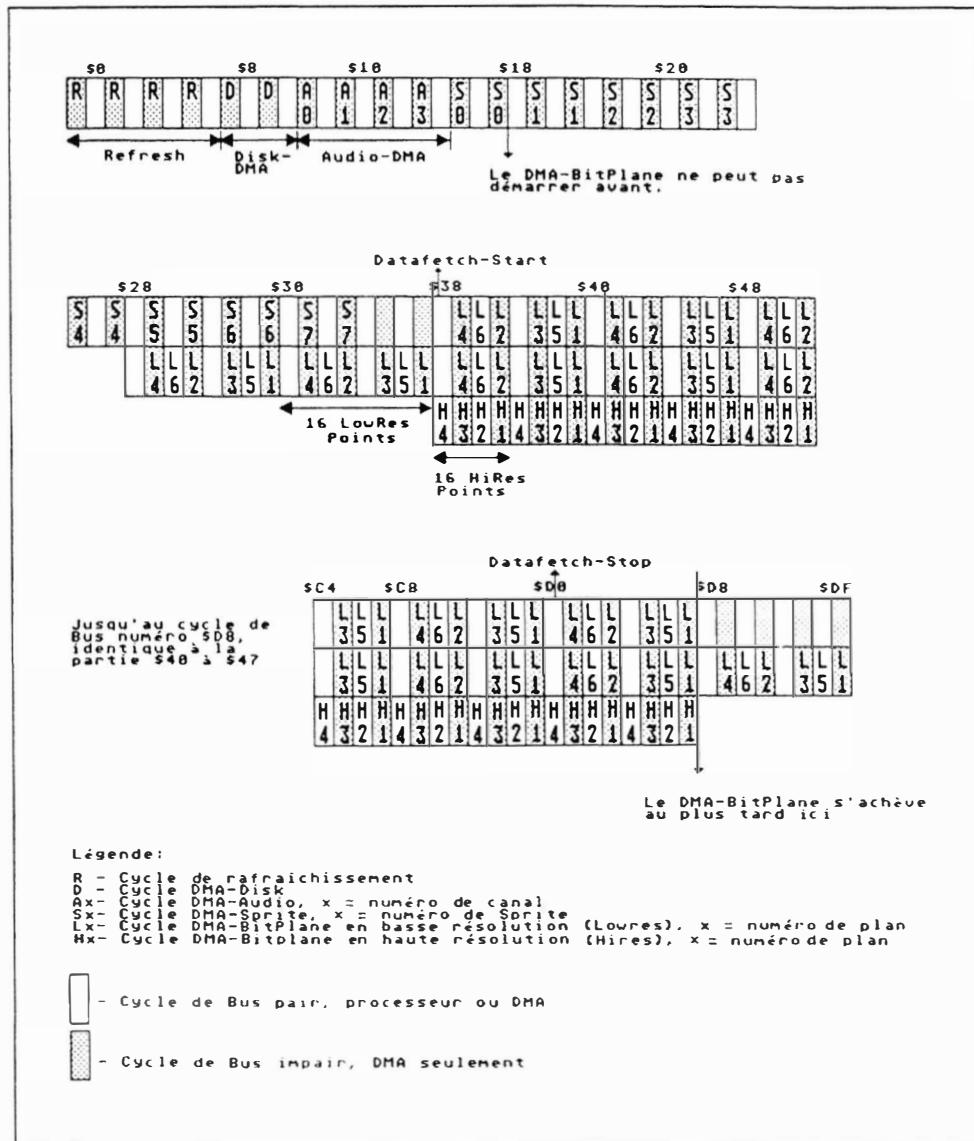


Figure 1 - 30

Ce qui se passe nous est indiqué par le schéma. Les caractères à l'intérieur de chaque cycle représentent les canaux DMA correspondant. Alors que le contrôleur DMA n'utilise que les cycles impairs, le processeur doit avoir accès aux cycles pairs. Les accès DMA

sont toujours prioritaires. Les accès DMA blitter et DMA copper ont lieu uniquement pendant les cycles pairs, mais n'ont aucune durée déterminée. Le DMA copper occupe tous les cycles pairs, jusqu'à ce qu'il ait fini sa tâche. Il est d'ailleurs prioritaire sur le BLITTER. Ce dernier fonctionne de la même manière, à ceci près qu'il peut allouer quelques cycles libres au 68000.

Comme cela apparaît sur le schéma, les accès DMA sprite, audio et disquette occupent des cycles impairs, n'influençant pas la vitesse du processeur. Les 4 cycles de bus indiqués par la lettre R correspondent aux cycles de rafraîchissement. Ils servent à raviver le contenu mémoire de la CHIP-RAM.

La répartition du DMA bitplane est quelque peu complexe. Pour reproduire les 16 premiers points sur l'écran, tous les bitplanes doivent être lus. Pendant que ces 16 points s'affichent, les bitplanes pour les 16 prochains points doivent être lus à leur tour. En basse résolution, 2 points sont émis à chaque cycle, signifiant ainsi que tous les 8 cycles, un bitplane est lu.

Les cycles impairs suffisent tant que 4 bitplanes et moins sont activés. Lorsque 5 ou 6 bitplanes sont utilisés, 2 cycles pairs doivent être alors employés, afin que toutes les données puissent être lues sur une durée de 8 cycles. L'utilisation est plus restreinte en mode haute résolution, seuls 4 points étant reproduits par cycles. Deux bitplanes hires peuvent donc être activés lors de l'utilisation des cycles impairs, l'utilisation des cycles pairs amenant à un maximum de 4 bitplanes actifs. Le processeur perd ici plus de la moitié de ses accès au bus. Sa vitesse diminuera d'un même facteur, en partant du principe que le programme se trouve dans la CHIP-RAM, car ses accès à une éventuelle FAST RAM ou à la ROM KICKSTART ne sont pas diminués pour autant.

Les points indiqués Datafetch-start et Datafetch-stop correspondent au début et à la fin des accès DMA des bitplanes. Ils déterminent la longueur et la position horizontale de l'image reproduite.

Plus le DMA bitplane débute tôt et finit tard, plus le nombre de mots de données lus et donc le nombre de points émis, seront importants. Les résolutions normales de 320 et 640 points par lignes se laissent modifier. Si on met à \$30 la valeur du datafetch-start, le canal DMA bitplane utilise normalement le cycle réservé au DMA sprite. De ce fait, suivant la valeur du datafetch-start, il peut disparaître jusqu'à sept sprites. Seul le sprite 0 ne se laisse pas éliminer de cette façon, celui-ci étant utilisé en tant que pointeur souris.

La ligne supérieure sur le schéma reproduit la répartition des cycles DMA, sur une largeur normale de 320 points en basse résolution. Le début du DMA bitplane, le datafetch-start, se trouve à \$38 et la fin, le datafetch-stop, se trouve à \$D0. A l'intérieur, le cycle marqué par L1 correspond à la lecture des données du bitplane 1, L2 à la lecture du bitplane 2 etc... Si les bitplanes correspondants ne sont pas activés, leur cycle DMA sera éliminé.

La deuxième ligne reproduit le déroulement du Raster sur une ligne, dans laquelle le point datafetch a été déplacé vers l'extérieur. Jusqu'au datafetch-start, le déroulement est le même que celui de la ligne supérieure, le DMA bitplane débutant à \$28. Ceci a pour conséquence l'élimination des sprites 5 à 7. La position du datafetch-stop ne pourra être décalée que jusqu'à la valeur maximale \$D8.

La troisième ligne montre la répartition des cycles DMA en haute résolution, où les valeurs datafetch correspondent à celles de la première ligne.

Lors du retour vertical du faisceau d'électrons, les accès DMA bitplane n'ont pas lieu.

Les registres de contrôle DMA

Les canaux DMA sont activés ou désactivés suivant le registre central de contrôle.

Adresses du registre DMACON : écriture \$096 ; lecture \$02

Bit	Nom	Fonction
15	SET/CLR	Bits allumer/éteindre
14	BBUSY	Blitter travaille (seulement en lecture)
13	BZERO	résultat de toutes les opérations du Blitter est 0 (seulement en lecture)
12 et 11		inutilisé
10	BLTPRI	DMA blitter est prioritaire sur le processeur
9	DMAEN	activer DMA complet (bits 0 à 8)
8	BPELN	activer DMA bitplane
7	COPEN	activer DMA copper
6	BLTEN	activer DMA blitter
5	SPREN	activer DMA sprite
4	DSKEN	activer DMA disque
3-0	AUDxEN	activer DMA audio par canal son (le numéro de bit correspond au numéro de canal)

Le registre DMACON n'est pas utilisé comme un registre normal. On ne peut qu'activer des bits ou les effacer. Ceci sera établi par le bit 15 du mot de donnée écrit dans le registre DMACON. Si ce bit est à 1, tous les bits activés dans le mot de donnée le seront aussi dans le registre DMACON. Si le bit 15 est à 0, tous les bits activés du registre DMACON seront effacés. Les autres bits de ce registre ne sont pas influencés.

Le bit 9, DMAEN, est utilisé comme interrupteur général. S'il est à 0, tous les canaux DMA sont inactifs, et ceci malgré les bits 0 à 8. Un canal DMA est sélectionné seulement si le canal correspondant et le bit DMAEN sont activés.

Exemple

Le DMA bitplane est activé (BPLEN=1), mais sans bit DMAEN. La valeur du registre DMACON est alors \$0100. Le DMA disque est alors sélectionné. DSKEN et DMAEN sont alors activés et BPLEN est effacé.

```
MOVE.W #$0100,$DFF096 : bit BPLEN est effacé (SET/CLR = 0)
MOVE W #$0210,$DDF096 : DSKEN et DMAEN sont activés (SET/CLR = 1)
```

Le registre DMACON contient alors la valeur voulue \$0210.

Les bits 13 et 14 ne peuvent qu'être lus. Ils donnent des renseignements sur l'état du Blitter (plus de précision au chapitre sur le Blitter).

Le bit 10 gère la priorité du Blitter sur le processeur. S'il est activé, le Blitter a la priorité absolue sur le processeur. Ceci signifie que ce dernier n'a aucune possibilité d'accès à un registre d'un circuit ou à la CHIP-RAM durant toutes les opérations du Blitter. Lorsqu'il est désactivé, on alloue au processeur un cycle tous les 4 cycles de bus pairs. Ceci empêche que le processeur soit arrêté trop longtemps, surtout si un accès à une routine du système d'exploitation ou à un programme de la FAST RAM est absolument nécessaire (structure de données du système d'exploitation ou vecteur d'exception du 68000).

La position actuelle du faisceau d'électron

Etant donné que le timing du DMA est orienté suivant la position du Raster, on doit savoir à quel endroit de la ligne se trouve le faisceau d'électrons. AGNUS possède pour ceci un compteur interne, qui contient la position horizontale et verticale du faisceau sur l'écran suivant laquelle le système s'aligne. Le processeur a la possibilité d'accès à ce compteur grâce à deux registres :

VHPOS \$006 (lecture, VHPOSR) et \$02C (écriture, VHPOSW)

Bit n :	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fonction :	V7	V6	V5	V4	V3	V2	V1	V0	H8	H7	H6	H5	H4	H3	H2	H1

VPOS \$004 (lecture, VPOSR) et \$02A (écriture, VPOSW)

Bit n :	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fonction :	LOF	V8

Les bits H1 à H8 reproduisent la position horizontale du faisceau correspondant directement à chaque cycle de bus (schéma précédent), et possédant les coordonnées de deux ou quatre points, respectivement, en basse et en haute résolution. La valeur d'une position horizontale peut varier de \$0 à \$E3 (0 à 227). Le temps mort horizontal correspond à la zone \$F-\$35.

Les bits caractérisant la position verticale, soit la ligne actuelle, sont répartis en deux registres. Les bits inférieurs V0 à V7 se trouvent dans le registre VHPOS, le bit V8 de

plus fort poids se trouvant dans le registre VPOS. Ensemble, ils donnent le numéro de la ligne actuelle sur l'écran.

Les lignes de 0 à 312 sont possibles. Le temps mort vertical atteindra alors la ligne 25.

Le bit LOF (long frame) indique si le bit reproduit est du type LONG FRAME (demi-image composée des lignes impaires) ou du type SHORT FRAME (demi-image composée des lignes paires). Ce bit n'est nécessaire qu'en mode interlace, en temps normal, celui-ci est à 1.

Le registre POS est utilisé par le crayon lumineux (lightpen). Il retient la position de ce dernier, lorsque l'entrée lightpen d'AGNUS est activée et lorsque le crayon est maintenu contre l'écran. Son contenu sera bloqué tant qu'un faisceau d'électrons ne sera pas passé devant la pointe du crayon.

A la fin du temps mort vertical, c'est-à-dire à la ligne 26, le compteur sera débloqué. Si on veut lire la position du crayon, on doit suivre les indications suivantes :

- ✓ Attendre à la ligne 0 (début du temps mort vertical). Ceci peut se faire au moyen d'une interruption verticale blanking (cf. chapitre suivant).
- ✓ Lire les deux registres compteurs.

Si la position verticale se trouve entre 0 et 25, aucun signal lightpen ne sera réceptionné. Si la valeur est en dehors de ce temps mort, la position du crayon est reproduite.

Pour finir ce chapitre, quelques détails sur les cycles de rafraîchissement:

Agnus possède un compteur 8 bits de rafraîchissement intégré. On peut y accéder par l'adresse registre \$28 (attention ! le contenu mémoire peut être perdu). Au début de chaque ligne, Agnus procède à un rafraîchissement du contenu mémoire des lignes toutes les 4 millisecondes, en ravivant 4 adresses sur le bus d'adresse CHIP-RAM.

Pendant que l'adresse de la ligne sera émise sur le bus d'adresse CHIP-RAM, Agnus met l'adresse du registre strobe déterminé sur le bus d'adresse registre. Ce signal strobe sert à communiquer aux autres circuits, DENISE et PAULA, le moment où une ligne ou une image débute. Ceci est nécessaire, étant donné que le compteur de position à l'écran se trouve dans Agnus et qu'il n'existe aucun signal de transfert de synchronisation sur les autres circuits. Il existe 4 adresses strobe différentes.

Adr.	Circuit	Fonction
\$38	D	Temps mort vertical d'une short frame
\$3A	D	Temps mort vertical
\$3C	D P	Cette adresse strobe est générée à chaque ligne du RASTER en dehors du temps mort vertical
\$3E	D	Indicateur d'une longue ligne Raster (228 cycles)

Pendant le premier cycle de rafraîchissement, on accède toujours à l'une des trois adresses strobe ci-dessus. En temps normal, ce sera \$3C, qui se trouve à l'intérieur du temps mort \$38 ou \$3A, qu'il s'agisse d'une short ou long frame.

La quatrième adresse nous indique un nouveau caractère du rafraîchissement. En effet une ligne du Raster a une longueur calculée de 227.5 cycles de bus. Comme on ne peut découper un cycle en deux, les lignes alternent entre 227 et 228 cycles de bus. L'adresse strobe \$3E signale les 228 cycles d'une ligne et sera générée pendant les deux cycles de rafraîchissement.

1.5.3. Les interruptions

Toutes les unités d'entrée/sortie ainsi que les deux CIA ont la possibilité de libérer une interruption. Un circuit spécial, à l'intérieur de PAULA, prend en charge la gestion de chaque source d'interruption et engendre ainsi les signaux d'interruption du 68000. Le vecteur d'interruption du processeur sera alors utilisé, ces dernières pouvant être d'un niveau 0 à 6. L'interruption non masquable (NMI), de niveau 7, n'est pas prévue. Les deux registres correspondent, l'un au registre demande d'interruption (INTREQ interrupt-request), et l'autre au registre masque d'interruption (INTENA interrupt-enable). La répartition des bits est la même dans les deux registres.

Répartition des bits des registres interrupt-request et interrupt-enable :

INTREQ	= \$09C (écriture)
INTREQR	= \$01E (lecture)
INTENA	= \$09A (écriture)
INTENAR	= \$01C (lecture)

Bit	Nom	Niveau	Fonction
15	SET/CLR		Écriture/lecture (cf. registre DMACON)
14	INTEN	(6)	Interruption autorisée
13	EXTER	6	Interruption du CIA-B ou port d'extension
12	DSKSYN	5	Valeur connue de synchronisation disque
11	RBF	5	Buffer d'entrée du port série est plein
10	AUD3	4	Données audio du canal 3 émises
9	AUD2	4	Données audio du canal 2 émises
8	AUD1	4	Données audio du canal 1 émises
7	AUD0	4	Données audio du canal 0 émises
6	BLIT	3	Blitter terminé
5	VERTB	3	Début du temps mort

Bit	Nom	Niveau	Fonction
4	COPER	3	Réserve aux interruptions du COPPER
3	PORTS	2	Interruption du CIA-A ou port d'extension
2	SOFT	1	Réserve aux interruptions du software
1	DSKBLK	1	Transfert DMA disque terminé
0	TBE	1	Buffer de sortie du port d'extension vide

Les 13 bits inférieurs caractérisent toutes les sources d'interruption. Les interruptions CIA sont rassemblées sous une seule interruption. Les bits du registre DMAREQ nous informent, dans ce cas, de quel type d'interruption il s'agit. Pour libérer une interruption processeur, il faut que les bits correspondants du registre DMAENA soit activés en même temps que le bit INTEN. Ce dernier agit comme un interrupteur général pour les 14 autres sources d'interruption, celles-ci pouvant être dissociées de chaque bit du registre INTENA. Les interruptions ne peuvent être libérées que lorsque le bit INTEN est à 1.

Si les bits INTEN des registres INTENA et INTREQ sont activés ensemble, une interruption processeur est libérée. Les numéros des vecteurs d'interruption se trouvent dans la colonne niveau du tableau. Voici encore pour rappel, les adresses des 7 niveaux de vecteurs :

Vecteur n°	Adresse DEC/HEX	Niveau d'interruption
25	100/\$64	vecteur niveau 1
26	104/\$68	vecteur niveau 2
27	108/\$6C	vecteur niveau 3
28	112/\$70	vecteur niveau 4
29	116/\$74	vecteur niveau 5
30	120/\$78	vecteur niveau 6
31	124/\$7C	vecteur niveau 7

Les interruptions qui requièrent un traitement rapide sont de haut niveau. Pour modifier les bits des deux registres, on doit accéder au registre DMACON et travailler avec le bit SET/CLR.

Après le traitement d'une interruption, les bits libérés du registre INTREQ doivent être remis par le processeur. Au contraire, ces mêmes bits ne sont pas désactivés automatiquement par lecture du registre de contrôle des interruptions du CIA.

Lorsqu'on active un bit du registre INTREQ au moyen de l'instruction MOVE, le résultat est le même que si l'interruption correspondante avait été déclenchée. C'est de cette

façon qu'on génère une interruption par le software (SOFT, bit 2). Même le COPPER ne peut qu'engendrer des interruptions par écriture sur ce registre.

Le bit 14 du registre INTREQ est particulier, en ceci qu'il n'a aucune fonction particulière comme dans le registre INTENA. Mais lorsqu'on l'active par écriture dans le registre INTREQ et que le bit INTEN du registre INTENA est à l'état HIGH, une interruption de niveau 6 est engendrée.

A chaque interruption du CIA-A, le bit 3 du registre DMAREQ est activé (bit 13 pour le CIA-B). La source d'interruption du CIA correspondant est communiquée par lecture du registre de contrôle des interruptions du CIA.

Les interruptions 3 et 13 peuvent être libérées par des cartes d'extension se trouvant sur le port.

Le bit n°5 correspond à l'interruption transparent vertical. Celle-ci se déclenche au début de chaque demi-image, lors du temps mort vertical (ligne 0) et ceci 50 fois par seconde.

Les autres types d'interruptions seront étudiés dans les chapitres correspondants.

1.5.4. Le coprocesseur COPPER

Le COPPER est un simple coprocesseur dont le rôle est la surveillance des différents registres des circuits spécialisés et l'écriture de leur contenu avec des valeurs déterminées. Le COPPER a la possibilité, par exemple, de modifier le contenu de registres à n'importe quelle position du raster de l'écran. Il peut aussi diviser ce dernier en multiples zones de résolution et de couleur différentes. Cette possibilité est employée lorsque plusieurs écrans sont utilisés. Le COPPER est décrit comme un coprocesseur, puisqu'il dispose d'un programme qui se trouve en mémoire, où des instructions y sont traitées, comme cela se passe pour un vrai processeur.

Cependant le COPPER ne reconnaît que 3 instructions différentes avec lesquelles on peut faire quantité de choses.

MOVE

L'instruction MOVE écrit une valeur immédiate dans n'importe quel registre d'un circuit spécialisé.

WAIT

L'instruction WAIT permet d'attendre que le faisceau d'électrons atteigne une position déterminée à l'écran.

SKIP

L'instruction SKIP permet de sauter l'instruction suivante, lorsque le faisceau d'électrons a atteint une position déterminée à l'écran. Avec cette instruction, on peut programmer des branchements particuliers.

Les programmes du COPPER sont appelés listes du COPPER (ou COPPER-LIST). Les instructions s'y suivent les unes après les autres et se décomposent toujours en 2 termes.

Exemples

```
WAIT (x1,y1) : attend jusqu'à ce que la position écran x1,y1 soit atteinte.
MOVE #0,$180 : écrit la valeur 0 dans le registre couleur d'arrière plan.
MOVE #1,$181 : écrit la valeur 1 dans le registre couleur 1.
WAIT (x2,y2) : attend jusqu'à ce que la position écran x2,y2 soit atteinte.
etc ...
```

La COPPER-LIST ne suffit pas seulement. En effet, il existe quelques registres importants, qui contiennent des paramètres nécessaires au COPPER.

Les registres du COPPER

Adresse	Nom	Fonction
\$080	COP1LCH	Ces deux registres contiennent à la suite
\$082	COP1LCL	l'adresse 18 bits de la première COPPER-LIST
\$084	COP2LCH	Ces deux registres contiennent à la suite
\$086	COP2LCL	l'adresse 18 bits de la deuxième COPPER-LIST
\$088	COPJMP1	Charge l'adresse de la première liste dans le compteur du COPPER
\$08A	COPJMP2	Charge l'adresse de la deuxième liste dans le compteur du COPPER
\$02E	COPCON	Ce registre ne contient qu'un bit (BIT 0). Si ce dernier est activé, le COPPER peut accéder aux adresses de registre de \$040 à \$7E (registres appartenant au Blitter).

Seul l'accès écriture est permis sur les registres du COPPER.

Les deux registres COPxLC renferment chacun l'adresse d'une COPPER-LIST. Etant donné que cette dernière est du type 18 bits, deux registres sont nécessaires. L'accès se fait avec l'instruction MOVE.L. La COPPER-LIST doit se trouver, comme toutes les données propres aux circuits spécialisés, dans les 512 Ko de la CHIP-RAM.

Le COPPER utilise un compteur programme interne comme pointeur de l'instruction actuelle. Il traite les données sous forme de deux termes ou mots. Afin que le COPPER puisse débuter à une adresse déterminée, l'adresse de départ de la liste doit être transférée dans le compteur programme. C'est l'utilisation réservée aux deux registres COPJMPx. Ils mettent en place des registres strobe, c'est-à-dire une valeur, qui ne sera pas écrite dans un registre, mais qui servira exclusivement à libérer une action particulière. Cette valeur est constante et ne dépend que de l'accès d'un tel registre. Ces deux registres servent donc, dans le COPPER, à transférer le contenu des registres correspondants COPxLC dans le compteur programme.

Si on écrit dans le registre COPJMP1, l'adresse contenue dans COP1LC sera transférée dans le compteur. En conséquence, l'exécution du programme COPPER pourra continuer. Le processus est le même pour les registres COPJMP2 et COP2LC.

Au début du temps mort vertical, à la ligne 0, le compteur programme sera chargé automatiquement avec la valeur de COP1LC, permettant au COPPER d'exécuter le même programme à chaque image.

Structure des instructions

	MOVE		WAIT		SKIP	
Bit	BW1	BW2	BW1	BW2	BW1	BW2
15	x	DW15	VP7	BFD	VP7	BFD
14	x	DW14	VP6	VM6	VP6	VM6
13	x	DW13	VP5	VM5	VP5	VM5
12	x	DW12	VP4	VM4	VP4	VM4
11	x	DW11	VP3	VM3	VP3	VM3
10	x	DW10	VP2	VM2	VP2	VM2
9	x	DW9	VP1	VM1	VP1	VM1
8	RA8	DW8	VP0	VM0	VP0	VM0
7	RA7	DW7	HP8	HM8	HP8	HM8
6	RA6	DW6	HP7	HM7	HP7	HM7
5	RA5	DW5	HP6	HM6	HP6	HM6
4	RA4	DW4	HP5	HM5	HP5	HM5
3	RA3	DW3	HP4	HM4	HP4	HM4
2	RA2	DW2	HP3	HM3	HP3	HM3
1	RA1	DW1	HP2	HM2	HP2	HM2
0	0	DW0	1	0	1	1

Légende

x	Ce bit est inutilisé. Il doit être initialisé avec 0
RA	Adresse registre
DW	Mot de donnée
VP	Position verticale du faisceau d'électrons
VM	Bit masque vertical
HP	Position horizontale du faisceau d'électrons
HM	Bit masque horizontal
BFD	Blitter Finish Disable

L'instruction MOVE

L'instruction MOVE est caractérisée par la mise à 0 du bit 0 du premier mot. Au moyen de cette instruction, il est possible d'écrire une valeur immédiate dans un registre d'un circuit spécialisé. L'adresse registre de ce dernier est représentée par les 9 premiers bits du premier mot. C'est pour cette raison que le bit 0 doit rester à 0. Le deuxième mot de l'instruction contient l'octet de donnée, qui sera écrit dans le registre de même nom.

Il existe plusieurs restrictions sur l'adresse registre. En temps normal, le Blitter n'influence pas les registres se trouvant dans la zone \$000-\$07F. Si on initialise le bit 0 du registre COPCON, le COPPER a la possibilité d'accéder aux registres se trouvant dans la zone d'adresse de \$040 à \$07F. Le COPPER pourra ainsi utiliser le Blitter. Un accès aux registres inférieurs est de toute évidence interdit (\$000 à \$03F).

L'instruction WAIT

L'instruction WAIT est caractérisée par la mise à 1 du bit 0 du premier mot et la mise à 0 du bit 0 du deuxième mot. Elle oblige le Blitter à attendre l'exécution de la prochaine instruction, jusqu'à ce que la position du faisceau d'électrons souhaitée soit atteinte. Si cette dernière est déjà dépassée alors que l'instruction apparaît, le COPPER sautera de suite à la prochaine instruction.

Cette position est déterminée suivant les lignes verticales et les colonnes horizontales. La résolution verticale correspond à une ligne du Raster. Etant donné qu'il n'est prévu que 8 bits pour la position verticale et qu'il existe 313 lignes, l'instruction WAIT ne peut différencier les 256 premières lignes des 57 restantes.

Si on veut accéder à une de ces lignes, on doit s'aider de deux instructions WAIT.

- 1) WAIT sur la ligne 255.

2) WAIT sur la ligne souhaitée, tout en négligent le bit n°9.

Les positions horizontales sont au nombre de 112, étant donné que les deux bits inférieurs, HP0 et HP1, ne peuvent être utilisés. Le mot de l'instruction MOVE ne contient que les bits HP2 à HP8, c'est-à-dire que les coordonnées horizontales d'une instruction WAIT correspondent à un point sur 4 en basse résolution.

Le deuxième terme renferme le masque bit. On peut, grâce à lui, déterminer la position horizontale et verticale du bit amené à être comparé avec le faisceau d'électrons. Seuls les bits de positions, c'est-à-dire activés dans le masque de bits, seront pris en compte. Ceci permet de nombreuses possibilités :

WAIT position verticale \$0F et masque vertical \$0F

indique que toutes les 16 lignes, la condition WAIT sera exécutée, toujours lorsque les 4 derniers bits sont à 1, étant donné que les bits 4 à 6 ne rentrent plus dans la comparaison (masque bits 4 à 6 sont à 0). Le 7ème bit de la position verticale ne se laisse pas masquer. Pour cette raison, l'exemple précédent ne fonctionne que dans la zone des lignes 0 à 127 et dans la zone des lignes 256 à 313.

Le bit BFD (Blitter Finish Disable) a la fonction suivante : si le COPPER veut utiliser une opération Blitter, il doit savoir si ce dernier a fini son travail. Si le bit BFD est désactivé, le COPPER attend à chaque instruction WAIT que le Blitter ait terminé son opération en cours, puis examine les autres instructions WAIT.

On peut l'éviter en activant le bit BFD, le COPPER ignorant alors le statut actuel du Blitter. Si on désire que le COPPER n'influence pas les registres du Blitter, on met ce bit à 1.

L'instruction SKIP

L'instruction SKIP a la même structure que l'instruction WAIT. Seul le bit 0 du deuxième terme est activé, différenciant les deux instructions. Il teste si la position du faisceau d'électrons est plus grande ou égale à celle se trouvant dans le terme de l'instruction. Si cette comparaison est positive, le COPPER saute la prochaine instruction. Sinon, il continuera le traitement du programme avec l'instruction suivante. Cette instruction permet donc la création de branchements. L'instruction suivante peut être un accès à un registre COMJMP, par lequel un saut sera libéré.

La structure d'une COPPER-LIST

Une COPPER-LIST est composée d'une suite d'instructions MOVE, WAIT et SKIP. Son adresse de départ se trouve dans le registre COPLC1. Pour arrêter une liste, la dernière instruction doit être suivie d'une instruction WAIT, avec comme paramètre, une position du faisceau impossible. Le traitement de la liste se terminera jusqu'à ce qu'une image charge à nouveau l'adresse du registre COPLC1 dans le compteur programme du COPPER. WAIT (\$0,\$FE) est un exemple d'arrêt de liste, étant donné qu'une position horizontale supérieure à \$E4 n'est pas possible.

L'interruption COPPER

Comme cela a déjà été précisé, il existe un bit dans le registre d'interruption, réservé à l'interruption du COPPER. Celle-ci peut être libérée par une instruction MOVE dans le registre INTREQ :

```
MOVE #$8010,INTREQ      ;SET/CLR et COPPER activé
```

On peut modifier tous les autres bits de ce registre de la même manière, mais le bit 4 a été prévu spécialement pour le COPPER.

Une interruption du COPPER peut servir à communiquer au processeur, qu'une position image déterminée est atteinte. Ce type d'interruption RASTER peut donc être facilement programmée.

Le DMA COPPER

Le COPPER prend ses instructions de la mémoire via un canal DMA. Il occupe les cycles pairs et est prioritaire sur le BLITTER et le 68000. Chaque instruction demande deux cycles, étant donné qu'une instruction est composée de deux termes. WAIT nécessite un cycle de plus pour attendre que la position souhaitée soit atteinte par le faisceau. Pendant cette phase d'attente, le COPPER libère le bus.

Le bit COPEN du registre DMACON autorise les accès DMA COPPER. Si ce bit est désactivé, le COPPER libère le bus et n'exécute plus d'instructions. Si on active ce bit, l'exécution du programme débute à l'adresse se trouvant dans le compteur programme. Il est d'ailleurs absolument nécessaire, avant l'accès DMA COPPER, de s'occuper de cette adresse, source d'erreur possible. L'exécution d'une zone mémoire quelconque par le COPPER peut planter le système. La séquence d'initialisation du COPPER est la suivante :

LEA \$DFF000,A5	; adresse de base du registre dans A5
MOVE.W #\$0080,DMACON(A5)	; DMA COPPER désactivé
MOVE.L #COPPERLIST,COP1LCH(A5)	; adresse de la COPPER-LIST activé
CLR.W COPJMP1(A5)	; cette adresse est transférée dans le
	; compteur du COPPER
MOVE.W #\$8080,DMACON(A5)	; DMA COPPER activé

Programme d'exemple

En conclusion, voici un programme d'exemple. Il affiche des bandes de couleur à l'aide des instructions WAIT et MOVE. Ce programme a été écrit avec l'assembleur PROFIMAT pour Amiga.

```
;*** exemple d'utilisation d'une COPPER-LIST ***
;registres circuits spécialisés

INTENA - $9A    ; registre interrupt enable (écriture)
DMACON - $96    ; registre de contrôle DMA (écriture)
COLOR00 - $180  ; registre de couleur 0
```



```
move.w #$8280,DMACON(a5)

;attente bouton gauche souris enfoncé

Wait :
btst #6,CIAAPRA    ; bit testé
bne.s Wait         ; non actif ? alors attendre

;*** fin du programme ***

;COPPER-LIST à nouveau activée

move.l #GRname,a1      ; nom de la librairie dans a1
clr.l d0              ; version 0 (la dernière)
jsr OpenLibrary(a6)   ; librairie graphique ouverte
move.l d0,a4          ; adresse de graphicbase dans a4
move.l StartList(a4),COP1LC(a5) ; chargement de l'adresse de
                                ; StartList
clr.w COPJMP1(a5)
move.w #$83E0,DMACON(a5) ; activation des canaux DMA
                        ; nécessaires
jsr Permit(a6)        ; Task Switching autorisé

;mémoire libre pour COPPER-LIST

      move.l CLadr,a1  ; libérer CLsize octets à partir de l'adresse
CLadr
      moveq #CLsize,d0
      jsr FreeMem(a6)  ; mémoire à nouveau libre
fin :
      clr.l d0        ; drapeau d'erreur désactivé
      rts             ; quitter le programme

;*****'
;      Données
;*****'

;variable

CLadr :
DC.L 0

;constante

GRname :
  DC.B "graphics.library",0
  EVEN
;COPPER-LIST

CLstart :
  DC.W COLOR00,$000F
  DC.W $780F,$FFFF
  DC.W COLOR00,$00FO
  DC.W $D70F,$FFFF
  DC.W COLOR00,$0F00
  DC.W $FFFF,$FFFF
CLend :
```

```
CLsize = CLend - CLstart  
END
```

Ce programme installe la COPPER-LIST et attend jusqu'à ce que le bouton gauche de la souris soit pressé. Il n'est pas évident, sur l'Amiga, de quitter un programme sans un reset.

Premièrement, il est nécessaire d'avoir de la mémoire. On y place les COPPER-LIST, toutes les données ayant trait aux circuits spécialisés qui doivent se trouver dans la CHIP-RAM. Si on n'est pas sûr que le programme s'y trouve, il est important d'y copier les listes.

Lorsqu'on possède un système d'exploitation multi-tâches, on ne peut pas écrire n'importe où dans la mémoire. Il faut d'abord la solliciter, ce qui sera fait au moyen de la routine AllocMem. Celle-ci met dans d0 l'adresse de la zone CHIP-RAM sollicitée, zone où la COPPER-LIST sera copiée.

Puis l'appel de 'Forbid' désactive le Task-Switching, c'est-à-dire que l'Amiga ne traite plus que notre programme. Cette précaution empêche tout autre programme de perturber le bon déroulement du programme présent.

Ensuite, le COPPER est initialisé et démarré.

Le programme teste alors le bouton gauche de la souris, en questionnant le bit de port correspondant du CIA-A.

Si le bouton est enfoncé, le processeur quitte la boucle d'attente.

Pour revenir à nouveau à l'état de départ, une liste spéciale sera chargée et démarrée dans le COPPER. Cette liste se nomme STARTUP-COPPERLIST et initialise l'écran. Son adresse se trouve dans une zone de variables du système d'exploitation réservée aux fonctions graphiques. En fin de programme, le Task-Switching sera à nouveau activé au moyen de 'Permit' et la mémoire occupée sera vidée avec 'FreeMem'.

Ce programme possède un grand nombre de fonctions peu évidentes du système d'exploitation, qui se laissent facilement éluder, quand le programme tourne régulièrement. Ce qu'il faut bien comprendre, c'est la partie concernant le COPPER. Les routines du système seront quant à elles expliquées dans les prochains chapitres.

Tapez ce programme et expérimentez-le avec la COPPER-LIST. Modifiez l'instruction WAIT, ou joignez-y une nouvelle à votre gré, et si le cœur vous en dit, essayez d'y rajouter une instruction SKIP.

Encore une précision sur la liste : les deux instructions WAIT renferment \$E comme position horizontale. Ceci correspond au début du temps mort horizontal, le COPPER l'utilisant pour modifier la couleur en dehors de la zone visible. Si on met 0 pour la position horizontale, la commutation des couleurs sera visible sur la bordure droite de l'écran.

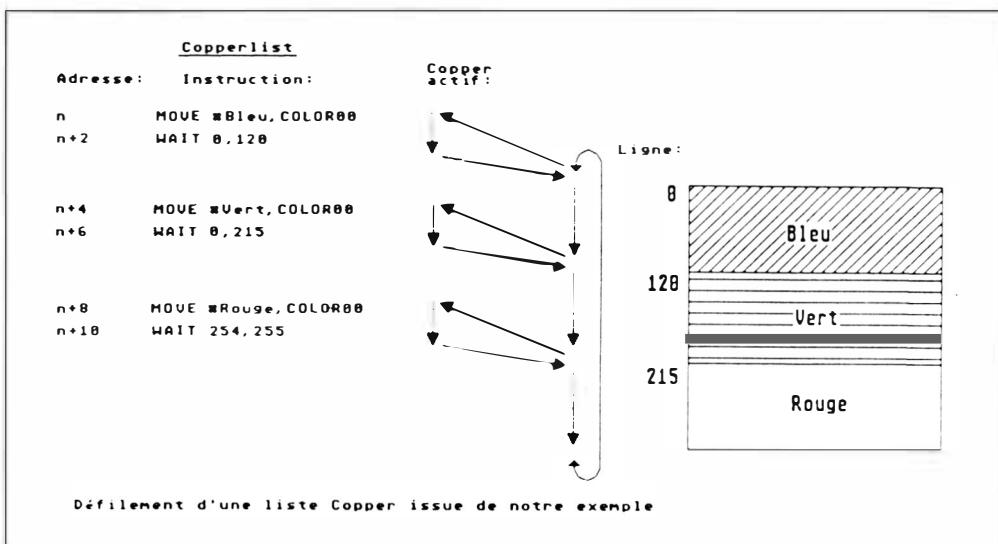


Figure 1 - 31

1.5.5. Playfields

La sortie écran de l'Amiga possède deux éléments de base : les SPRITES (lutins) et les PLAYFIELDS (champs de jeu). Ce chapitre concerne essentiellement la structure et la programmation de toutes les possibilités des playfields. Les sprites seront étudiés dans le chapitre suivant.

Le playfield est l'élément de base d'une représentation à l'écran. Il peut comprendre au minimum 1 et au maximum 6 bitplanes (la structure du bitplane est expliquée au chapitre 1.5.2). Un playfield reproduit donc une image graphique, constituée d'un nombre variable de zone mémoire ou bitplane. L'Amiga présente donc un nombre important de possibilités différentes de playfields :

- ✓ entre 2 et 4096 couleurs à l'écran.
- ✓ résolution allant de 16 sur 1 à 704 sur 625 points.
- ✓ possibilité de deux playfields entièrement indépendant l'un de l'autre.
- ✓ scrolling libre dans les deux directions (Smooth-scrolling).

Toutes ces possibilités peuvent se répartir en deux groupes.

- ① La combinaison des bitplanes pour le calcul de chaque points d'image (la reproduction des modèles de bit des bitplanes de l'image).

- ② Détermination de la forme, de la taille et de la position du ou des playfields (structure du playfield).

Les différentes possibilités de reproduction

Suivant l'utilisation d'un ou de plusieurs bitplanes, chaque point sera représenté par plus ou moins de bits. Cette valeur sera modifiée par la possibilité des 4096 couleurs, chaque pixel de l'image n'ayant qu'une couleur propre.

L'Amiga génère ses couleurs en mixant trois couleurs de base, le rouge, le vert et le bleu. Chacune de ces trois composantes pouvant prendre 16 degrés d'intensité différents, on obtient 4096 nuances différentes ($16 \times 16 \times 16 = 4096$). Chaque mémorisation de couleur nécessite 4 bits par composante, autrement dit, un total de 12 bits par nuance.

Si on veut attribuer une des 4096 couleurs à chaque point, 12 bits sont nécessaires. Etant donné que seuls 6 bits sont possibles, il est impératif de les convertir afin que le point visible puisse renfermer une couleur.

La palette couleur

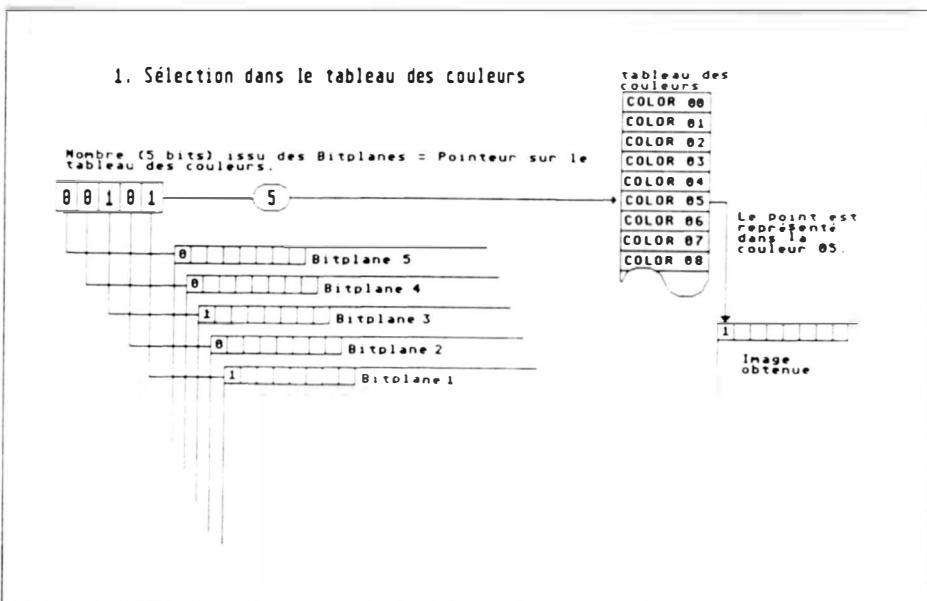


Figure 1 - 32

On emploie ainsi une palette ou tableau des couleurs. Celle-ci contient, sur l'Amiga, 32 entrées qui admettent chacune une valeur couleur 12 bits. La valeur du premier registre couleur COLOR00 correspond à la couleur de l'arrière plan ou fond et à la couleur du cadre.

Les registres couleur 0 à 31 ne sont accessibles qu'en mode écriture.

Adresse	Registre palette couleur
\$180	COLOR00
\$182	COLOR01
...	...
\$1BE	COLOR31

Structure d'un élément du tableau :

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COLORxx	x	x	x	x	R3	R2	R1	R0	G3	G2	G1	G0	B3	B2	B1	B0

R0-R3	valeur 4 bits pour la composante rouge
G0-G3	valeur 4 bits pour la composante verte
B0-B3	valeur 4 bits pour la composante bleu
Les 4 bits supérieurs indiqués par x sont inutilisés	

La valeur issue du bitplane n'est utilisée qu'en tant que pointeur. Etant donné qu'il existe 32 registres couleur, seuls 5 bitplanes peuvent être combinés dans ce mode. Le bit inférieur issu du bitplane est appelé LSB, le bit supérieur étant le MSB.

Cette façon de sélectionner les couleurs sur un tableau ne permet la présence que de 32 couleurs choisies parmi 4096 sur la même image. En mode haute résolution, seuls 4 bitplanes peuvent être activés ensemble. La limite est ici 16 couleurs. Certains registres couleur resteront alors inutilisés.

Nbre de bitplane	Nbre de couleur	Registres couleurs utilisés
1	2	COLOR00-COLOR01
2	4	COLOR00-COLOR03
3	8	COLOR00-COLOR07
4	16	COLOR00-COLOR15

Nbre de bitplane	Nbre de couleur	Registres couleurs utilisés
5	32	COLOR00-COLOR31

Le mode Extra-Half-Bright

En mode basse résolution, 6 bitplanes peuvent être utilisés. Ceci correspond à un domaine de valeurs de 2^6 ou 0 à 63. Or, seul 32 registres couleurs sont disponibles. On emploie dans ce cas un mode spécial appelé Extra-Half-bright.

Les bits inférieurs (bits 0 à 4 des plans 1 à 5) servent de pointeur sur les registres couleurs. Le contenu de ces derniers est aussitôt versé à l'image si le bit 5 (du plan 6) est à 0. Si ce bit est à 1, la valeur de la couleur sera divisée par deux, avant d'être communiquée à l'image.

La division par deux correspond à un décalage vers la droite d'un bit de la valeur couleur des trois composantes. Etant donné que chaque composante sera moitié moins importante, la couleur reproduite sera à peu près identique, seule la luminosité étant affectée. La correspondance du nom anglais de ce mode est en fait : mode particulier à luminosité réduite de moitié.

Exemple

bit n :	5 4 3 2 1 0
valeur des bitplanes :	1 0 0 1 0 0

Ceci correspond à l'entrée palette n°4 (00100 correspond à 8 en binaire).

COLOR04 doit contenir la valeur suivante (couleur orange) :

R3 R2 R1 R0 G3 G2 G1 G0 B3 B2 B1 B0
1 1 1 0 0 1 1 0 0 0 0 0 0

Comme le bit 5 est à 1, cette valeur est décalée d'un bit vers la droite :

R3 R2 R1 R0 G3 G2 G1 G0 B3 B2 B1 B0
0 1 1 1 0 0 1 1 0 0 0 0 0

Cette valeur correspond encore à la couleur orange, mais moitié moins claire.

A travers le choix des valeurs correspondantes dans les registres couleur, il est simple de caractériser la couleur d'un point parmi les 64 possibles, en mode Extra-Half-Bright. Dans le registre couleur, on trouve les nuances claires et si le bit 5 est activé, on initialisera des teintes plus foncées.

Le mode Hold-And-Modify

Ce mode permet l'affichage des 4096 couleurs simultanément. Il n'est possible qu'en mode basse résolution, étant donné qu'il nécessite 6 bitplanes. Ce mode profite du fait

que les changements de couleur d'une image sont toujours constants. Certains nécessitent des passages graduels de couleurs claires vers de plus foncées ou inversement.

En mode Hold-and-Modify, aussi nommé HAM, la couleur des premiers points sera modifiée progressivement. On peut donc favoriser des dégradés de couleurs en augmentant, par exemple, la composante bleue d'un pas à chaque pixel. On est évidemment limité dans le fait qu'on ne peut accéder qu'à une composante à la fois, et modifier d'un point à l'autre soit la valeur rouge, verte ou bleue et jamais plus en même temps. Pour obtenir un passage graduel du sombre au clair, on doit modifier les trois composantes en mixant de nombreuses fois les couleurs. Ceci ne peut se faire en mode HAM, qu'en mettant la valeur souhaitée d'une composante à un point. Trois points sont alors nécessaires.

On a toujours le compromis de modifier directement la couleur d'un pixel, en pointant une des 16 couleurs de la palette.

Comment interpréter la valeur issue des bitplanes en mode HAM ?

Les deux bits supérieurs (bits 4 et 5 des bitplanes 5 et 6) déterminent l'utilisation des 4 bits inférieurs (bitplanes 1 à 4). Si les bits 4 et 5 sont à 0, les 4 bits restants seront employés comme pointeurs sur la palette couleur, 16 couleurs pouvant être directement choisies. Si lors d'une combinaison des bits 4 et 5, le résultat diffère de 0, la valeur couleur du dernier point sera prise (à gauche du pixel actuel), deux des composantes restants inchangées, la troisième étant modifiée par les 4 bits inférieurs. Le choix entre les trois composantes dépend exclusivement des deux bits supérieurs.

Cette explication peut sembler compliquée. Voici donc un tableau qui explique l'utilisation des différentes combinaisons de bits.

Bit N° :

5	4	3	2	1	0	Fonction
0	0	C3	C2	C1	C0	Les bits C0 à C3 seront utilisés comme pointeur d'un registre couleur dans la zone COLOR00 à COLOR15, d'une manière identique à celle d'un choix de couleur normal.
0	1	B3	B2	B1	B0	Les composantes rouge et verte du dernier pixel restent inchangées. La valeur bleue précédente sera modifiée par la nouvelle valeur B3-B0.
1	0	R3	R2	R1	R0	Les composantes bleue et verte du dernier pixel restent inchangées. La valeur rouge précédente sera modifiée par la nouvelle valeur R3-R0.

5	4	3	2	1	0	Fonction
1	1	G3	G2	G1	G0	Les composantes rouge et bleue du dernier pixel restent inchangées. La valeur verte précédente sera modifiée par la nouvelle valeur G3-G0.

Pour le premier point d'une ligne, la couleur cadre (COLOR00) sera utilisée comme couleur des premiers points.

Le mode Dual-Playfield

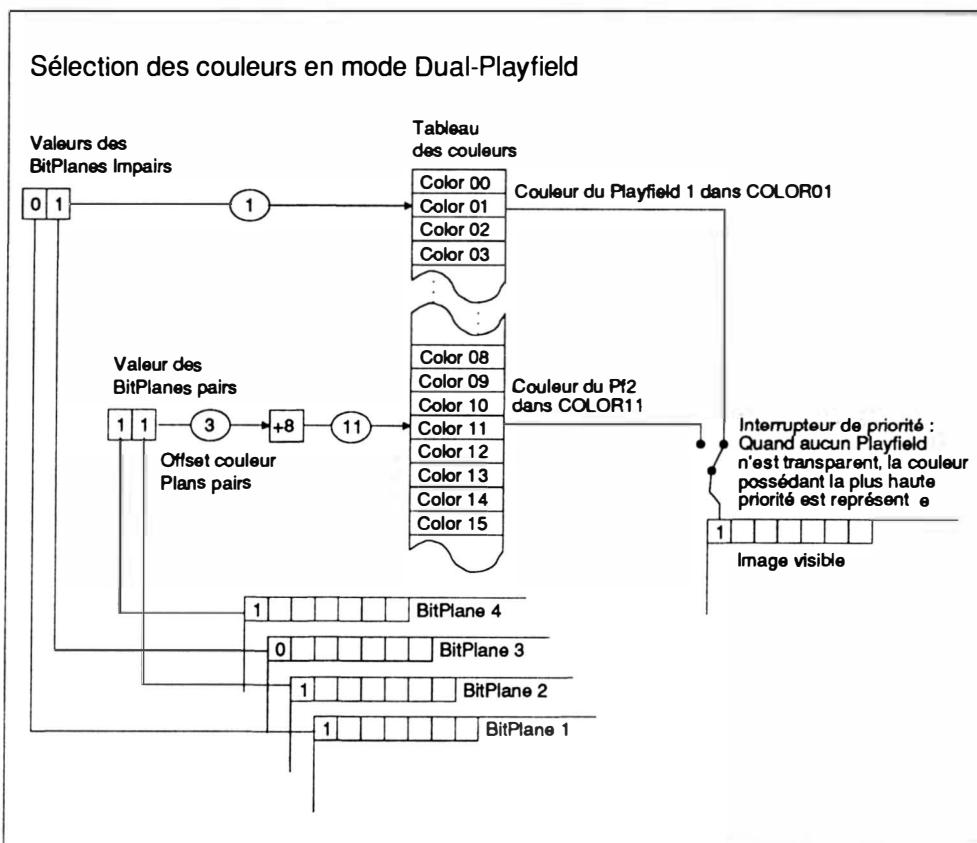


Figure 1 - 33

Les modes décrits jusqu'ici n'utilisaient, exclusivement, qu'un seul playfield. Le mode dual-playfield permet la reproduction de deux playfields, entièrement indépendants l'un

de l'autre. Il en résulte l'existence de deux images sur le même moniteur et en même temps. Celles-ci pourront être utilisées facilement, rapidement et séparément.

Ceci peut être intéressant, notamment pour les jeux où, par exemple, un effet jumelle peut être facilement créé. Le playfield, en avant plan, sera de couleur noire, et s'il reste un espace au milieu, une partie du playfield formant l'arrière plan pourra être observable. Chacun des deux playfields est reproduit avec la moitié des bitplanes activés.

Le playfield n°1 sera formé des bitplanes impairs (odd planes), le playfield n°2 étant représenté par les bitplanes pairs (even planes). Si un nombre impair de bitplanes est exploité, le playfield 1 aura à sa disposition un bitplane de plus.

La sélection des couleurs se fait de la manière suivante :

La valeur des bits d'un point, des plans impairs (playfield 1) ou des plans pairs (playfield 2), correspond au pointeur de registre couleur. Etant donné que chaque playfield comprend au maximum 3 bitplanes, seules 8 couleurs sont disponibles. Ces dernières seront accessibles, pour le champ 1, aux huit premières entrées de la palette (COLOR00-COLOR07). Un offset de 8 sera rajouté à la valeur issue des bitplanes du playfield 2, étant donné que les registres couleur accessibles se trouvent dans les positions 8 à 15.

Si un point a la valeur 0, sa couleur ne sera pas issue du registre COLOR00 (ou COLOR08), mais sera représentée avec la couleur transparente. Ceci signifie que les éléments de l'arrière plan seront directement visibles. Ces derniers pourront être soit des sprites, soit la couleur (COLOR00) de l'arrière plan.

Le mode Dual-Playfield est utilisable en mode haute résolution. Chaque champ ne possède plus alors que 4 couleurs. La répartition des registres couleur n'est pas pour autant modifiée, les 4 registres couleur supérieurs n'étant tout simplement pas utilisés (COLOR04 à 07 et COLOR12 à 15).

Répartition des bitplanes avec le mode Dual-Playfield :

Bitplanes	Plans dans le playfield 1	Plans dans le playfield 2
1	PLAN 1	aucun
2	PLAN 1	PLAN 2
3	PLAN 1 et 3	PLAN 2
4	PLAN 1 et 3	PLAN 2 et 4
5	PLAN 1,3 et 5	PLAN 2 et 4
6	PLAN 1,3 et 5	PLAN 2, 4 et 6

Sélection des couleurs avec le mode Dual-Playfield

Playfield 1				Playfield 2				
Plans	5	3	1	Plans	5	3	1	Registre couleur
0	0	0	0	Transparent	0	0	0	Transparent
0	0	1	0	COLOR01	0	0	1	COLOR09
0	1	0	0	COLOR02	0	1	0	COLOR10
0	1	1	0	COLOR03	0	1	1	COLOR11
1	0	0	0	COLOR04	1	0	0	COLOR12
1	0	1	0	COLOR05	1	0	1	COLOR13
1	1	0	0	COLOR06	1	1	0	COLOR14
1	1	1	0	COLOR07	1	1	1	COLOR15

Structure d'un playfield

Comme cela a déjà été vu, un playfield est composé d'un nombre déterminé de bitplanes. La description de ces derniers (cf. Chapitre 1.5.2) montre qu'ils sont conçus comme des zones mémoires continues où la largeur d'écran d'une ligne est représentée par un nombre de mots. Ce nombre est de 20 en plus basse résolution (320 points répartis en 20 mots de 16 points) et de 40 en plus haute (640/16). Afin d'établir la structure exacte d'un playfield, les indications suivantes sont nécessaires :

- ✓ définition de la taille souhaitée de l'image
- ✓ mise en place de la taille du bitplane
- ✓ choisir le nombre de bitplanes
- ✓ initialiser la palette couleur
- ✓ déterminer le mode (Hires, Lores, HAM, etc...)
- ✓ structurer les COPPER-LIST
- ✓ initialiser le COPPER
- ✓ activer le COPPER et le DMA bitplane

Détermination de la taille de l'image

L'Amiga autorise une position libre des coins supérieur gauche et inférieur droit du playfield. La position et la taille de l'image restent donc variables. La résolution correspond verticalement à une ligne du Raster et horizontalement, à un pixel basse résolution. Deux registres contiennent cette valeur.

DIWSTRT (Display Window Start)

contient la position verticale et horizontale de départ de la fenêtre d'écran, c'est-à-dire la ligne et la colonne où la reproduction du playfield débute.

DIWSTOP (Display window STOP)

renferme la position de fin + 1. Ceci correspond à la première ligne et colonne après le playfield.

En dehors de la zone visible, la couleur du cadre sera effective (elle correspond à la couleur d'arrière plan et est issue du registre COLOR00).

DIWSTRT \$08E (écriture seulement)

Bit n :	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	V7	V6	V5	V4	V3	V2	V1	V0	H7	H6	H5	H4	H3	H2	H1	H0

DIWSTOP \$90 (écriture seulement)

Bit n :	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	V7	V6	V5	V4	V3	V2	V1	V0	H7	H6	H5	H4	H3	H2	H1	H0

La position de départ déterminée dans DIWSTRT est limitée au premier quart de l'écran, c'est-à-dire aux lignes et colonnes comprises entre 0 et 255, étant donné que les MSB manquants, V8 et H8, sont initialisés à 0. Le cas est le même pour la position de fin, à part qu'ici H8 est mis à 1, permettant un accès à la zone comprise entre 256 et 458. Le cas de la position de fin verticale est différent. Celle-ci peut être inférieure ou supérieure à la position 256. Pour cette raison, le MSB V8 est généré suivant l'inversion du bit V7. Ainsi une position de fin comprise entre les valeurs 128 et 312 sera possible.

Si celle-ci est comprise entre 256 et 312, V7 sera mis 0 et V8 à 1. Si V7 est à 1 et V8 à 0, la position sera comprise entre 128 et 255.

La fenêtre d'écran normale possède un coin supérieur gauche de position horizontale 129 et verticale 41. Le coin inférieur droit correspond à la position 448/296, c'est-à-dire que DIWSTOP doit être initialisé avec les coordonnées 449 et 297. Les valeurs hexadécimales correspondantes de ces deux registres sont \$2981 et \$29C1. Avec ces dernières, la page de l'amiga de 640 points sur 256 (idem 320 par 256) est centrée exactement au milieu de l'écran.

Pourquoi n'utilise-t-on pas toute l'étendue possible de l'écran ?

Il y a en fait plusieurs raisons. Premièrement, un moniteur normal n'exploite pas toute l'image, la zone visible débutant après un certain nombre de lignes et de colonnes, ceci correspondant aux différents temps mort. De plus un écran cathodique n'a pas de coins carrés. Si on voulait faire correspondre la taille de l'image avec celle de l'écran, une partie de l'image se perdrait dans les coins.

Une autre limite concerne les valeurs DIWSTRT et DIWSTOP à travers le temps mort. Celui-ci correspond exactement à la zone verticale des lignes 0 à 25, la partie visible de l'écran se réduisant aux lignes 26 à 312 (\$1A à \$138). Il en est de même avec les colonnes, l'espace 30 à 106 (\$1E à \$6A) n'étant pas visible du fait du temps mort. Seules sont donc possibles les positions horizontales après 107 (\$6B).

Après avoir déterminé la position de la fenêtre d'écran, il faut établir le début et la fin des accès DMA bitplane. Afin qu'un point s'affiche à un moment désiré sur l'écran, les données des bitplanes doivent être lues de façon opportune. Verticalement, ce n'est pas un problème, étant donné que le DMA écran commence et s'arrête aux mêmes lignes que celles précisées dans les registres DIWSTRT et DIWSTOP, c'est-à-dire aux limites de la fenêtre d'écran.

Horizontalement, c'est plus compliqué. Pour qu'un point soit représenté sur l'écran, l'électronique nécessite le mot actuel de chaque bitplane. Avec 6 bitplanes en basse résolution, 8 cycles de bus sont nécessaires pour lire tous les plans.

La haute résolution ne nécessite que 4 cycles (rappel : pendant un cycle de bus, 2 points en basse résolution ou 4 points en haute résolution seront reproduits).

De plus, le hardware a encore besoin d'un demi cycle, avant que les données ne soient affichées à l'écran. L'accès DMA doit donc commencer 8.5 cycles (17 points) avant le début de l'affichage (4.5 cycles ou 9 points en haute résolution).

Le cycle de bus du premier accès DMA bitplane d'une ligne sera mis dans le registre DDFSTART (Display Data Fetch Start), le dernier sera dans DDFSTOP (Display Data Fetch Stop) :

DDFSTART \$092 (écriture seulement)

DDFSTOP \$094 (écriture seulement)

Bit n :	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fonction :	x	x	x	x	x	x	x	x	H8	H7	H6	H5	H4	H3	x	x

La résolution demande 8 cycles de bus en mode Lores, où H3 sera toujours à 0 et 4 en mode Hires. Dans ce dernier cas, H3 sert de bit de plus faible poids. La raison de la limitation de la résolution réside dans la distribution des accès DMA bitplane. En mode Lores, chaque bitplane sera lu tous les 8 cycles de bus. Pour cette raison, la valeur DDFSTART doit être un entier multiple de 8 (H1 à H3=0).

C'est la même chose en mode Hires, mais avec, cette fois-ci, 4 cycles de bus (H1 à H2=0). La différence entre DIWSTRT et DIWSTOP doit toujours être divisible par 8, étant donné que le hardware partage les lignes en zone de 8 cycles, et ceci indépendamment de la résolution.

En mode Hires, les accès DMA bitplane se déroulent encore 8 cycles après DIWSTOP, ainsi 32 points seront toujours lus.

Les valeurs correctes se déduisent comme suit :

Estimation de DDFSTART et de DDFSTOP en mode Lores

Hstart = début horizontal de la fenêtre d'écran.
DDFSTART = (Hstart/2 - 8.5) and \$FFF8
DDFSTOP = DDFSTART + (points par ligne/2 - 8)

Ceci correspond pour Hstart = \$81 et 320 points par ligne :

DDFSTART = (\$81/2 - 8.5) and \$FFFB = \$38
DDFSTOP = \$38 + (320/2 - 8) = \$D0

Estimation de DDFSTART et de DDFSTOP en mode Hires

DDFSTART = (Hstart/2 - 4.5) and \$FFFC
DDFSTOP = DDFSTART + (points par ligne/4 - 8)

Ceci correspond pour Hstart = \$81 et 640 points par ligne :

DDFSTART = (\$81/2 - 4.5) and \$FFFC = \$3C
DDFSTOP = \$3C + (640/4 - 8) = \$D4

DDFSTART ne doit pas être inférieur à \$18. La limite maximale de DDFSTOP est \$D8. Une valeur inférieure à \$28 n'a aucun sens, étant donné que les points seront reproduits pendant le temps mort horizontal, ce qui n'est pas possible (exception faite pour le scrolling). Comme les positions DDFSTART inférieures à \$34 chevauchent les cycles DMA des bitplanes et des sprites, certains sprites ne sont pas représentables après DDFSTART.

Décalage de la fenêtre d'écran

Si on désire décaler horizontalement la fenêtre d'écran au moyen de DIWSTART et STOP, il peut arriver que la différence entre DIWSTART et DDFSTART ne soit pas exactement égale à 8.5 cycles de bus (17 points), étant donné qu'on peut déterminer DDFSTART suivant un pas de 8 cycles. A ce moment là, une partie du premier mot de données disparaîtra dans la zone gauche, près de la limite de la fenêtre d'écran. Afin d'éviter cela, il y a la possibilité de décaler les données vers la droite, avant la sortie à l'écran, pour qu'elles correspondent avec le début de la fenêtre d'écran. La programmation de ce décalage sera expliquée dans le paragraphe concernant le scrolling.

Détermination des adresses Bitmap

Les valeurs dans DDFSTART et DDFSTOP déterminent le nombre de mots de données par ligne. Pour chaque bitmap, l'adresse de départ doit être établie afin que le contrôleur DMA puisse savoir à partir de quel endroit les données de point doivent être lues. 12 registres renferment ces adresses. Deux registres (BPLxPTH et BPLxPTL)

correspondent, à chaque fois, au même bitplane x. Ceux-ci peuvent être dénommés, plus simplement, BPLxPT (bitplane x pointer, pointeur du bitplane x).

Adresse	Nom	Fonction	
\$0E0	BPL1PTH	adresse de départ	(bits 16-18)
\$0E2	BPL1PTL	du bitplane 1	(bit 0-15)
\$0E4	BPL2PTH	adresse de départ	(bits 16-18)
\$0E6	BPL2PTL	du bitplane 2	(bit 0-15)
\$0E8	BPL3PTH	adresse de départ	(bits 16-18)
\$0EA	BPL3PTL	du bitplane 3	(bit 0-15)
\$0EC	BPL4PTH	adresse de départ	(bits 16-18)
\$0F0	BPL4PTL	du bitplane 4	(bit 0-15)
\$0F2	BPL5PTH	adresse de départ	(bits 16-18)
\$0F4	BPL5PTL	du bitplane 5	(bit 0-15)
\$0F6	BPL6PTH	adresse de départ	(bits 16-18)
\$0F8	BPL6PTL	du bitplane 6	(bit 0-15)

Le contrôleur DMA procède de la façon suivante lors de la reproduction des bitplanes : le DMA bitplane reste inactif, jusqu'à ce que la première ligne de la fenêtre d'écran soit atteinte (DIWSTRT).

Puis, il cherche dans DDFSTRT la colonne déterminée par les mots de données des différents bitplanes, tout en respectant le timing observé sur le schéma du fonctionnement du Raster. Il utilise BPLxPT comme pointeur sur les données se trouvant dans la CHIP-RAM. Après lecture de chaque mot, BPLxPT augmente d'un mot. Les mots lus passent dans le registre BPLxDAT. Ces derniers seront utilisés uniquement par le canal DMA. Si les 6 registres BPLxDAT sont occupés par les mots de données issus des bitplanes, les données sont transférées, bit à bit, vers la logique vidéo de DENISE, qui, suivant le choix du mode, sélectionne l'une des 4096 couleurs et l'affiche à l'écran.

En atteignant DDFSTOP, le DMA bitplane marque une pause jusqu'à DDFSTRT de la prochaine ligne, ce processus étant répété jusqu'à la fin de la dernière ligne de la fenêtre écran (DIWSTOP).

BPLxPT pointe alors sur le premier mot du bitplane. Etant donné que BPLxPT doit pointer à nouveau sur le premier mot du bitplane correspondant de la prochaine image, il doit être réinitialisé. C'est le rôle rapide et sans problème du COPPER. Une COPPER-LIST, pour un playfield à 4 bitplanes, est du type :

AdrPlanexH = adresse du plan x, bits 16-18
 AdrPlanexL = adresse du plan x, bits 0-15

```

MOVE #AdrPlane1H,BPL1PTH      ;pointeur sur le bitplane 1
MOVE #AdrPlane1L,BPL1PTL      ;initialisation
MOVE #AdrPlane2H,BPL2PTH      ;pointeur sur le bitplane 2
MOVE #AdrPlane2L,BPL2PTL      ;initialisation
MOVE #AdrPlane3H,BPL3PTH      ;pointeur sur le bitplane 3
MOVE #AdrPlane3L,BPL3PTL      ;initialisation
MOVE #AdrPlane4H,BPL4PTH      ;pointeur sur le bitplane 4
MOVE #AdrPlane4L,BPL4PTL      ;initialisation
WAIT ($FF,$FE)                ;fin de la COPPER-LIST (attente d'une
                                ;position impossible à l'écran)

```

La réinitialisation de BPLxPT est absolument nécessaire. Si on ne veut pas utiliser de listes, on doit laisser le processeur achever le travail, au moyen d'une interruption écran vide vertical.

Scrolling et grande taille des playfields

Tous les playfields étudiés jusqu'ici ont toujours eu la taille exacte de l'écran. Pourtant, il est souvent judicieux d'avoir un playfield de grande taille en mémoire, dont seule une partie est visible à l'écran, et que l'on peut décaler dans n'importe quelle direction. Ceci peut se faire sans problème sur l'Amiga. Le schéma suivant le montre, dans la direction des axes X et Y.

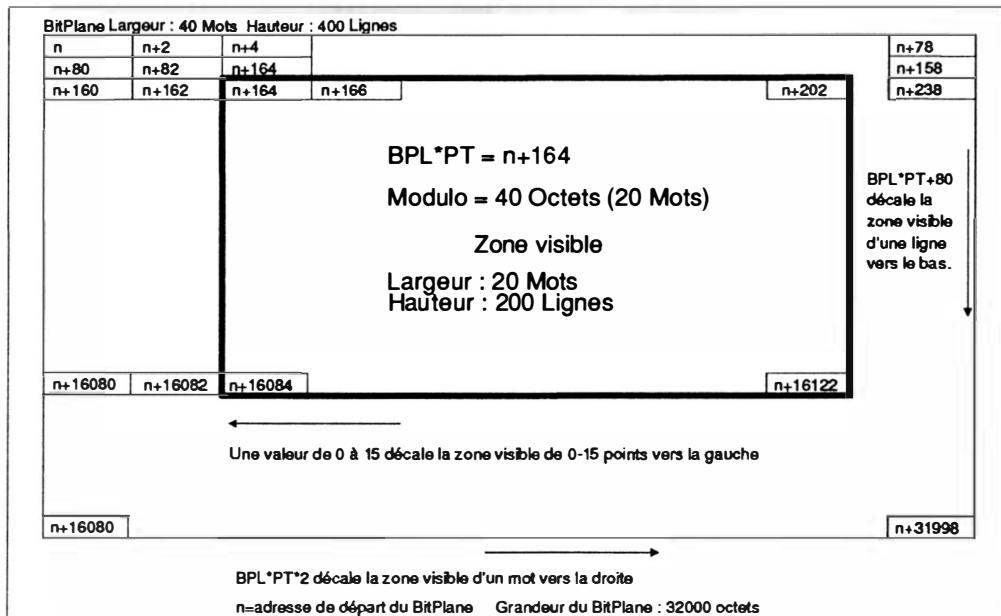


Figure 1 - 34

Hauteur du playfield et scrolling vertical

L'extension verticale est facilement réalisable. On initialise les bitplanes nécessaires en mémoire, mais cette fois-ci avec plus de lignes que l'écran ne peut reproduire. Par exemple, une fenêtre standard comporte 256 lignes, qu'on peut doubler pour former un playfield étendu en hauteur sur 512 lignes. Afin d'amener ce playfield sur la fenêtre d'écran, il faut modifier la valeur de BPLxPT.

Si la fenêtre d'écran doit reproduire la zone s'étalant de la ligne 100 à la ligne 356, BPLxPT doit pointer le premier mot de la ligne 100. Une résolution de 320 points correspond à 20 mots (40 octets) par ligne. Si on multiplie par 100 lignes, le résultat sera 4000. En rajoutant l'adresse de départ du playfield, on obtient alors la valeur que doit renfermer BPLxPT. Pour déplacer le playfield dans la fenêtre d'écran, on modifie cette valeur avec chaque image, d'une ou plusieurs lignes, suivant la vitesse désirée. Etant donné qu'on ne peut modifier BPLxPT qu'en dehors de la zone visible, on doit se servir de la COPPER-LIST énoncée plus haut.

On peut alors modifier l'adresse de n'importe quel point dans la COPPER-LIST, ce dernier l'écrivant automatiquement dans le registre BPLxPT. On doit cependant faire attention à ne pas modifier les listes pendant que le COPPER accède à ces instructions. En effet, si à ce moment précis, on transforme un mot d'une adresse, le COPPER lira l'ensemble, c'est-à-dire une adresse erronée.

Largeur du playfield et scrolling horizontal

Pour le scrolling horizontal et les playfields d'extension large, des registres spéciaux sont à l'honneur (exclusivement accessibles en écriture).

\$108 BPL1MOD valeur modulo des bitplanes impairs
\$10A BPL2MOD valeur modulo des bitplanes pairs

BPLCON1 \$102

Bit n :	15-8	7	9	6	5	4	2	1	0
Fonction :	inutilisé	P2H3	P2H2	P2H1	P2H0	P1H3	P1H2	P1H1	P1H0

P1H0-P1H3 position des plans pairs (4 bits)
P2H0-P2H3 position des plans impairs (4 bits)

La valeur modulo, issue des registres BPLxMOD autorise la zone mémoire du coin droit. Ce principe sera encore souvent utilisé par le hardware de l'Amiga. Il permet la définition d'une petite zone mémoire à l'intérieur d'une grande zone où les lignes et colonnes sont définies. Prenons par exemple, une grande zone mémoire de 640 points de large, et d'une hauteur de 256 points, constituant un playfield. Ceci correspond en fait à 256 lignes de 40 mots. La petite zone s'identifie donc à la taille normale de la fenêtre d'écran, c'est-à-dire 320 points par 200, ou 20 mots par lignes. Le problème est qu'après l'affichage d'une ligne, BPLxPT est augmenté de 20 mots. Pour pointer le début de la

prochaine ligne de notre playfield, il faudrait qu'il soit en fait augmenté de 40 mots. On est donc obligé de rajouter une valeur de 20 mots à BPLxPT après reproduction de chaque ligne. Cette addition peut être réalisée automatiquement par l'Amiga. Il suffit d'écrire la différence entre les deux

longueurs de lignes dans le registre modulo, cette valeur étant ajoutée automatiquement à la valeur de BPLxPT après affichage de chaque ligne.

Largeur du playfield : 80 octets (40 mots)

Largeur de la fenêtre d'écran : 40 octets (20 mots)

Valeur modulo : 40 octets (cette valeur doit toujours être un nombre pair)

Start = adresse de départ de la première ligne du playfield.

Affichage de la 1ère ligne :

Mot :	0	1	2	3	...	19
BPLxPT :	Start	Start+2	Start+4	Start+6		Start+38

Après l'affichage du dernier mot, BPLxPT sera augmenté d'un mot :

$BPLxPT = Start + 40$

A la fin de la ligne, la valeur modulo sera rajoutée au contenu du registre BPLxPT :

$BPLxPT = BPLxPT + \text{modulo}$ $BPLxPT = Start+40 + 40 = Start+80$

Affichage de la 2ème ligne :

Mot :	0	1	2	3	...	19
BPLxPT :	Start+80	Start+82	Start+84	Start+86		Start+118

et ainsi de suite ...

En haut, la moitié gauche du grand champ sera reproduite dans la fenêtre écran. Si on veut débuter à une position horizontale différente, il suffit de rajouter à la valeur de départ de BPLxPT le nombre de mots souhaité, la valeur modulo restant la même.

Si la valeur de départ est la même que plus haut, BPLxPT aura une valeur de départ de Start+40, avec laquelle la première moitié du playfield sera pointée.

Affichage de la première ligne :

Mot :	0	1	2	3	...	19
BPLxPT :	Start+40	Start+42	Start+44	Start+46		Start+78

Après affichage du dernier mot :

$BPLxPT = Start + 80$

Addition de la valeur modulo

$BPLxPT = BPLxPT + \text{modulo}$ $BPLxPT = Start+80 + 40 = Start+120$

Affichage de la deuxième ligne :

Mot	0	1	2	3	...	19
BPLxPT	Start+120	Start+122	Start+124	Start+126		
	Start+158					

Les valeurs modulo des bitplanes pairs et impairs se laissent initialiser séparément. Ceci autorise la présence de deux playfields de grande extension, dans le mode Dual-Playfield. Si on ne travaille pas dans ce mode, il suffit de mettre la même valeur dans les fichiers BPLxMOD.

A l'aide des registres BPLxPT et BPLxMOD, l'écran peut être décalé d'un pas de 16 points. Un scrolling plus fin, d'un pas d'un point, est possible avec le registre BPLCON1. Les 4 bits inférieurs renferment la valeur du scrolling des plans pairs, les bits 4 à 7, celle des plans impairs. Ces valeurs de scrolling ralentissent la sortie des données de points des plans correspondant. Si ces valeurs sont nulles, les données seront émises tous les 8.5 cycles (4.5 en Hires) après la position DDFSTRT et si, au contraire, elles ne sont pas nulles, les données apparaîtront, le nombre de points contenus dans BPLCON1 plus tard, c'est-à-dire décalé de la valeur du nombre de points, vers la droite.

On peut obtenir un scrolling très fluide vers la droite du contenu de l'écran lorsqu'on augmente la valeur de BPLCON1 progressivement, de 0 à 15, puis en la remettant à 0. On aura diminué la valeur de BPLxPT d'un mot.

Le scrolling vers la gauche est autorisé lorsqu'on décrémente la valeur du scrolling, de 15 à 0 par exemple. On aura augmenté la valeur de BPLxPT d'un mot. BPLCON1 doit être modifié en dehors de la zone visible. On utilisera à cet effet, soit l'interruption 'écran vide vertical' du processeur, soit le COPPER. On pourra alors modifier la valeur dans la COPPER-LIST à tout moment, cette dernière étant toujours transférée dans le registre BPLCON1, lors du temps mort vertical.

Si on décale l'image vers la droite au moyen de la valeur BPLCON1, les points en trop seront effacés correctement sur le bord gauche, mais les nouveaux points sur la droite n'apparaîtront pas, étant donné qu'aucune donnée de point ne sera lue. Afin d'éviter cela, la valeur de DDFSTRT doit être rallongée de 8 cycles de bus (4 cycles en mode Hires), par rapport au début normal.

On calcule, comme auparavant, la valeur DDFSTRT de la fenêtre d'écran, en la réduisant de 8 (4 Hires). A la valeur normale \$38 se substituera la nouvelle valeur \$30 (le sprite 7 sera désactivé). Ce mot à lire en plus n'est normalement pas visible. Lorsque la valeur du scrolling est à 0, ses points s'affichent sur une position libre, le bord gauche de la fenêtre d'écran. Si celle-ci a une largeur de 320 points, 21 mots de donnée seront lus, à la place des 20 normaux. Il faudra en tenir compte pour l'évaluation des bitplanes et de la valeur modulo.

A l'aide de la valeur du scrolling, on peut positionner la fenêtre d'écran n'importe où horizontalement. Si la différence entre DIWSTRT et DFFSTRT est supérieure à 17 points, les données seront décalées vers la droite du montant de la différence.

Le mode Interlace

Malgré le double affichage du nombre des lignes en mode interlace, ce dernier ne diffère de la représentation normale, en technique de programmation, que par une modification de la valeur modulo et par une nouvelle COPPER-LIST. Comme cela a été expliqué au chapitre 1.5.2, le mode interlace se caractérise par l'alternance d'affichage des lignes paires et impaires.

Comme on peut reproduire, normalement, un playfield interlace en mémoire, on fixe la valeur modulo du nombre de mots d'une ligne. Après l'affichage d'une ligne, la valeur de BPLxPT sera augmentée de la longueur de cette dernière, ce qui correspond à un saut d'une ligne. A chaque image, seules toutes les deux lignes sont représentées. On devra donc initialiser BPLxPT en concordance avec le type de demi-image qui sera affiché, c'est-à-dire en alternant suivant la première ou la deuxième ligne du playfield, afin que, soit les lignes impaires, soit les lignes paires puissent être pointées. Dans le cas Long Frame (lignes impaires), BPLxPT sera fixé sur la ligne 1 ; dans le cas Short Frame (lignes paires), BPLxPT sera fixé sur la ligne 2. La COPPER-LIST est un peu plus compliquée pour un playfield interlace. En effet, 2 listes sont nécessaires pour les deux types de demi-image, afin de pouvoir être alternés avec chaque image.

COPPER-LIST pour un playfield en mode interlace :

```
ligne1 = adresse de la première ligne du bitplane
ligne2 = adresse de la deuxième ligne du bitplane
```

Copper1 :

```
MOVE #ligne1Hi,BPLxPTH    :pointeur BPLxPTH sur
MOVE #ligne1Lo,BPLxPTL    :sur la première ligne
... autres instructions du COPPER
MOVE #Copper2Hi,COP1LCH  :adresse de la liste
MOVE #Copper2Lo,COP1LCL  :initialisée à Copper2
WAIT ($FF,$FE)           :fin de la première liste
```

Copper2 :

```
MOVE #ligne2Hi,BPLxPTH    :pointeur BPLxPTH sur
MOVE #ligne2Lo,BPLxPTL    :sur la deuxième ligne
... autres instructions du COPPER
MOVE #Copper1Hi,COP1LCH  :adresse de la liste
MOVE #Copper1Lo,COP1LCL  :initialisée à Copper1
WAIT ($FF,$FE)           :fin de la deuxième liste
```

Le COPPER change de liste après chaque image, grâce à la liste d'instructions se trouvant à la fin, qui charge l'adresse de l'autre liste dans COP1LC. Cette adresse sera chargée automatiquement dans le compteur programme du COPPER, au début de chaque image. L'initialisation du mode interlace doit être fait avec soin, afin que la COPPER-LIST réservée aux lignes impaires traite bien une Long Frames :

- ✓ Initialiser COP1LC avec Copper1.
- ✓ Le bit LOF (bit 15) du registre VPOS (\$2A) doit être mis à 0. Ceci assure la présence d'une LongFrame comme première demi-image, après démarrage du mode interlace et le passage de la liste sur Copper1. Le bit LOF sera inversé à chaque demi-image. Si on le fixe à 0, au début de la prochaine demi-image, il passera à 1.

- ✓ Mode interlace sélectionné.
- ✓ Attente de la première ligne de la prochaine image (ligne 0).
- ✓ DMA COPPER activé.

Toutes les autres fonctions des registres restent inchangées en mode interlace. Toutes les instructions concernent les lignes des demi-images actuelles (0-312 pour LongFrame et 0-311 pour ShortFrame). Si on sélectionne le mode interlace sans modifier les autres registres, on remarque un léger tremblement de l'image.

Ceci est dû au fait que chaque ligne d'une demi-image est remplacée par une autre. De plus, les deux images contiennent les mêmes données. Ce n'est qu'au moyen des COPPER-LIST qu'on arrive à doubler le nombre de lignes, tout en modifiant la taille des bitplanes et en choisissant la valeur modulo correspondante, ceci de telle façon que chaque demi-image sera représentée par d'autres données.

Le mode interlace est accompagné d'un fort scintillement, étant donné que chaque ligne n'est rafraîchie que toutes les deux images, c'est-à-dire 25 fois par seconde. On peut atténuer ce scintillement en choisissant des couleurs différentes avec le moins de contrastes possibles, l'œil de l'homme n'arrivant que difficilement à repérer un changement d'images aux couleurs faiblement contrastées.

Les registres de contrôle

Pour activer les différents modes, il existe trois registres de contrôle BPLCON0 à BPLCON2. BPLCON1 contient la valeur du scrolling. Les deux autres sont structurés de la manière suivante :

BPLCON0 \$100

Bit n°	Nom	Fonction
15	Hires	Mode haute résolution sélectionné (Hires = 1).
14	BPU2	Les trois bits BPUx forment ensemble
13	BPU1	un compteur 3 bits
12	BPU0	nombre de bitplanes utilisés (0 à 6).
11	HOMOD	Mode Hold and Modify sélectionné (HOMOD = 1).
10	DBPLF	Mode Dual-Playfield sélectionné (DBPLF = 1).
9	COLOR	Sortie Vidéo couleur (COLOR = 1).
8	GAUD	Genlock audio sélectionné (GAUD = 1).
7-4	----	Inutilisés
3	LPEN	Entrée crayon lumineux activé (LPEN = 1).

Bit n°	Nom	Fonction
2	LACE	Mode interlace sélectionné (LACE = 1).
1	ERSY	Synchronisation externe sélectionnée (ERSY = 1).
0	----	Inutilisé

Hires

Ce bit sert à la sélection du mode haute résolution (Hires, 640 points par ligne).

BPL0-BPL2

Ces trois bits forment un compteur qui renferme le nombre de bitplanes activés. Seules les valeurs comprises entre 0 et 6 sont possibles.

HOMOD et DBPLF

Ces deux bits sélectionnent les modes correspondants. Ces derniers ne doivent pas être activés en même temps. Le mode Extra-Half-Bright sera sélectionné automatiquement, lorsque tous les 6 bitplanes seront activés.

LACE

Lorsque le bit LACE sera initialisé, le bit LOF-Frame du registre VPOS, sera inversé au début de chaque nouvelle image, afin que le changement entre Long et Short Frame puisse avoir lieu.

COLOR

Le bit COLOR active la sortie du signal couleur d'AGNUS. Lorsque ce dernier délivre ce signal, le mixeur vidéo peut générer un signal vidéo couleur, sinon le signal reste noir et blanc. La sortie RGB ne sera pas influencée par ce signal.

ERSY

Le bit ERSY commute les connexions des signaux de synchronisation horizontale et verticale, du mode sortie en mode entrée. Ainsi l'image de l'Amiga pourra être synchronisée par un signal extérieur.

L'interface GENLOCK utilise ce bit pour pouvoir mixer l'image de l'Amiga avec n'importe quelle image vidéo. Le bit GAUD est aussi utilisé par cette interface (cf. section 1.3.2).

BPLCON2 \$104

Bit n :	15-7	6	5	4	3	2	1	0
Fonction :	inuti.	PF2PRI	PF2PR2	PF2PR1	PF2PRO	PF1P2	PF1P1	
	PF1PO							

PF2P0-PF2P2 et PF1P0-PF1P2 déterminent la priorité des sprites dans le traitement d'un playfield (cf. prochain chapitre).

Si le bit PF2PRI est activé, les plans pairs sont prioritaires par rapport aux plans impairs, c'est-à-dire qu'ils s'affichent devant les plans impairs. Ce bit n'a de résultats visibles qu'en mode Dual-Playfield.

Affichage d'écran activé

Après avoir initialisé les registres vus plus haut avec les valeurs désirées, on doit encore activer le canal DMA bitplane et dans le cas où on utilise le COPPER, le canal DMA COPPER. C'est le rôle de l'instruction MOVE suivante, qui active les bits DMAEN, BPLEN et COPEN du registre de contrôle DMACON :

```
MOVE.W #$8310,$DFF096
```

Exemples de programmes

Programme 1 : Démonstration du mode Extra-Half-Bright

Ce programme génère un playfield en mode basse résolution, c'est-à-dire 320 points par 200. Il utilise 6 bitplanes, permettant ainsi la sélection du mode Extra-Half-Bright. Au départ, le programme gère la mémoire nécessaire. Etant donné que les adresses des bitplanes sont maintenant connues, la COPPER-LIST ne sera pas issue du programme, mais directement placée dans la CHIP-RAM. Elle ne renferme que les instructions servant à initialiser le registre BPLxPT.

Afin que l'on puisse voir les 64 couleurs possibles, le programme dessine à des positions déterminées, des blocs de 16 points sur 16, dans toutes les couleurs. Le registre VHPOS sera utilisé comme générateur de nombres aléatoires.

```
;*** Dual-Playfield et Scrolling ***
;registres des circuits spécialisés
INTENA - $9A      ; registre d'autorisation des interruptions (écriture)
DMACON - $96      ; registre demande d'interruptions (lecture)
COLOR00 - $180    ; registre couleur 0
VHPOSR - $6       ; position du faisceau d'électrons (lecture)

;registre COPPER
COP1LC - $80      ; adresse de la 1ère liste
COP2LC - $84      ; adresse de la 2ème liste
COPJMP1 - $88     ; saut à la 1ère liste
```

```
COPJMP2 = $8A      ; saut à la 2ème liste

;registre bitplane

BPLCON0 = $100    ; registre 0 de contrôle bitplane
BPLCON1 = $102    ; 1 (valeur pour le scrolling)
BPLCON2 = $104    ; 2 (sprite <> priorité playfield)
BPL1PTH = $0E0    ; pointeur sur le
BPL1PTL = $0E2    ; premier bitplane
BPL1MOD = $108    ; valeur modulo pour les plans impairs
BPL2MOD = $10A    ; valeur modulo pour les plans pairs
DIWSTRT = $08E    ; début de la fenêtre écran
DIWSTOP = $090    ; fin de la fenêtre écran
DDFSTRT = $092    ; début DMA bitplane
DDFSTOP = $094    ; fin DMA bitplane

;CIA-A registre port A (bouton souris)

CIAAPRA = $BFE001

;Exec Library Base Offsets

OpenLibrary = -30-522   ; LibName, Version / a1, d0
Forbid      = -30-102
Permit      = -30-108
AllocMem    = -30-168   ; ByteSize, Requirements / d0, d1
FreeMem     = -30-180   ; MemoryBlock, ByteSize / a1, d0

;graphics base

StartList = 38

;autres labels

Execbase = 4
Planesize = 40*256      ; taille des bitplanes : 40 octets sur 256
lignes
CLsize    = 13*4         ; la COPPER-LIST contient 13 instructions
Chip      = 2             ; CHIP-RAM sollicitée
Clear     = Chip+$10000  ; CHIP-RAM réservée

Start :

;Allouer de la mémoire pour les bitplanes

move.l Execbase,a6
move.l #Planesize*6,d0      ; mémoire nécessaire à tous les plans
move.l #Clear,d1            ; la mémoire doit être remplie de 0
jsr AllocMem(a6)           ; mémoire sollicitée
move.l d0,Planeadr          ; mise en mémoire de l'adresse du 1er plan
beq Fin                     ; Erreur -> FIN

;Solliciter la mémoire pour la COPPER-LIST

moveq #CLsize,d0            ; taille de la COPPER-LIST
moveq #Chip,d1
jsr AllocMem(a6)
move.l d0,CLadr
beq FreePlane                ; erreur -> RAM bitplane libérée
```

```

;Installation de la COPPER-LIST

moveq #5,d4          ; 6 plans - 6 boucles
move.l d0,a0          ; adresse de la COPPER-LIST dans a0
move.l Planeadr,d1
move.w #BPL1PTH,d3    ; premier registre dans d3

MakeCL :
move.w d3,(a0)+      ; BPLxPTH dans la RAM
addq.w #2,d3          ; prochain registre
swap d1
move.w d1,(a0)+      ; Haut Mot de l'adresse plan dans la RAM
move.w d3,(a0)+      ; BPLxPTL dans la RAM
addq.w #2,d3          ; prochain registre
swap d1
move.w d1,(a0)+      ; Bas Mot de l'adresse plan dans la RAM
add.l #Planesize,d1   ; adresse du prochain plan calculée

dbf d4,MakeCL

move.l #$FFFFFFFE,(a0) : fin de la COPPER-LIST

;DMA activé et Task Switching bloqué

jsr Forbid(a6)
lea $DFF000,a5
move.w #$03E0,DMACON(A5)

;COPPER initialisé

move.l Caddr,COP1LC(a5)
clr.w COPJMP1(a5)

;création de la palette couleur

moveq #31,d0          ; compteur des registres couleur
lea COLOR00(a5).a1
moveq #1,d1            ; première couleur
SetTab:
move.w d1.(a1)+        ; couleur dans le registre correspondant
mulu #3.d1             ; calcul de la prochaine couleur
dbf d0,SetTab

;playfield initialisé

move.w #$3081,DIWSTRT(a5)   ; valeurs standard pour
move.w #$30C1,DIWSTOP(a5)    ; la fenêtre écran
move.w #$0038,DDFSTRT(a5)    ; et le DMA bitplane
move.w #$00D0,DDFSTOP(a5)
move.w #%0110001000000000,BPLCON0(a5) ; 6 bitplanes
clr.w BPLCON1(a5)          ; pas de scrolling
clr.w BPLCON2(a5)          ; pas de priorité
clr.w BPL1MOD(a5)           ; modulo de tous les plans - 0
clr.w BPL2MOD(a5)

;DMA activé

move.w #$8380,DMACON(a5)

```

```

;Bitplane modifié

moveq #40,d5          ; octets par ligne
clr.l d2              ; commencer avec la couleur 0

Loop :

clr.l d0
move.w VHPUSR(a5),d0      ; valeur aléatoire dans d0
and.w #$3FFE,d0          ; éliminer les bits superflus

cmp.w #$2580,d0          ; comparaison à la taille d'un plan
bcs cont                ; si plus petit, alors on continue
and.w #$1FFE,d0          ; sinon, effacer le bit supérieur

Cont :
move.l Planeadr,a4
add.l d0,a4
moveq #5,d4
move.l d2,d3

Block :
clr.l d1
lsr #1,d3              ; un bit issu du numéro de couleur dans le
flag x
negx.w d1               ; d1 égalisé avec le flag x
moveq #15,d0             ; 16 lignes par bloc
move.l a4,a3              ; adresse bloc dans le registre de travail

Fill :
move.w d1,(a3)           ; mot dans le bitplane
add.l d5,a3              ; calcul de la prochaine ligne
dbf d0,Fill

add.l #Planesize,a4       ; prochain bitplane
dbf d4,Block

addq.b #1,d2              ; prochaine couleur
btst #6,CIAAPRA          ; test bouton souris
bne Loop                 ; sinon -> on continue

;*** fin de programme ***

;ancienne COPPER-LIST activée

move.l #GRname,a1          ; paramètre pour initialiser OpenLibrary
clr.l d0
jsr OpenLibrary(a6)         ; ouverture de la librairie graphique
move.l d0,a4
move.l StartList(a4),COP1LC(a5) ; adresse du début de la liste
clr.w COPJMP1(a5)
move.w #$8060,DMACON(a5)    ; réinitialisation du canal DMA
jsr Permit(a6)              ; Task Switching autorisé

;mémoire libérée des COPPER-LIST

move.l Cladr,a1            ; paramètre initialisant FreeMem
moveq #Clsize,d0           ; mémoire libérée
jsr FreeMem(a6)

```

```

;mémoire des bitplanes libérée

FreePlane :
    move.l Planeadr,a1
    move.l #Planesize*6,d0
    jsr FreeMem(a6)

Fin :
    clr.l d0
    rts           ; quitter le programme

;variables

CLaddr:   dc.l 0
Planeadr: dc.l 0

;constante

GRname:   dc.b "graphics.library",0

END

;fin du programme

```

Programme 2 : Dual-Playfield et Scrolling

Ce programme utilise plusieurs effets en même temps : premièrement, il définit un écran Dual-Playfield avec un bitplane basse résolution par playfield. Ensuite, il augmente la taille de la fenêtre d'écran, afin que les bordures ne soient plus visibles. Enfin, il scrolle le playfield 1 horizontalement et le playfield 2 verticalement.

Au début et à la fin, on utilisera les mêmes routines d'occupation de la mémoire que plus haut.

Les deux playfields seront occupés par un damier, avec des cases de 16 points sur 16.

La grande boucle du programme qui produit le scrolling attend tout d'abord une ligne du temps mort vertical, où le système d'exploitation a déjà traité toutes les éventuelles routines d'interruption et où le COPPER a déjà initialisé le pointeur BPLxPT. Puis il compte le nombre de lignes à scroller, calcule le nouveau pointeur pour le champ deux, et l'écrit dans la COPPER-LIST.

Les positions de scrolling sont obtenues par séparation du compteur scroll entre les 4 bits inférieurs et le reste. Les 4 bits de plus faible poids seront transférés dans le registre BPLCON1, où ils représenteront la valeur du scrolling. A partir du 5ème bit, le nouveau pointeur BPLxPT sera calculé et copié dans la COPPER-LIST.

Les deux compteurs de scrolling, vertical et horizontal, seront augmentés progressivement de 0 à 31 et puis remis à 0. Ceci aura pour résultat un effet de scrolling très fluide, étant donné que le contenu des playfields, suivant le modèle utilisé, sera répété tous les 32 points.

```
;*** Dual-Playfield et Scrolling ***  
  
;registres circuits spécialisés  
  
INTENA - $9A      ; registre d'autorisation d'interruptions (écriture)  
DMACON - $96      ; registre de demande d'interruptions (lecture)  
COLOROO - $180    ; registre de contrôle DMA (écriture)  
VPOSR  - $4       ; registre 0 palette couleur  
  
;registre COPPER  
  
COP1LC - $80      ; adresse de la 1ère liste  
COP2LC - $84      ; adresse de la 2ème liste  
COPJMP1 - $88     ; saut à la 1ère liste  
COPJMP2 - $8A     ; saut à la 2ème liste  
  
;registre bitplane  
  
BPLCON0 - $100    ; registre 0 de contrôle bitplane  
BPLCON1 - $102    ; 1 (valeur de scrolling)  
BPLCON2 - $104    ; 2 (sprite <> priorité playfield)  
BPL1PTH - $0E0    ; pointeur sur le 1er  
BPL1PTL - $0E2    ; bitplane  
BPL1MOD - $108    ; valeur modulo pour les plans impairs  
BPL2MOD - $10A    ; valeur modulo pour les plans pairs  
DIWSTRT - $08E    ; début de la fenêtre écran  
DIWSTOP - $90     ; fin de la fenêtre écranbitplane  
DDFSTRT - $092    ;  
DDFSTOP - $094    ;  
  
;CIA-A registre port A (bouton souris)  
  
CIAAPRA - $BFE001  
  
;Exec Library Base Offset  
  
OpenLibrary - -30-522      ; LibName, Version / a1, d0  
Forbid      - -30-102  
Permit      - -30-108  
AllocMem    - -30-168      ; ByteSize, Requirements / d0, d1  
FreeMem     - -30-180      ; MemoryBlock, ByteSize / a1, d0  
  
;graphics base  
  
StartList - 38  
  
;autres labels  
  
Execbase - 4  
Planesize - 52*345        ; taille des bitplanes  
Planewidth - 52  
CLsize     - 5*4           ; la COPPER-LIST contient 5 instructions  
Chip       - 2             ; CHIP-RAM sollicitée  
Clear      - Chip+$10000   ; CHIP-RAM réservée  
  
;Initialisation  
  
Start :
```

```

;réserver de la mémoire pour les bitplanes

move.l Execbase,a6
move.l #Planesize*2,d0      ; mémoire nécessaire aux plans
move.l #Clear,d1
jsr AllocMem(a6)           ; mémoire sollicitée
move.l d0,Planeadr
beq.l Fin                  ; erreur -> Fin

;réservation de mémoire pour la COPPER-LIST

moveq #CLsize,d0
moveq #Chip,d1
jsr AllocMem(a6)
move.l d0,CLadr
beq FreePlane              ; erreur -> mémoire des plans libérée

;mise en place de la COPPER-LIST

moveq #1,d4                ; deux bitplanes
move.l d0,a0
move.l Planeadr,d1
move.w #BPL1PTH,d3

MakeCL:
move.w d3,(a0)+
addq.w #2,d3
swap d1
move.w d1,(a0)+
move.w d3,(a0)+
addq.w #2,d3
swap d1
move.w d1,(a0)+
add.l #Planesize,d1        ; adresse du prochain plan

dbf d4,MakeCL

move.l #$FFFFFFFE,(a0)    ; fin de la COPPER-LIST

;*** programme principal ***

;DMA activé et Task Switching bloqué

jsr Forbid(a6)
lea $DFF000,a5
move.w #$01E0,DMACON(a5)

;COPPER initialisé

move.l CLadr,COP1LC(a5)
clr.w COPJMP1(a5)

;Playfield initialisé

move.w #0,COLOR00(a5)
move.w #$0F00,COLOR00+2(a5)
move.w #$000F,COLOR00+18(a5)

move.w #$1A64,DIWSTRT(a5)   ; 26,100

```

```

move.w #$39D1,DIWSTOP(a5)           : 313,465
move.w #$0020,DDFSTRT(a5)          : lecture d'un mot de plus
move.w #$0008,DDFSSTOP(a5)
move.w #%0010001100000000,BPLCON0(a5) ; Dual-Playfield activé
clr.w BPLCON1(a5)                 : valeur de départ du scrolling à 0
clr.w BPLCON2(a5)                 : playfield 1 en avant du playfield 2

move.w #4,BPL1MOD(a5)             : valeur modulo - 2 mots
move.w #4,BPL2MOD(a5)

;DMA activé

move.w #$8180,DMACON(a5)

;remplissage des bitplanes avec un damier
move.l Planeadr,a0
move.w #Planesize/2-1,d0          ; compteur de boucle
move.w #13*16,d1                  ; maximum = 16 lignes
move.l #$FFFF0000,d2              ; damier
move.w d1,d3

Fill :
move.l d2,(a0)+
subq.w #1,d3
bne.s cont
swap d2                         ; changer de modèle
move.w d1,d3

Cont:
dbf d0.Fill

;Playfields scrollés
clr.l d0                         ; position de scroll vertical
clr.l d1                         ; position de scroll horizontal
move.l CLadr,a1                   ; adresse de la COPPER-LIST
move.l Planeadr,a0                ; adresse du premier bitplane

Wait :
move.l VPOSR(a5),d2               ; lecture de la position
and.l #$0001FF00,d2               ; bits horizontaux sélectionnés
cmp.l #$00001000,d2               ; attendre la ligne 16
bne.s wait

;scrolling vertical du playfield 1

addq.b #2,d0                      ; compteur scrolling vertical incrémenté
cmp.w #$80,d0                      ; test d'arrivée à la valeur 128 (4*32)
bne.s PlusHaut
clr.l d0                           ; retour à 0

PlusHaut :
move.l d0,d2                      ; compteur scrolling copié
lslr.w #2,d2                       ; copie divisée par 4
mulu #52,d2                        ; nombre d'octets/ligne * position de scroll
add.l a0,d2                         ; plus adresse du premier plan

add.l #Planesize,d2                ; plus taille du plan
move.w d2,14(a1)                   ; donne l'adresse de fin de la COPPER-LIST

```

```

swap d2
move.w d2,10(a1)

;scrolling horizontal du playfield 2

addq.b #1,d1           ; compteur scrolling horizontal incrémenté
cmp.w #$80,d1
bne.s PlusLoin
clr.l d1

PlusLoin :
move.l d1,d2
lsr.w #2,d2
move.l d2,d3           ; position scrolling copiée
and.w #$FFF0,d2        ; 4 bits inférieurs forcés à 0
sub.w d2,d3             ; 4 bits inférieurs dans d3 isolés
move.w d4,BPLCON1      ; dernière valeur dans BPLCON1
move.w d3,d4             ; nouvelle valeur de scrolling dans d4
lsl.w #3,d2             ; nouvelle adresse pour la COPPER-LIST
add.l a0,d2              ; calcul
move.w d2,6(a1)          ; et écriture dans la COPPER-LIST
swap d2
move.w d2,2(a1)

btst #6,CIAAPRA         ; souris activée (bouton pressé)
bne.s Wait               ; non -> continuer

;*** fin de programme ***

;ancienne COPPER-LIST réactivée

move.l #GRname,a1        ; initialisation d'OpenLibrary
clr.l d0
jsr OpenLibrary(a6)       ; librairie graphique ouverte
move.l d0,a4
move.l StartList(a4),COP1LC(a5)
clr.w COPJMP1(a5)
move.w #$83E0,DMACON(a5)
jsr Permit(a6)            ; Task Switching rétabli

;mémoire des COPPER-LIST libérée

move.l CLadr,a1
moveq #CLsize,d0
jsr FreeMem(a6)

Freeplane :
move.l Planeadr,a1
move.l #Planesize*2,d0
jsr FreeMem(a6)

Fin:
clr.l d0
rts                      ; quitter le programme

;variables

CLadr : dc.l 0
Planeadr : dc.l 0

```

```

test :      dc.l 0
;constante
GRname :   dc.b "graphics.library",0
END
;Fin du programme

```

1.5.6. Sprites

Les sprites sont des éléments graphiques qui peuvent être utilisés indépendamment des playfields. Chaque sprite peut avoir une largeur de 16 points et une hauteur maximale analogue à celle de la fenêtre d'écran. Il peut occuper n'importe quelle position à l'écran. Normalement, les sprites se trouvent en avant du playfield et leurs points en cachent le graphisme. Le pointeur de la souris en est un exemple. Sur l'Amiga, seuls 8 sprites peuvent être reproduits, avec la possibilité de 3 couleurs. Il existe aussi un moyen de combiner deux sprites en un nouveau, qui acceptera lui, 15 couleurs.

La structure des sprites

Le choix des couleurs

Le choix des couleurs d'un sprite ressemble à celui du mode Dual-Playfield. Le sprite a une largeur de 16 points, ce qui représente deux mots de donnée, ce qui peut correspondre à deux 'mini-bitplanes'. Ainsi la couleur d'un point sera déterminée, comme pour les bitplanes, par la combinaison des bits le représentant.

La couleur du premier point (point le plus à gauche du sprite) sera déterminée par les deux bits de plus fort poids (bit 15) des deux mots de donnée. Les deux bits de plus faible poids correspondent à la couleur du dernier point. Chaque point est donc représenté par deux bits, ce qui amène la possibilité de 4 combinaisons différentes, donc de 4 couleurs différentes. Pour attribuer une couleur à ces points au travers de ces 4 valeurs, on utilisera à nouveau la palette couleur, étant donné qu'il n'existe pas de registre couleur propre aux sprites. Les couleurs des sprites seront représentées par la deuxième moitié des registres, c'est-à-dire les registres couleurs correspondant aux numéros variant de 16 à 31. Ainsi, les sprites et les playfields pourront entrer en conflit lorsque ce dernier comprendra plus de 16 couleurs.

Le tableau suivant nous montre les liens entre les registres couleurs et les sprites :

Sprite N°	Données sprite	Registre couleur
0 & 1	0 0	Transparent
	0 1	COLOR17

Sprite N°	Données sprite	Registre couleur
	1 0	COLOR18
	1 1	COLOR19
2 & 3	0 0	transparent
	0 1	COLOR21
	1 0	COLOR22
	1 1	COLOR23
4 & 5	0 0	transparent
	0 1	COLOR25
	1 0	COLOR26
	1 1	COLOR27
6 & 7	0 0	transparent
	0 1	COLOR29
	1 0	COLOR30
	1 1	COLOR31

Les deux sprites qui se suivent ont les mêmes couleurs.

Comme en mode Dual-Playfield, la combinaison de bits de deux 0 ne reproduit pas une couleur, mais s'affiche en transparence. A la place de cette couleur, on peut voir directement les couleurs des éléments se trouvant en position d'arrière plan, que ce soit d'autres sprites, un playfield ou plus simplement la couleur du fond.

Si les trois couleurs ne suffisent pas, on peut combiner deux sprites. Les deux combinaisons de bits des deux sprites donnent ensemble une valeur 4 bits. Les deux mots de donnée du sprite ayant le plus grand numéro correspondent aux deux bits de plus fort poids de la valeur 4 bits. Cette dernière pointe sur 15 registres de la palette couleur, la valeur nulle correspondant à l'état transparent.

Ces registres sont les mêmes pour les 4 combinaisons possibles de sprite : COLOR16 à COLOR31.

Données sprite	Registre couleur	Données sprite	Registre couleur
0 0 0 0	transparent	1 0 0 0	COLOR24
0 0 0 1	COLOR17	1 0 0 1	COLOR25
0 0 1 0	COLOR18	1 0 1 0	COLOR26
0 0 1 1	COLOR19	1 0 1 1	COLOR27

Données sprite	Registre couleur	Données sprite	Registre couleur
0 1 0 0	COLOR20	1 1 0 0	COLOR28
0 1 0 1	COLOR21	1 1 0 1	COLOR29
0 1 1 0	COLOR22	1 1 1 0	COLOR30
0 1 1 1	COLOR23	1 1 1 1	COLOR31

Le DMA Sprite

La programmation des sprites sur l'Amiga peut se réaliser assez facilement. La plus grande partie du travail est réalisée par le biais des canaux DMA sprite. Pour représenter un sprite à l'écran, une liste de données sprite spéciale est nécessaire en mémoire. Elle contient toutes les données importantes du sprite. Pour afficher ce dernier, seule l'adresse de cette liste est à communiquer au contrôleur DMA.

Le contrôleur DMA possède un canal DMA propre à chaque sprite.

Il ne peut malheureusement que lire deux mots de données, pendant l'affichage d'une ligne par le Raster. C'est la raison pour laquelle le sprite n'a qu'une largeur de 16 points et est limité à 4 couleurs. Etant donné que ces deux mots de données sont lus à chaque affichage d'une ligne, la hauteur d'un sprite n'est limité que par la fenêtre écran.

Structure d'une liste de données sprite

Une telle liste de données se compose de lignes renfermant chacune 2 mots de données. A chaque ligne affichée par le Raster, une de la liste sera lue via le DMA. Celle-ci peut contenir, soit deux mots de contrôle, soit deux mots avec les données points, lorsque le sprite doit être à reproduit à l'instant.

Les mots de contrôle déterminent la position horizontale et les première et dernière lignes du sprite.

Après lecture de ces mots par le contrôleur DMA et lorsque celui-ci les a placé dans les registres correspondants, il attend jusqu'à ce que le faisceau d'électrons ait atteint le début de la première ligne du sprite. Puis, à chaque ligne, deux mots de données seront lus, et par le hardware de DENISE, une ligne de sprite sera affichée à la position correspondante, jusqu'à la dernière définie. Les deux prochains mots correspondent à des mots de contrôle. S'ils sont à 0, le canal DMA termine son activité. Mais il est aussi possible de donner une nouvelle position sprite. Le contrôleur DMA attendra alors à nouveau la ligne de début d'affichage et le processus se répétera aussi longtemps que la liste ne se finira pas avec deux mots nuls.

Structure d'une liste de données sprite (Start = adresse de départ de la liste dans la CHIP-RAM) :

Adresse	Contenu
Start+4	1er et 2ème mot de donnée de la 1ère ligne du sprite
Start+8	1er et 2ème mot de donnée de la 2ème ligne du sprite
Start+12	1er et 2ème mot de donnée de la 3ème ligne du sprite
Start+4*n	1er et 2ème mot de donnée de la nième ligne du sprite
Start+4*(n+1)	0,0 fin de la liste de données sprite

Structure du premier mot de contrôle :

Bit n° :	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fonction :	E7	E6	E5	E4	E3	E2	E1	E0	H8	H7	H6	H5	H4	H3	H2	H1

Structure du deuxième mot de contrôle :

Bit n° :	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fonction :	L7	L6	L5	L4	L3	L2	L1	L0	AT	0	0	0	0	E8	L8	HO

H0-H8	position horizontale du sprite (HSTART)
E0-E8	première ligne du sprite (VSTART)
L0-L8	dernière ligne + 1 du sprite (VSTOP)
AT	bit de contrôle d'attache

Il existe donc 9 bits de position horizontale et verticale pour chaque sprite. Ces bits ne sont pourtant pas répartis de façon pratique entre les deux registres de contrôle.

La résolution correspond verticalement à une ligne du Raster et horizontalement, à un point en mode Lores. Ces valeurs ne peuvent être modifiées, étant donné qu'elles sont indépendantes du mode des playfields.

La hauteur des sprites est limitée d'après les valeurs DIWSTRT et STOP, définissant la fenêtre d'écran. Si les coordonnées se trouvant dans les mots de contrôle dépassent ces derniers, les sprites ne seront plus ou partiellement représentés, suivant les limites de la fenêtre.

La position de départ, horizontale et verticale, concerne le coin supérieur gauche du sprite. La position de fin verticale correspond à la première ligne après le sprite, c'est-à-dire à la dernière + 1. Le nombre de lignes d'un sprite est donc VSTOP-VSTART.

L'exemple suivant reproduit un sprite aux coordonnées 180, 160, à peu près au centre de l'écran. Il doit avoir une hauteur de 8 lignes, la valeur de VSTOP étant alors 168.

Lorsqu'on réunit les deux mots de donnée, on obtient un nombre compris entre 0 et 3 qui représente une des 3 couleurs et l'état transparent (0) du sprite. On peut donc écrire les sprites de la manière suivante :

```
000000222200000
0000220000220000
0002200330022000
0022003113002200
0022003113002200
0002200330022000
0000220000220000
0000002222000000
```

Dans la liste de données, on doit fournir les deux mots séparés :

```
Start:
dc.w $A05A,$A800 ; HSTART - $B4, VSTART - $A0, VSTOP-$A8
dc.w %0000 0000 0000,0000,0000 0011 1100 0000
dc.w %0000 0000 0000 0000,0000 1100 0011 0000
dc.w %0000 0001 1000 0000,0001 1001 1001 1000
dc.w %0000 0011 1100 0000,0011 0010 0100 1100
dc.w %0000 0011 1100 0000,0011 0010 0100 1100
dc.w %0000 0001 1000 0000,0001 1001 1001 1000
dc.w %0000 0000 0000 0000,0000 1100 0011 0000
dc.w %0000 0000 0000 0000,0000 0011 1100 0000
dc.w 0,0 ;fin de la liste de données sprite
```

Le bit AT, du deuxième mot de contrôle, établit si deux sprites sont combinés l'un avec l'autre. Il n'a qu'une fonction dans les sprites à numéro impair (sprites 1, 3, 5, 7). S'il est activé pour le sprite1, par exemple, ses bits de données seront interprétés avec ceux du sprite0, en tant que pointeur 4 bits sur la palette couleur. L'ordre des bits sera le suivant :

Sprite 1 (numéro impair), deuxième mot de donnée :	bit 3 (MSB)
Sprite 1 premier mot de donnée :	bit 2
Sprite 0 (numéro pair), deuxième mot de donnée :	bit 1
Sprite 0 premier mot de donnée :	bit 0 (LSB)

Pour que deux sprites puissent être combinés, leur position doit concorder. Si ce n'est pas le cas, on se placera automatiquement en mode trois couleurs. Le plus simple est de mettre les mêmes mots de contrôle dans les deux listes de données sprite.

Voici pour exemple, une liste de données sprite en 16 couleurs :

Pour plus de simplicité, le sprite représenté ne comprend que 4 lignes. Les chiffres reproduisent à nouveau les couleurs des points correspondant. Pour pouvoir représenter les 15 couleurs, on utilisera la notation hexadécimale.

```

0011111111111100
1123456789ABCD11
11EFEFEFEFEFEF11
0011111111111100

```

Voici le détail des mots de données nécessaires à la deuxième ligne du sprite :

Couleurs du sprite :	1123456789ABCD11
Sprite1, mot de donnée 2 :	0000000011111100
Sprite1, mot de donnée 1 :	0000111100001100
Sprite0, mot de donnée 2 :	0011001100110000
Sprite0, mot de donnée 1 :	1101010101010111

Les listes de données pour ce même sprite sont les suivantes :

La position horizontale (HSTART) est à nouveau 180. La première ligne du sprite (VSTART) correspond à 160, la dernière ligne à 164.

```

StartSprite0:
dc.w $A05A,$A400 ;HSTART-$B4, VSTART-$A0, VSTOP-$A4, AT-0
dc.w %0011 1111 1111 1100, %0000 0000 0000 0000
dc.w %1101 0101 0101 0111, %0011 0011 0011 0000
dc.w %1101 0101 0101 0111, %0011 1111 1111 1100
dc.w %0011 1111 1111 1100, %0000 0000 0000 0000
StartSprite1:
dc.w $A05A,$A480 ;HSTART-$B4, VSTART-$A0, VSTOP-$A4, AT-1
dc.w %0000 0000 0000 0000, %0000 0000 0000 0000
dc.w %0000 1111 0000 1100, %0000 0000 1111 1100
dc.w %0011 1111 1111 1100, %0011 1111 1111 1100
dc.w %0000 0000 0000 0000, %0000 0000 0000 0000

```

Plusieurs sprites sur un canal DMA

Après l'affichage d'un sprite, son canal est à nouveau libre. Dans l'exemple plus haut, les dernières données sprite seront lues à la ligne 163. Puis le canal DMA sprite sera désactivé par la présence des deux zéros. Comme cela a été vu, il est possible d'utiliser à nouveau le canal, en écrivant à la place des deux mots nuls, deux nouveaux mots de contrôle. Il y a tout de même une restriction, car une ligne reste libre entre la première du nouveau sprite et la dernière du sprite affiché. Si celle-ci est reproduite à la ligne 163, la nouvelle ne commencera pas avant la ligne 165. La raison réside dans la présence intermédiaire de la ligne comprenant les mots de contrôle, qui seront eux aussi lus. Le fonctionnement du DMA sprites se déroule donc de la façon suivante :

Ligne	Données sur le canal DMA
162	Avant dernière ligne du 1er sprite
163	Dernière ligne du 1er sprite
164	Mot de contrôle du 2ème sprite
165	1ère ligne du 2ème sprite
166	2ème ligne du 2ème sprite

L'exemple suivant place le sprite 3 couleurs de notre première liste de données à deux positions différentes.

```

Start:
;Premier sprite sur le canal DMA à la ligne 160 ($A0)
;position horizontale: 180 ($B4)
dc.w $A05A,$A800 ; HSTART - $B4, VSTART - $A0, VSTOP-$A8
dc.w %0000 0000 0000 0000,%0000 0011 1100 0000
dc.w %0000 0000 0000 0000,%0000 1100 0011 0000
dc.w %0000 0001 1000 0000,%0001 1001 1001 1000
dc.w %0000 0011 1100 0000,%0011 0010 0100 1100
dc.w %0000 0011 1100 0000,%0011 0010 0100 1100
dc.w %0000 0001 1000 0000,%0001 1001 1001 1000
dc.w %0000 0000 0000 0000,%0000 1100 0011 0000
dc.w %0000 0000 0000 0000,%0000 0011 1100 0000
;deuxième sprite à la suite sur le canal DMA
;à la ligne 176 ($B0), et à la position horizontale 300 ($12C)
dc.w $B096,$B800 ; HSTART - $12C, VSTART - $B0, VSTOP-$B8
dc.w %0000 0000 0000 0000,%0000 0011 1100 0000
dc.w %0000 0000 0000 0000,%0000 1100 0011 0000
dc.w %0000 0001 1000 0000,%0001 1001 1001 1000
dc.w %0000 0011 1100 0000,%0011 0010 0100 1100
dc.w %0000 0000 0000 0000,%0000 1100 0011 0000
dc.w %0000 0000 0000 0000,%0000 0011 1100 0000
dc.w 0,0 ;fin de la liste de données sprite

```

Initialisation du sprite

Après avoir créé un sprite, en reproduisant une liste de données correctes dans la CHIP-RAM, et après avoir choisi les couleurs dans la palette, on doit encore communiquer au contrôleur DMA l'adresse où se trouve la liste, afin qu'il puisse initialiser le sprite. Ainsi chaque canal DMA sprite possède une paire de registres, dans laquelle l'adresse de départ de la liste de données doit être écrite :

Registre SPRxPT (SPRitexPoinTeur, pointeur sur la liste de données pour le canal DMA sprite).

Adresse	Nom	Fonction
\$120	SPROPTH	pointeur sur la liste de données sprite (bits 16-18)
\$122	SPROPTL	pour le canal DMA sprite 0 (bits 0-15)
\$124	SPR1PTH	pointeur sur la liste de données sprite (bits 16-18)
\$126	SPR1PTL	pour le canal DMA sprite 1 (bits 0-15)
\$128	SPR2PTH	pointeur sur la liste de données sprite (bits 16-18)
\$12A	SPR2PTL	pour le canal DMA sprite 2 (bits 0-15)

Adresse	Nom	Fonction
\$12C	SPR3PTH	pointeur sur la liste de données sprite (bits 16-18)
\$12E	SPR3PTL	pour le canal DMA sprite 3 (bits 0-15)
\$130	SPR4PTH	pointeur sur la liste de données sprite (bits 16-18)
\$132	SPR4PTL	pour le canal DMA sprite 4 (bits 0-15)
\$134	SPR5PTH	pointeur sur la liste de données sprite (bits 16-18)
\$136	SPR5PTL	pour le canal DMA sprite 5 (bits 0-15)
\$138	SPR6PTH	pointeur sur la liste de données sprite (bits 16-18)
\$13A	SPR6PTL	pour le canal DMA sprite 6 (bits 0-15)
\$13C	SPR7PTH	pointeur sur la liste de données sprite (bits 16-18)
\$13E	SPR7PTL	pour le canal DMA sprite 7 (bits 0-15)

SPRxPT n'est accessible qu'en mode écriture.

Le contrôleur DMA utilise ces registres comme pointeur sur l'adresse actuelle de la liste de données sprite. Au début de chaque image, les registres renferment l'adresse du premier mot de contrôle. A chaque mot de données lu, ils sont augmentés d'un mot (16 bits), pour qu'ils pointent le mot suivant à la fin de la liste. Si le même sprite doit être représenté, à la même position, le pointeur devra être remis à l'adresse de départ.

Comme pour les pointeurs bitplanes BPLxPT, c'est le COPPER qui assure cette tâche, pendant le temps mort vertical. La partie correspondante de la COPPER-LIST sera identique à ce qui suit :

StartSpritexH = adresse de départ de la liste de données
du sprite x bits 16-19

StartSpritexL = bits 0-15

:Début de la COPPER-LIST

MOVE #StartSprite0H,SPROPTH ;initialisation du canal 0 DMA sprite

MOVE #StartSprite0L,SPROPTL ;initialisation du canal 1 DMA sprite

MOVE #StartSprite1H,SPROPTH ;initialisation du canal 2 DMA sprite

MOVE #StartSprite1L,SPROPTL ;initialisation du canal 3 DMA sprite

MOVE #StartSprite2H,SPROPTH ;initialisation du canal 4 DMA sprite

MOVE #StartSprite2L,SPROPTL ;initialisation du canal 5 DMA sprite

.....

MOVE #StartSprite7H,SPROPTH ;initialisation du canal 7 DMA sprite

MOVE #StartSprite7L,SPROPTL ;initialisation du canal 8 DMA sprite

.....

WAIT \$FFFF ;fin de la COPPER-LIST

Il n'y a aucune possibilité d'activer et de désactiver chaque canal DMA sprite séparément. Le bit SPREN (bit n° 5) du registre DMACON active les 8 canaux DMA en même temps. Si on ne veut pas tous les employer, on doit laisser travailler les canaux inutilisés avec des listes vides. On fixera les pointeurs sur deux mots mémoire à contenu nul, les deux mots de fin de liste du sprite utilisé, par exemple.

Les pointeurs SPRxPT doivent toujours être initialisés lors du temps mort vertical. Même quand les listes de données ne renferment rien d'autre que deux valeurs nulles, SPRxPT pointera toujours à la fin d'une image, sur le premier mot se tenant après elles.

Evidemment, l'initialisation des SPRxPT peut aussi être réalisée par le processeur, après l'interruption 'écran vide vertical'.

Puis en dernier, il faut initialiser les 8 canaux DMA sprites, au moyen du bit SPREN du registre DMACON, par l'instruction move suivante :

```
MOVE.W #$8220,$DFF096 ;SPREN et DMAEN du registre DMACON initialisés
```

Mouvement des sprites

La position d'un sprite est déterminée par la valeur se trouvant dans les deux mots de contrôle de la liste de données sprite. Afin que le sprite acquiert un mouvement, il faut modifier cette valeur d'un certain pas. Ceci peut être exécuté par le processeur directement, au moyen de l'instruction MOVE. On doit cependant être prudent, car il peut apparaître des problèmes si on modifie les mots de contrôle à n'importe quel moment :

Le processeur modifie le premier mot et avant qu'il n'ait le temps de modifier le deuxième, le contrôleur DMA lit les deux valeurs. Comme ces deux dernières ensemble ne conviennent plus, l'affichage sera altéré.

On peut éviter ceci en modifiant les mots de contrôle pendant le temps mort vertical (pendant l'interruption 'écran vide vertical', après que le COPPER ait initialisé SPRxPT).

La priorité sprites/playfield(s)

La priorité entre les sprites et un playfield détermine qui sera affiché entre ou derrière les éléments de l'écran. Le sprite avec la plus grande priorité se trouve devant tous les autres et ne peut être dissimulé. La priorité est déterminée par le numéro du sprite. Le numéro 0 correspond à la plus grande priorité.

Pour les playfields, c'est un bit de contrôle qui détermine lequel sera devant l'autre.

Le dernier cas à traiter est celui de la priorité entre les sprites et le playfield. Sur l'Amiga, il est possible de positionner un playfield entre n'importe quel sprite. Lors de la détermination de la priorité du playfield sur les sprites, ces derniers seront rassemblés et traités par paires. Ce sera à nouveau les mêmes combinaisons utilisées pour les sprites à 15 couleurs, c'est-à-dire, une paire formée du sprite à numéro impair et de son prédécesseur à numéro pair :

Sprite 0&1, sprite 2&3, sprite 4&5, sprite 6&7

On peut considérer ces quatre paires sous forme d'un tableau. Les trous correspondent aux points transparents des bitplanes ou des sprites et aux parties de l'écran, que la taille du sprite ne permet pas de cacher. La rangée d'un tel tableau n'est pas modifiable. On peut, par contre, insérer deux autres éléments, les playfields, à n'importe quelle place entre les paires de sprites. Cinq positions sont alors possibles :

Position	Suite des priorités					
	PLF	SPR0&1	SPR2&3	SPR4&5	SPR6&7	
0	PLF	SPR0&1	SPR2&3	SPR4&5	SPR6&7	
1	SPR0&1	PLF	SPR2&3	SPR4&5	SPR6&7	
2	SPR0&1	SPR2&3	PLF	SPR4&5	SPR6&7	
3	SPR0&1	SPR2&3	SPR4&5	PLF	SPR6&7	
4	SPR0&1	SPR2&3	SPR4&5	SPR6&7	PLF	

Le registre BPLCON2 renferme la priorité des Playfields par rapport aux sprites.

BPLCON2 \$104 (écriture seulement)

Bit n :	15-7	6	5	4	3	2	1	0
Fonction :	inutilisé	PF2PRI	PF2P2	PF2P1	PF2P0	PF1P2	PF1P1	PF1P0

PF2PRI

Si ce bit est activé, le playfield 2 s'affiche devant le playfield 1.

PF1P0 à PF1P2

Ces trois bits forment un nombre 3 bits, qui correspond à la position du playfield n°1 (bitplanes impairs) entre les quatre paires de sprites. Seules les valeurs 0 à 4 sont possibles.

PF2P0 à PF2P2

Ces trois bits correspondent à la même fonction que les trois bits PF1Px, mais cette fois-ci pour le playfield n°2 (bitplanes pairs).

Exemple :

BPLCON2 = \$0003

Ceci signifie que le playfield n°1 se trouve devant le n°2, que PF2Px = 2, PF1Px = 3. La suite logique entre les différents éléments est donc :

PLF2 SPR0&1 SPR2&3 SPR4&5 PLF1 SPR6&7

Cette suite est donc paradoxale, étant donné que le bit PLF2PRI est à 0 et que le n1 devrait précéder le n2, mais elle est juste.

Lorsqu'un des sprites 0 à 5 se trouve entre les playfields 1 et 2, ce dernier se tient devant le n1, tout en respectant sa priorité. Etant donné que le n1 se trouve devant le n2, les sprites visibles se tiennent à leur place, même si ceux-ci sont derrière le playfield n2. Si ce dernier se trouve à une place commune d'un sprite, il couvrira suivant le rang de priorité le(s) sprite(s) d'une façon normale. Il en résulte donc que la priorité playfield/playfield passe avant la priorité sprite/playfield.

Si on n'utilise pas le mode Dual-Playfield, il n'y aura qu'un seul playfield qui sera formé d'un ensemble de bitplanes pair(s) et impair(s). Les bits PLF2PRI et PL2Px n'auront alors plus aucune fonction.

Collisions entre éléments graphiques

Il est souvent très utile de savoir si deux sprites entrent en collision l'un avec l'autre ou avec un élément de l'arrière plan. Ceci facilite, par exemple, la reconnaissance d'un coup au but dans un programme de jeu.

Lorsque les points de deux sprites se chevauchent à une position donnée sur l'écran, c'est-à-dire qu'ils possèdent les mêmes coordonnées tout en n'étant pas transparents, il y a collision. Cette dernière est aussi possible entre deux playfields ou un champ et un sprite.

Toutes les collisions reconnues sont mises en mémoire dans le registre de données collision CLXDAT :

CLXDAT \$00E (lecture seulement)

Bit n°	Collision entre
15	inutilisé
14	sprite 4 (ou 5) et sprite 6 (ou 7)
13	sprite 2 (ou 3) et sprite 6 (ou 7)
12	sprite 2 (ou 3) et sprite 4 (ou 5)
11	sprite 0 (ou 1) et sprite 6 (ou 7)
10	sprite 0 (ou 1) et sprite 4 (ou 5)
9	sprite 0 (ou 1) et sprite 2 (ou 3)
8	playfield 2 (bitplane pair) et sprite 6 (ou 7)
7	playfield 2 (bitplane pair) et sprite 4 (ou 5)

Bit n°	Collision entre
6	playfield 2 (bitplane pair) et sprite 2 (ou 3)
5	playfield 2 (bitplane pair) et sprite 0 (ou 1)
4	playfield 1 (bitplane impair) et sprite 6 (ou 7)
3	playfield 1 (bitplane impair) et sprite 4 (ou 5)
2	playfield 1 (bitplane impair) et sprite 2 (ou 3)
1	playfield 1 (bitplane impair) et sprite 0 (ou 1)
0	playfield 1 et playfield 2

Alors que chaque point non transparent d'un sprite peut libérer une collision, on peut choisir librement dans le(s) champ(s) de jeu, la couleur du point qui caractérisera la reconnaissance d'une collision. De plus, il est possible d'autoriser ou non le test de collision pour tous les sprites à numéro impair. Toutes ces possibilités sont issues des bits du registre de contrôle collision CLXCON.

CLXCON \$098 (*écriture seulement*)

Bit n°	Nom	Fonction
15	ENSP7	test de collision autorisé pour le sprite 7
14	ENSP5	test de collision autorisé pour le sprite 5
13	ENSP3	test de collision autorisé pour le sprite 3
12	ENSP1	test de collision autorisé pour le sprite 1
11	ENBP6	comparaison entre le bitplane 6 et MVBP6
10	ENBP5	comparaison entre le bitplane 5 et MVBP5
9	ENBP4	comparaison entre le bitplane 4 et MVBP4
8	ENBP3	comparaison entre le bitplane 3 et MVBP3
7	ENBP2	comparaison entre le bitplane 2 et MVBP2
6	ENBP1	comparaison entre le bitplane 1 et MVBP1
5	MVBP6	valeur pour la collision avec le bitplane 6
4	MVBP5	valeur pour la collision avec le bitplane 5
3	MVBP4	valeur pour la collision avec le bitplane 4
2	MVBP3	valeur pour la collision avec le bitplane 3
1	MVBP2	valeur pour la collision avec le bitplane 2

Bit n°	Nom	Fonction
0	MVBP1	valeur pour la collision avec le bitplane 1

Les bits ENSPx (enable sprite x) déterminent si les collisions des sprites correspondants, à numéro impair, seront testées ou non. Par exemple, si le bit ENSP1 est activé, la collision entre le sprite 1 et un autre sera enregistrée. Le bit correspondant du registre de données collision sera alors activé, comme pour le sprite 0. Il n'est donc pas possible de savoir, par l'intermédiaire du contenu du registre, lequel des deux sprites, 0 ou 1, a libéré la collision. Retenez bien cela dans l'utilisation des sprites.

Si on a combiné deux sprites, en vue de former un sprite 15 couleurs, le bit correspondant ENSPx doit être activé, afin de permettre un test de collision correct.

Dans un playfield, on peut établir soi-même quelle combinaison de bitplane est en mesure de libérer une collision ou non. Les bits ENBPx (enable bitplane x) déterminent quel bitplane sera sélectionné pour le test de collision. Si tous les bits ENBPx sont activés, la collision est possible avec tous les points. Les combinaisons de bits correspondent aux bits MVBPx (Match Value Bitplane x).

Les bits ENBPx déterminent si les bits des plans x doivent être comparés avec les valeurs des bits MVBPx. Si les bits de tous les plans d'un point concordent avec les bits MVBPx, ce point pourra libérer une collision. Pour plus de clarté, voici un exemple :

Les bits ENBPx sont activés, tout comme les bits MVBPx. Chaque point du playfield pourra libérer une collision, si sa combinaison de bits est 111111. Si seuls les trois bits inférieurs sont activés, une collision pourra être libérée lorsque les points du playfield auront la combinaison 000111.

Si le test de collision doit être autorisé pour tous les points aux combinaisons de bits suivantes, 000111, 000110, 000100 ou 000101, les bits MVBPx doivent correspondre au nombre suivant : 000100.

En effet ces deux bits inférieurs devront toujours exécuter les modalités de la collision, du fait de leur état à 0, la combinaison des bits ENBPx devant quant à elle correspondre à 111100.

Exemple de combinaison de bits possibles :

ENBPx	MVBPx	Collision possible
111111	111111	111111
111111	111000	111000
111100	1111xx	111100, 111101, 111110, 111111
011111	x00000	000000, 100000

ENBPx	MVBPx	Collision possible
000000	xxxxxx	collision possible pour toutes combinaisons de bits

(La valeur du bit caractérisé par x n'a pas d'importance).

Si tous les bitplanes ne sont pas actifs, les bits ENBPx des plans inutilisés doivent être mis à 0.

Les différentes possibilités de combinaisons des bits ENBPx et MVBPx permettent une grande quantité de reconnaissances de collisions différentes. On peut, par exemple, initialiser le registre CLXCON d'une façon telle que seuls les sprites à points rouges et verts peuvent entrer en collision avec le playfield.

Il est aussi possible d'autoriser une collision, lorsque les points transparents du playfield 1 chevauchent les points de couleur noir du playfield 2 etc...

Autres registres sprite

Il existe pour chaque sprite, en dehors des registres SPRxPT, 4 autres registres. Les données qu'ils contiennent leur sont communiquées automatiquement par le contrôleur DMA. Mais il est aussi possible d'y accéder via le processeur.

SPRxPOS	premier mot de contrôle
SPRxCTL	deuxième mot de contrôle
SPRxDATA	premier mot de données d'une ligne (mot low)
SPRxDATB	deuxième mot de données d'une ligne (mot high)

(x correspond à un des numéros 0-7 possibles de sprite. Les adresses de ces registres se trouvent dans la liste des registres du chapitre 1.5.1).

Le contrôleur DMA écrit directement les deux mots de contrôle d'un sprite dans deux registres : SPRxPOS et SPRxCTL. Lorsqu'une valeur sera écrite dans le registre SPRxCLT, que ce soit par l'intermédiaire du DMA ou du 68000, la sortie sprite de DENISE sera désactivée. Le sprite ne sera plus envoyé à l'écran.

Le contrôleur DMA attend alors la ligne établie dans VSTART. Puis il écrit les deux premiers mots de données dans les registres SPRxDATA et SPRxDATB. C'est alors que DENISE activera à nouveau la sortie sprite. Il attend alors la position horizontale souhaitée en comparant les registres SPRxCTL et SPRxPOS avec la colonne actuelle de l'écran, et affiche le sprite à sa bonne place.

Le contrôleur DMA écrit, à chaque nouvelle ligne, deux nouveaux mots de données dans SPRxDATA/B, jusqu'à ce que la dernière ligne du sprite (VPOS) soit passée. Puis il prend les prochains mots de contrôle et les place dans SPRxPOS et SPRxCTL. Ainsi, le sprite sera à nouveau désactivé, jusqu'à ce que la position VSTART soit atteinte. Si ces deux mots de contrôle sont nuls, le contrôleur DMA termine ses accès DMA sprites,

par le biais des canaux correspondants, jusqu'au début de l'affichage de la prochaine image. A la fin du temps mort vertical, il recommencera à l'adresse se trouvant actuellement dans SPRxPT.

Reproduction de sprites sans DMA

On peut afficher un sprite sans canaux DMA, sans problème. Il faut alors écrire directement les mots de contrôle souhaités dans les registres SPRxPOS et SPRxCTL. La position HSTART et le bit AT devront obligatoirement renfermer des valeurs valides. VSTART et VSTOP seront utilisés exclusivement par les canaux DMA.

On peut activer la sortie sprite, à n'importe quelle ligne, dès qu'on écrit les deux mots de données dans SPRxDATA/B. Comme SPRxDATA active la sortie sprite, il est préférable d'écrire le deuxième mot de données dans SPRxDATB en premier. Si on ne modifie pas les données des deux registres, elles seront affichées à chaque ligne, pour devenir un montant vertical.

Si on désire désactiver à nouveau le sprite, il suffit d'écrire n'importe quelle valeur dans SPRxPOS.

*** Sprite Démo ***

;Registres circuits spéciaux

```
INTENA - $9A ;Registre Interruptenable (écriture)
INTREQ - $1e ;Registre Interruptrequest (lecture)
DMACON - $96 ;Registre contrôleur DMA (écriture)
COLOR00 - $180 ;Registre palette couleurs 0
VPOSR - $4 ;Position du faisceau (lecture)
JOYODAT - $A ;Position de la souris pour Port 0
```

;Registres Copper

```
COP1LC - $80 ;Adresse de la 1ère COPPER-LIST
COP2LC - $84 ;Adresse de la 2ème COPPER-LIST
COPJMP1 - $88 ;Saut à la COPPER-LIST 1
COPJMP2 - $8a ;Saut à la COPPER-LIST 2
```

;Registres Bitplane

```
BPLCON0 - $100 ;Registre de contrôle Bitplane 0
BPLCON1 - $102 ;1 (Valeurs de défilement)
BPLCON2 - $104 ;2 (Sprite<>Priorité Playfield)
BPL1PTH - $0E0 ;Pointeur sur le 1er Bitplane
BPL1PTL - $0E2 ;
BPL1MOD - $108 ;Valeur modulo pour les Bit-Planes impairs
BPL2MOD - $10A ;Valeur modulo pour les Bit-Planes pairs
DIWSTRT - $08E ;Début de la fenêtre d'affichage
DIWSTOP - $090 ;Fin de la fenêtre d'affichage
DDFSTRT - $092 ;Bit-Plane DMA Début
DDFSTOP - $094 ;Bit-Plane DMA Stop
```

;Registres Sprite

```

        SPROPTH = $120 ;Pointeur sur la liste des données Sprite pour
Sprite 1
        SPROPTL = $122
SPR1PTH = $124
SPR1PTL = $126

;CIA-A Registre de port A (Bouton de la souris)

        CIAAPRA = $bfe001

;Exec Library Base Offsets

        OpenLibrary = -30-522 ;LibName,Version/a1,d0
        Forbid = -30-102
        Permit = -30-108
        AllocMem = -30-168 ;ByteSize,Requirements/d0,d1
        FreeMem = -30-180 ;MemoryBlock,ByteSize/a1,d0

;graphics base

        StartList = 38

;Autres labels

        Execbase = 4
        Planesize = 52*345 ;Taille du Bitplane
        Planewidth = 52
        CLsize = 19*4 ;Taille de la COPPER-LIST en octets
        Chip = 2 ;Demander Chipram
        Clear = Chip+$10000 ;Effacer d'abord Chipram

;*** Début du programme ***

Start:
;Demander de la mémoire pour les bitplanes

        move.l Execbase,a6
        move.l #Planesize,d0 ;Besoin en mémoire des bitplanes
        move.l #clear,d1
        jsr AllocMem(a6) ;Demander de la mémoire
        move.l d0,Planeadr
        beq.L End ;Erreur! -> Fin

;Demander de la mémoire pour la COPPER-LIST

        moveq #CLsize,d0
        moveq #chip,d1
        jsr AllocMem(a6)
        move.l d0,CLadr
        beq.L FreePlane ;Erreur! -> FreePlane

;Demander de la mémoire pour la liste des données Sprite

        moveq #Sprsize,d0
        moveq #chip,d1

```

```
jsr AllocMem(a6)
move.l d0,Spradr
beq.l FreeCL

;Créer une COPPER-LIST dans le chipram

;Bitplanepointer

    move.l CLadr,a0
    move.w #bpl1pt1,d2
    move.l Planeadr,d1
    bsr setadr

;Pointeur sur le 1er sprite

    move.w #sprOpt1,d2
    move.l Spradr,d1
    bsr setadr

;Autres pointeurs de sprite (non utilisés)

    moveq #6,d0
    move.w #spr1pt1,d3

spr_set:
    move.l Spradr+Sprsize-4,d1
    move.w d3,d2
    bsr setadr
    addq.w #4,d3
    dbf d0,spr_set
    move.l #$ffffffe,(a0)

;Copier la liste des données sprite

    move.w #Sprsize/4-1,d0
    lea Sprstart,a0
    move.l Spradr,a1

spr_copy:
    move.l (a0)+,(a1)-
    dbf d0,spr_copy

;*** Programme principal ***

;Bloquer DMA et Taskswitching

    jsr forbid(a6)
    lea $dff000,a5
    move.w #$0300,dmacon(a5)

;Initialiser le Copper

    move.l CLadr,cop1lc(a5)
    clr.w copjmp1(a5)

;Initialiser le Playfield
```

```

move.w #$0,color00(a5)           ;Couleurs Playfield
move.w #$0f00,color00+2(a5)
move.w #$000f,color00+34(a5)    ;Couleurs Sprite
move.w #$00ff,color00+36(a5)
move.w #$00f0,color00+38(a5)
move.w #$1a64,diwstrt(a5) ;26,100
move.w #$39d1,diwstop(a5) ;313,465
move.w #$0028,ddfstrt(a5)
move.w #$00d8,ddfstop(a5)
move.w #%0001001000000000,bplcon0(a5)
clr.w bplcon1(a5)
move.w #8,bplcon2(a5)
move.w #2,bpl1mod(a5)

;DMA actif

move.w #$83a0,dmacon(a5)         ;Bitplane & Sprite DMA activés

;Remplir les Bitplanes d'un motif à carreaux

move.l planeaddr,a0
move.w #planesize/4-1,d0 ;Compteur de la boucle
move.w #13*16,d1
move.l #$ffff0000,d2      ;Motif à carreaux
move.w d1,d3

fill: move.l d2,(a0)+
subq.w #1,d3
bne.s weiter
swap d2                      ;Changer de motif
move.w d1,d3

weiter: dbf d0,fill

;Attendre sur la ligne raset 16 (après les interrupts Exec)

wait: btst #6,ciaapra ;Bouton de la souris pressé?
beq.s end
move.l vposr(a5),d2
and.l #$0001FF00,d2
cmp.l #$00001000,d2
bne.S wait

;Sprite

move.w joy0dat(a5),d0 ;Position de la souris
move.w d0,d1
and.w #$ff,d0
lsr.w #8,d1
add.w #150,d0 ;Ajouter offset pour la position 0
add.w #30,d1 ;pour que le sprite reste toujours visible
jsr setcor ;Présente la position du sprite dans d0,d1

bra.S wait          ;Non -> poursuivre

;*** Partie finale ***

;Réactiver l'ancienne COPPER-LIST

```

```
end:    move.l #GRname,a1    ;Définir paramètres pour OpenLibrary
        clr.l d0
        jsr OpenLibrary(a6) :Ouvrir Graphics Library
        move.l d0,a4
        move.l StartList(a4),cop1lc(a5)
        clr.w copjmpl(a5)
        move.w #$83a0,dmacon(a5)
        jsr permit(a6)

;Libérer la mémoire pour les données des sprites

        move.l Spradr,a1
        moveq #Sprsize,d0
        jsr FreeMem(a6)

;Libérer la mémoire pour la liste des Coppers

FreeCL:
        move.l CLadr,a1    ;Définir paramètres pour FreeMem
        moveq #CLsize,d0
        jsr FreeMem(a6)

;Libérer la mémoire pour les bitplanes

FreePlane:
        move.l Planeadr,a1
        move.l #Planesize,d0
        jsr FreeMem(a6)

Fin:
        clr.l d0
        rts                  ;Quitter le programme

;Sous-programmes

;setadr écrit les commandes Copper pour l'initialisation d'un compteur
DMA

;dans la COPPER-LIST
;a0 - Pointeurs sur la COPPER-LIST (incrémenté par setadr)
;d1 - Adresse à écrire (par exemple bitplane)
;d2 - Adresse du registre des pointeurs, low (par exemple bp1lpt1)
setadr:
        move.w d2,(a0)+ ;move pt1
        move.w d1,(a0)+ ;Adrbits 1-15
        swap d1
        subq.w #2,d2      ;passer à pth
        move.w d2,(a0)+ ;move pth
        move.w d1,(a0)+ ;Adrbits 16-18
        rts
;"setcor" écrit dans le chip ram les coordonnées X,Y du sprite
;dans la liste des données du sprite
;d0,d1 - Coordonnées X,Y
;Adresse de la liste des données du sprite: Spradr
;Hauteur du sprite en lignes: Sprhigh
;a0,d2,d3 sont utilisés intérieurement
setcor:
        movem.l d0-d3/a0,-(sp)
```

```

move.w d0,d3      ;bit H0 dans le second mot de contrôle
and.w #1,d3       ;effacer le reste
lsr.w #1,d0       ;H1-H8 en position
move.w d0,d2       ;dans le premier mot de contrôle
and.w #$ff,d2      ;effacer E0-E7
move.w d1,d0
add.l #Sprhigh,d0 ;dernière ligne du sprite dans d0
as1.w #8,d1
bcc noE8
bset #2,d3        ;définir E8 dans le second mot
noE8:  or.w d1,d2   ;E0-E7 dans le premier mot
asl.w #8,d0
bcc noL8          ;définir L8
bset #1,d3
noL8:  or.w d0,d3   ;L0-L7 dans le second mot
move.l Spradr,a0   ;transfert des nouvelles valeurs dans la mémoire
move.w d2,(a0)+
move.w d3,(a0)

movem.l (sp)+,d0-d3/a0
rts

;Variables
CLadr:    dc.l 0
Planeadr: dc.l 0
Spradr:   dc.l 0
test:     dc.l 0

;Constantes
GRname: dc.b "graphics.library",0

;Liste des données du sprite
even

Sprstart:
dc.w $a05a,$a800
dc.w %0000000000000000,%00000001111000000
dc.w %0000000000000000,%00000110000110000
dc.w %0000000110000000,%0001000110001000
dc.w %0000000111100000,%00011001001001100
dc.w %0000000111100000,%00011001001001100
dc.w %0000000110000000,%00001000110001000
dc.w %0000000000000000,%00000110000110000
dc.w %0000000000000000,%00000001111000000

Sproff: dc.w 0,0
Sprend:

Sprsize = Sprend-Sprstart
Sprhigh = 9

```

1.5.7. Le blitter

Le nom Blitter est un raccourci pour l'expression anglaise : 'BLock Image Transferer', ce qui signifie 'transfert de blocs de données image'. Son nom traduit exactement le rôle qu'il joue au sein de l'Amiga, c'est-à-dire, décaler et copier des blocs de données en mémoire, ces données étant pour la plupart, de type graphique. Le Blitter peut aussi réunir plusieurs zones mémoires de façon logique et remplacer le résultat en mémoire. Il exécute toutes ces tâches très rapidement. Il peut réaliser un décalage de données en traitant plus de 16 millions de données dans la même seconde.

De plus, le Blitter peut aussi remplir des surfaces et tracer des lignes. La combinaison de ces deux possibilités autorise le dessin de n'importe quelle figure à plusieurs côtés et ceci beaucoup plus rapidement que n'aurait pu le faire le 68000.

Le système d'exploitation utilise le Blitter pour pratiquement toutes les opérations graphiques. Ce dernier s'occupe de l'affichage du texte, de dessiner les gadgets, de décaler les fenêtres etc...

Il a aussi la responsabilité du décodage des données disquette, ce qui prouve que le Blitter ne se limite pas aux données graphiques.

L'utilisation du blitter dans la copie de données

Le Blitter copie toujours les données suivant le même modèle : une à trois zones mémoires différentes, qui représentent la source des données, seront réunies, suivant les conditions logiques choisies et le résultat sera à nouveau réécrit en mémoire. Ce dernier peut aller d'une simple copie à une union complexe de plusieurs zones de données. Les adresses des zones sources de données se nomment A, B et C, la zone cible étant D. Celles-ci peuvent être choisies librement dans toute la CHIP-RAM (adresses de 0 à \$7FFF).

Le nombre de mots qui peuvent être traités en une opération du Blitter, peut atteindre 65536. Il peut donc y avoir transfert en mémoire de 128 Koctets en une seule fois.

Le Blitter forme donc des zones mémoires 'à angle droit', c'est-à-dire que la mémoire est découpée en colonnes et en lignes, comme pour un bitmap. Il est aussi possible de traiter une petite zone incluse dans un grand bitmap, en utilisant des valeurs modulo.

Ces dernières ont aussi été utilisées dans la définition des bitplanes d'un playfield et avaient alors la largeur de fenêtre d'écran.

Pour initialiser une opération du Blitter, les démarches suivantes sont importantes :

- ✓ Choix du mode du Blitter : copie des données.
- ✓ Sélection de la zone de données source (l'utilisation de trois zones mémoires n'est pas obligatoire) et de la zone cible.
- ✓ Choix de la liaison logique.
- ✓ Définition d'autres paramètres (scrolling, masques, sens d'adressage).

- ✓ Détermination de la fenêtre dans laquelle l'opération Blitter sera exécutée, et démarrage du Blitter.

Etablir une fenêtre Blitter

Vous vous demandez peut-être pourquoi nous commençons par la description du dernier point de la suite de démarches énoncées plus haut. En fait, la définition de la fenêtre souhaitée est l'élément de base de toutes les autres opérations. Mais lorsqu'on programme le Blitter, cette valeur sera écrite à la fin dans le registre correspondant, étant donné que le Blitter débutera ses opérations avec cette valeur. C'est pour cette raison que ce point se trouve à la fin de la liste. Mais pour la compréhension des autres valeurs, il est cependant nécessaire de connaître la notion de fenêtre Blitter.

La fenêtre Blitter est la zone mémoire traitée par le Blitter lors de ses opérations. Elle est structurée comme un bitplane, c'est-à-dire découpée en lignes et colonnes, où une colonne correspond à un mot. Le nombre de mots dans une fenêtre est donc le produit des lignes par les colonnes : $L \times C$.

A travers ce découpage de la zone mémoire, le traitement des bitplanes par le Blitter est grandement facilité et ceci d'une telle manière qu'on peut parler ici de zones mémoires linéaires.

Le découpage en lignes et colonnes n'est là que pour faciliter la programmation. En vérité, l'ordre de toutes les lignes en mémoire correspond à une suite régulière d'adresses. Pour un petit champ de données, qui n'est pas découpé en lignes et colonnes, il est possible d'initialiser une fenêtre d'une taille 1.

Le Blitter traite sa fenêtre ligne par ligne. Les opérations Blitter débutent avec le premier mot de la première ligne et finissent avec le dernier mot de la dernière ligne.

Le registre BLTSIZE renferme la taille de la fenêtre.

BLTSIZE \$058 (écriture seulement)

Bit n° :	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fonction :	H9	H8	H7	H6	H5	H4	H3	H2	H1	H0	W5	W4	W3	W2	W1	W0

H0-H9 Ces dix bits reproduisent la hauteur (Height) de la fenêtre Blitter. La fenêtre peut avoir une hauteur variant de 1 à 1024 lignes ($2^{10} = 1024$). La hauteur maximale est sélectionnée en fixant Height à 0. Toutes les autres valeurs correspondent directement au nombre de lignes. Une hauteur de 0 ligne est impossible.

W0-W5 Ces six bits représentent la largeur (Width) de la fenêtre. Cette dernière peut comporter de 1 à 64 mots ($2^6 = 64$). Si on convertit cette valeur en points graphiques (1 mot = 16 points), on obtient un maximum de 1024 points. Ce dernier est sélectionné si on met Width à 0.

Pour trouver la valeur du registre BLTSIZE à partir de la hauteur et de la largeur, on se sert de la formule suivante :

$$\text{BLTSIZE} = \text{Hauteur} * 64 + \text{Largeur}$$

Si on veut atteindre les deux maxima Height = 1024 et Width = 64, on est obligé de la modifier quelque peu.

$$\text{BLTSIZE} = (\text{Hauteur AND } \$3FF) * 64 + (\text{Largeur AND } \$3F)$$

Le registre BLTSIZE devra toujours être initialisé en dernier. Le Blitter démarrera automatiquement lors de l'accès écriture à ce registre.

Zone de données Source et Cible

Lors d'une opération Blitter, les données issues de zones mémoires différentes seront reliées entre elles. Même lorsque la fenêtre Blitter établit le nombre et l'ordre de traitement des données, le positionnement des trois zones sources et de la zone cible de cette fenêtre doit être déterminé.

Prenons pour exemple un petit graphique à angles droits, que le Blitter doit copier dans la mémoire d'écran et qui se trouve à un endroit quelconque de la CHIP-RAM. Il n'y aura, pour cette simple tâche, qu'une zone source. Le choix de la fenêtre Blitter est simple. Le graphique devant être copié entièrement, la taille de la fenêtre correspondra à la hauteur et à la largeur du graphique en mémoire.

Pour que le Blitter sache aussi où ce graphique se trouve, il suffit d'écrire l'adresse du premier mot de la ligne supérieure dans le registre correspondant.

La définition de la zone cible est sensiblement différente. Le graphique devant être copié dans la mémoire d'écran, il sera obligatoirement transféré dans le bitplane actuel. Le problème est que le bitplane est plus étendu que notre simple graphique.

La copie directe dans le bitplane par le Blitter, pourrait paraître quelque peu curieuse. Ainsi, à côté de l'adresse de la zone cible, la largeur du graphique devra aussi être communiquée au Blitter, au moyen d'une valeur modulo. Cette dernière sera additionnée à l'adresse du pointeur, après chaque traitement de ligne. Par conséquent, les mots non influencés du bitplane seront sautés et le pointeur se tiendra à nouveau au début de la prochaine ligne. Comme la zone mémoire source et la zone cible peuvent avoir une largeur différente, il existe pour chaque zone, un registre modulo indépendant.

Recopie d'un graphique dans un BitPlane

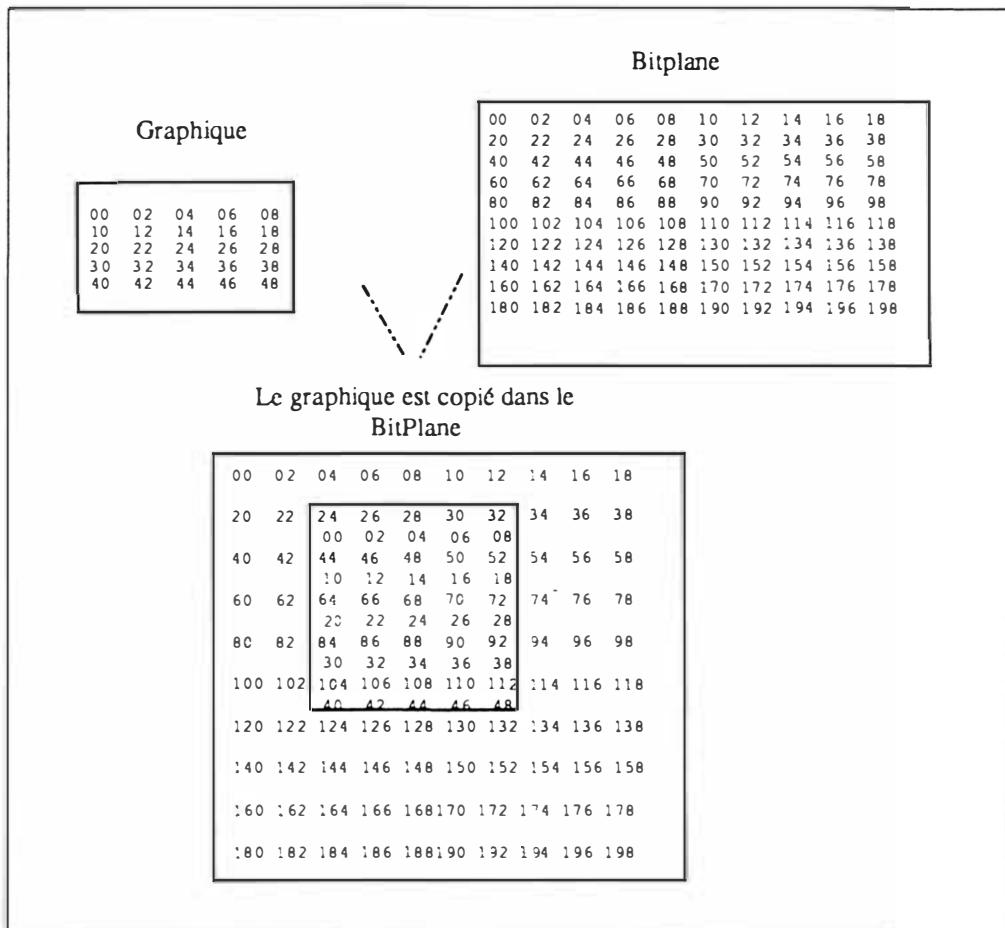


Figure 1 - 35

Ce schéma illustre bien notre exemple. Le graphique est composé de 5 lignes d'une largeur de 10 mots. Le bitplane a une hauteur de 10 lignes sur 20 mots. Le choix de la fenêtre Blitter de l'adresse de départ et de la valeur modulo se fera de la manière suivante :

La fenêtre Blitter doit correspondre au graphique pour que ce dernier puisse être copié entièrement, c'est-à-dire que la hauteur de la fenêtre sera de 5 lignes et que la largeur sera de 10 mots. La valeur qui sera mise en dernier dans le registre BLTSIZE, correspondra à 330 ou, en valeur hexadécimale, à \$014A.

L'adresse de départ de la source de données est la même que celle du premier mot de données du graphique. Etant donné que la largeur d'une ligne du graphique est la même que celle de la fenêtre Blitter, la valeur modulo de la source reste à 0.

Il faut maintenant déterminer la valeur modulo de la zone cible de données. On établit pour cela la différence entre la largeur de la zone cible et de la fenêtre Blitter. Dans notre exemple, 20 mots - 10 mots : la valeur modulo de la zone cible correspondra à 10 mots. Dans le registre modulo du Blitter, la valeur modulo doit être donnée en octets : Valeur modulo = Modulo en mots *2.

En dernier, le Blitter nécessite encore l'adresse de départ de la zone cible. Elle détermine la position à laquelle le graphique sera copié dans le bitplane. Elle correspond donc à la somme de l'adresse de départ du bitplane et de l'adresse du mot, qui doit être placé dans le coin gauche supérieur du graphique. Sur le schéma, l'adresse de départ de la zone cible correspond à l'adresse du bitplane plus 24.

Déroulement d'une opération Blitter

Une fois que les adresses et la valeur modulo ont été établies, le Blitter commence la copie des données, après initialisation du registre BLTSIZE. Il prend le mot se trouvant à l'adresse de départ de la source de données et le met en mémoire à partir de l'adresse cible. Puis il additionne un mot aux deux adresses et copie le prochain mot. Ceci sera répété jusqu'à ce que le nombre de mots par ligne, contenus dans BLTSIZE, soit traité. Avant que le Blitter ne recommence à la prochaine ligne, il additionne la valeur modulo au pointeur d'adresse, afin que la prochaine ligne débute bien à la bonne adresse.

Lorsque toutes les lignes ont été copiées, le Blitter se désactive et attend le prochain transfert.

Le registre adresse renferme, après une opération du Blitter, l'adresse du dernier mot, plus deux, plus la valeur modulo.

Les registres adresses se nomment BLTxPT, où x correspond à l'une des trois sources A, B ou C, ou à la zone cible D. Le registre d'adresses est formé d'un registre renfermant les bits 0-15 et d'un autre renfermant les bits 16-18 :

Adresse	Nom	Fonction	
048	BLTCPPTH	Adresse de départ de la zone source de données C.	(bits 16-18)
04A	BLTCPPL	zone source de données C.	(bits 0-15)
04C	BLTBPTH	Adresse de départ de la zone source de données B.	(bits 16-18)
04E	BLTBPL	zone source de données B.	(bits 0-15)
050	BLTAPTH	Adresse de départ de la zone source de données A.	(bits 16-18)
052	BLTAPTL	zone source de données A.	(bits 0-15)
054	BLTDPTH	Adresse de départ de la zone cible D.	(bits 16-18)

Adresse	Nom	Fonction
056	BLTDPTL	zone cible de données D. (bits 0-15)
Chacune des quatre zones possède un registre modulo propre :		
060	BLTCMOD	Valeur modulo pour la source C
062	BLTBMOD	Valeur modulo pour la source B
064	BLTAMOD	Valeur modulo pour la source A
068	BLTDMOD	Valeur modulo pour la zone cible D

Copie par incrémentation ou décrémentation d'adresse

Dans notre exemple, le Blitter travaille par incrémentation d'adresse, c'est-à-dire qu'il commence à une adresse de départ, qui sera augmentée continuellement d'un certain pas et ce jusqu'à l'adresse de fin. Logiquement, cette dernière est plus haute que l'adresse de départ.

Il se peut qu'un tel adressage nous mène à l'erreur : en effet, la copie d'une zone mémoire à une haute adresse peut avoir pour conséquence le chevauchement partiel des zones source et cible.

Voici un exemple :

Adresse	Données		Résultat	
	sources	cibles	Souhaité	Accidentel
0	source 1			
2	source 2			
4	source 3			
6	source 4	cible 1	source 1	source 1
8	source 5	cible 2	source 2	source 2
10		cible 3	source 3	source 3
12		cible 4	source 4	!! source 1 !!
14		cible 5	source 5	!! source 2 !!

Les 5 mots de données source doivent être transférés à l'adresse de données cible. Si le Blitter commence à la source 1, il écrira sur la source 4, lorsqu'il aura mis la source 1 en mémoire à l'adresse cible souhaitée. Ceci résulte du fait que cible 1 et source 4 possèdent les mêmes adresses, les deux zones se chevauchant. Il se passe la même chose pour source 5 et cible 2.

Lorsque le Blitter arrive à l'adresse de la source 4, il y trouvera la source 1. Ainsi cette dernière sera chargée à la place de la source 4 dans la zone cible 4. Ce sera à nouveau identique pour la source 2 dans la zone cible 5, alors que les sources 4 et 5 seront perdues.

Comme solution à ce problème, le Blitter possède en plus du mode ascending (adressage par incrémentation), le mode descending (adressage par décrémentation). Dans ce dernier mode, il débute à l'adresse se trouvant dans le pointeur BLTxPT, puis à chaque mot copié, il le diminue de deux octets. La valeur modulo ne sera pas additionnée, mais soustraite. L'adresse de fin se trouve donc avant l'adresse de départ.

Il faudra, bien sûr, en tenir compte lors de l'initialisation de BLTxPT. En mode normal on prend comme référence le coin supérieur gauche.

En mode descending, on prendra le coin inférieur droit, étant donné que l'adressage se fera par décrémentation.

Les valeurs modulo et BLTSIZE se déterminent de la même façon que celles du mode ascending.

Suivant les modes choisis, on peut faire les remarques suivantes :

- ① Aucun chevauchement entre les zones source et cible :

Que ce soit en mode ascending ou descending, les deux travaillent correctement dans ce cas.

- ② Chevauchement partiel entre les zones source et cible : (la zone cible se trouve avant la zone source).

Seul le mode ascending travaille correctement.

- ③ Chevauchement partiel entre les zones source et cible : (la zone cible se trouve derrière la zone source).

Seul le mode descending travaille correctement.

La sélection des liaisons logiques

Comme cela a déjà été vu, il peut y avoir trois sources de données qui seront reliées pour donner les données cible. Ce lien logique ne se déroule que bit à bit, c'est-à-dire que les trois bits de données A, B et C doivent donner le bit cible D.

Le Blitter connaît 256 différentes liaisons. Elles peuvent se différencier en deux groupes :

- ① On peut appliquer 8 équations booléennes différentes sur les trois bits de données. Chacune libère comme résultat un 1, après combinaison de A, B et C.

- ② Les huit résultats des équations, citées plus haut, seront réunis par un OU logique. Le résultat sera le bit cible D.

La notion d'équation booléenne correspond à une formule mathématique qui reproduit une combinaison d'une liaison logique.

Cette façon de calculer est appelée algèbre booléenne, du mathématicien Georges BOOLE (1815 à 1864). Les explications à venir des fonctions logiques du Blitter ne peuvent être comprises sans quelques connaissances d'algèbre booléenne. Les équations booléennes seront toutefois exposées.

Trois bits donnent huit combinaisons possibles. Chacune des huit équations révèle un résultat vrai (résultat = 1) dans une des combinaisons. On peut choisir, au moyen de huit bits de contrôle LF0 à LF7, si le résultat de l'équation doit être sélectionné ou non, pour former D. Chaque résultat sous forme de bit, les bits correspondants étant à 1, seront liés entre eux par un OU logique. Une liaison OU signifie que le résultat est 1, si au moins une entrée est à 1. Autrement dit, un OU logique libère un 0 lorsque toutes les entrées sont à 0.

Au moyen des 8 bits LFx, on peut ainsi choisir par quelle combinaison des trois bits d'entrée A, B et C, on obtient le bit de sortie D, égal à 1.

Les 8 équations booléennes d'entrée sont caractérisées en anglais par le terme, 'Minterms'.

Le tableau suivant donne un aperçu des combinaisons d'entrée, obtenues avec les bits LFx (dans la colonne Minterm, le caractère minuscule désigne une liaison NOT du bit d'entrée correspondant. En temps normal, cet état est caractérisé par une barre horizontale au dessus du caractère).

La colonne 'bits d'entrée' renferme les combinaisons de bits telles qu'elles seront exécutées par les équations correspondantes. L'ordre des bits est le suivant : A, B puis C.

	LF7	LF6	LF5	LF4	LF3	LF2	LF1	LF0
Minterm :	ABC							
Bits d'entrée :	111	110	101	100	011	010	001	000

Le choix d'un Minterm est simple. Il suffit d'activer tous les bits LFx, de telle façon que leur combinaison d'entrée forme le bit de sortie D égal à 1.

Dans notre premier exemple, les données sources A devaient être copiées directement dans la zone cible D. Les sources B et C ne sont pas utilisées. La sélection du Minterm se passera donc de la manière suivante :

D doit être à 1, lorsque A = 1. Seuls les 4 termes supérieurs obéissent à ces conditions, étant donné que A est à 1. Comme B ne joue aucun rôle, on peut choisir deux termes, où B peut être soit à 1 soit à 0, qui sont cependant identiques. B n'a plus aucune conséquence sur D, le reste de l'équation n'étant en rien influencé par les deux états possibles de B, et le résultat ne dépendant que de ce reste. Il se passe la même chose pour le bit C. Si on se réfère au tableau, on remarque que les 4 termes, LF4 à LF7, doivent être activés. Le résultat ne dépendra que de A, car après chaque combinaison de B et C, une de ces quatre équations sera toujours vrai pour A = 1 et D sera égal à 1. Si A = 0 les quatre équations seront fausses et D sera égal à 0.

Si vous avez quelques connaissances en algèbre booléenne, vous pouvez obtenir le Minterm suivant une approche différente. La formule à obtenir est A=D. Etant donné que B et C sont toujours présents dans le Blitter, vous devez les intégrer dans l'équation de la manière suivante :

$$A * (b+B) * (c+C) = D$$

Le terme $x+X$ est toujours vrai (égal à 1) et sera utilisé lorsque la valeur de x sera sans importance pour le résultat D. Afin d'obtenir le Minterm, on est obligé de développer la multiplication :

1. $A * (b+B) * (c+C) = D$
2. $(A * b + A * B) * (c+C) = D$
3. $A * b * c + A * B * c + A * b * C + A * B * C = D$

En enlevant le signe multiplicateur, on obtient :

$$Abc + AbC + Abc + ABC = D$$

Il ne reste plus alors qu'à activer les bits LFx des Minterms correspondants. Comme on peut le voir, on accède aussi au résultat par le biais de l'algèbre booléenne.

Voici d'autres exemples de combinaisons des bits LFx, utiles pour les opérations du Blitter :

- ✓ Inversion d'une zone de données : a=D

Combinaison LFx à obtenir : 00001111

En algèbre booléenne :

$$\begin{aligned} a &= D \\ a * (b+B) * (c+C) &= D \\ (a * b + a * B) * (a + C) &= D \\ abc + abC + abc + aBC &= D \end{aligned}$$

- ✓ Copie d'un graphique dans un bitplane, sans modifier son contenu ; ceci correspond à un OU logique entre le graphique A et le bitplane B : A+B = D.

Combinaison LFx à obtenir : 11111100

En algèbre booléenne :

$$\begin{aligned}
 A + B &= D \\
 A(b+B)(c+C) + B(a+A)(c+C) &= D \\
 (Ab+AB)(c+C) + (Ba+BA)(c+C) &= D \\
 Abc+AbC+ABC+ABC+Bac+BaC+BAC+BAC &= D \\
 Abc+ABC+AbC+ABC+aBc+aBC &= D
 \end{aligned}$$

Pour finir, voici encore les règles d'obtention des bits LFx nécessaires :

- ① Trouver par laquelle des 8 combinaisons de ABC, on obtient D=1.
- ② Activer les bits LFx de la combinaison correspondante.
- ③ Si les trois sources ne sont pas indispensables, toutes les combinaisons peuvent être choisies, dans lesquelles les bits inutilisés apparaissent et où les bits souhaités possèdent la bonne valeur.

Décalage des valeurs d'entrée

Pour certaines tâches, le fait que le Blitter soit relié au mot limite peut poser un problème. Il peut arriver, par exemple, qu'une zone déterminée se trouvant à l'intérieur d'une bitmap doit être décalée d'un certain nombre de points. Dans ce cas, le Blitter ne peut déplacer les bits de données que sur la partie d'un mot. Autrement le Blitter est obligé d'écrire un graphique à une position déterminée de la mémoire d'écran, les coordonnées ne concordant évidemment pas avec une limite de mot.

Pour résoudre ce problème, le Blitter a la possibilité de décaler les mots de données des sources A et B de 15 bits vers la droite. Il est alors en situation de mettre les données à chaque position de point souhaitée. Tous les bits qui seront décalés vers la droite par ce processus de déplacement, arriveront dans le mot suivant, où les bits ont été libérés.

Pratiquement toute la ligne sera décalée. Ce déplacement se nomme décalage en cylindre. Le Blitter ne nécessite pas de temps en plus pour ce genre de processus et ceci indépendamment du nombre de bits décalés. Le décalage de données ne limite en aucun cas la vitesse du Blitter.

Exemple de décalage de données sur trois bits :

AVANT

Mot de données 1 00011111 10011100	Mot de données 2 00010101 01111111	Mot de données 3 11100001 11100101
---------------------------------------	---------------------------------------	---------------------------------------

APRES

Mot de données 1 xxx00011 11110011	Mot de données 2 10000010 10101111	Mot de données 3 11111100 00111100
---------------------------------------	---------------------------------------	---------------------------------------

Les trois bits `xxx` dépendent du mot de données précédent, d'où ils proviennent.

Les masques

Il est possible que le Blitter soit obligé de copier un graphique de la mémoire d'écran, dont les bords ne correspondent pas au mot limite. Les données qui se trouvent à gauche de la bordure du graphique, tout en appartenant aux premiers mots de données, ne doivent pas être copiées. Afin de rendre ceci possible, le Blitter peut traiter le premier et dernier mot de données d'une ligne avec un masque. Ceci signifie qu'on peut choisir les bits de ces mots, qui seront pris en charge. On pourra ainsi éliminer les bits indésirables des bordures.

Cette possibilité de masquer n'existe que pour la source A. Deux registres contiennent les masques des deux bordures. Seuls les bits activés seront pris en compte, les autres étant éliminés.

\$044 BLTAFWM Blitter Source A First Word Mask

(Masque pour le premier mot de données)

\$046 BLTALWM Blitter Source A Last Word Mask

(Masque pour le dernier mot de données de la ligne)

Les bits 0-15 renferment les bits masque correspondants.

Exemple :

('1' correspond à un bit activé, '.' correspond à un bit masqué).

Données graphiques du bitplane :

Colonne 1	Colonne 2	Colonne 3
.....11111111	1111111111111111	1.....11
i11111.....1111	11.....1111	1111.....1111
....11.....11	1111.....111	11111.....11111111
....11.....1	11111.....11	1111111111111111
....11.....1	11111.....11	1111111111111111
....11.....11	1111.....111	11111.....11111111
111111.....1111	11.....1111	1111.....1111
.....11111111	1111111111111111	1.....11

Premier mot masque :
0000000011111111

Dernier mot masque :
1111110000000000

Résultat :

Colonne 1	Colonne 2	Colonne 3
.....11111111	1111111111111111	1.....
.....1111	11.....1111	1111.....
.....11	1111.....111	11111.....
.....1	11111.....11	111111.....
.....1	11111.....11	1111111.....
.....11	1111.....111	111111.....
.....1111	11.....1111	1111.....
.....11111111	1111111111111111	1.....

Après avoir masqué les éléments indésirables de l'image sur les bordures, on obtient le graphique souhaité.

ATTENTION !!!

Si la largeur de la fenêtre d'écran correspond exclusivement à un mot (BLTSIZE Width=1), les deux masques sont appliqués ensemble. Ils influencent ensemble le mot d'entrée. Seuls les bits d'entrées, activés dans les deux masques, seront pris en compte.

Les registres de contrôle Blitter

Le Blitter possède deux registres de contrôle, BLTCON0 et BLTCON1. On y trouve différents bits de contrôle nécessaires à la gestion du Blitter.

BLTCON0 \$040

Bit n°	Nom	Fonction
15	ASH3	Ces quatre bits renferment la valeur de décalage
14	ASH2	des données d'entrée de la source A
13	ASH1	ASH0-3 = 0 ne produit aucun décalage
12	ASH0	
11	USEA	Active le canal DMA de la source A
10	USEB	Active le canal DMA de la source B
9	USEC	Active le canal DMA de la source C
8	USED	Active le canal DMA de la zone cible D
7	LF7	Choix Minterm ABC (combinaison de bit ABC: 111)
6	LF6	Choix Minterm ABC (combinaison de bit ABC: 110)
5	LF5	Choix Minterm AbC (combinaison de bit ABC: 101)
4	LF4	Choix Minterm Abc (combinaison de bit ABC: 100)

Bit n°	Nom	Fonction
3	LF3	Choix Minterm aBC (combinaison de bit ABC: 011)
2	LF2	Choix Minterm aBc (combinaison de bit ABC: 010)
1	LF1	Choix Minterm abC (combinaison de bit ABC: 001)
0	LF0	Choix Minterm abc (combinaison de bit ABC: 000)

BLTCON1 \$042

Bit n°	Nom	Fonction
15	BSH3	Ces quatre bits renferment la valeur
14	BSH2	de décalage des données d'entrée de la source B
13	BSH1	BSH0-3 = 0 ne produit aucun décalage
12	BSH0	
10-5		inutilisé
4	EFE	Exclusive Fill Enable
3	IFE	Inclusive Fill Enable
2	FCI	Fill Carry In
1	DESC	DESC = 1 commute en mode descending
0	LINE	LINE = 1 active le mode lignes

Le bit LINE commute le Blitter en mode dessin de ligne. Si on utilise le Blitter pour copier des données, ce bit doit être à 0.

Avec le bit DESC, on peut choisir entre le mode d'adressage par incrémentation et le mode par décrémentation. Si DESC = 0, c'est le mode ascending qui sera utilisé, le mode descending étant utilisé avec le bit DESC = 1.

Les bits EFE et IFE activent la routine de remplissage des surfaces. Les deux doivent être à 0 si on veut utiliser le Blitter normalement.

Le bit FCI a une fonction liée au mode de remplissage.

Le DMA Blitter

Les données des zones source A, B et C et cible D ont accès à la mémoire, que ce soit en mode lecture ou écriture, au moyen de 4 canaux DMA. On peut activer ces DMA Blitter par l'intermédiaire du bit BLTEN (bit 6) du registre DMAON. Les 4 canaux

seront activés ensemble. Le Blitter possède pour ses transferts DMA, 4 registres de données :

Adresse	Nom	Fonction
\$000	BLTDDAT	Sortie de données D
\$070	BLTCDAT	Registre de données de la source C
\$072	BLTBDAT	Registre de données de la source B
\$076	BLTADAT	Registre de données de la source A

Le contrôleur DMA lit les données d'entrées nécessaires dans la mémoire et les écrit dans le registre de données. Si le Blitter a traité les données d'entrée, c'est BLTDDAT qui contiendra le résultat. Le contrôleur DMA transférera alors le contenu de ce registre dans la CHIP-RAM.

On commute le transfert DMA des 4 registres au moyen des bits USEx. Si USEA=0, on désactivera par exemple, le canal DMA sur le registre A. Le Blitter continuera à accéder à la valeur dans BLTADAT. Autrement dit, à chaque nouveau mot de la source active, le même mot sera issu de la source A. Pour cette raison, il est préférable de mettre les bits USEx à 0 pour les sources inutilisées, ainsi que d'exclure toute influence de celles-ci dans la sélection correspondante du Minterm.

Une autre possibilité est l'utilisation consciente du suivant, qu'après désactivation du canal DMA, ce sera toujours le même mot de données qui sera utilisé. On pourra ainsi, par exemple, remplir la mémoire suivant un modèle que l'on a écrit directement dans le registre BLTxDAT, à l'aide du processeur.

En dehors de BLTEN, 3 autres bits du registre DMACON appartiennent au Blitter :

Bit 10 BLTPRI

Le rôle de ce bit a déjà été expliqué dans le chapitre 'Eléments de base'. S'il est à 1, le Blitter a la priorité absolue sur le processeur.

Bit 14 BBUSY (ne peut être utilisé qu'en lecture)

BBUSY signale l'état du Blitter. S'il est à 1, cela signifie qu'une opération se déroule à ce moment-là.

Après activation de la fenêtre Blitter dans BLTSIZE, le Blitter débute ses accès DMA tout en activant BBUSY, jusqu'à ce que le dernier mot de la fenêtre soit traité et à nouveau réécrit en mémoire. Il termine alors ses accès et désactive BBUSY.

En même temps, le bit Blitter-Finished sera initialisé dans le registre Interrupt-Request.

Bit 13 BZERO

Le bit BZERO indique si, lors d'une opération Blitter, tous les bits résultats sont nuls. En d'autres termes, BZERO est initialisé lorsque, dans tous les mots de données, aucune des liaisons choisies ne libère un 1 comme résultat. Avec l'aide de ce bit, on peut organiser un test de collision. Il suffit pour cela d'activer les Minterms d'une telle manière que D ne soit égal à 1 que lorsque les deux sources sont à 1. Si les graphiques se chevauchent dans les deux zones sources, même d'un seul point, le résultat sera 1. A la fin de l'opération Blitter, on pourra ainsi déterminer s'il y a eu collision ou non. USED sera initialisé à 0, afin que les données de sortie ne soient pas écrites en mémoire.

Utilisation du Blitter pour remplir des surfaces

Le Blitter entend par surface une zone mémoire bidimensionnelle, qu'il doit remplir avec des points. Normalement, cette surface appartient à un graphique ou à un bitplane.

Afin de pouvoir remplir une surface, le Blitter doit connaître ses limites. Une définition compréhensible des lignes limites est absolument nécessaire pour le Blitter.

De nombreuses fonctions de remplissage existent dans la plupart des programmes de dessin, ou par exemple en Amiga-Basic avec l'instruction PAINT.

Avec elles, une zone de l'écran sera remplie à partir d'un point de départ, et ce jusqu'à ce que le programme rencontre une ligne frontière. Ainsi, n'importe quelle surface se laissera colorier, avec pour seule restriction qu'elle soit entièrement fermée par une ligne continue. Le Blitter n'a pas la possibilité d'exécuter une opération de remplissage aussi complexe. Il travaille seulement ligne par ligne, en remplittant les espaces vides entre deux bits activés, qui représentent les limites de la surface désirée. Les deux exemples suivant montre le déroulement du processus de remplissage du Blitter :

Opération de remplissage sans erreur :

Avant	Après
.....1.1.....111.....
.....1.....1.....1111111111.....
.....1.....1.....1111111111.....
.....1....1....1....1.....111111....111111.....
.....1....1....1....1.....111111....111111.....
.....1.....1.....111111111111.....
.....1.....1.....111111111111.....
.....1.....1.....111111111111.....

Opération de remplissage avec erreurs, du fait du mauvais choix des bits de limite.

Avant	Après
.....111.....	111111111111.....
.....111.....111.....	1111111111.....
.....11.....111.....11.....	11111111111111.....11.....
.....1.....1.....1.....1.....	1111111.....111111.....
.....1.....1.....1.....1.....	1111111.....111111.....
.....11.....11.....11.....11.....	1111111111111111.....11.....
.....1.....1.....1.....1.....	111111111111.....
.....1111111111111111.....	1111111111111111.....

Dans le premier exemple, une surface correctement limitée sera correctement remplie. Ce n'est évidemment pas le cas sur la figure n°2. On a dessiné ici une limite entièrement close et continue. Si on cherche à remplir une telle figure par le Blitter, il n'en résultera que le chaos.

La raison réside dans l'algorithme du Blitter. Il est assez simple. Le Blitter commence sur le côté droit de la ligne.

Pour déterminer si un bit doit être activé, il utilise le bit FillCarry (FC). Normalement, il est à zéro au départ. Le Blitter teste alors la valeur du bit le plus à droite. S'il est nul, la valeur du bit FC reste inchangée, et le bit de sortie prendra cette valeur, c'est-à-dire 0. Dans ce cas, le Blitter continuera avec le bit voisin, jusqu'à ce qu'il rencontre un bit d'entrée initialisé. Le bit FC sera mis à 1 et comme le bit de sortie correspond à la valeur actuelle de FC, il sera aussi mis à 1. Dès que le Blitter rencontrera à nouveau un bit initialisé, le bit FC sera à nouveau effacé et mis à 0. De cette manière, une zone se trouvant entre deux bits activés sera toujours remplie. Comme on peut le remarquer sur le deuxième exemple, la présence d'un nombre impair de bits perturbe la logique de remplissage.

La valeur de départ du bit FC détermine le bit FCI (FillCarryIN) dans le registre BLTCON1. Si FCI n'est pas activé, tout se déroule comme cela a été expliqué plus haut. Si FCI = 1, le Blitter commence à remplir la ligne, dès la bordure, jusqu'à ce qu'il rencontre un bit d'entrée activé. Le processus de remplissage se déroulera alors de la manière inverse.

Exemple du rôle du bit FCI :

Graphique de sortie	FCI=0	FCI=1
.....1.....1.....111111.....	1111111.....111111
.....1.....1....111111111111...	11111.....11111111
....1.....1.1.....1..	...111111.111111..	111.....111.....11
....1.....1.1.....1..	...111111.111111..	111.....111.....11
....1.....1....1....	...111111111111...	11111.....111111
....1.....1....1....	...111111111111...	11111.....111111

Sur les exemples ci-dessus, les bits d'entrée, c'est-à-dire les limites des bordures, restent présent après le remplissage de l'image. C'est toujours le cas lorsqu'on sélectionne le mode de remplissage par initialisation du bit ICE (Inclusive Fill Enable) dans le registre BLTCON1.

L'inverse s'obtient avec le mode ECE (Exclusive Fill Enable) et sera sélectionné en initialisant le bit de même nom dans le registre BLTCON1. Avec lui, le bit limite gauche d'une bordure d'une surface remplie (toujours quand le bit Fill-Carry change de 1 à 0) ne sera pas pris en compte dans l'image de sortie. La surface de cette dernière sera donc réduite d'un point. Il n'est possible qu'en mode ECE d'obtenir une surface d'une largeur d'un bit. Ceci n'est pas possible en mode ICE puisqu'au minimum deux points sont nécessaires pour former les limites d'une figure qui puisse s'afficher sans problème sur la sortie image.

Differences entre les modes ICE et ECE :

Sortie image	ICE	ECE
.....11.....1111.....111.....11
....1...1.....1.111111....1111111.....111
..1....11..1..1..1	...1111111111..11111111..111....111
1.....11..11....1	11111111111111111111	.1111111..1111..1111
..1.....1.....1	..111111111111111111	...111111111111111111
....1.....1.....1.111111111111111111111111111111111111
.....1....1.....11..11111....11...1111.....11..

Modèle d'entrée : 11010010

Bit Nr. d'entrée	bit	FCI = 0		FCI = 1	
		FC	ICE ECE	FC	ICE ECE
-	11010010	FC-FCI		FC-FCI	
0	0	0	0	1	1
1	1	1	10	0	11
2	0	1	110	0	011
3	0	1	1110	0	0011
4	1	0	11110	01110	1
5	0	0	011110	001110	1
6	1	1	1011110	1001110	0
7	1	0	11011110	01001110	1

(FC=FCI signifie que le bit FC accepte la valeur du bit FCI issue de BLTCON1, avant le début du processus de remplissage).

Voici la façon de démarrer une opération de remplissage du Blitter :

Le Blitter peut exécuter une opération de remplissage en même temps qu'un processus de copie. Il sera initialisé après avoir sélectionné l'un des deux modes, ICE ou ECE dans le registre BLTCON1. Le Blitter formera comme toujours, à partir des trois sources A, B et C et du Minterm choisi, les données de sortie D. Si aucun des deux modes de remplissage n'est actif, le Blitter prend en charge ces données directement dans son registre de données de sortie (BLTDDAT, \$000), les données accédant à la mémoire via les canaux DMA, lorsque USED = 1.

En mode de remplissage, les données de sortie D seront utilisées comme données d'entrée du circuit de remplissage. Le résultat de l'opération de remplissage sera écrit alors dans le registre de données de sortie BLTDDAT.

Pour exécuter une opération de remplissage, les démarches suivantes sont nécessaires :

- ✓ BLTxPT, BLTMOD et les Minterms doivent être choisis de telle manière que les données de sortie D doivent renfermer les bits corrects limites de la surface à remplir.
- ✓ Il faut choisir le mode descending (le Blitter remplit de la droite vers la gauche, ceci ne fonctionnant que lorsque les mots ont des adresses décrémentées).
- ✓ Il faut choisir le mode de remplissage souhaité, c'est-à-dire activer ICE ou ECE.
- ✓ LINE = 0 (mode lignes désactivé).
- ✓ BLTSIZE initialisé à la taille du graphique à remplir.

Le Blitter débute alors le processus de remplissage. Lorsqu'il aura terminé, il mettra le bit BLTBUSY à 0.

La vitesse du Blitter ne sera pas influencée par l'activité du mode de remplissage. Le Blitter a la possibilité de remplir une surface avec une vitesse maximum de 16 millions de points par seconde. La principale utilisation du mode de remplissage est le dessin de figures pleines. Celles-ci seront dessinées dans une zone mémoire libre au moyen du mode lignes. Le Blitter remplira alors ces graphiques avec plus de rapidité.

Utilisation du Blitter pour tracer des lignes

Le Blitter offre de nombreuses possibilités. En dehors de la capacité à copier des données et à remplir des surfaces, il possède un mode de dessin de lignes. Comme les autres modes du Blitter, il est assez rapide : jusqu'à 1 million de points par seconde. Seule l'utilisation du processeur 68020 peut permettre d'atteindre cette vitesse, et encore, avec peine. Pour le 68000 cette vitesse est impossible.

Pour dessiner une ligne, on doit relier deux points entre eux par un certain nombre de ces unités.

Etant donné que la résolution du graphisme d'ordinateur est limitée, on ne peut pas toujours choisir le point optimum. Les points réels se trouvent toujours un peu au-dessus ou au-dessous du point idéal. Une telle représentation de lignes ressemble à un escalier. Plus la résolution sera importante, plus les marches seront petites, mais on ne pourra jamais les éliminer totalement.

Exemple d'une ligne dans un graphisme d'ordinateur :

Les deux points	reliés entre eux forment une droite.
.....
.....1...111...
.....111.....
.....111.....
.....1.....111.....
.....

Le Blitter a la possibilité de dessiner de telles lignes, d'une longueur maximum de 1024 points. Mais on ne peut malheureusement pas donner tout simplement les coordonnées des deux extrémités. On doit définir la ligne Blitter correctement.

Celui-ci nécessite les octants à l'endroit où se trouve la ligne. Le partage des coordonnées en 8 parties, les octants, se retrouve dans plusieurs processeurs graphiques.

Schéma du choix des octants corrects

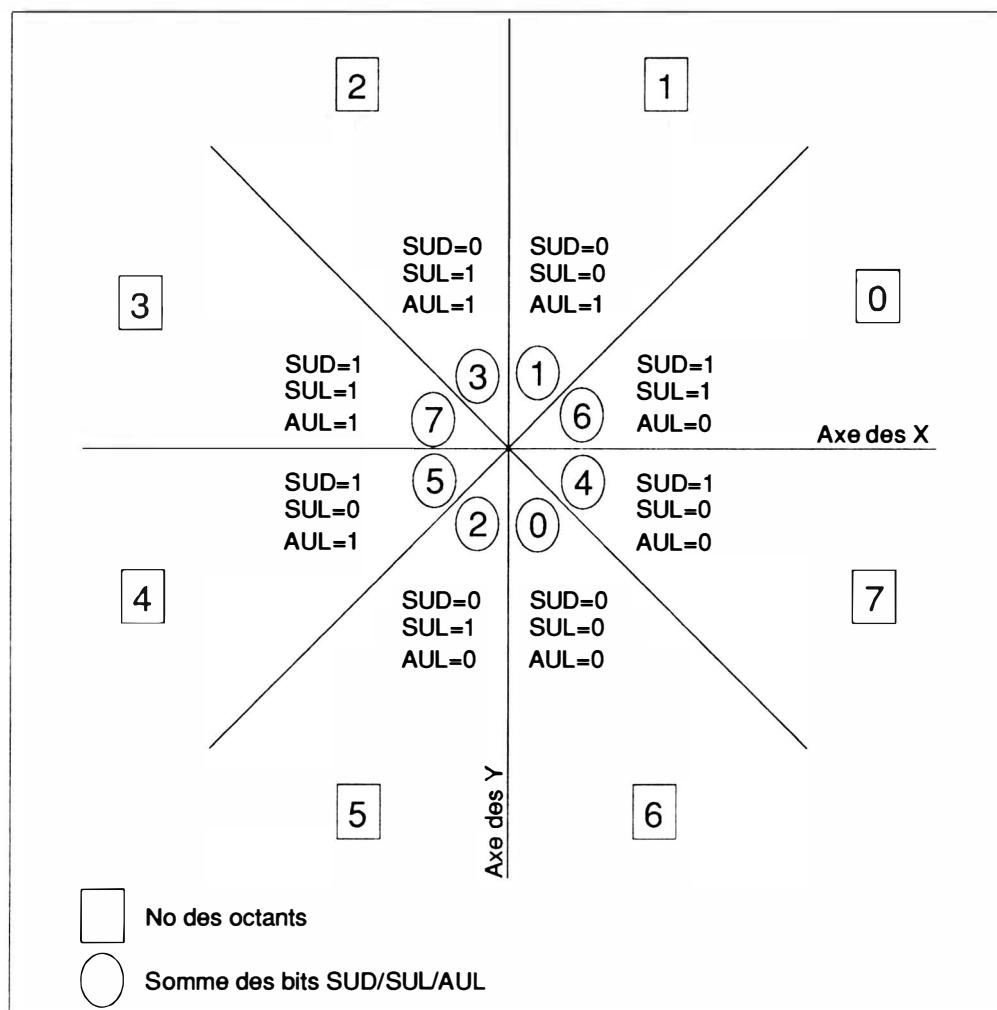


Figure 1 - 36

Schéma d'exemple d'une ligne

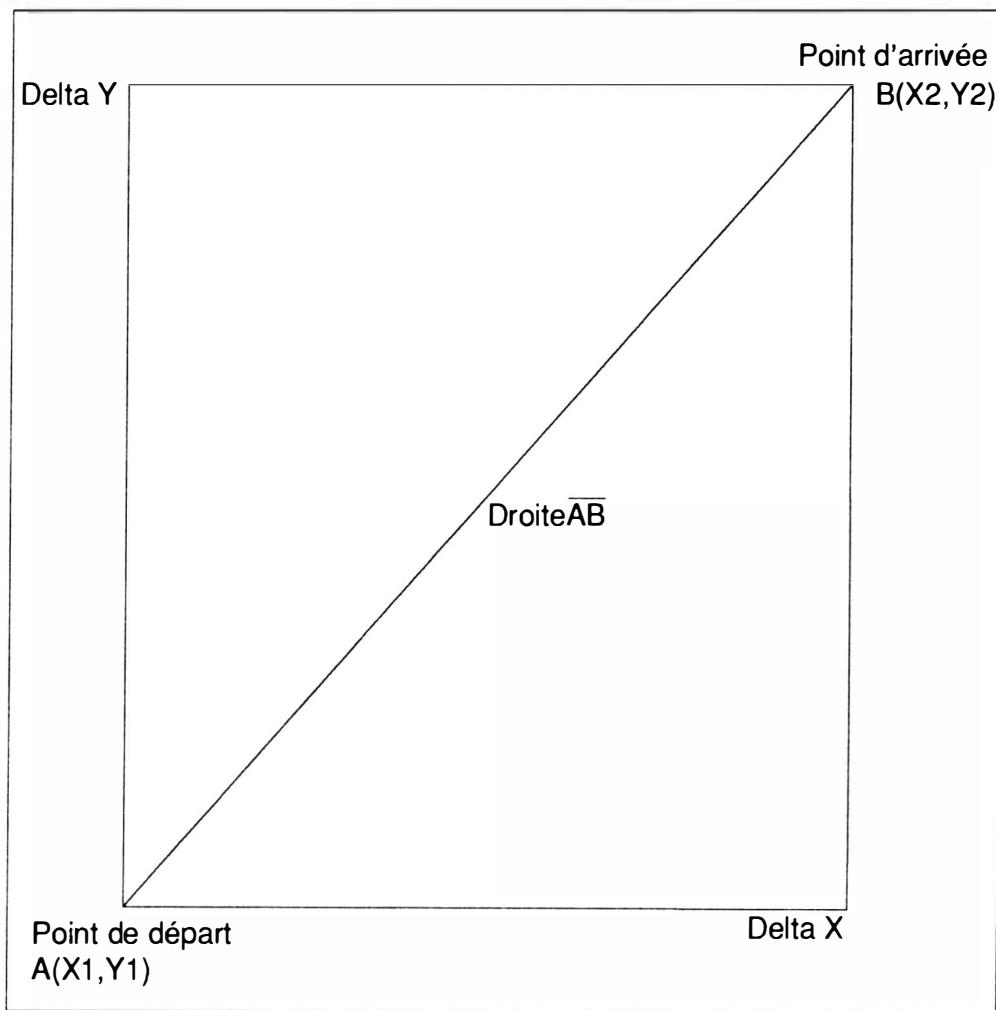


Figure 1 - 37

Le schéma du choix des octants nous montre cette répartition. Le point de départ d'une ligne se trouve à l'origine des axes de coordonnées (point de croisement des deux axes). Le point d'arrivée se trouve dans un des 8 octants. Au moyen de trois comparaisons logiques, on peut caractériser le numéro de ces octants, ainsi X_1 et Y_1 correspondent aux coordonnées du point de départ, X_2 et Y_2 étant les coordonnées du point d'arrivée :

Lorsque X_1 est inférieur à X_2 , le point d'arrivée peut se trouver dans un des octants suivants : 0, 1, 6 ou 7. Si X_1 est supérieur à X_2 , il se trouvera dans un des octants

suivants : 2, 3, 4 ou 5. Si X1 et X2 sont égaux, il se trouve sur l'axe des Y, tous les octants étant alors possibles.

De la même façon, si Y1 est inférieur à Y2, les octants possibles du point d'arrivée sont 0, 1, 2 ou 3. Lorsque Y1 est supérieur à Y2, les octants possibles seront 4, 5, 6 ou 7. Y1 = Y2 indique que tous les octants sont possibles.

A partir des deux dernières égalités, on peut caractériser les différences des X et Y : DeltaX = |X2-X1|, DeltaY = |Y2-Y1|. Lorsque DeltaX est supérieur à DeltaY, le point d'arrivée se trouve dans un des octants suivants : 0, 3, 4, ou 7. Si DeltaX est inférieur à DeltaY, il se trouvera dans un des octants suivants : 1, 2, 5 ou 6. DeltaX=DeltaY indique que tous les octants sont possibles.

De ces trois comparaisons, on pourra déterminer dans quel octant précisément, se trouve le point d'arrivée. Si ce point se trouve sur une ligne frontière entre deux, l'octant choisi n'a pas d'importance.

Détermination de l'octant :

Coordonnées de point	Octant	Code	Coordonnées de point	octant	code
Y1 <= Y2			Y1 >= Y2		
X1 <= X2	0	6	X1 >= X2	4	5
DeltaX>=DeltaY			DeltaX>=DeltaY		
Y1 <= Y2			Y1 >= Y2		
X1 <= X2	1	1	X1 >= X2	5	2
DeltaX<=DeltaY			DeltaX<=DeltaY		
Y1 <= Y2			Y1 >= Y2		
X1 >= X2	2	3	X1 <= X2	6	0
DeltaX<=DeltaY			DeltaX<=DeltaY		
Y1 <= Y2			Y1 >= Y2		
X1 >= X2	3	7	X1 <= X2	7	4
DeltaX>=DeltaY			DeltaX>=DeltaY		

Le chiffre dans la colonne 'Code' correspond au chiffre entouré d'un cercle sur le schéma du choix des octants. Le Blitter nécessite, pour chaque octant dans lequel se trouve le point d'arrivée de la ligne, une combinaison spéciale de trois bits.

Ces bits se nomment SUD (Sometimes Up or Down), SUL (Sometimes Up or Left) et AUL (Always Up or Left). 'Code' est en fait le nombre résultant de la combinaison de ces trois bits (SUD = MSB, AUL = LSB).

Pour programmer une ligne, on doit tout d'abord découvrir l'octant du point d'arrivée, puis écrire dans le Blitter la valeur Code correspondante.

Lignes avec modèles

Le Blitter utilise un masque pour dessiner une ligne, afin de déterminer si les points d'une ligne doivent être activés, effacés ou affichés suivant un modèle. Le masque a une largeur de 16 bits, celui-ci se répétant sur la ligne tous les 16 points. L'emploi d'un modèle et son influence sur l'affichage ne peuvent être expliqués au mieux que par quelques exemples :

('.-' = 0, '1' = 1, A = point de départ, B = point d'arrivée)

Sortie image : Masque = '1111111111111111'

.....11111111..... B11111111....118
.....111.....111.....111.....11111.....
.....11.....11.....11.....11.11.....
.....11.....11.....11.....111.....11.....
.....11.....11.....11.....111.....11.....
.....11.....11.....11111.....11.....
.....11.....11.....11111.....11.....
.....A.....11111111.....A11.....11111111.....

Les bits nuls du masque indiquent que les points correspondants de la ligne seront effacés :

Sortie image : Masque = '0000000000000000'

La combinaison de 0 et de 1 dans un masque donne un modèle de ligne :

Masque = '1111111000111000'

.A111111.....
.....111...1.....
.....111111.....
.....111...11.....
.....11111.....
.....111...111.....
.....1111...B

Dessin d'une ligne frontière

Dans le chapitre concernant le remplissage d'une surface par le Blitter, il est expliqué que les lignes frontières d'une surface doivent toujours avoir la largeur d'un pixel. Si on

dessine une telle ligne avec le Blitter, il peut arriver que plusieurs points se trouvent sur la même ligne horizontale. Afin d'éviter cela, le Blitter se laisse manipuler d'une telle façon qu'il dessine la ligne avec seulement un point par position horizontale :

Ligne normale	Ligne avec un point/position hor.
.....11111...
.....1111....1.....
.....1111.....1.....
.....1111.....1.....
1111.....	1.....

Définition de la pente

Afin que le Blitter sache où il doit tracer sa droite, il est nécessaire de définir la pente d'une droite. La façon la plus simple est de se baser sur trois termes issus des valeurs delta X et delta Y, valeurs qui ont été détaillées lors de l'explication des octants (delta Y et delta X reproduisent la largeur et la hauteur d'un rectangle dont la diagonale correspond à la droite. Cf schéma 37). On devra tout d'abord comparer les deux valeurs afin de savoir quelle est la plus petite et quelle est la plus grande. La plus petite sera appelée Pdelta, la plus grande Gdelta. Puis le Blitter exécutera les opérations suivantes :

- ① $2 * P\text{delta}$
- ② $2 * P\text{delta} - G\text{delta}$
- ③ $2 * P\text{delta} - 2 * G\text{delta}$

Le Blitter possède de plus un drapeau SIGN qui sera initialisé à 1 lorsque $2 * P\text{delta}$ sera inférieur à $G\text{delta}$.

Fonction des registres dans le mode Droite

Le Blitter utilise, lors du dessin d'une droite, des registres identiques à ceux utilisés lors de la copie des données, les fonctions étant toutefois différentes :

BLTAPTL

dans ce registre sera écrite la valeur de l'opération $2 * P\text{delta} - G\text{delta}$.

DLTCPCT et BLTDPT

ces deux registres (BLTCPCTH et BLTCPCTL, BLTDPTH et BLTDPTL) doivent être initialisés avec l'adresse de départ de la droite. Ceci correspond à l'adresse du mot à laquelle se trouve le point de départ de la droite.

BLTAMOD

dans ce registre on trouve le résultat de l'opération $2 * P\text{delta} - 2 * G\text{delta}$.

BLTBMOD

$2 * P_{\text{delta}}$.

BLTCMOD et BLTDMOD

dans ces deux registres modulo, on trouve la largeur de l'image dans laquelle la droite devra être dessinée. Cette dernière est toujours sous la forme d'un nombre pair d'octets. Pour un Bitplane normal, comportant 320 points (40 octets), sur l'axe des X, la valeur pour BLTCMOD sera égale à 40.

BLTSIZE

la largeur (Bit 0 à 5) devra être initialisée à 2. La hauteur (Bit 6 à 15) renferme la longueur de la droite en points. Une hauteur de 0 correspond à une droite d'une longueur de 1024 points.

La longueur correcte de la droite est toujours identique à la valeur de Gdelta.

Le dessin de la droite sera toujours démarré après l'écriture dans le registre BLTSIZE. Ce sera donc toujours le dernier registre à initialiser.

BLTADAT

ce registre doit être initialisé avec la valeur \$8000.

BLTBBDAT

ce registre correspond au masque avec lequel la droite devra être dessinée.

BLTAFWM

ce registre masque est initialisé avec la valeur \$FFFF.

BLTCON 0

N° de bit	Nom	Fonction
15	START3	ces 4 bits STARTx correspondent à la position du point
14	START2	de départ de la droite.
13	START1	(BLTCP/BLTDPT).
12	START0	En règle générale
11	USEA = 1	cette combinaison des bits USEx est
10	USEB = 0	nécessaire au mode droite.
9	USEC = 1	
8	USED = 1	

N° de bit	Nom	Fonction
7	LF7	ces bits LFx doivent être initialisés avec x
jusqu'à 0	LF0	\$CA(D= aC+ AB).

BLTCON1

N° de bit	Nom	Fonction
15	Texture3	Ceci correspond à la valeur nécessaire
14	Texture2	au décalage du masque. En temps
13	Texture1	normal Texture0-3 doit être égal à START
12	Texture0	0-3. Le modèle présent dans le registre
11 à 7 = 0	inutilisé	toujours 0.
6	SIGN	lorsque $2 * P\Delta x$ est inférieur à $G\Delta x$
5	inutilisé	toujours 0.
4	SUL	Ces trois bits doivent être initialisés avec les codes SUL/SUD/AUL
3	SUD	des octants correspondants.
2	AUL	
1	SIGN = 1	trace la droite avec un point par ligne
0	LINE = 1	commute le Blitter en mode tracé de droite

En conclusion voici un exemple de calcul :

Il s'agit de tracer une droite dans un Bitplane. Ce Bitplane possède une taille de 320 points par 200 et se trouve à l'adresse \$40000. Le point de départ de la droite possède les coordonnées X=20 et Y=185. Le dernier point se trouve aux coordonnées X=210 et Y=35 (les coordonnées se trouvent dans le coin inférieur gauche du BitPlane).

$\Delta x = 190$, $\Delta y = 150$

1ère opération : recherche de l'octant du dernier point

Il s'agit ici d'exécuter trois comparaisons : $X1 < X2$, $Y1 > Y2$ et $\Delta x > \Delta y$. Le résultat sera ici l'octant numéro 7 et une valeur pour les codes SUD/SUL/AUL = 4.

2ème opération : adresse du point de départ

Ceci se calcule de la façon suivante :

- ✓ l'adresse de départ du Bitplane + (le nombre de lignes - Y1 - 1) * nombre d'octets par ligne + 2*(X1/16).

La partie décimale de la division sera éliminée.

En remplaçant les variables par les chiffres :

$$\$40000 + (200-185-1)*40 + 2 = \$40232$$

Cette valeur sera mise dans les registres BLTCPT et BLTDPT. Le nombre d'octets par ligne sera mis dans les registres BLTCMOD et BLTDMOD.

3ème opération : point de départ de la droite dans START0-3

Opération nécessaire : X1 AND \$F

En numérique : START0-3 = 20 AND \$F = 4

4ème opération : valeur des registres BNTAPTL, BLTAMOD et BLTBMOD

$\Delta Y < \Delta X$, Pdelta = DeltaY et Gdelta = DeltaX
 BLTAPTL = $2 * Pdelta - Gdelta$, ce qui est égal à $2 * 150 - 190 = 110$
 BLTAMOD = $2 * Pdelta - 2 * Gdelta$ ce qui est égal $2 * 150 - 2 * 190 = -80$
 BLTBMOD = $2 * Pdelta = 300$
 $2 * Pdelta > Gdelta$ implique SIGN = 0

5ème opération : longueur de la droite dans BLTSIZE

Longueur = Gdelta = DeltaX = 190

La valeur du registre BLTSIZE se trouve sous la forme suivante :

longueur*64 + largeur.

La largeur lors du tracé de la droite doit toujours être initialisée à 2. BLTSIZE est égal à $\Delta X * 64 + 2 = 12162$, soit \$2F82.

6ème opération : Compilation des données pour les deux registres BLTCONx

Il s'agit ici d'initialiser la valeur START dans le registre BLTCON0 ainsi que \$CA pour les bits LFx et la valeur 1 0 1 1 dans USEx. Dans notre exemple, l'ensemble donnera \$ABCA.

BLTCON1 contient le code de l'octant et les bits de contrôle. Notre droite doit être dessinée en mode normal ce qui implique que SIGN = 0.

LINE doit être initialisé à 1. SIGN sera calculé et dans notre exemple, correspondra à 0. L'ensemble donnera \$0011. En langage assembleur, l'initialisation des registres sera fait de la façon suivante :

```
LEA $DFF000,A5 ;Adresse de base Blitter dans A5
MOVE.L #$40232, BLTCPH(A5) ;Adresse de départ dans BLTCPH
MOVE.L #$40232, BLTDPTH(A5) ;et dans BLTDPT
MOVE.W #40, BLTCMOD(A5) ;Largeur du Bitplane dans BLTCMOD
MOVE.W #40, BLTDMOD(A5) ;et dans BLTDMOD
MOVE.W #110, BLTAPTL(A5)
MOVE.W #-80, BLTAMOD(A5)
MOVE.W #300, BLTBMOD(A5)
MOVE.W #$SABCA, BLTCONO(A5)
MOVE.W #$11, BLTCON1(A5)
MOVE.W #12162, BLTSIZE(A5) ;Début de tracé de la droite par le Blitter
```

Autre mode de tracé

Jusqu'à présent la valeur des bits LF_x a toujours été \$CA. Ceci permet au point de la droite de suivre exactement le modèle du masque.

Mais il existe aussi d'autres combinaisons LF_x.

Afin de bien le comprendre, on doit, en premier lieu, savoir comment les Bits LF_x sont interprétés en mode tracé de droite :

Le Blitter ne peut adresser la mémoire que par des mots. En mode tracé de droite, les données d'entrée arrivent sur le canal C du Blitter. A cela se rajoute le registre B correspondant au masque. Le registre A détermine alors les points à tracer parmi les mots lus correspondants aux points de la droite. Celui-ci contient toujours un Bit initialisé, qui était décalé par le Blitter à la position souhaitée.

Initialisation de LF_x avec la valeur \$CA que tous les bits, dont les bits A correspondants sont égaux à 0, seront directement pris en charge par la source C. Si par contre A = 1, le bit masque correspondant sera utilisé comme bit cible.

Lorsqu'on sait comment utiliser les bits LF_x, on peut déterminer d'autres modes de dessin. Par exemple \$4A provoque l'inversion de tous les points de la droite.

Les cycles Blitter DMA

Comme cela a déjà été expliqué dans le chapitre précédent, le Blitter n'occupe que les cycles de bus pairs. Comme il y a par ailleurs aussi la priorité sur le 68000, il est intéressant de savoir combien de cycles peuvent être occupés par le processeur.

Ceci dépend surtout du nombre de canaux DMA du Blitter (A, B, C et D) qui sont actifs.

Le tableau suivant montre le déroulement d'une opération Blitter dans les 15 combinaisons possibles des canaux DMA du Blitter.

Les caractères A, B, C et D correspondent aux canaux DMA. Le chiffre 1 correspond au 1er mot de l'opération Blitter, le chiffre 2 correspond au 2ème et le chiffre 3 au dernier mot de données. Les deux traits d'union signifient que le cycle de bus n'est pas occupé par le Blitter.

Occupation des cycles de bus pairs par le Blitter :

Canaux OMA actifs	Occupation des cycles de bus pairs
aucun	-- -
D	D0 -- D1 -- D2 -- - - - - - - - - - - - - -
C	C0 -- C1 -- C2 -- - - - - - - - - - - - - -
C D	C0 -- - - C1 D0 -- C2 D1 -- D2 -- - - - -
B	B0 -- - - B1 -- - - B2 -- - - - - - - - - -
B D	B0 -- - - B1 D0 -- B2 D1 -- D2 -- - - - -
B C	B0 C0 -- B1 C1 -- B2 C2 -- - - - - - - - -
B C D	B0 C0 -- - - B1 C1 D0 -- B2 C2 D1 -- D2
A	A0 -- A1 -- A2 -- - - - - - - - - - - - - -
A D	A0 -- A1 D0 A2 D1 -- D2 -- - - - - - - - -
A C	A0 C0 A1 C1 A2 C2 -- - - - - - - - - - -
A C D	A0 C0 -- A1 C1 D0 A2 C2 D1 -- D2 -- - -
A B	A0 B0 -- A1 B1 -- A2 B2 -- - - - - - - - -
A B C	A0 B0 -- A1 B1 D0 A2 B2 D1 -- D2 -- - -
A B C	A0 B0 C0 A1 B1 C1 A2 B2 C3 -- - - - - - -
A B C D	A0 B0 C0 -- A1 B1 C1 D0 A2 B2 B3 D1 D2

Remarque :

Le tableau précédent est valide lorsque les considérations suivantes sont prises en compte :

- ① Le Blitter ne doit pas être dérangé par les accès DMA Copper ou Bitplane.
- ② Le Blitter fonctionne en mode normal (soit tracé de droite, soit remplissage de surface).
- ③ Le bit BLITPRI du registre DMACON est initialisé et le Blitter a la priorité absolue sur le 68000.

Explications :

Comme on peut le voir, il arrive que les données de sortie D0 arrivent dans la RAM après que les données A1, B1 et C1 aient été lues. Ceci provient du mode de traitement du Blitter en "Pipelining". Cela signifie que le traitement des données à l'intérieur du Blitter fonctionne en plusieurs étapes indépendantes les unes des autres. Chaque étape est liée à la sortie de la précédente et à l'entrée de la suivante. La première étape correspond à l'entrée des données (A0, B0, C0) où ces dernières seront traitées et transférées à la deuxième étape. Pendant que ces données sont traitées, dans la

deuxième étape, les données suivantes arrivent à la première étape (A1, B1, C1). Quand les premières données arrivent à la dernière étape, l'étape de sortie (D0), le Blitter est déjà en train de traiter les données suivantes. On trouve toujours lors d'une opération Blitter, et à tout moment, deux paires de données à une étape de traitement différente.

Cette étape permet aussi le calcul de la durée de déroulement d'une opération Blitter.

A chaque microseconde, le Blitter a, à sa disposition, deux cycles de bus. S'il doit par exemple copier une zone de 64 kilo Octets (32768 mots) de A vers D, le Blitter nécessite deux fois 32768 cycles.

Lorsque cette même zone doit être de plus combinée avec la source C, le Blitter nécessitera, $3 * 32768$ cycles, étant donné que pour chaque mot, il doit être lu un mot de donnée de la source C. Le tableau montre aussi que le Blitter n'a pas la possibilité d'utiliser tous les cycles de bus lorsque seul un canal DMA est actif. Cette étape permet aussi le calcul de la durée de déroulement d'une opération Blitter.

Exemples de programme

Programme 1 : tracé de droite avec le Blitter

Ce programme met en place une routine de tracé de droite avec le Blitter. Il montre comment calculer les valeurs nécessaires. Le programme est simple : en avant programme, on réservera la mémoire nécessaire et on structurera la COPPER-LIST. Seule la routine OwnBlitter est inconnue. Comme son nom l'indique, on peut, grâce à elle, se réserver le Blitter.

De la même façon, on retrouvera à la fin du programme la routine de désactivation, c'est-à-dire la routine DisownBlitter avec laquelle le Blitter retombe à nouveau sous le contrôle du système d'exploitation.

Le programme utilise une seul BitPlane Hires avec la résolution standard de 640 * 256 points. Dans la boucle principale, le programme trace des droites qui partent du bord de l'écran, passent par le point central, pour finir sur le côté opposé.

Lorsque l'écran est rempli, le programme décale le masque avec lequel la droite est tracée et recommence.

Remarque :

Les coordonnées données dans le programme partent du point 0,0 qui se trouve dans le coin supérieur gauche de l'écran, et ne correspondent pas aux coordonnées mathématiques.

;*** Tracé de droite par le Blitter

;Registres Custom chip

INTENA = \$9A ;Registre InterruptEnable (écriture)
DMACON = \$96 ;Registre de contrôle DMA (écriture)

```

DMACONR -,$2      ;Registre de contrôle DMA (lecture)
COLOR00 - $180    ;Registre palette de couleur 0
VHPOSR - $6       ;Position faisceau (lecture)

;Registre Copper

COP1LC - $80      ;Adresse de la 1ère COPPER-LIST
COP2LC - $84      ;Adresse de la 2ème COPPER-LIST
COPJMP1 - $88      ;Saut à la 1ère COPPER-LIST
COPJMP2 - $8a      ;Saut à la 2ème COPPER-LIST

;Registre Bitplane

BPLCON0 - $100    ;Registre de contrôle Bitplane 0
BPLCON1 - $102    ;1 (Valeur de scrolling)
BPLCON2 - $104    ;2 (Sprite<>Priorité playfield)
BPL1PTH - $0E0     ;Pointeur sur le Bitplane 1
BPL1PTL - $0E2     ;
BPL1MOD - $108    ;Valeur Modulo pour BitPlane impair
BPL2MOD - $10A    ;Valeur Modulo pour BitPlane pair
DIWSTRT - $08E    ;Départ fenêtre écran
DIWSTOP - $090    ;Fin fenêtre écran
DDFSTRT - $092    ;BitPlane DMA Start
DDFSTOP - $094    ;BitPlane DMA Stop

;Registres Blitter

BLTCONO - $40     ;Registre contrôle Blitter 0
BLTCON1 - $42     ;Registre contrôle Blitter 1
BLTCPPTH - $48     ;Pointeur sur source C
BLTCPCTL - $4a     ;
BLTBPTH - $4c     ;Pointeur sur source B
BLTBPTL - $4e     ;
BLTAPTH - $50     ;Pointeur sur source A
BLTAPTL - $52     ;
BLTDPTH - $54     ;Pointeur sur données cible D
BLTDPTL - $56     ;
BLTCMOD - $60     ;Valeur Modulo pour source C
BLTBMOD - $62     ;Valeur Modulo pour source B
BLTAMOD - $64     ;Valeur Modulo pour source A
BLTDMOD - $66     ;Valeur Modulo pour cible D
BLTSIZE - $58     ;Hauteur et largeur de la fenêtre blitter
BLTCDAT - $70     ;Registre de données source C
BLTBDAT - $72     ;Registre de données source B
BLTADAT - $74     ;Registre de données source A
BLTAFWM - $44     ;Masque 1er mot de données source A
BLTALWM - $46     ;Masque 1er mot de données source B

;CIA-A Registre Port A (bouton souris)

CIAAPRA - $bfe001

;Exec Library Base Offsets

OpenLibrary - -30-522 ;LibName,Version/a1,d0
Forbid      - -30-102
Permit      - -30-108
AllocMem    - -30-168 ;ByteSize,Requirements/d0,d1
FreeMem     - -30-180 ;MemoryBlock,ByteSize/a1,d0

```

```
;Graphics Library Base Offsets

OwnBlitter    - -30-426
DisownBlitter - -30-432

;graphics base

StartList - 38

;Autres labels

Execbase - 4
Planesize - 80*256 ;Taille Bitplane: 80 octets par 256 lignes
Planewidth - 80
CLsize - 3*4 ;La COPPER-LIST comporte 3 instructions
Chip - 2 ;Solliciter la Chip-RAM
Clear - Chip+$10000 ;Effacer Chip-RAM précédente

;*** Avant programme ***

Start:

;Solliciter la mémoire pour les bitplanes

move.l Execbase,a6
move.l #Planesize,d0 ;mémoire pour le plan
move.l #clear,d1
jsr AllocMem(a6)      ;Sollicitation de la mémoire
move.l d0,Planeadr
beq fin ;Erreur! -> Fin

;Solliciter la mémoire pour les bitplanes

moveq #CLsize,d0
moveq #chip,d1
jsr AllocMem(a6)
move.l d0,CLadr
beq FreePlane ;Erreur! -> FreePlane

;Mise en place de la COPPER-LIST

move.l d0,a0          ;Adresse de la COPPER-LIST dans a0
move.l Planeadr,d0    ;Adresse du Bitplane
move.w #bp1lpth,(a0)+ ;Première instruction Copper dans la RAM
swap d0
move.w d0,(a0)+        ;Mot supérieur de l'adresse Bitplane dans la RAM
move.w #bp1lpt1,(a0)+  ;Deuxième instruction dans la RAM
swap d0
move.w d0,(a0)+        ;Mot inférieur de l'adresse Bitplane dans la RAM
move.l #$ffffffe,(a0) ;Fin de la COPPER-LIST

;Solliciter le Blitter

move.l #GRname,a1
clr.l d0
jsr OpenLibrary(a6)
move.l a6,-(sp)       ;ExecBase sur la pile
move.l d0,a6          ;GraphicsBase dans a6
```

```
move.l a6,-(sp) ;GraphicsBase sur la pile
jsr OwnBlitter(a6) ;Prise en charge Blitter

;*** Programme général ***
;Bloquer les commutations de tâche et accès DMA

move.l 4(sp),a6 :ExecBase dans a6
jsr forbid(a6) ;Désinitialisation du Task-Switching
lea $dff000,a5
move.w #$03e0,dmacon(a5)

;Initialisation du Copper

move.l CLadr,cop1lc(a5)
clr.w copjmp1(a5)

;Choix des couleurs

move.w #$0000,color00(a5) :Arrière plan noir
move.w #$0fa0,color00+2(a5) ;Droites jaunes

;Initialisation du Playfield

move.w #$3081,diwstrt(a5) ;30,129
move.w #$30c1,diwstop(a5) ;30,449
move.w #$003c,ddfstrt(a5) ;Ecran Hires normal
move.w #$00d4,ddfstop(a5)
move.w #%1001001000000000,bplcon0(a5)
clr.w bplcon1(a5)
clr.w bplcon2(a5)
clr.w bpl1mod(a5)
clr.w bpl2mod(a5)

;DMA activé

move.w #$83C0,dmacon(a5)

;Tracé de lignes

;Détermination des valeurs de départ

move.l Planeadr.a0 ;Paramètre Constant pour DrawLine
move.w #Planewidth.a1 ;dans registre correspondant
move.w #255,a3 ;Taille du Bitplane dans registre
move.w #639,a4
move.w #$0303,d7 ;Modèle de départ

Loop: rol.w #2,d7 ;Décalage du modèle
move.w d7,a2 ;Modèle dans registre pour DrawLine

clr.w d6 ;Effacer la variable de la boucle
BoucleX:
clr.w d1 ;Y1 = 0

move.w a3,d3 ;Y2 = 255
move.w d6,d0 ;X1 = variable boucle
move.w a4,d2
```

```
sub.w d6,d2          ;X2 = 639-Variable boucle
bsr DrawLine         ;Tracé de droite
addq.w #4,d6         ;Incrémenter variable de boucle
cmp.w a4,d6          ;Test supérieur ou égal à 639
ble.s bouclerx       ;Sinon, boucle continue

clr.w d6              ;Effacer la variable boucle
BoucleY:
move.w a4,d0          ;X1 = 639
clr.w d2              ;X2 = 0
move.w d6,d1          ;Y1 = Variable boucle
move.w a3,d3          ;Y2 = 255-Variable boucle
sub.w d6,d3

bsr DrawLine         ;Tracé de droite
addq.w #2,d6          ;Incrémenter la variable boucle
cmp.w a3,d6          ;Test > 255
ble.s BoucleY         ;Sinon, la boucle continue
btst #6,ciaapra      ;Bouton souris ?
bne Loop              ;Non -> continue

;*** Partie finale ***
;Attente jusqu'à ce que le blitter ait terminé
Wait: btst #14,dmaconr(a5)
bne Wait

;COPPER-LIST précédente à nouveau activée

move.l (sp)+,a6        ;GraphicsBase pris sur la pile
move.l StartList(a6),cop1lc(a5)
clr.w copjmpl(a5)      ;COPPER-LIST Startup activé
move.w #$8020,dmacon(a5)
jsr DisownBlitter(a6)  ;Blitter libéré
move.l (sp)+,a6        ;ExecBase pris sur la pile
jsr Permit(a6)         ;Task Switching autorisé

;Mémoire libérée pour la COPPER-LIST

move.l CLadr,a1        ;Paramètre pour FreeMem
moveq #CLsize,d0
jsr FreeMem(a6)         ;Mémoire libérée

;Mémoire libérée pour bitplane

FreePlane:
move.l Planeadr,a1
move.l #Planesize,d0
jsr FreeMem(a6)

Fin:
clr.l d0
rts    ;Fin du programme

;Variables
```

```

CLadr:    dc.l 0    ;Adresse de la COPPER-LIST
Planeadr: dc.l 0    ;Adresse du Bitplane

;Constantes

GRname: dc.b "graphics.library",0
even

;*** DrawLine Routine ***
;DrawLine dessine une droite avec le Blitter.
;Les paramètres suivants seront nécessaires :
;d0 = X1  Coordonnées X du point de départ
;d1 = Y1  Coordonnées Y du point de départ
;d2 = X2  Coordonnées X du dernier point
;d3 = Y2  Coordonnées Y du dernier point
;a0 pointe sur le 1er mot du Bitplane
;a1 contient la largeur du Bitplane (octets)
;a2 est écrit dans le registre masque
;d4 à d6 seront utilisés en tant que registres de traitement

DrawLine:

;Calcul de l'adresse de départ de la droite

move.l a1,d4      ;Largeur du registre de travail
mulu d1,d4        ;Y1 * Nombre d'octets par ligne
moveq #-10,d5      ;Tracer : $F0
and.w d0,d5        ;4 bits inférieurs de X1 masqués
lsr.w #3,d5        ;Division par 8 du reste
add.w d5,d4        ;Y1 * Nombre d'octets par ligne + X1/8
add.l a0,d4        ;+ adresse de départ du Bitplane
;d4 contient l'adresse de départ
;de la droite
;Calcul de l'octant et des Deltas

clr.l d5          ;Effacer le registre de travail
sub.w d1,d3        ;Y2-Y1  DeltaY dans D3
rox1.b #1,d5       ;Signe de DeltaY dans d5
tst.w d3           ;Flag N restauré
bge.s y2gy1        ;Si DeltaY positif, branchement à Y2gy1
neg.w d3           ;Inversion DeltaY
y2gy1:
sub.w d0,d2        ;X2-X1  DeltaX dans D2
rox1.b #1,d5       ;Signe de DeltaX dans d5
tst.w d2           ;Flag N restauré
bge.s x2gx1        ;Si DeltaX positif, branchement à X2gx1
neg.w d2           ;Inversion DeltaX
x2gx1:
move.w d3,d1        ;DeltaY dans d1
sub.w d2,d1        ;DeltaY-DeltaX
bge.s dygdx        ;Si DeltaY > DeltaX, branchement à dygdx
exg d2,d3          ;Delta < dans d2
dygdx: rox1.b #1,d5 ;d5 contient le résultat des 3 opérations
move.b Table_Octants(pc,d5),d5 ;prise en compte de l'octant correspondant
add.w d2,d2          ;petit Delta * 2

;Test si le Blitter a déjà terminé la dernière opération

```

```

WBInit: btst #14,dmaconr(a5) ;Test du bit BBUSY
bne.s WBInit           ;Attente jusqu'à ce qu'il soit à 0

move.w d2,bltbmod(a5)  ;2* PDelta dans BLTBMOD
sub.w d3,d2            ;2* PDelta - GDelta
bge.s signnl          ;Si 2*PDelta > GDelta : saut à signal
or.b #$40,d5           ;Initialisation de SignFlag
signnl: move.w d2,bltaptl(a5) ;2*PDelta-GDelta dans BLTAPTL
sub.w d3,d2            ;2* PDelta - 2* GDelta
move.w d2,bltamod(a5)  ;dans BLTAMOD

;Initialisation des registres

move.w #$8000,bltadat(a5)
move.w a2,bltbdat(a5)  ;Masque issu de a2 dans BLTBDAT
move.w #$ffff,bltafwm(a5)
and.w #$000f,d0         ;4 Bits inférieurs de X1 décalés
ror.w #4,d0             ;vers STARTO-3
or.w #$0bca,d0          ;Initialisation USEx et LFx
move.w d0,bltcon0(a5)
move.w d5,bltcon1(a5)   ;Octant dans le Blitter
move.l d4,bltcpth(a5)   ;Adresse de départ de la droite
move.l d4,bltdpth(a5)   ;dans BLTCPT et BLTDPT
move.w a1,bltcmod(a5)   ;Largeur du Bitplane dans
move.w a1,bltdmod(a5)   ;les 2 registres Modulo

;Initialisation de BLTSIZE et démarrage du Blitter

lsl.w #6,d3    ;Longueur * 64
addq.w #2,d3   ;plus (Width - 2)
move.w d3,bltsize(a5)

rts

;Table d'octants avec LINE = 1:
;La table d'octants contient pour chaque octant
;la valeur code correspondante qui est déjà décalée
;à la bonne position. De plus, le bit LINE est initialisé.
```

Table_Octants :

```

dc.b 0 *4+1 ;y1<y2, x1<x2, dx<dy - Oct6
dc.b 4 *4+1 ;y1<y2, x1<x2, dx>dy - Oct7
dc.b 2 *4+1 ;y1<y2, x1>x2, dx<dy - Oct5
dc.b 5 *4+1 ;y1<y2, x1>x2, dx>dy - Oct4
dc.b 1 *4+1 ;y1>y2, x1<x2, dx<dy - Oct1
dc.b 6 *4+1 ;y1>y2, x1<x2, dx>dy - Oct0
dc.b 3 *4+1 ;y1>y2, x1>x2, dx<dy - Oct2
dc.b 7 *4+1 ;y1>y2, x1>x2, dx>dy - Oct3
```

Programme 2 : Remplissage d'une surface avec le Blitter

Ce programme ressemble beaucoup au premier programme ; il montre comment le Blitter génère un polygone et comment il remplit cette surface.

La partie la plus importante est identique à celle du premier programme, seule la boucle de tracé du polygone ainsi que le remplissage ont été préservés. La partie à rajouter

débute au commentaire "Tracé de lignes" jusqu'au commentaire "*** Partie finale ***". Cette zone devra se mettre à la place et à la ligne où apparaît "Partie 1".

De plus, la table d'octants se trouvant à la fin du programme devra être mise à partir de la ligne où se trouve le titre "Partie 2".

La nouvelle table d'octants est nécessaire étant donné que le Blitter nécessite pour le remplissage d'une surface, une limite possédant un point par ligne. Dans la nouvelle table d'octants, les bits LIMMSIGN sont initialisés en plus. Le programme indiqué par "Partie 1" trace deux droites et remplit la zone intermédiaire par le Blitter. Il attend ensuite une pression sur un bouton de la souris.

```
;*** Remplissage d'une surface avec le Blitter ***
```

Partie 1:

```
;Dessin d'un triangle vide
```

```
;Valeur de départ
```

```
move.l Planeadr,a0      ;Paramètre Constant pour initialiser
move.w #Planewidth,a1    ;la routine LineDraw
move.w #$ffff,a2          ;Masque = $FFFF -> pas de modèle
```

```
;* Droites limites dessinées *
```

```
;Droite de 320,10 à 600,230
```

```
move.w #320,d0
move.w #10,d1
move.w #600,d2
move.w #230,d3
bsr.L drawline ;Dessin de la droite
```

```
;Droite de 319,10 à 40,230
```

```
move.w #319,d0
move.w #10,d1
move.w #40,d2
move.w #230,d3
bsr.L drawline ;Dessin de la droite
```

```
;* Remplissage d'une surface *
```

```
;Attente jusqu'à ce que le blitter ait terminé le dessin de la dernière
droite
```

```
Wline: btst #14,dmaconr(a5) ;Test BBUSY
bne.S Wline
```

```
add.l #Planesize-2,a0      ;Adresse des derniers mots
move.w #$09f0,bltcon0(a5)  ;USEA et D, LFx: D - A
move.w #$000a,bltcon1(a5)  ;Initialisation de Fill
move.w #$ffff,bltafwm(a5)  ;inclusive et lastwordmask
move.w #$ffff,bltalwm(a5)
move.l a0,bltapth(a5)       ;Adresse du dernier mot du bitplane
move.l a0,bltdpth(a5)       ;dans le registre pointeur d'adresses
move.w #0,bltamod(a5)       ;Pas de Modulo
move.w #0,bltdmod(a5)
```

```

move.w #$ff*64+40,bltsize(a5) ;Démarrage du Blitter

;Attente événement souris

fin: btst #6,ciaapra ;Test bouton souris ?
bne.S fin ;Non -> continue

Fin de la partie 1.

Partie 2:

;Table_Octants avec SING = 1 et LINE = 1:
Table_Octants :

dc.b 0 *4+3 ;y1<y2, x1<x2, dx<dy = Oct6
dc.b 4 *4+3 ;y1<y2, x1<x2, dx>dy = Oct7
dc.b 2 *4+3 ;y1<y2, x1>x2, dx<dy = Oct5
dc.b 5 *4+3 ;y1<y2, x1>x2, dx>dy = Oct4
dc.b 1 *4+3 ;y1>y2, x1<x2, dx<dy = Oct1
dc.b 6 *4+3 ;y1>y2, x1<x2, dx>dy = Oct0
dc.b 3 *4+3 ;y1>y2, x1>x2, dx<dy = Oct2
dc.b 7 *4+3 ;y1>y2, x1>x2, dx>dy = Oct3

```

1.5.8. La sortie du son

Eléments de base de la musique électronique

Lorsque nous entendons quelque chose, que ce soit de la musique, un bruit ou des paroles, il s'agit en fait d'une vibration de l'air, l'onde sonore, qui atteint notre oreille. Un instrument de musique normal engendre cette vibration, soit directement, comme par exemple une flûte où on y insuffle de l'air, soit indirectement, une partie de l'instrument générant le son, ce dernier étant alors émis dans l'air. C'est ce qui se passe, par exemple, pour tous les instruments à corde.

Un instrument électronique engendre des vibrations électriques dans ses circuits de commutation, qui dans leurs allures, correspondent aux sons souhaités. Ces vibrations deviennent audibles, lorsqu'on les transforme en vibrations sonores, au moyen d'un haut parleur. L'Amiga utilise celui qui est présent dans le moniteur. Malheureusement, du fait de sa taille et de sa qualité, il n'est pas en mesure de convertir les vibrations électriques en ondes sonores identiques. C'est pour cette raison qu'il est préférable de relier l'Amiga à un amplificateur, qui offrira, en même temps qu'un grand confort d'écoute, la stéréophonie.

Voici les paramètres qui établissent le son de l'ordinateur :

Fréquence

La fréquence d'un son détermine si celui-ci est haut ou bas. Physiquement, la fréquence correspond au nombre de vibrations par seconde, l'unité étant le HERTZ. Une vibration par seconde = un HERTZ, un KiloHertz correspondant à 1000 Hertz. L'oreille humaine

perçoit des sons entre 16 et 16000 Hertz. Celui qui possède quelques notions de musique sait que le LA de référence possède une fréquence de 440 Hertz. Le lien entre la fréquence et la hauteur d'un son est le suivant : à chaque octave, la fréquence est doublée. Le LA seconde a donc une fréquence de 880 Hertz, alors que le LA se trouvant une octave en dessous, a une fréquence de 220 Hertz.

La fréquence d'un son ne doit pas être forcément constante. Elle peut, par exemple, osciller périodiquement entre plusieurs Hertz, pour un même son. Cet effet se nomme Vibrato.

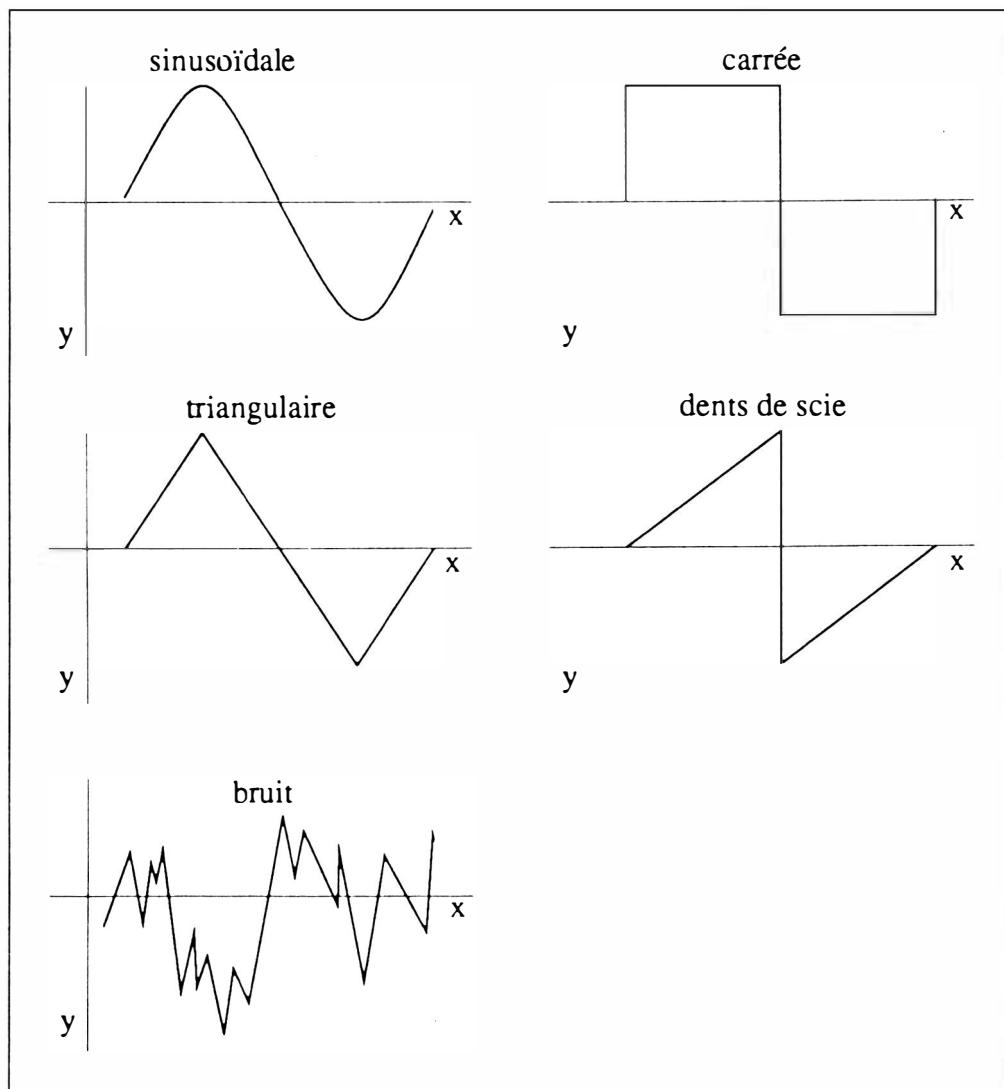
Intensité

Le deuxième paramètre d'un son est son intensité. Par ce terme, on comprend l'amplitude d'une vibration. L'unité est ici le décibel (db). La zone audible s'étant de 1 à 120 db. Tous les 10 db environ, l'intensité est doublée. L'intensité sonore est aussi désignée par le terme pression acoustique.

L'intensité peut être influencée par de nombreux facteurs. Le plus simple correspond naturellement au bouton d'intensité du moniteur. Sa seule fonction est de modifier l'amplitude des vibrations électriques. Mais la distance entre l'auditeur et le haut-parleur a aussi un effet sur l'intensité. Plus on s'éloigne de ce dernier, plus le son devient faible. De plus, l'installation d'une salle, la présence de portes ouvertes ou fermées ou autres, peut aussi modifier l'amplitude de l'onde sonore. Pour cette raison, l'intensité absolue n'est pas importante, les intensités relatives entre plusieurs sons l'étant beaucoup plus.

Il existe un rapport entre l'intensité d'un son et sa fréquence. La raison est la sensibilité de l'oreille humaine. Les sons hauts et bas sont plus difficilement audibles que les sons moyens, même si physiquement, ils possèdent la même pression acoustique en décibels. Cette zone moyenne de sons s'étend environ de 1000 à 3000 hertz. A l'intérieur de cette zone fréquentielle se trouvent les vibrations de la voix humaine, raison pour laquelle la sensibilité dans cet intervalle est importante.

L'intensité d'un son peut se modifier périodiquement, à l'intérieur d'un milieu connu, cet effet se nommant Trémolo. L'allure de l'intensité peut être modifiée entre le début et la fin d'un son. Ainsi, celui-ci peut débuter très fort et s'assourdir progressivement. Mais il peut aussi débuter très fort, puis baisser avec une mesure déterminée et s'arrêter brusquement. Il peut aussi commencer très faiblement et augmenter progressivement son intensité. Il y a, comme on peut le voir, énormément de combinaisons possibles.

*Le timbre***Schéma des formes d'ondes typiques***Figure 1 - 38*

Le troisième et dernier paramètre d'un son est quelque peu compliqué. Il s'agit du timbre, qui joue un rôle important. Il y a une centaine d'instruments différents qui peuvent jouer un son avec la même fréquence et à la même intensité. Le son résonnera pourtant de

manière différente. La raison résidera dans la forme de la vibration. Le schéma suivant montre 4 formes d'ondes importantes. Chacune, quelle qu'elle soit, apparaît comme un mélange d'oscillations sinusoïdales, qui, entre elles, se tiennent à des proportions fréquentielles fixes. Pour une onde carrée, par exemple, la première partie du son est à la fréquence fondamentale, la deuxième a le triple de la fréquence, mais plus qu'un tiers de l'amplitude. La troisième partie du son possède une fréquence multipliée par cinq et un cinquième de l'amplitude etc...

Schéma de composition de différentes formes d'ondes, issue de vibrations sinusoïdales.

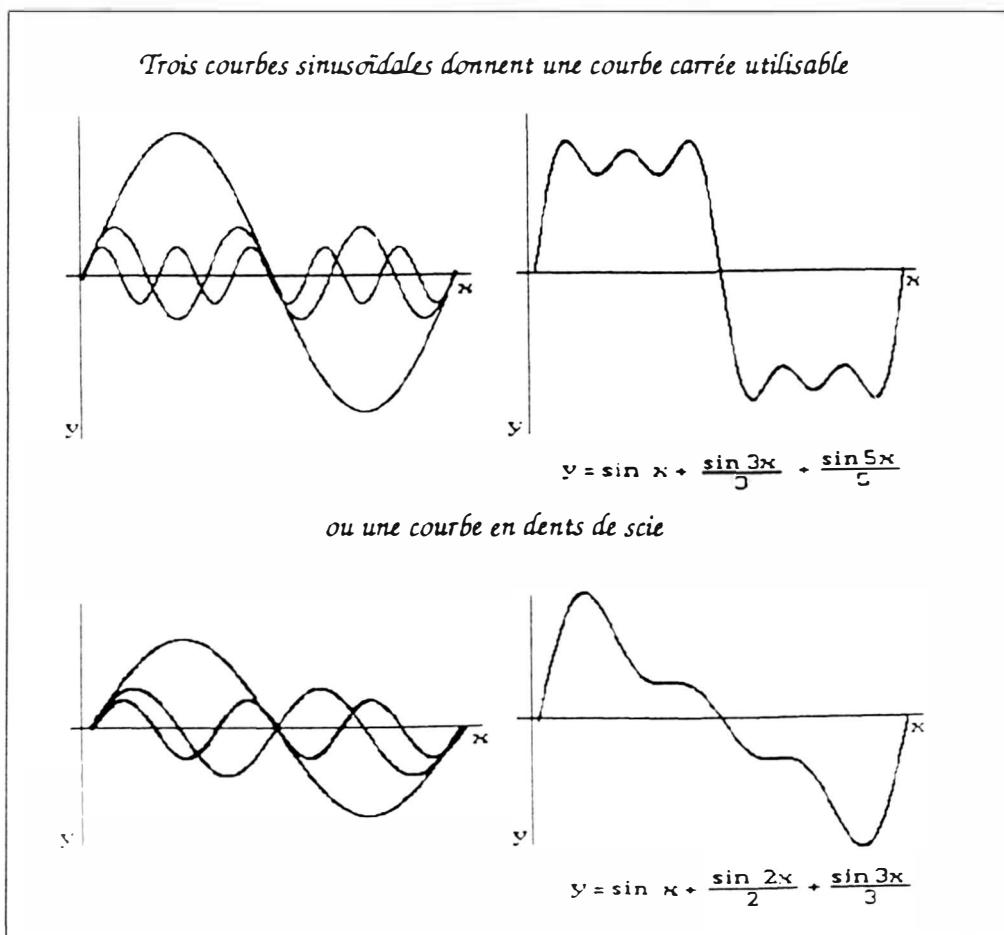


Figure 1 - 39

Ce schéma nous montre ainsi les différentes compositions pour l'obtention d'une vibration carrée ou en dents de scie. Pour plus de simplicité, ces ondes ne sont formées que des trois premières parties du son.

Comme cela a été dit, toutes les formes d'ondes périodiques se composent d'oscillations sinusoïdales (ou série de Fourier). Celles-ci forment pour un son ses harmoniques. L'oscillation sinusoïdale pure apparaît seulement, en toute logique, avec la première partie du son. Une vibration carrée comporte une infinité d'harmoniques. Le nombre d'harmoniques, leur rapport fréquentiel et d'amplitudes déterminent le timbre d'un son. La série d'harmoniques est très importante, car l'oreille humaine ne réagit qu'avec des vibrations sinusoïdales. Un son dont la forme d'onde s'éloigne d'une sinusoïde pure, sera dans l'oreille avant d'être entendu, décomposé en parties de son ou partiel. On devra prendre en compte ce fait dans les explications à venir.

Les bruits

En dehors des sons, on trouve le bruit. Un son peut se définir très précisément et se générer électroniquement, ce qui pour un bruit est plus difficile à obtenir. Il possède soit une fréquence déterminée, soit une intensité à enveloppe définie et de plus, la forme de l'onde n'est pas unitaire. Il reproduit en fait une combinaison arbitraire d'événements sonores. Le souffle est l'élément de base du bruit car c'est un mélange d'une infinité de vibrations, dans lesquelles les fréquences et positions de phase n'ont aucun rapport entre elles. Par exemple, le vent souffle car les millions de molécules d'air s'entrechoquent les unes avec les autres ou avec la surface de la terre, la conséquence étant la formation et le déplacement de vibrations. Cet ensemble forme un son de combinaison qu'on nomme souffle du vent.

La création des sons sur l'Amiga

Schéma de digitalisation d'une forme d'onde

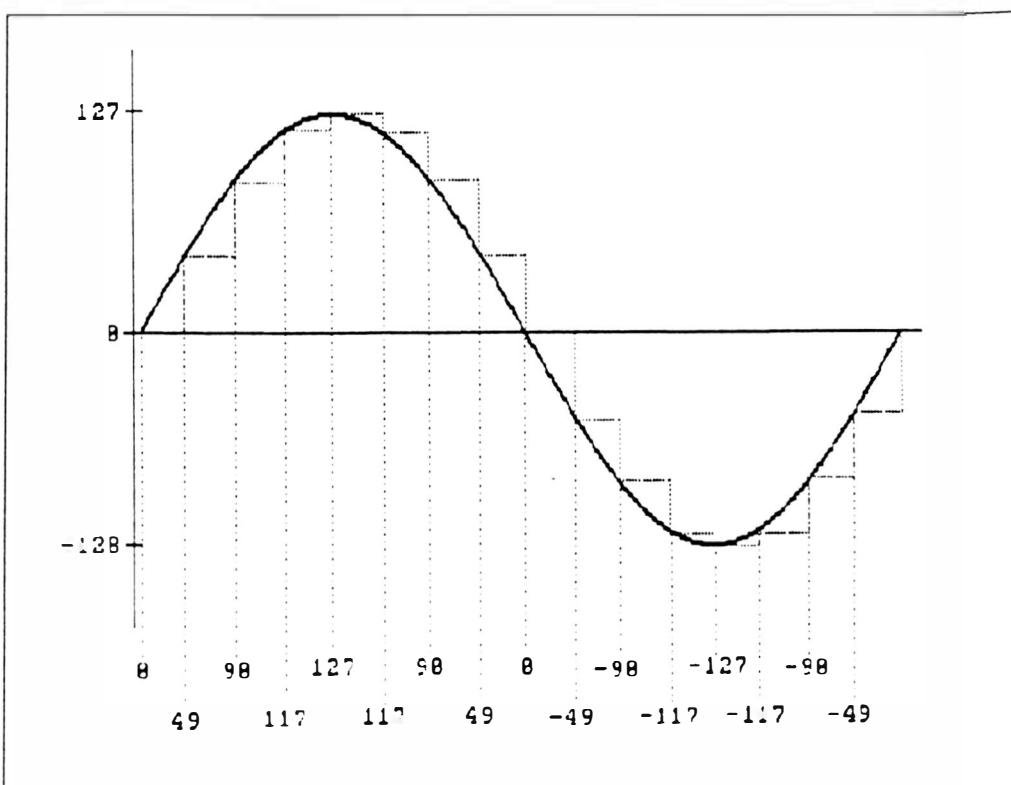


Figure 1 - 40

Le critère important pour l'analyse des capacités acoustiques d'un ordinateur est sa complexité. L'optimum serait que les trois paramètres d'un son (fréquence, intensité, timbre) se laissent manipuler librement.

Pour l'Amiga, on a cherché à s'approcher le plus près possible de cette condition. Pour ne pas être assujetti à une forme d'onde, l'équivalent digital sera mis en mémoire et sera modifié dans les oscillations électriques correspondantes, au moyen d'un transformateur digital/ analogique.

Autrement dit, la vibration sera digitalisée et stockée en mémoire. Pour la sortie, les données digitalisées seront à nouveau transformées en données analogiques et enfin transmises à l'amplificateur.

Le schéma 1-38 présente différentes formes d'ondes. Lorsque l'une d'elles doit être modifiée par l'ordinateur, son allure doit être reproduite à partir de calculs.

C'est pour cette raison qu'on partage l'oscillation d'une onde en un nombre pair de sections de taille égale. On débutera toujours ce fractionnement au passage à zéro de la courbe. Pour chacune des sections, on transfère la valeur *Y* correspondante en mémoire. On aura ainsi une suite de nombres, qui sur la courbe, représenteront un moment déterminé. Le terme anglais pour ces valeurs digitalisées est 'sample', ce qui signifie échantillon.

Lors de l'émission, l'Amiga modifie à nouveau la valeur des nombres issus de la mémoire, en tension de sortie correspondante. Etant donné que la vibration, issue de la digitalisation, ne peut se décomposer qu'en un nombre limité d'échantillons, la courbe de sortie ne pourra être reconstruite qu'avec ce nombre. C'est pourquoi cette dernière prend une allure en forme de marche, qu'on peut voir sur le schéma 1-40.

La qualité de reproduction d'une tonalité, proche de l'original, dépend essentiellement de deux facteurs :

Le premier est la résolution du signal digitalisé. On entend par là l'échelle de valeurs utilisée pour un échantillon. Sur l'Amiga, cette échelle est constituée d'un nombre 8 bits, pouvant varier de -128 à +127. Chaque valeur d'entrée peut donc prendre une des 256 valeurs en mémoire. Comme la résolution du signal d'entrée analogique est théoriquement infinie et que chaque échantillon est limité, il apparaît ici une erreur. On la nomme erreur d'arrondi (ou de quantification). Ainsi, lorsqu'une valeur d'entrée se trouvera n'importe où, entre deux nombres ne correspondant en aucun cas à l'un des 256 pas de digitalisation, elle sera arrondie. L'erreur de quantification maximale se monte à 1/256 de la valeur digitalisée (on dit aussi : l'erreur atteint 1 LSB).

L'erreur de quantification est associée au bruit de quantification. Ce bruit se manifeste en augmentant avec la taille de l'erreur de quantification.

Une zone de valeur de 8 bits permet déjà une très bonne reproduction de la vibration originale. Une qualité Hifi nécessite une plus grande résolution. Une platine CD travaille, par exemple, avec 16 bits.

Le deuxième paramètre, pour une tonalité digitale de qualité, est le 'sampling-rate'. On entend par ce terme le nombre d'échantillons par seconde, un nombre élevé donnant naturellement une meilleure restitution. Le taux d'échantillonnage se laisse librement déterminer sur l'Amiga, à l'intérieur de limites préétablies. On doit tout d'abord, considérer le nombre d'échantillons que l'on veut employer pour la digitalisation d'un son. Sur notre exemple (schéma 1-40), il y a 16 valeurs. Il en résulte une courbe sinusoïdale formée de marches peu différentes du signal normal.

La sortie de sons digitalisés

L'Amiga possède 4 canaux audio, qui travaillent suivant le même principe :

Un son digitalisé en provenance de la mémoire, via les canaux DMA, sera émis au moyen d'un transformateur digital/analogique. Cette émission se répète continuellement, de telle sorte que la vibration se stabilise. Les canaux 0 et 3 ainsi que les canaux 1 et 2 forment respectivement le canal stéréo gauche et le canal stéréo droit.

A chaque canal audio correspond un canal DMA. Etant donné que les accès DMA de l'Amiga se déroulent mot par mot, on réunit deux échantillons sur un même mot de données. C'est pour cette raison qu'un nombre pair d'échantillons est nécessaire. La partie de poids fort du mot (bits 8-15) sera toujours prise en compte avant les bits inférieurs (bits 7-0). La liste de données pour notre son digitalisé apparaîtra en mémoire de la façon suivante :

(Start correspond à l'adresse de départ de la liste dans la CHIP-RAM) :

```
Start:
dc.b 0,49      ;1er mot de données, échantillon 1 et 2
dc.b 90,117    ;2ème mot de données, échantillon 3 et 4
dc.b 127,117   ;3ème mot de données, échantillon 5 et 6
dc.b 90,49      ;4ème mot de données, échantillon 7 et 8
dc.b 0,-49      ;5ème mot de données, échantillon 9 et 10
dc.b -90,117   ;6ème mot de données, échantillon 11 et 12
dc.b -127,-117 ;7ème mot de données, échantillon 13 et 14
dc.b -90,-49    ;7ème mot de données, échantillon 15 et 16
```

Le transformateur digital/analogique traite les échantillons par nombres 8 bits. Comme la technique digitale le précise plus haut, les échantillons doivent être émis sous la forme de deux compléments. Cette conversion est exécutée par l'assembleur, de telle manière que les valeurs négatives puissent être écrites directement dans la liste des données.

On doit alors choisir un des 4 canaux audio sur lequel le son sera émis. Ensuite, le canal DMA audio sera initialisé. Ce sont 5 registres qui établissent les paramètres d'exploitation. Les deux premiers forment une paire de registres d'adresse, identiques aux registres déjà rencontrés dans les autres accès DMA. Ils se nomment AUDxLCH et AUDxLCL, où x correspond au numéro de canal DMA.

Registre	Nom	Fonction
\$0A0	AUD0LCH	Pointeur sur les données audio (bits 16-18)
\$0A2	AUD0LCL	du canal 0 (bits 0-15)
\$0B0	AUD1LCH	Pointeur sur les données audio (bits 16-18)
\$0B2	AUD1LCL	du canal 1 (bits 0-15)
\$0C0	AUD2LCH	Pointeur sur les données audio (bits 16-18)
\$0C2	AUD2LCL	du canal 2 (bits 0-15)
\$0D0	AUD3LCH	Pointeur sur les données audio (bits 16-18)
\$0D2	AUD3LCL	du canal 3 (bits 0-15)

L'initialisation d'un pointeur d'adresse peut se faire avec l'instruction MOVE.L :

```
LEA $DDF000,a5 ;adresse de base des circuits spécialisés dans a5
MOVE.L #Start,AUDOLCH(a5) ;'Start' mis dans AUDOLC
```

Enfin, il faut communiquer au contrôleur DMA, la longueur d'une oscillation digitalisée, c'est-à-dire le nombre d'échantillons la représentant. Les registres correspondants sont AUDxLEN :

Adresse	Nom	Fonction
\$0A4	AUDOLEN	nombre de mots de données audio du canal 0
\$0B4	AUD1LEN	nombre de mots de données audio du canal 1
\$0C4	AUD2LEN	nombre de mots de données audio du canal 2
\$0D4	AUD3LEN	nombre de mots de données audio du canal 3

La longueur n'est pas donnée en nombre d'octets, mais en nombre de mots. C'est la raison pour laquelle le nombre d'octets doit être divisé par 2, avant d'être écrit dans le registre AUDxLEN.

L'initialisation du registre AUDxLEN peut se faire avec l'instruction MOVE. Pour éviter le décompte des mots, on définit deux labels : 'Start' correspondant à l'adresse de départ de la liste de données, 'End' correspondant à l'adresse de fin + 1 (cf. exemple de liste de données). Les adresses de base des circuits spécialisés se trouvent dans le registre a5 (\$DDF000) :

```
MOVE.W #(End-Start)/2,AUDOLEN(a5)
```

On s'occupe ensuite de l'intensité des sons. Sur l'Amiga, on peut choisir l'intensité de chaque canal séparément. Ainsi, 65 pas sont à disposition, de 0 (inaudible) à 64 (intensité maximum). Les registres correspondants se nomment AUDxVOL :

Adresse	Nom	Fonction
\$0A8	AUD0VOL	intensité du canal audio 0
\$0B8	AUD1VOL	intensité du canal audio 1
\$0C8	AUD2VOL	intensité du canal audio 2
\$0D8	AUD3VOL	intensité du canal audio 3

Si on initialise le canal audio avec une intensité moyenne, on obtient :

```
MOVE.W #32,AUDOVOL(a5)
```

Le dernier paramètre manquant est le taux d'échantillonnage (sampling-rate). Il détermine l'intervalle de temps séparant l'émission de deux octets de données vers le transformateur digital/analogique. Ce taux détermine ainsi la fréquence des sons. Comme cela a déjà été défini au départ, la fréquence correspond aux nombre de vibrations par seconde. Chaque oscillation est constituée d'un nombre variable d'échantillons ; dans notre exemple, le nombre se monte à 16. Lorsque le taux d'échantillonnage reproduit le nombre d'échantillons par seconde, la fréquence du son correspond à ce taux divisé par le nombre d'échantillons par oscillation :

$$\text{Fréquence du son} = \text{taux d'échantillonnage} / \text{échantillons par oscillation}$$

Malheureusement, le taux d'échantillonnage n'est pas donné exactement en Hertz. Le contrôleur DMA saura, par contre, le nombre de cycles de bus présents entre deux sorties. Un cycle a une durée exacte de 279,365 nanosecondes (milliardèmes de seconde), ou 2.79365×10^{-7} , ou 0.000000279365 secondes.

Pour arriver au taux d'échantillonnage par l'intermédiaire du nombre des cycles de bus, il suffit d'obtenir la valeur inverse du taux et on aura ainsi la durée d'un échantillon. Si on divise cette valeur par la durée en secondes d'un cycle de bus entre deux échantillons, on obtient la période d'échantillon :

$$\text{Période d'échantillon} = 1 / \text{taux d'échantillonnage} \times 2.79365 \times 10^{-7}$$

Si on veut, par exemple, émettre un son d'une fréquence de 440 Hz, c'est-à-dire un LA, le taux d'échantillonnage se calculera comme suit :

$$\begin{aligned}\text{taux d'échantillonnage} &= \text{fréquence} \times \text{échantillons par oscillation} \\ \text{taux d'échantillonnage} &= 440 \text{ Hz} \times 16 = 7040 \text{ Hz}\end{aligned}$$

La période d'échantillon sera vite déduite en mettant les valeurs à la bonne place :

$$\text{période d'échantillon} = 1 / (7040 \times 2.79365 \times 10^{-7}) = 508.4583$$

Etant donné que seules les valeurs entières sont acceptées, le résultat arrondi sera 508. Ainsi la fréquence de sortie ne sera plus exactement 440 Hz, mais l'écart se montera à 0.4 Hz.

La période d'échantillon peut théoriquement accepter les valeurs entre 0 et 65535. La zone réelle sera cependant limitée vers le haut. Chaque canal audio possède un slot DMA par ligne du RASTER, c'est-à-dire qu'à chaque ligne du RASTER, un mot, respectivement deux échantillons, peut être lu dans la mémoire. Ainsi la valeur minimale pour la période est de 124. La fréquence d'échantillon correspond alors à 28867 Hertz. Si on raccourcit la période, il peut arriver qu'un mot de données soit émis deux fois, étant donné que le prochain ne pourra être lu à temps.

Les registres de période d'échantillon se nomment AUDxPER :

Adresse	Nom	Fonction
\$0A6	AUD0PER	période d'échantillon du canal audio 0

Adresse	Nom	Fonction
\$0B6	AUD1PER	période d'échantillon du canal audio 1
\$0C6	AUD2PER	période d'échantillon du canal audio 2
\$0D6	AUD3PER	période d'échantillon du canal audio 3

L'instruction `MOVE.W #508,AUD0PER(a5)` transfère notre calcul du taux d'échantillonnage dans le registre AUD0PER. Ainsi, tous les registres du canal audio 0 seront munis de la bonne valeur, pour l'émission de notre son. Pour l'entendre, il faut activer les accès DMA sur le canal DMA audio 0. C'est ce que permettent 4 bits du registre DMACON :

Nº bit DMACON	Nom	Nº canal DMA audio
3	AUD3EN	3
2	AUD2EN	2
1	AUD1EN	1
0	AUD0EN	0

Si le canal DMA audio 0 doit être activé, le bit AUD0EN doit être mis à 1. Au même moment, il sera nécessaire d'initialiser le bit DMAEN (cf. Eléments de base).

`MOVE.W #$8201,DMACON(a5)` ;initialisation de AUD0EN et de DMAEN

A ce moment, le contrôleur DMA commencera à prendre les données audio de la mémoire et à les transférer vers le transformateur digital/ analogique. Le son pourra être entendu via le haut-parleur. Si on désire le désactiver, il suffit de mettre le bit AUD0EN à 0.

Lorsqu'on met AUDxEN à 1, les accès DMA débutent à l'adresse contenue dans AUDxLC. Il y a évidemment une exception : si le canal DMA est activé (AUDxEN=1) et si on met le bit rapidement à 0, puis nouveau à 1, sans que le canal DMA n'ait lu, entre-temps, un nouveau mot, le contrôleur DMA poursuit à l'ancienne adresse.

Interruption Audio

Les accès DMA audio débutent toujours avec l'octet de données se trouvant à l'adresse AUDxLC. Si autant de mots de données sont issus de la mémoire qu'émis, comme l'établit AUDxLEN, les accès DMA recommenceront à l'adresse AUDxLC.

Contrairement aux registres adresse du Blitter ou des BitPlanes, le contenu des registres AUDxLC ne peut pas être modifié lors des accès. Il existe en fait pour chaque canal DMA audio, un registre adresse de plus. Avant que le contrôleur DMA ne prélève le

premier octet de données de la mémoire, il copie la valeur du registre AUDxLEN dans ses registres adresse internes. Le registre AUDxLEN est aussi transféré dans un compteur interne. Lorsque ces deux transferts sont réalisés, une interruption est libérée. Comme cela a été vu au chapitre 'interruptions', il existe pour chaque canal audio, un bit d'interruption propre. L'interruption processeur de niveau 4 est essentiellement réservée pour ces bits.

Pendant que le contrôleur DMA prélève mot de données sur mot de données dans la mémoire, le processeur peut s'occuper des nouvelles données de AUDxLC et de AUDxLEN, étant donné que les deux registres sont stockés en mémoire de façon interne. Lorsque le compteur, qui a été initialisé au départ avec la valeur de AUDxLEN, arrive à 0, les données de AUDxLC et AUDxLEN seront à nouveau lues. Le processeur a donc ainsi assez de temps pour modifier la valeur des deux registres si nécessaire. La sortie son est donc possible sans interruption.

Après chaque oscillation complète, une interruption sera donc libérée. Avec un son à haute fréquence, les interruptions apparaissent souvent. On devra activer le bit Interrupt-Enable de l'interruption audio, seulement si cette dernière est absolument nécessaire. Le processeur risque alors de saturer, à force d'appels d'interruptions.

Modulation de l'intensité et de la fréquence

La possibilité de moduler la fréquence ou l'intensité permet de générer des effets de tonalité déterminée. Un canal DMA travaille donc toujours en tant que modulateur, qui modifie les paramètres des autres canaux. L'oscillateur de modulation prend les données de la mémoire comme cela a été expliqué plus haut. Mais au lieu de les transférer vers le transformateur digital/analogique, il les écrit, soit dans le registre fréquence, soit dans le registre intensité de l'oscillateur, qu'il doit modifier (AUDxVOL ou AUDxLEN). Il peut aussi influencer ces deux registres en même temps.

Dans ce cas, les données issues de la liste seront alternées dans les registres AUDxVOL et AUDxLEN. Les mots de données ont le même format que ceux du registre cible :

Volume :	bits 7-15	inutilisés
	bits 0-6	valeur d'intensité comprise entre 0 et 64
Fréquence :	bits 0-15	période d'échantillon

Le tableau suivant montre l'utilisation des mots de données de l'oscillateur de modulation dans les trois cas possibles :

		Oscillateur module :	
Nº mot de données	Fréquence	Intensité	Fréquence & intensité
1	période 1	Volume 1	Volume 1
2	période 2	Volume 2	Période 1
3	période 3	Volume 3	Volume 2

		Oscillateur module :	
Nº mot de données	Fréquence	Intensité	Fréquence & intensité
4	période 4	Volume 4	Période 2

Pour activer un canal audio en tant que modulateur, on doit initialiser le(s) bit(s) correspondant(s) du registre contrôle audio-disque (ADKCON). Chaque canal ne module que le suivant, c'est-à-dire : le canal 0 module le canal 1, 1 module 2 et 2 module 3. Le canal 3 peut aussi être commuté en modulateur, mais ses données n'étant pas utilisées pour une modulation, elles seront perdues. Enfin, si on utilise un canal audio en temps que modulateur, sa sortie audio sera désactivée.

Le registre ADKCON renferme aussi, comme son nom l'indique, des bits de gestion, pour le contrôleur disque. Ils ne sont pas utilisés ici et seront expliqués dans le chapitre suivant.

ADKCON \$09E (écriture), \$010 (lecture)

Bit n°	Nom	Fonction
15	SET/CLR	Bits activés (SET/CLR=1) ou désactivés
14 à 8		Bits de gestion du contrôleur disque
7	USE3PN	le canal audio 3 ne module rien
6	USE2P3	le canal audio 2 module la période du canal 3
5	USE1P2	le canal audio 1 module la période du canal 2
4	USE0P1	le canal audio 0 module la période du canal 1
3	USE3VN	le canal audio 3 ne module rien
2	USE2V3	le canal audio 2 module le volume du canal 3
1	USE1V2	le canal audio 1 module le volume du canal 2
0	USE0V1	le canal audio 0 module le volume du canal 1

Si on utilise un canal pour en moduler un autre, ses mots de données seront écrits dans le registre correspondant et serviront à la modulation. Sinon les deux canaux travailleront indépendamment l'un de l'autre.

Problème de la création des sons digitalisés

Schéma de la digitalisation de plusieurs oscillations, servant à l'amélioration de la qualité des sons

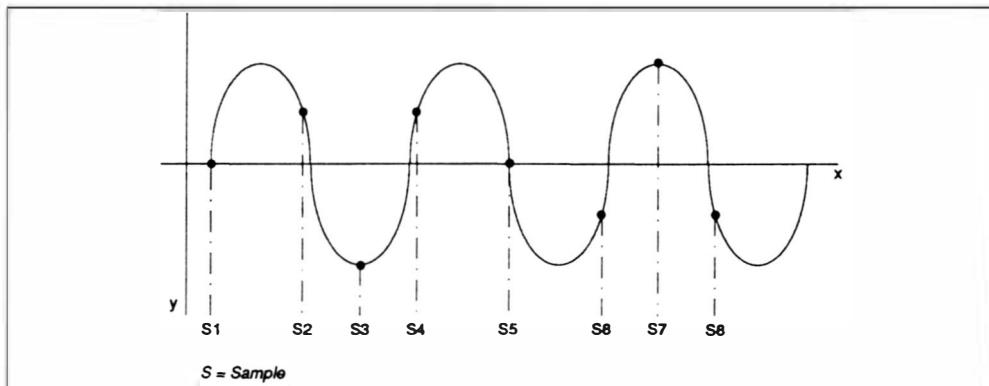


Figure 1 - 41

Cet exemple d'oscillation est défini au moyen de 16 échantillons. Le taux d'échantillonnage maximum correspond à 28867 Hz. Ceci nous donne une fréquence maximum de $28867/16 = 1804,1$ Hz.

Si on veut aller plus haut, il faut diminuer le nombre d'échantillons. Si notre sinusoïde était définie avec la moitié des échantillons, la limite fréquentielle doublerait pour atteindre 3608,3 Hz. Seulement, 8 bits de données est un petit nombre pour reproduire une sinusoïde.

Pour des sons plus hauts, le nombre d'échantillons se réduit encore. Pour 7216,75 Hz, seuls 4 sont présents. A ce moment, les formes d'ondes ne peuvent plus être différencierées dans leur représentation.

Toutefois, ces différenciations ne sont plus perceptibles par notre oreille, en hautes fréquences. En effet, plus cette dernière est haute, plus les différentes tonalités seront difficilement audibles.

Pourtant, la qualité des sons pourra être améliorée lorsqu'on utilisera, en haute fréquence, plusieurs oscillations pour former l'onde souhaitée (cf. schéma 1-41).

Schémas du filtre passe-bas

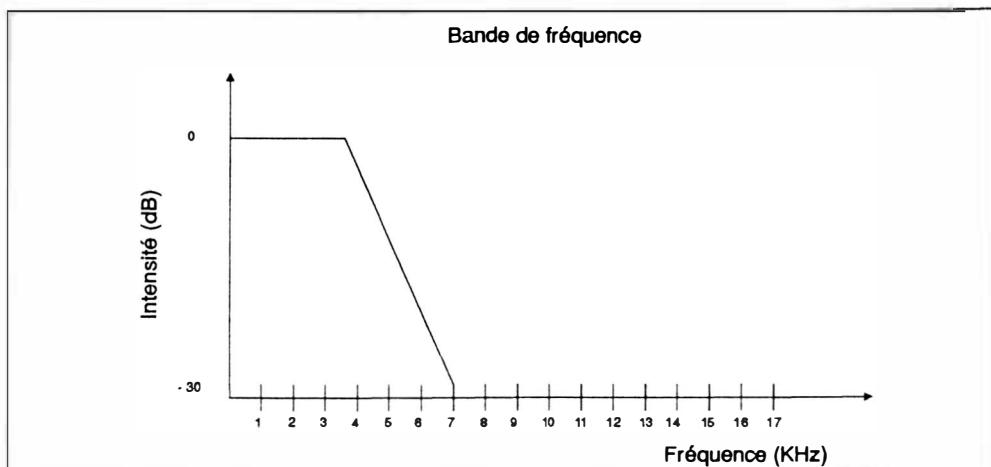


Figure 1 - 42

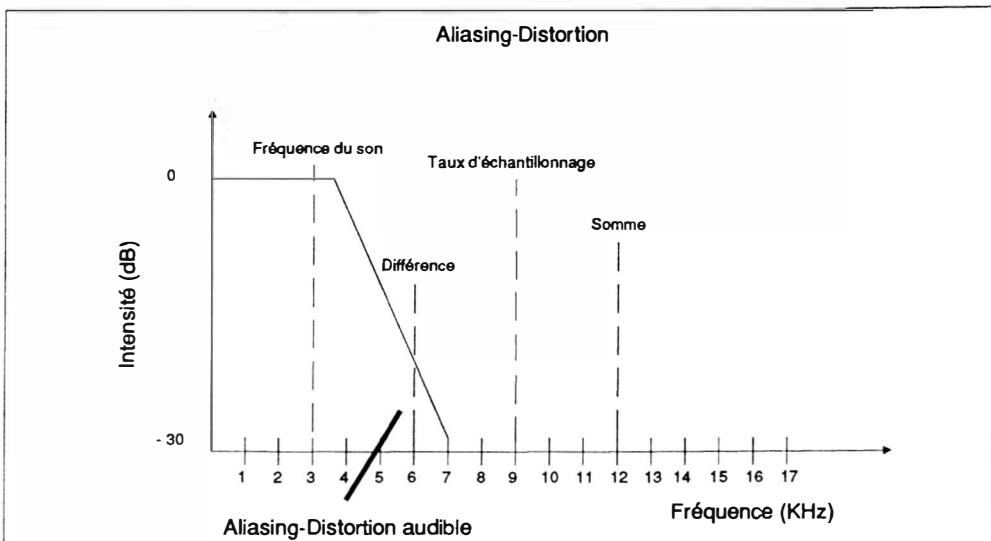


Figure 1 - 43

La frontière fréquentielle de la sortie son de l'Amiga sera encore limitée en d'autres circonstances. Lors de l'analogie des différentes données digitales, il apparaît deux

fréquences parasites, par interaction entre le taux d'échantillonnage et la fréquence désirée. L'une correspond à la somme entre le taux et la fréquence, l'autre à la différence.

Ce phénomène porte le nom de 'Aliasing distorsion'.

Par exemple, pour un son d'une fréquence de 3 KHz et un taux de 12 KHz, la différence se montera à 9 et la somme à 15 KHz.

Pour éliminer cette fréquence parasite, on a installé un filtre passe bas entre la sortie du transformateur analogique/digital et le boîtier audio. Sa fonction est représentée par les schémas 1-42 et 1-43. Toutes les fréquences inférieures à 4 KHz peuvent passer sans être parasitées. Entre 4 et 7 KHz, le signal sera de plus en plus atténué. Au dessus de 7 KHz, plus aucune fréquence ne passera. Si on prend notre exemple plus haut : le son 3 KHz ne sera pas atténué, mais la somme et la différence fréquentielle qui se montent à, respectivement, 15 et 9 KHz, ne passeront pas la limite de fréquence du filtre à 7 KHz.

De ce fait, ces deux fréquences parasites ne seront pas perceptibles à partir du haut-parleur. Si on cherche à produire un son de 3 KHz, avec un taux d'échantillonnage de 9 KHz, la différence fréquentielle correspondra à $9-3 = 6$ KHz, valeur qui ne sera pas éliminée par le filtre. Si on veut être certain que cette différence reste toujours supérieure à la limite du filtre, on doit respecter la règle suivante :

$$\text{taux d'échantillonnage} > \text{plus haute composante fréquentielle} + 7 \text{ KHz}$$

Il ne suffit pas que la différence entre la fréquence d'échantillonnage et la fréquence de sortie souhaitée soit supérieure à 7 KHz. Lorsqu'on utilise une forme d'onde, qui comprend beaucoup d'harmoniques, chacune de ces dernières produit une telle différence. Pour cette raison, on sera obligé de mettre dans la formule ci-dessus, la plus grande composante fréquentielle de la forme d'onde utilisée.

Le filtre passe-bas ne refoule pas seulement les fréquences parasites, il limite aussi la bande de fréquences de l'Amiga. En effet, les sons rares d'un morceau de musique apparaissent avec une fréquence de base comprise entre 4 et 7 KHz, mais les harmoniques des formes d'ondes déterminées se trouvent déjà à des fréquences bien plus basses. Cela se répercute particulièrement sur les ondes carrées. Le schéma 1-39. montre la formation d'une onde carrée, à partir de la combinaison de plusieurs ondes sinusoïdes, qui se trouvent dans le même rapport de fréquences. Sur le schéma, la carrée se compose exclusivement des trois partiels (parties de sons), une onde carrée réelle étant composée d'une infinité d'harmoniques. Si l'harmonique de haute fréquence est découpée ou limitée par le filtre, il en résultera un signal déformé. Dans le cas extrême où la fréquence de base de l'onde carrée approche la limite fréquentielle du filtre, il peut arriver que seule la première harmonique subsiste. L'onde carrée originelle deviendra donc une onde sinusoïde.

Développement d'intensité d'un son

La tonalité d'un instrument sera déterminée non seulement par la forme de l'onde, mais aussi par le développement d'intensité du son. Sur le premier point, l'Amiga est capable

de tout prendre en charge. Le point important est la programmation du développement d'intensité.

Le développement d'intensité d'un son se divise en quatre parties : la phase d'attaque, la phase d'affaiblissement, la phase de maintien et la phase de relâchement.

Dès que le son est joué, la phase d'attaque débute. Elle détermine la rapidité de l'intensité à passer de 0 jusqu'à la valeur maximale. Puis l'intensité baissera pendant la phase d'affaiblissement jusqu'à la valeur de maintien. Pendant la phase de maintien, le son retentira avec cette intensité. La phase de relâchement marque la fin du son, où l'intensité passera de la valeur de maintien à 0.

Ce déroulement peut être représenté sous la forme d'une courbe, que l'on nomme courbe enveloppe.

Il existe trois possibilités différentes de réalisation d'une telle courbe sur l'Amiga :

① Modulation de l'intensité

On utilise un deuxième canal audio, afin de moduler le son (par exemple, le canal 0 sera le modulateur du canal 1). Le canal 1 servira donc à émettre le son. L'intensité de départ sera mise à 0.

La courbe enveloppe souhaitée sera divisée en deux parties : la phase d'attaque et la phase de relâchement. Son développement sera digitalisé (ceci fonctionnant de la même manière qu'avec une forme d'onde) et stocké en mémoire dans deux listes de données. Si le son doit être joué, il suffit de mettre le canal 0 sur l'adresse des données de la première phase et de le démarrer. Comme l'intensité du canal 1 est modulée, l'intensité du son suivra exactement la phase d'attaque. Lorsque cette dernière atteint la valeur de maintien, la liste de données du canal 0 a été traitée. Ce dernier génère alors une interruption et voudra recommencer la liste de données par le début. A ce moment, le processeur doit réagir à l'interruption et au moyen du bit AUD0EN du registre DMA0CON, désactiver le canal 0.

Pour désactiver le son, le canal 0 est initialisé à l'adresse de la phase de relâchement et redémarré à nouveau. On attendra alors l'interruption qui montre que cette phase est terminée et on désactivera le canal 0.

Les registres du canal 0 devront être initialisés de la manière suivante, pour ce processus :

USE0V1 Ce bit du registre ADKCON doit être mis à 1, afin que le canal 0 puisse moduler l'intensité du canal 1.

AUD0LC Ce registre sera initialisé une première fois avec l'adresse de la liste de données de la phase d'attaque, puis une deuxième fois, avec l'adresse de la liste de la deuxième phase.

AUDOLEN	Ce registre contiendra, comme les adresses dans AUDOLC, la longueur des données de la première phase, puis de la deuxième phase.
AUD0VOL	Ce registre n'a aucune fonction, étant donné que la sortie audio du canal 0 est désactivée.
AUD0PER	Le contenu de ce registre détermine la vitesse avec laquelle les données d'intensité seront issues de la mémoire. On pourra ainsi contrôler la durée des 2 phases.

Au moyen de cette méthode, il est possible de former une enveloppe de courbe parfaite. Il y a pourtant un inconvénient de taille : la sortie d'un son nécessite deux canaux audio. Si on est obligé d'utiliser 4 canaux audio différents, il faudra opter pour la deuxième méthode.

② Gestion de l'intensité au moyen du processeur

La courbe enveloppe sera transférée dans la mémoire de la même manière. Cependant, on modifie cette fois-ci l'intensité à partir du processeur. Ce dernier prélève la valeur actuelle de l'intensité dans la mémoire et l'écrit dans le registre d'intensité du canal audio correspondant, ceci de manière régulière et périodique. On doit alors laisser se dérouler le programme comme une routine d'interruption. Ceci peut se passer lors de l'interruption écran vide vertical, ou alors on utilisera une interruption minuterie du CIA-B.

L'inconvénient de cette méthode est la nécessité d'un temps de calcul, étant donné que la gestion de l'intensité ne se fait plus via les canaux DMA. Mais c'est une méthode qui conviendra dans la plupart des cas d'utilisation.

③ Structure de la courbe enveloppe à partir des données Vibration

Cette méthode est avantageuse pour tout ce qui est courte tonalité ou bruitage. Au lieu de digitaliser une oscillation de la forme d'onde souhaitée, on écrit le déroulement entier en mémoire. Ceci peut être fait soit à l'aide d'une évaluation par un programme, soit à l'aide d'un digitaliseur audio. Avec ce dernier, un bruit pourra être digitalisé au moyen d'un micro et d'un transformateur analogique/digital.

Plusieurs appareils sont proposés pour l'Amiga par différentes firmes. Lorsque les données sont dans l'Amiga, on peut à chaque hauteur de son, jouer sur la vitesse. On pourra ainsi imiter des effets complexes tels que les rires où les cris humains.

Cette méthode a aussi son inconvénient : elle requiert des calculs fastidieux, ou un périphérique de plus pour stocker en mémoire la forme digitalisée d'une tonalité. De plus, le besoin en place mémoire est assez important. Une sonorité d'une durée d'une seconde, avec un taux d'échantillonnage de 20 KHz, est issue d'une liste de données prenant 20 Koctets de mémoire.

Trucs, Astuces et autres

La qualité d'un son

La zone de valeur des données digitales s'étend de -128 à 127. Cette zone doit être entièrement utilisée, le plus souvent possible. Il est préférable d'avoir une oscillation digitalisée avec une amplitude égale à 256. Sinon la qualité du son diminuera perceptiblement, étant donné que la taille de l'erreur de quantification est inversement proportionnelle à cette zone, et que le souffle dû à cette quantification atteint très vite une dimension de brouillage.

C'est pour cette raison qu'il faut éviter d'utiliser l'amplitude d'une oscillation digitalisée pour la gestion de l'intensité et que chaque canal possède son registre AUDxVOL. Si on diminue l'intensité avec ce dernier, le rapport entre le son souhaité et le bruit parasite sera entièrement conservé, ainsi que la qualité sonore de l'Amiga.

Changement de forme d'onde sans brouillage

Pour éviter le brouillage des sons, comme les craquements ou les sauts d'intensité, on doit appliquer les règles suivantes :

Chaque oscillation ne doit être digitalisée qu'entre deux passages à 0, c'est-à-dire qu'on ne commence à transférer les données en mémoire qu'avec un point d'intersection avec l'axe des X. Si on se tient à cette règle, il est évident que les formes d'ondes débuteront ou se termineront avec la même valeur en mémoire, c'est-à-dire 0. Ainsi, plusieurs oscillations différentes pourront apparaître à la suite et ceci sans entendre des bruits parasites.

Deuxièmement, il faut faire attention à ce que l'intensité des deux oscillations soit à peu près identique. Cette intensité correspondra à la valeur effective de la vibration. La valeur effective d'une oscillation ou vibration est égale à l'amplitude d'un signal carré, la superficie en-dessous de la courbe étant à peu près la même que celle de l'oscillation.

Cette valeur effective détermine l'intensité d'une vibration. Seulement dans le cas d'une carrée, cette valeur est égale à l'amplitude. Si on passe d'une forme d'onde à une autre forme possédant une valeur effective plus haute, cette dernière résonnera beaucoup plus fort que la précédente.

La valeur effective d'une oscillation se laisse calculer assez facilement, à partir des données digitalisées :

On additionne les montants complets des octets et on les divise par le nombre d'octets de données.

Si on veut utiliser la zone complète de valeur 8 bits du transformateur digital/analogique pour toutes les formes d'ondes, leur valeur effective ne pourra pas toujours correspondre. On devra alors s'ajuster, lors d'un changement de forme d'onde, à la valeur d'intensité du registre AUDxVOL correspondant.

Jouer une note

Normalement, un morceau de musique est composé d'une suite de notes. Si on veut exécuter une telle partition sur l'Amiga, on doit transformer les valeurs des notes en périodes d'échantillonnage correspondantes. Pour éviter des calculs fastidieux, on préfère utiliser un tableau qui comprend les valeurs des périodes d'échantillonnage pour tous les demi-ton d'une octave :

Tableau des valeurs de période d'échantillon des notes de musique :

Note	Fréquence (Hz)	Période d'échantillon pour AUDxLEN = 16	
C : do	261.7	427	(262.0)
C# : do dièse	277.2	404	(276.9)
D : ré	293.7	381	(293.6)
D# : ré dièse	311.2	359	(311.6)
E : mi	329.7	339	(330.0)
F : fa	349.3	320	(349.6)
F# : fa dièse	370.0	302	(370.4)
G : sol	392.0	285	(392.5)
G# : sol dièse	415.3	269	(415.8)
A : la	440.0	254	(440.4)
A# : la dièse	466.2	240	(466.0)
B : si	493.9	226	(495.0)
C : do	523.3	214	(522.7)

(les valeurs entre parenthèses indiquent les fréquences réelles des périodes d'échantillonnage correspondantes).

Voici encore une petite remarque pour le calcul des valeurs ci-dessus :

La fréquence d'un demi-ton est toujours supérieure d'un facteur racine douzième de 2 que le précédent.

$$440 \text{ (La)} * 2^{(1/12)} = 466.2 \text{ (la dièse)}$$

$$466.2 \text{ (La\#)} * 2^{(1/12)} = 493.9 \text{ (Si)}$$

etc ...

Une octave correspond toujours à une fréquence doublée.

Si on veut jouer maintenant une note d'une octave qui ne se trouve pas dans le tableau, on a deux possibilités :

- ① On modifie la période d'échantillonnage. Pour chaque octave vers le haut, on doit diviser la valeur par deux. Une octave plus bas correspond à doubler la valeur. Ceci est assez simple, mais on atteint vite les limites connues. Pour un champ de données de 32 octets ($AUDxLEN = 16$), comme dans notre tableau, la plus petite période d'échantillonnage possible est déjà atteinte avec un Fa seconde. On devra alors diminuer la liste de données.

Dans ce cas, un problème risque d'apparaître dans la zone des sons bas, étant donné que la fréquence parasite de l'Aliasing distortion est perceptible.

La deuxième solution semble être la meilleure :

- ② On compose pour chaque octave, une liste de données propres. La valeur de la période d'échantillonnage restera ainsi constante pour chaque octave. Elle ne servira qu'au choix des demi-tons. S'il existe un son d'une octave supérieure dans le tableau, on utilise une liste de données moitié moins grande. Respectivement, elle aura une longueur doublée pour une octave inférieure.

L'étendue normale des sons comprend environ 8 octaves, c'est-à-dire que 8 listes de données sont nécessaires par forme d'onde.

Pour équilibrer cette dépense de temps importante, on aura toujours, avec ce procédé, une sonorité optimale indépendamment de la hauteur du son.

Générer des hautes fréquences

La période d'échantillonnage minimale correspond à 124. La raison est que le DMA audio n'a pas la capacité de lire les données audio à temps, avec une période d'échantillonnage plus courte. Le mot de données précédent sera émis plusieurs fois. Cet effet peut être très bien utilisé. Comme le mot de donnée lu renferme 2 échantillons, on peut engendrer un signal carré de haute fréquence. Avec une période d'échantillonnage de 1, on obtient une fréquence d'échantillonnage de 3.58 MHz et une fréquence de sortie de 1.74 MHz. Pour pouvoir utiliser ces hautes fréquences comme signal de sortie, on est obligé de les faire passer sur le filtre passe bas. Pour ce faire, on pourra employer l'entrée AUDIN (broche 16) du connecteur série (RS232).

Pour générer de telles hautes fréquences, on devra mettre dans le registre AUDxVOL, l'intensité maximum ($AUDxVOL = 64$).

Jouer de la musique polyphonique

Comme l'Amiga possède 4 canaux audio entièrement indépendants, il est possible de générer 4 sons différents au même moment. On pourra ainsi exécuter directement un morceau de musique à quatre voix.

On peut faire mieux. En effet, 4 canaux audio ne signifient pas que le maximum de voix corresponde à 4. On a vu précédemment qu'une forme d'onde est reproduite par une

combinaison de plusieurs signaux sinusoïdaux. De la même façon que ces harmoniques forment une onde, on peut, par combinaison de plusieurs formes d'ondes, générer une sonorité à plusieurs voix. Les signaux de sortie des canaux audio 0 et 3 seront mixés ensemble, dans un canal stéréo, inclus dans PAULA. Ainsi les formes d'ondes des deux canaux seront combinées, pour former un seul signal à deux voix.

Ce qu'il est possible de faire avec des signaux analogiques électroniques, l'est aussi par calcul sur des données digitales. On peut additionner des données digitales de deux formes d'ondes différentes, et émettre le résultat sur le canal audio. On aura alors deux voix par canal audio, ce qui n'est pas une limite, car théoriquement, n'importe quel nombre de voix est possible sur le même canal audio.

Ce nombre est vite limité par la vitesse de calcul, mais 16 voix sont tout de même possibles.

La détermination de la somme des signaux est assez simple. A chaque moment, la valeur actuelle de tous les sons sera additionnée, puis le résultat sera divisé par leur nombre. On obtiendra un signal carré de manière analogue lorsqu'on additionnera les signaux sinusoïdaux du même rapport de fréquence.

Sortie Audio sans DMA

Comme pour tous les canaux DMA, il existe pour le DMA audio, des registres de données dans lesquels les canaux DMA transfèrent des données et où le processeur peut avoir un accès écriture.

Les registres de données audio :

Adresse	Nom	Fonction
\$0AA	AUD0DAT	Ces quatre registres renferment toujours les mots de données
\$0BA	AUD1DAT	les mots de données
\$0CA	AUD2DAT	paire d'échantillons. Celui qui correspond aux bits supérieurs (8-15) sera toujours transféré en premier.
\$0DA	AUD3DAT	

Pour pouvoir constituer les registres de données audio à partir du processeur, on doit désactiver les accès DMA en mettant AUDxEN à 0. La génération des interruptions audio sera alors modifiée.

Elles apparaîtront toujours après la sortie des deux échantillons dans les registres AUDxDAT et non comme avant, au début de chaque liste de données audio.

Si on ne charge pas à temps les nouvelles données dans AUDxDAT, les deux derniers échantillons ne seront pas répétés par la routine DMA, mais la sortie restera à la valeur du dernier octet de données (l'octet de poids faible de AUDxDAT).

La programmation directe des registres de données audio coûte très cher en temps de calcul. A part les cas spéciaux, il est préférable d'utiliser les accès DMA audio.

Quelques résultats

Valeurs des registres AUDxVOL en décibels (0 dB = pleine intensité) :

AUDxVOL dB	AUDxVOL dB	AUDxVOL dB	AUDxVOL dB
64	0.0	48	-2.5
63	-0.1	47	-2.7
62	-0.3	46	-2.9
61	-0.4	45	-3.1
60	-0.6	44	-3.3
59	-0.7	43	-3.5
58	-0.9	42	-3.7
57	-1.0	41	-3.9
56	-1.2	40	-4.1
55	-1.3	39	-4.3
54	-1.5	38	-4.5
53	-1.6	37	-4.8
52	-1.8	36	-5.0
51	-2.0	35	-5.2
50	-2.1	34	-5.5
49	-2.3	33	-5.8
16	-12.0		
15	-12.6		
14	-13.2		
13	-13.8		
12	-14.5		
11	-15.3		
10	-16.1		
9	-17.0		
8	-18.1		
7	-19.2		
6	-20.6		
5	-22.1		
4	-24.1		
3	-26.6		
2	-30.1		
1	-36.1		

(AUDxVOL = 0 correspond à une valeur dB minimum infinie).

Lorsque AUDxVOL = 64, la valeur digitale de 127 correspond à une tension de sortie d'environ 400 millivolts, -128 correspondant à -400 millivolts. Une modification de 1 LSB produit une fluctuation d'environ 3 millivolts.

Exemple de programmes

Programme 1 : Génération d'un son sinusoïdal simple

Ce programme génère un son sinusoïdal avec une fréquence de 440 Hz. On utilisera le même tableau d'échantillons employé dans notre premier exemple. La plus grande partie du programme concerne l'allocation de la CHIP-RAM pour la liste des données audio.

Le son sera émis par le canal audio 0, jusqu'à ce que le bouton de la souris soit enfoncé.
Le programme restituera alors la zone mémoire employée.

```
;*** génération d'un son simple ***

;registres des circuits spécialisés

INTENA - $9A          ; registre d'autorisation d'interruption (écriture)
DMACON - $96          ; registre de contrôle DMA (écriture)

;registres audio

AUDOLC - $A0           ; adresse de la liste de données audio
AUDOLEN - $A4           ; longueur de la liste de données audio
AUDOPER - $A6           ; période d'échantillonnage
AUDIOVOL - $A8          ; intensité

ADKCON - $9E           ; gestion de la modulation

;registre port A du CIA-A (bouton de la souris)

CIAAPRA - $BFE001

;Exec Library Base Offsets

AllocMem - -30-168     ;ByteSize,Requirements/d0,d1
FreeMem - -30-180      ;MemoryBlock,ByteSize/a1,d1

;autres labels

Execbase - 4
Chip - 2                ;CHIP-RAM sollicitée

;*** avant programme ***

Start:

;solliciter la mémoire pour la liste de données audio

move.l Execbase,a6
moveq #Alsize,d0        ;taille de la liste des données audio
moveq #Chip,d1
jsr AllocMem(a6)         ;mémoire sollicitée
beq Fin                  ;erreur -> fin du programme

;copie de la liste de données audio dans la CHIP-RAM

move.l d0,a0              ;adresse dans la CHIP-RAM
move.l #ALstart,a1         ;adresse dans le programme
moveq #Alsize-1,d1         ;compteur de boucle

Loop: move.b (a1)+,(a0)   ;liste de données dans la CHIP-RAM
dbf d1,Loop

;*** programme général ***

;initialisation des registres audio
```

```

lea $DFF000,a5
move.w #$000F,DMACON(a5)      ;DMA audio désactivé
move.l d0,AUDOLC(a5)          ;adresse de la liste des données activée
move.w #Alsize/2,AUDOLEN(a5)   ;longueur en mot
move.w #32,AUDIOVOL(a5)       ;intensité moyenne
move.w #508,AUDOPER(a5)       ;fréquence: 440 Hz

move.w #$00FF,ADKCON(a5)      ;modulation désactivée

;activer les accès DMA audio

move.w #$8201,DMACON(a5)      ;canal 0 activé

;attendre l'action du bouton de la souris

Wait: btst #6,CIAAPRA
bne Wait

;désactiver les accès DMA audio

move.w #$0001,DMACON(a5)      ;canal 0 désactivé

;*** Fin de programme ***

move.l d0,a1                  ;adresse de la liste de données
moveq #Alsize,d0               ;longueur
jsr FreeMem(a6)               ;mémoire occupée, libérée

Fin: clr.l d0
rts

;liste de données audio

Alstart:
dc.b 0,49
dc.b 90,117
dc.b 127,117
dc.b 90,49
dc.b 0,-49
dc.b -90,-117
dc.b -127,-117
dc.b -90,-49
Alfin:
Alsize = Alfin - Alstart     ;longueur de la liste de données audio

```

Programme 2 : Son sinusoïdal avec Vibrato

Ce programme correspond à une suite du précédent. Il émet le même son sinusoïdal sur le canal 1. Le canal 0 module la fréquence du canal 1 et génère ainsi un vibrato. Les données du vibrato reproduisent une oscillation sinusoïdale digitalisée, la valeur de la période d'échantillonnage du point 0 correspondant à un FA, c'est-à-dire 508.

```

;*** générer un vibrato ***
;registres des circuits spécialisés

INTENA = $9A                 ; registre d'autorisation d'interruption (écriture)

```

```

DMACON = $96          ;registre de contrôle DMA (écriture)

;registres audio

AUDOLC = $A0          ;adresse de la liste de données audio
AUDIOLEN = $A4         ;longueur de la liste de données audio
AUDOPER = $A6           ;période d'échantillonnage
AUDIOVOL = $A8          ;intensité

AUD1LC = $B0
AUD1LEN = $B4
AUD1PER = $B6
AUD1VOL = $B8

ADKCON = $9E          ;gestion de la modulation

;CIA-A registre port A (bouton de la souris)

CIAAPRA = $BFE001

;Exec Library Base Offsets

AllocMem = -30-168     ;ByteSize,Requirements/d0,d1
FreeMem = -30-180       ;MemoryBlock,ByteSize/a1,d0

;autres labels

Execbase = 4
Chip = 2

;*** avant programme ***
Start:

;solliciter la mémoire pour les listes de données
move.l Execbase,a6
move.l #Size,d0          ;longueur des deux listes
moveq #Chip,d1
jsr AllocMem(a6)         ;mémoire sollicitée
beq fin

;copie de la liste de données audio dans la CHIP-RAM

move.l d0,a0              ;adresse dans la CHIP-RAM
move.l #ALstart,a1         ;adresse dans le programme
move.w #Size-1,d1           ;compteur de boucle

Loop: move.b (a1)+,(a0)+ ;listes dans la CHIP-RAM
dbf d1,Loop

;*** programme général ***
;initialisation des registres audio

move.l d0,d1              ;adresse de la liste de données audio
add.l #ALsize,d1            ;adresse du tableau Vibrato
lea $DFF000,a5
move.w #$000F,DMACON(a5)    ;accès DMA désactivés
move.l d1,AUDOLC(a5)        ;pointeur sur le tableau vibrato

```

```
move.w #Vibsize/2,AUDOLEN(a5) ;longueur du tableau vibrato
move.w #8961,AUDOPER(a5)      ;fréquence vibrato
move.l d0,AUD1LC(a5)          ;liste de données audio dans canal 1
move.w #Alsizel/2,AUD1LEN(a5)  ;longueur de la liste
move.w #32,AUD1VOL(a5)        ;intensité moyenne
move.w #$00FF,ADKCON(a5)      ;autres modulations activées
move.w #$8010,ADKCON(a5)      ;canal 0 module la période du canal 1
;accès DMA audio activés
move.w #$8203,DMACON(a5)      ;canaux 0 et 1 activés
;attente de l'action du bouton de la souris
Wait: btst #6,CIAAPRA
bne Wait
;accès DMA audio désactivés
move.w #$0003,DMACON(a5)      ;canaux 0 et 1 désactivés
;*** fin du programme ***
move.l d0,a1                  ;adresse des listes
move.l #Size,d0                ;longueur
jsr FreeMem(a6)                ;mémoire libérée
Fin: clr.l d0
rts
;liste de données audio
Alstart:
dc.b 0,49
dc.b 90,117
dc.b 127,117
dc.b 90,49
dc.b 0,-49
dc.b -90,-117
dc.b -127,-117
dc.b -90,-49
Alfin:
Alsizel = Alfin - Alstart     ;longueur de la liste de données audio
;tableau vibrato
Vibstart:
dc.w 508,513,518,522,524,525,524,522,518,513
dc.w 508,503,498,494,492,491,492,494,498,503
Vibfin:
Vibsize = Vibfin - Vibstart   ;longueur du tableau vibrato
Size = Alsizel + Vibsize      ;longueur totale des deux listes
;fin du programme
```

1.5.9. Souris, joysticks et paddles

La souris, le joystick ou le paddle peuvent être, tous les trois, raccordés à l'Amiga. Ces derniers seront traités ensemble, avec leurs registres respectifs. Les références du gameport, auquel on raccordera ces périphériques, ont été décrites dans le chapitre des connexions. On commencera par la souris :

La souris

La souris est le périphérique le plus employé. Voici l'explication de son fonctionnement et du lien existant entre elle et la flèche à l'écran :

Si on retourne la souris, on remarque une boule dense, entourée de caoutchouc, qui roule lors d'un déplacement de la souris. Ce mouvement de la boule sera enregistré sur deux axes perpendiculaires. Ils sont disposés de telle manière qu'ils pivotent, l'un pour un mouvement le long de l'axe des X, l'autre pour n'importe quelle direction de l'axe Y. Si on déplace la souris en diagonale, les deux axes tournent suivant les composantes X et Y du mouvement de la souris.

A l'extrémité de chaque axe, un disque perforé est disposé. Lors d'une rotation, il interrompt sans cesse un faisceau lumineux. Ce dernier sera transformé en signal électrique et intensifié pour finalement accéder à l'ordinateur, au moyen du câble de la souris.

L'Amiga a maintenant la possibilité d'établir si et avec quelle vitesse la souris se déplace. Mais il ne sait toujours pas dans quelle direction, c'est-à-dire vers la droite ou la gauche, vers le haut ou le bas, est le mouvement.

Une petite astuce résoud ce problème. Sur chaque disque perforé sont disposées deux cellules photoélectriques, qui sont déplacées l'une vers l'autre, d'une demi perforation. Si le disque tourne dans un sens, un des faisceaux sera interrompu avant l'autre.

Si on change le sens du mouvement, l'état des signaux photo-électriques se modifiera aussi. Ainsi, l'Amiga pourra déterminer le sens du mouvement de la souris.

Cette dernière délivre donc 4 signaux, 2 par axe. Ils portent les noms suivants :

- ✓ vertical pulse
- ✓ vertical quadrature pulse
- ✓ horizontal pulse
- ✓ horizontal quadrature pulse

Schéma du signal de la souris

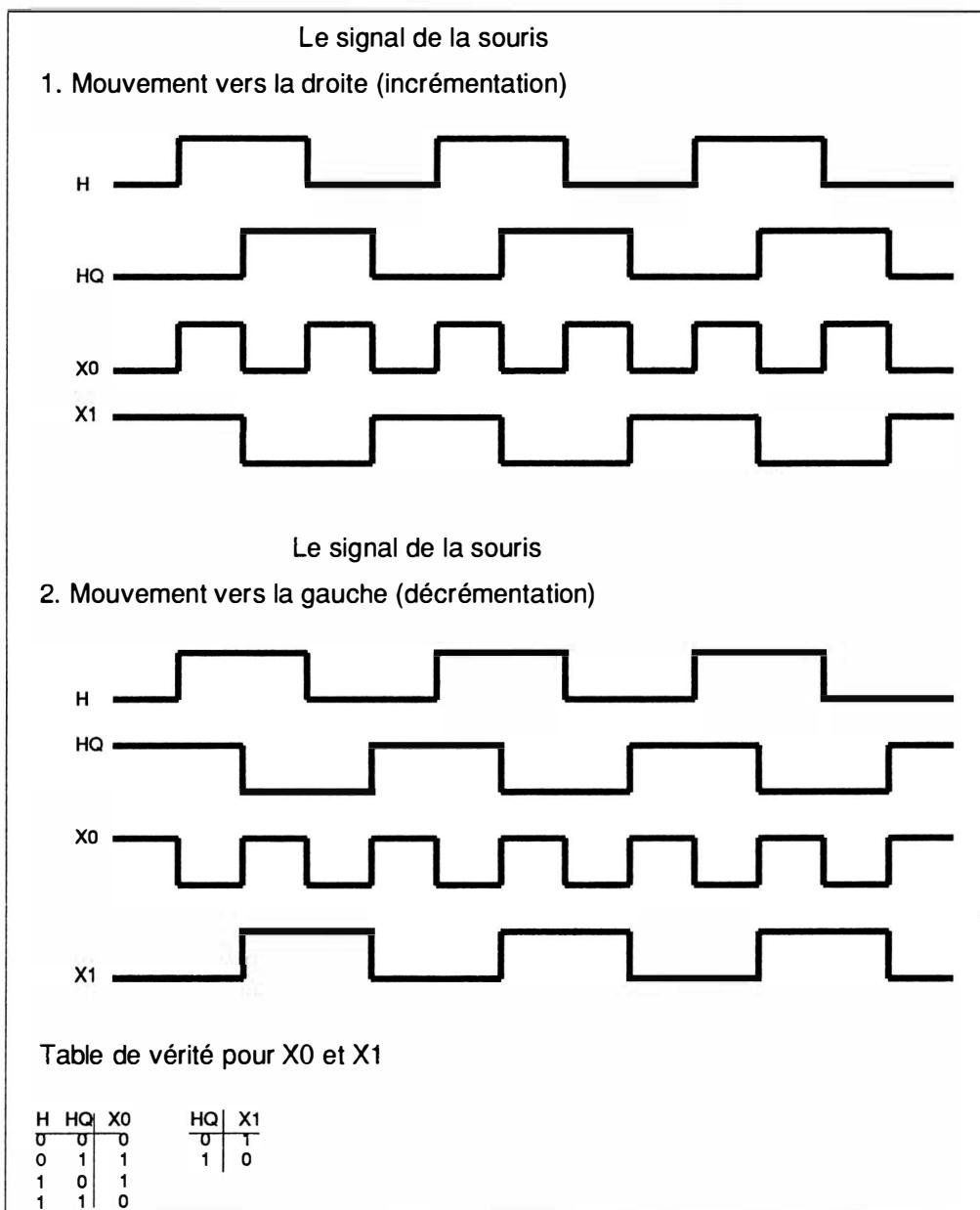


Figure 1 - 44

Le schéma montre les positions de phase des signaux horizontaux 'pulse' (H) et 'quadrature pulse' (Q) ; ceci est aussi valide pour les signaux verticaux. On remarque clairement le décalage de phase de H et HQ, pour chaque direction de mouvement. L'Amiga acquiert deux nouveaux signaux, X0 et X1, issus des impulsions H et HQ, par opérations logiques. X1 correspond à l'inverse de HQ, alors que X0 est le résultat d'un OU exclusif entre H et HQ, c'est-à-dire que X0 sera toujours à 1 lorsque les niveaux de H et HQ seront différents. Avec ces deux signaux, l'Amiga gère un compteur 6 bits, qui s'incrémentera ou se décrémentera suivant la direction de X1. L'ensemble X0, X1 correspond à une valeur 8 bits, qui représente la position actuelle de la souris.

Si la souris est déplacée vers la droite ou vers le bas, cette valeur sera incrémentée. Si la souris est déplacée vers la gauche ou vers le haut, cette valeur sera décrémentée.

Il existe 4 compteurs semblables, inclus dans DENISE, deux par gameport, où peut être branchée la souris. Ils se nomment JOYDAT0 et JOYDAT1.

JOY0DAT \$00A
(souris sur le gameport 0)

JOY1DAT \$00C
(souris sur le gameport 1)

Bit : 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Fonction : Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0 X7 X6 X5 X4 X3 X2 X1 X0

(Les deux registres ne sont accessibles qu'en lecture).

Y0-7 compteur pour les mouvements verticaux (direction Y).
X0-7 compteur pour les mouvements horizontaux (direction X).

La souris génère deux cents impulsions compteur par pouce (79 par centimètre) c'est-à-dire que la limite du compteur est vite atteinte. Un nombre 8 bits permet d'obtenir 256 valeurs. On ne peut déplacer la souris que de 4 centimètres, avant de dépasser ce chiffre de 256. Ceci est possible, aussi bien par incrémentation (saut de 0 à 255) que par décrémentation (saut de 255 à 0). C'est pour cette raison qu'il faut interroger le registre compteur à des intervalles déterminés et vérifier s'il y a eu dépassement ou non.

Le système d'exploitation prend ce test en charge, pendant l'interruption écran vide vertical.

On pourra ainsi considérer que le mouvement de la souris, entre deux tests, ne dépasse pas 127 pas du compteur. L'ordinateur compare alors le nouvel état avec le précédent. Si la différence est supérieure à 127, le compteur a un dépassement indiquant que la souris a été déplacée vers la droite (ou vers le haut). Si elle est inférieure à -127, le compteur a un dépassement, indiquant un mouvement vers la gauche (ou vers le haut).

Etat compteur précédent	Etat compteur actuel	Déférence réelle	Mouvement de souris	Dépassemant haut/bas
100	200	-100	+100	non
200	100	+100	-100	non
50	200	-150	-105	bas
200	50	+150	+105	haut

Déférence = état précédent - état actuel

S'il y a dépassement vers le bas, le mouvement réel de la souris sera calculé comme suit :

-255 - différence, ce qui donne par calcul : $-255 - (50-200) = -105$

S'il y a dépassement vers le haut, le mouvement réel de la souris sera calculé comme suit :

255 - différence, ce qui donne par calcul : $255 - (200-50) = +105$

Un mouvement positif de la souris correspond à un déplacement vers la droite (ou vers le bas), alors qu'une valeur négative correspond à un déplacement vers la gauche (ou vers le haut).

Le compteur souris peut être déterminé par le software. On pourra ainsi écrire n'importe quelle valeur dans ce dernier au moyen du registre JOYTEST.

Ce dernier influence les deux gameports au même moment, c'est-à-dire que les compteurs horizontaux et verticaux seront initialisés avec la même valeur (JOY0DAT = JOY1DAT).

JOYTEST \$036 (écriture)

Bit :	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fonction :	Y7	Y6	Y5	Y4	Y3	Y2	xx	xx	X7	X6	X5	X4	X3	X2	xx	xx

Comme on peut le remarquer, seuls 6 bits du compteur seront influencés. Ceci est logique si on se rappelle que les deux bits inférieurs proviennent directement du signal souris et non d'un registre interne se trouvant en mémoire, que l'on pourrait modifier.

Les joysticks

Lorsqu'on examine l'organisation des gameports, on s'aperçoit que les quatre directions du joystick correspondent aux quatre signaux de la souris. Les signaux du joystick seront donc traités de la même façon que ceux de la souris et au moyen des mêmes registres. Ainsi chaque paire de signaux sera combinée avec les bits X0 et X1 (ou Y0 et Y1). Les positions du joystick seront communiquées de la manière suivante :

Joystick à droite	X1=1	(bit 1 JOYxDAT)
Joystick à gauche	Y1=1	(bit 9 JOYxDAT)
Joystick vers l'arrière	X0 EOR X1 = 1	(bits 0 et 1 JOYxDAT)
Joystick vers l'avant	Y0 EOR Y1 = 1	(bits 8 et 9 JOYxDAT)

Pour établir la position du joystick soit vers l'avant, soit vers l'arrière, on doit exécuter une opération logique OU exclusif entre respectivement, Y0 et Y1, et X0 et X1.

Si le résultat correspond à 1, c'est que le joystick se trouve dans la position testée. Le programme en assembleur, ci-dessous, prend en charge le test de position sur le gameport 1 :

```
;test du bit n1
bne gauche           ;s'il est activé, le joystick est à gauche
move.w d0,d1          ;copie de d0 dans d1
lsr.w #1,d1          ;Y1 et X1 sur la position de Y0 et X0
eor.w d0,d1          ;ou exclusif : Y1 eor Y0 et X1 eor X0
btst #0,d1           ;résultat du test X1 eor X0
bne arrière          ;résultat - 1 -> joystick vers l'arrière
btst #8,d1           ;résultat du test Y1 eor Y0
bne avant             ;résultat - 1 -> joystick vers l'avant
bra milieu            ;le joystick est en position centrale
```

La liaison OU exclusif de ce programme est employée de la manière suivante :

Une copie du registre JOY1DAT est réalisée dans d1, afin de décaler le contenu d'un bit vers la droite. Ainsi X1 dans d1 et X0 dans d0 possèdent la même position de bit. Le processus est le même pour les signaux Y1 et Y0. Une opération EOR entre d0 et d1 teste immédiatement X1 avec X0 et Y1 avec Y0. Puis le résultat de ce test sera mis dans d1, pour être traité par la suite avec l'instruction BTST.

Ce programme ne prend pas en compte la position diagonale.

Les paddles

L'Amiga possède deux entrées analogiques par gameport, auxquelles on peut raccorder deux résistances variables ou potentiomètres. Ces derniers possèdent pour chaque position, une résistance déterminée que le hardware communique à PAULA. Un paddle renferme un tel potentiomètre, qui est activé grâce à un bouton de commande. Les joysticks analogiques fonctionnent de la même façon où chaque potentiomètre des directions X et Y donne la position réelle.

Deux registres renferment les 4 valeurs 8 bits des entrées analogiques, POT0DAT (gameport 0) et POT1DAT (gameport 1) :

POT0DAT \$012 et POT1DAT \$014

Bit n° :	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fonction :	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	X7	X6	X5	X4	X3	X2	X1	X0

(Ces deux registres ne sont accessibles qu'en lecture).

Etant donné qu'un ordinateur ne peut traiter que des signaux digitaux, celui-ci nécessite un circuit spécial lorsqu'il doit traiter un signal analogique. Sur l'Amiga, la détermination de la valeur de la résistance externe fonctionne de la manière suivante :

Les potentiomètres doivent avoir une valeur de résistance maximum de 470 kilo-Ohm (+/- 10%). Il est relié d'un côté avec la sortie +5 volts et de l'autre avec l'une des 4 entrées paddle du gameport. Ces dernières sont reliées aux entrées correspondantes de PAULA, et à 4 condensateurs qui se trouvent entre l'entrée et la masse.

La mesure commence au moyen d'un bit spécial de départ. Paula met alors rapidement les entrées paddle sur la masse, ce qui permet de décharger les condensateurs, tout en initialisant les registres POTxDAT. Puis, ce compteur est incrémenté d'une unité, avec chaque ligne de l'écran, pendant que les condensateurs se chargent à nouveau lentement sur les résistances. Si la tension des condensateurs dépasse une valeur déterminée, le compteur correspondant sera stoppé. C'est pour cette raison que l'état du compteur correspond exactement à la taille de la résistance (une petite résistance correspond à un état bas du compteur, une grande résistance correspond à un état haut du compteur).

Le bit de départ se trouve dans le registre POTGO :

POTGO \$034 (écriture) et POTGOR \$016 (lecture)

Bit	Nom	Fonction
15	OUTRY	gameport 1 POTY commuté en sortie
14	DATRY	bit de données gameport 1 POTY
13	OUTRX	gameport 1 POTX commuté en sortie
12	DATRX	bit de données gameport 1 POTX
11	OUTLY	gameport 0 POTY commuté en sortie
10	DATLY	bit de données gameport 0 POTY
9	OUTLX	gameport 0 POTX commuté en sortie
8	DATLX	bit de données gameport 0 POTX
7 à 1		inutilisés
0	START	déchargement des condensateurs début de la mesure.

(Un accès écriture sur POTGO initialise les deux registres POTxDAT).

Normalement, le bit START est initialisé à 1, pendant le temps mort vertical. Pendant l'affichage de l'image, les condensateurs se chargent, atteignant leur valeur seuil et stoppant le compteur. Pendant le temps mort suivant, on peut lire dans les registres POTxDAT, l'état du potentiomètre.

Les 4 entrées analogiques se laissent programmer comme des entrées/sorties normales digitales. Les bits de gestion et de données correspondants, se trouvent dans le registre POTGO, comme le bit START. Au moyen des bits OUT_{xx} (OUT_{xx} = 1), on peut commuter chaque signal sur le mode sortie. Ainsi chaque ligne sera séparée du circuit de gestion des condensateurs et la valeur du bit DAT_{xx} (POTGO) pourra y être émise.

La lecture des bits DAT_{xx} de POTGOR donnera toujours l'état actuel du signal concerné.

Si on utilise les ports analogiques comme sortie, on doit faire attention à ce qui suit :

Comme les 4 ports analogiques internes sont reliés aux condensateurs pour la mesure de la résistance (47 nF), il peut s'écouler 300 microsecondes jusqu'à ce que le signal prenne le niveau souhaité, et ceci à chaque fois que le condensateur a une inversion de charge.

Le bouton de ces outils périphériques

Chacun des trois outils étudiés plus haut possèdent un ou plusieurs boutons. Le tableau suivant montre les registres concernant le statut du bouton de la souris, du paddle ou du joystick.

Gameport 0 :	
Bouton gauche souris	CIA, port parallèle A, bit de port 6
Bouton droit souris	POTGOR, DATLY
Troisième bouton souris	POTGOR, DATLX
Bouton feu joystick	CIA-A, port parallèle A, bit de port 6
Bouton gauche paddle	JOY0DAT, bit 9 (1 = bouton activé)
Bouton droit paddle	JOY0DAT, bit 1 (1 = bouton activé)

Gameport 1 :	
Bouton gauche souris	CIA, port parallèle A, bit de port 7
Bouton droit souris	POTGOR, DATRY
Troisième bouton souris	POTGOR, DATRX
Bouton feu joystick	CIA-A, port parallèle A, bit de port 7
Bouton gauche paddle	JOY1DAT, bit 9 (1 = bouton activé)
Bouton droit paddle	JOY1DAT, bit 1 (1 = bouton activé)

(Lorsque ce n'est pas précisé, les bits sont 0-actifs, c'est-à-dire que 0 correspond à un bouton activé).

1.5.10. Le connecteur série

Comme cela a été vu au chapitre 1.3.4, l'Amiga possède un connecteur RS232 standard. Les signaux de ce boîtier peuvent être divisés en deux groupes de signaux :

1. Les signaux de données séries
2. Les signaux Handshake

Exammons d'abord le groupe 2. Le connecteur RS232 possède une grande quantité de signaux Handshake. La plupart ne seront d'ailleurs pas du tout utilisés. De plus, les contenus de ces signaux ne sont pas les mêmes sur chaque boîtier RS232. Leur fonction et programmation ont déjà été décrites dans le chapitre 1.3.4.

Sur les deux signaux du groupe 1 se déroule le transfert de toutes les informations. Le signal RXD reçoit les données et le signal TXD les émet. La communication RS232 peut donc se faire dans les deux sens au même moment, lorsque deux appareils sont reliés ensemble aux signaux RXD et TXD. On branchera ainsi le signal RXD d'un des appareils avec le signal TXD de l'autre et inversement.

Principe du transfert de données série RS232

Schéma du déroulement de transfert de données séries.

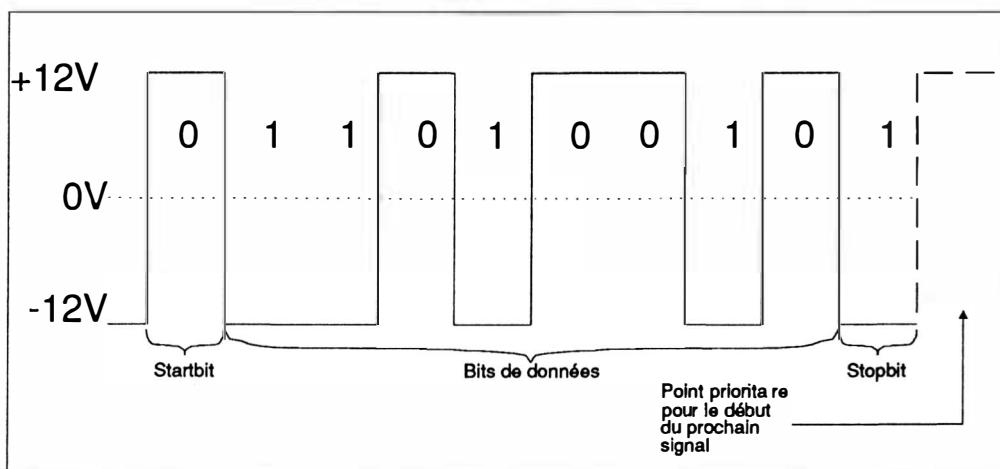


Figure 1 - 45

Etant donné que le transfert de données pour une direction n'occupe qu'un seul signal, les mots de données doivent être convertis en courant de données, le transfert s'effectuant alors bit après bit. La norme RS232 ne prévoit aucun signal d'horloge. Pour

que le destinataire sache quand le prochain bit doit être lu, le temps par bit doit être constant, c'est-à-dire que la vitesse de lecture ou d'émission de données doit être déterminée. C'est ce qu'on fait en établissant le baudrate (taux). Ce dernier détermine le nombre de bits transférés par seconde. Les taux de transfert usuels sont 300, 1200, 2400, 4800, 9600 bauds.

On n'est cependant pas obligé de s'en tenir à ces valeurs, mais il faut faire attention, lors de l'utilisation de taux peu communs, à ce que l'émetteur et le destinataire autorisent et utilisent la même vitesse de transmission.

Pour que le transfert fonctionne, le destinataire doit encore connaître le moment où un octet débute et celui où il se finit. Le schéma 1-45 montre le timing du transfert d'un octet sur les signaux de données. Chaque octet commence avec un bit de départ, qui ne se différencie en rien des bits de données, si ce n'est qu'il est toujours à 0. Puis suivent les bits dans l'ordre croissant, c'est-à-dire, du LSB au MSB. La fin est marquée par un ou deux bits d'arrêt. Le destinataire reconnaît l'alternance des octets par le changement de niveau de 1 à 0, qui résulte de la suite bit d'arrêt, bit de départ.

Le circuit qui exécute le transfert série, se nomme Universal Asynchronous Receive Transmit (UART). Il est inclus dans PAULA et ses registres se trouvent dans la zone registre du circuit spécialisé :

Les registres UART

SERPER \$032 (écriture)

Bit	Nom	Fonction
15	LONG	La longueur des données reçues est de 9 bits
0 à 14	RATE	Ce nombre 15 bits correspond au Baudrate (taux)

SERDAT \$030 (écriture)

Ce registre contient les données émises.

SERDATR \$018 (écriture)

Bit	Nom	Fonction
15	OVRUN	Dépassement du registre à décalage de destination
14	RBF	Le tampon des données reçues est plein
13	TBE	Le tampon des données émises est vide

Bit	Nom	Fonction
12	TSRE	Le registre à décalage d'émission est vide
11	RXD	Correspond au niveau du signal RXD
10	---	inutilisé
9	STP	Bit d'arrêt
8	STP ou DB8	Dépend de la longueur des données
7	DB7	Le tampon des données reçues/ bit de données n°7
6	DB6	Le tampon des données reçues/ bit de données n°6
5	DB5	Le tampon des données reçues/ bit de données n°5
4	DB4	Le tampon des données reçues/ bit de données n°4
3	DB3	Le tampon des données reçues/ bit de données n°3
2	DB2	Le tampon des données reçues/ bit de données n°2
1	DB1	Le tampon des données reçues/ bit de données n°1
0	DB0	Le tampon des données reçues/ bit de données n°0

Un bit du registre ADKCON appartient aussi à la gestion UART :

ADKCON \$09E (écriture) ADKCONR \$010 (lecture)

Bit n°11 : UARTBRK

Ce bit stoppe la sortie série et met TXD à 0.

Le déroulement du transfert de données UART de l'Amiga

La réception

La réception de données séries se déroule en deux stades. Les bits, arrivant sur la broche RXD, seront pris en charge par le registre à décalage à la vitesse déterminée par le baudrate et reconstitués pour données un mot de données parallèle. Si le registre à décalage est plein, son contenu sera écrit dans le tampon de données reçues. Il sera ainsi libre de suite pour les prochaines données.

Le processeur peut lire ce tampon, mais non le registre à décalage. Les bits de données correspondants, DB0-7, se trouvent dans le registre SERDATR.

L'Amiga peut réceptionner 8 ou 9 bits de données. L'UART est commuté en mode 9 bits, au moyen du bit LONG du registre SERPER (LONG = 1).

La longueur fixée des données détermine le format utilisé dans le registre SERDATR. Pour 9 bits, le bit 8 de SERDAT contient le 9ème bit de données, alors que le bit d'arrêt correspond au bit 9. Pour 8 bits de données, le bit 8 contient déjà le bit d'arrêt, le bit 9 étant réservé au deuxième bit d'arrêt.

L'état du registre à décalage et du tampon de données est donné par deux bits présents dans SERDATR :

RBF correspond au signal 'receive-buffer-full', c'est-à-dire que dès qu'un mot de données, issu du registre à décalage, est transféré dans le buffer, ce bit se met à 1, signalisant ainsi au microprocesseur qu'il doit lire les données dans SERDATR.

Ce bit existe aussi dans les registres d'interruption (RBF, INTREQ/INTEN bit n 11). Après que le processeur ait lu les données, il doit remettre RBF à 0 dans INTREQ. Ceux des registres INTREQ et SERDATR seront aussi remis alors à 0.

```
move.w #$0800,$DFF000+INTREQ ;désactiver RBF dans INTREQ et SERDATR
```

Si on s'abstient, le registre à décalage réceptionnera un nouveau mot de données complet, l'UART initialisant le bit OVRUN. Ceci signale qu'aucune donnée suivante ne peut être prise en compte, étant donné que le buffer (RBF=1) et le registre à décalage (OVRUN=1) sont pleins. OVRUN sera mis à 0 lorsque RBF sera désactivé. Puis RBF se remettra à 1, étant donné que le contenu du registre à décalage sera aussitôt transféré dans le buffer.

L'émission

Le processus d'émission se déroule aussi suivant deux phases. Le tampon d'émission de données se trouve dans le registre SERDAT. Dès qu'un mot de données y est inscrit, il sera transféré dans le registre à décalage de sortie. C'est ce que signale le bit TBE. TBE correspond à Transmit Buffer Empty et indique que SERDAT est prêt à prendre en compte les prochaines données. TBE est aussi présent dans les registres d'interruption (TBE, INTREQ/INTEN bit n0). Comme RBF, TBE sera désactivé par le biais du registre INTREQ.

Dès que le registre à décalage a émis le mot de données, un nouveau mot sera prélevé automatiquement dans le buffer d'émission de données. Si ceci n'est pas possible, l'UART met le bit TSRE à 1 (Transmit-Shift-Register-Empty). Ce bit sera remis à 0 par la désactivation du bit TBE.

La longueur des mots de données et le nombre de bits d'arrêt déterminent le format des données dans SERDAT. On écrit simplement le mot de données sur les 8 ou 9 bits inférieurs de SERDAT, ces derniers étant suivis d'un ou deux bits d'arrêt (bit(s) mis à 1). Un mot de données 8 bits avec deux bits d'arrêt sera de la forme suivante :

Bit	:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fonction	:	0	0	0	0	0	0	1	1	D7	D6	D5	D4	D3	D2	D1	D0

D0-7 sont les 8 bits de données.

Les deux 1 correspondent aux deux bits d'arrêt.

Un mot de données 9 bits avec un bit d'arrêt sera de la forme suivante :

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fonction	0	0	0	0	0	0	1	D8	D7	D6	D5	D4	D3	D2	D1	D0

Un mot de données 8 bits avec un bit d'arrêt sera de la forme suivante :

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fonction	0	0	0	0	0	0	0	1	D7	D6	D5	D4	D3	D2	D1	D0

Le bit LONG du registre SERPER a juste une influence sur la longueur des données réceptionnées. Le format des données émises sera déterminé seul avec la valeur présente dans le registre SERDAT.

Détermination de la vitesse en bauds

Le taux d'émission ou de réception des données doit être écrit dans les 15 bits inférieurs du registre SERPER. On ne peut malheureusement pas établir directement ce taux, mais on doit choisir le nombre de cycles de bus présents entre deux bits (1 cycle de bus a une durée de 2.79365×10^{-7} secondes). Si tous les n cycles de bus, une donnée doit être émise, on devra écrire dans le registre SERPER la valeur n-1. Avec la formule suivante, on pourra transformer le taux en valeur utilisable :

$$\text{SERPER} = (1 / (\text{vitesse} * 2.79365 * 10^{-7})) - 1$$

Par exemple pour un taux de 4800 baud :

$$\text{SERPER} = (1 / (4800 * 2.79365 * 10^{-7})) - 1 = 744.74$$

La valeur calculée sera ensuite arrondie et écrite dans le registre SERPER :

```
move.w #745,$DFF000+SERPER ;SERPER activé, LONG=0
```

ou

```
move.w #$8000+745,$DFF000+SERPER ;LONG=1
```

1.5.11. Le contrôleur de disquette

La gestion Hardware du lecteur de disquette se divise en deux parties.

La première correspond aux signaux de gestion, qui activent l'unité, le moteur, qui bougent la tête de lecture/écriture, etc...

Ils sont tous reliés aux signaux correspondants de port du CIA. Leur fonction et initialisation ont déjà été décrites au chapitre 1.3.5.

Ce qui sera expliqué dans ce chapitre concerne essentiellement les signaux de données. C'est grâce à eux que les données seront enregistrées sur disquette ou chargées à partir

de cette dernière. C'est un circuit spécial inclus dans PAULA, le contrôleur disquette, qui se charge du traitement de ces données.

Il possède un canal DMA qui se charge de lire ou d'écrire les données disquette.

La programmation du DMA disquette

Avant d'initialiser un accès DMA disquette, on doit vérifier que le précédent est terminé. Si on arrête le déroulement d'un accès écriture, on peut détruire les données de la piste correspondante. Soyez donc attentif à ce que le dernier accès DMA soit arrêté.

La première chose à faire est de déterminer l'adresse mémoire du buffer de données. Le DMA disquette utilise une des paires de registres adresses suivantes, DSKPTH et DSKPTL, comme pointeur sur la CHIP-RAM.

\$20 DSKPTH	pointeur de données disquette (bits 16-18)
\$22 DSKPTL	pointeur de données disquette (bits 0-15)

Puis il faut initialiser le registre DSKLEN. Ce dernier est organisé de la manière suivante :

DSKLEN \$024 (écriture)

Bit	Nom	Fonction
15	DMAEN	Initialisation des accès DMA disquette
14	WRITE	Ecriture des données sur la disquette
0-13	LENGTH	Nombre de mots de données à transférer

LENGTH

Les 14 bits inférieurs du registre DSKLEN renferment le nombre de mots de données à transférer.

WRITE

Au moyen de WRITE (WRITE = 1), on commute le contrôleur disquette du mode lecture en mode écriture.

DMAEN

Lorsqu'on met le bit DMAEN à 1, le transfert de données commence. On doit cependant faire attention aux faits suivants :

1. Le bit Disk-DMA-Enable du registre DMACON (DSKEN, bit n4) doit toujours être activé.

2. Pour éviter un accès écriture, initialisé par mégarde, on doit activer deux fois, et à la suite, le bit DMAEN. Puis l'accès DMA pourra débuter. Le bit WRITE, pour plus de sécurité, ne doit être activé que lors d'un accès écriture. Une séquence d'initialisation ordonnée pour un accès DMA disquette sera de la forme suivante :
 1. Désactiver DMAEN en mettant DSKLEN à 0.
 2. Tant que DSKEN du registre DMACON n'est pas activé, on doit exécuter ce qui suit à sa place.
 3. Mettre l'adresse souhaitée dans DSKPTH et DSKPTL.
 4. Mettre la bonne valeur (incluant LENGTH, WRITE et DMAEN) dans DSKLEN.
 5. Réécrire la même valeur dans DSKLEN.
 6. Attendre que les accès DMA disquette soit terminés.
 7. Par mesure de sécurité, remettre DSKLEN à 0.

Afin que le processeur sache à quel moment le contrôleur disquette aura terminé le transfert des mots de données (dont le nombre est déterminé par LENGTH), il y aura interruption DSKBLK (Disk BlocK Finished, bit n°1 dans INTREQ/INTEN). Elle sera libérée après lecture ou écriture du dernier mot de données.

Le statut actuel du contrôleur disquette se trouve dans le registre DSKBYTR:

DSKBYTR \$01A (lecture)

Bit	Nom	Fonction
15	BYTEREADY	Ce bit signale que l'octet de données est valide pour les 8 bits inférieurs.
14	DMAON	DMAON indique si l'accès DMA disquette est initialisé. Dès que DMAON = 1, DMAEN dans DSKLEN et DSKEN dans DMACON doivent être initialisés.
13	DSKWRITE	Donne l'état de WRITE du registre DSKLEN.
12	WORDEQUAL	Données disquettes = DSKSYNC.
11 à 8		Inutilisés.
7 à 0	DATA	Octets de données actuels de la disquette.

Au moyen des 8 bits DATA et du drapeau BYTEREADY, on peut lire les données sur la disquette par le processeur. A chaque fois qu'un octet données complet est arrivé à destination, le contrôleur disquette active le bit BYTEREADY. Ainsi, le processeur sait que l'octet présent dans les 8 bits DATA est valide. Après lecture du registre DSKBYTR, le bit BYTEREADY sera désactivé.

Il peut arriver qu'on ne veuille pas lire une piste entière. On aura alors la possibilité de démarrer les accès DMA à une position déterminée. On écrira, à ce moment là, le mot de données où le contrôleur disquette doit commencer, dans le registre DSKSYNC.

DSKSYNC \$07E (écriture)

Le registre DSKSYNC contient le mot de données, où le transfert doit débuter.

Après initialisation des accès DMA disquette, le contrôleur disquette commence à lire les données sur la disquette, mais sans les écrire en mémoire. A la place, il les compare avec le mot de données présent dans DSKSYNC. Lorsque les deux concordent, il commencera un transfert normal. On pourra ainsi programmer le contrôleur disquette de telle façon, qu'il soit obligé d'attendre la marque de synchronisation au début d'un bloc de données.

Le bit WORDEQUAL du registre DSKBYTR sera mis à 1 dès que la donnée lue et celle présente dans DSKSYNC concordent. Etant donné que cette concordance ne dure que 2 microsecondes, le bit WORDEQUAL ne sera activé que pendant un temps sensiblement égal. De plus, une interruption sera générée chaque fois que WORDEQUAL sera à 1.

Le bit n° 12 du registre INTREQ (respectivement INTEN) correspond au bit d'interruption DSKSYNC. Il sera activé lorsque les données disques concorderont avec DSKSYNC.

Paramètres d'exploitation

Les données ne peuvent pas être écrites sur disquette, avec le format utilisé en mémoire. Elles doivent être décodées au préalable. Normalement, l'Amiga travaille avec le code MFM, mais il peut aussi utiliser le code GCR. Deux opérations sont nécessaires pour sélectionner le code souhaité :

- ① Une routine doit coder les données avant l'écriture sur disquette, et décoder les données lues.
- ② Le contrôleur disquette doit mettre au point le codage correspondant. C'est ce que permet le registre ADKCON.

ADKCON \$09E (écriture) ADKCONR \$010 (lecture)

Bit	Nom	Fonction
15	SET/CLR	Bits activés (SET/CLR=1) ou désactivés
14-13	PRECOMP	Ces bits contiennent la valeur PRECOMP : Bit 14 Bit 13 Temps PRECOMP 0 0 nul 0 1 140 nanosecondes 1 0 280 nanosecondes 1 1 560 nanosecondes
12	MFMPREC	GCR = 0, MFM = 1
11	UARTBRK	Cf. UART
10	WORDSYNC	WORDSYNC = 1 active la synchronisation GCR du contrôleur disquette sur le mot présent dans le registre DSKSYNC.
9	MSBSYNC	MSBSYNC = 1 active la synchronisation MSB
8	FAST	Taux d'horloge du contrôleur disquette FAST = 1: 2 microsecondes/bit (MFM) FAST = 0: 4 microsecondes/bit (GCR)
7-0		Cf. chapitre 1.5.8

Le registre de données du contrôleur de disquette

Le contrôleur DMA transfère les données de la mémoire vers un registre données correspondant. Le contrôleur disquette possède un registre données pour celles issues de la disquette et un pour celles écrites sur disquette.

DSKDAT \$026 (écriture)

Ce registre renferme les données qui doivent être écrites sur disquette.

DSKDATR \$008 (lecture)

Ce registre renferme les données issues de la disquette. C'est un registre du type Early Read Register, qui ne peut pas être accessible au processeur.

1.6. L'architecture d'extension

Si vous avez déjà utilisé une carte d'extension dans l'Amiga, vous aurez sans doute remarqué qu'on n'a plus besoin de s'occuper de quoi que ce soit avec ces cartes. Il suffit de les monter et de démarrer. Une extension RAM est ainsi ajoutée automatiquement à la mémoire libre d'ensemble, dès que vous avez allumé la machine et booté le système. Si l'on introduit une autre carte, on ne rencontre pas de problème particulier, même si cette seconde carte provient d'un autre constructeur. On n'est pas obligé, comme sur les autres ordinateurs, de régler d'abord un grand nombre d'interrupteurs DIP ou de jumpers, afin que les deux cartes ne se trouvent pas à la même adresse.

C'est la même chose avec les logiciels. Seuls sont chargés les drivers pour lesquels le matériel correspondant est effectivement monté dans l'appareil. Pour permettre tout ceci, Commodore a développé une structure matériel-logiciel qui soutient cette "auto-configuration" des cartes d'extension depuis le Kickstart 1.2.

1.6.1. Le hardware

Le problème principal de toutes les cartes d'extension concerne le secteur d'adresse dont elles ont besoin. On entend par là les adresses auxquelles la carte doit être adressée par les logiciels. Sur l'Amiga, le secteur prévu pour les cartes d'extension va de \$200000 à \$9FFFFF. Mais comme une carte d'extension ne sait pas quelles sont les autres extensions qui se trouvent déjà dans ce secteur d'adresses, on risque d'aboutir à des collisions. Supposons que vous ayez une extension de RAM de 2 Mo à l'adresse \$200000. On veut maintenant introduire une seconde carte, et étendre la mémoire à 4 Mo. Si l'on introduit pour cela à nouveau la même carte, cela ne sert évidemment à rien, car celle-ci occupera à nouveau le secteur qui commence en \$200000. Il faut donc modifier l'adresse de l'une des cartes, par exemple en la reportant à \$400000. Dans ce cas, elle fonctionnera. C'est également ainsi que l'on fait sur la plupart des ordinateurs. Les cartes d'extension ont de petits interrupteurs sur la carte-mère, à l'aide desquels on peut régler l'adresse de base. Malheur à celui qui a égaré la documentation concernant la carte, et en est réduit à se demander désespérément quelle était la fonction des différents interrupteurs !

Avec l'Amiga, au contraire, chaque carte possède des éléments matériels permettant à l'ordinateur de savoir quelle place est exigée en mémoire par la carte. Si cette place est encore disponible, les éléments matériels sont en mesure d'affecter une adresse correspondante à la carte. Si celle-ci est une extension de RAM, cette mémoire sera ajoutée à la liste des secteurs libres en mémoire. En outre, un logiciel pilote, par exemple un driver de disque dur peut être lu automatiquement par l'une des ROM qui se trouvent sur la carte d'extension.

Le système d'exploitation de l'Amiga lit toujours les informations des cartes d'extension à partir de l'adresse \$E80000. Comment cela fonctionne-t-il si plusieurs cartes sont introduites en même temps ?

Chaque carte d'extension possède une broche CFGIN et une broche CFGOUT (Configuration IN et Configuration OUT, cf. la description des slots du A2000). Après un reset, toutes les cartes placent leur sortie CFGOUT sur high. Puisque la sortie CFGOUT de chaque carte est liée à l'entrée CFGIN de la suivante, on a sur toutes les cartes DFGIN = high (les slots vides sur A2000 sont sautés automatiquement, et l'on n'est pas obligé d'introduire les cartes dans l'ordre pour que l'auto-configuration fonctionne). La seule exception est la première carte, montée dans le A1000 ou le A500. Or pour toutes les cartes, on a la règle suivante : elle ne peut répondre à un accès à l'adresse \$E80000 que si son entrée CFGIN est sur low et sa sortie CFGOUT est sur high. Il s'ensuit qu'après un reset, c'est d'abord la carte introduite dans le premier slot qui répond (ou encore celle qui se trouve reliée au connecteur d'extension sur A2000, ou la carte reliée directement au port d'extension sur A500/A1000). Lorsque l'Amiga a lu sur ces cartes toutes les données qui lui sont nécessaires, il vérifie si la place en mémoire réclamée existe. Si c'est le cas, il écrit le début de son adresse dans un registre spécial sur la carte d'extension. De cette façon, le processus d'auto-configuration est terminé, la sortie CFGOUT est placée sur low, et il permet ainsi à la carte suivante de commencer son propre processus d'auto-configuration.

L'adresse que l'Amiga affecte à une carte se trouve toujours à une limite de la mémoire, en fonction de la taille de la mémoire exigée. Une extension de RAM de 2 Mo reçoit donc une adresse qui se trouve à un multiple de 2 Mo: \$200000, \$400000, \$600000 ou \$80000. Les seules exceptions sont les cartes de 4 et 8 Mo qui, du fait de leur taille, ne pourraient pas être accueillies autrement.

On peut maintenant atteindre la carte ainsi configurée sous l'adresse qui lui est affectée. Si la place en mémoire qu'elle exige n'est pas libre, ce qui n'arrive néanmoins que rarement avec un espace libre d'adresses supérieur à 8 Mo, elle est réduite au silence par l'Amiga, par l'intermédiaire d'une autre adresse, nommée *Shut_up_forever*, et sa sortie CFGOUT est placée sur low, pour que l'auto-configuration puisse se poursuivre. Lorsqu'aucune carte ne répond plus à \$E80000, l'Amiga sait qu'il a saisi toutes les cartes d'extension et que le processus d'auto-configuration est donc terminé

La structure de l'auto-configuration

Nous avons dit que l'auto-configuration commence à \$E80000. Pour ne pas surcharger le coût de la carte d'extension, on utilise seulement quatre bits de données : D12-D15. Chaque octet de données est donc divisé suivant les deux nibbles supérieurs de deux mots successifs :

Adresse	D12-D15	D11-D0
\$E80000	Nibble supérieur de l'octet	Non occupé
\$E80002	Nibble inférieur de l'octet	Non occupé

Les adresses sont distribuées de la façon suivante (l'adresse de base durant l'auto-configuration est \$E80000, ensuite c'est la carte qui décide si ces données sont accessibles à la nouvelle adresse ou non) :

Adresses	Fonction
00/02	Bits 0-2 : Taille du secteur souhaité en mémoire :
	000 - 8 Mo
	001 - 64 Ko
	010 - 128 Ko
	011 - 256 Ko
	100 - 512 Ko
	101 - 1 Mo
	110 - 2 Mo
	111 - 4 Mo
	Bit 3 : un 1 dans ce bit signifie que la carte d'extension suivante se trouve sur le même support, donc dans le même slot.
	Bit 4 : ROM valide sur le support
	Bit 5 : Ajouter un secteur en mémoire à la liste des secteurs libres (extensions RAM)
	Bits 6 et 7 : Type de la carte d'extension :
	00, 01 et 10 pour les cartes à venir
	11 : carte normale
04/06	Numéro de produit : avec cet octet, un constructeur peut numérotter ses cartes d'extension successives, pour qu'elles puissent être reconnues par les logiciels.
08/0A	Bits 0-5 : toujours 0
	Bit 6 : un 0 signifie que cette carte peut être désactivée (Shut_up_forever)
	Bit 7 : Si ce bit est sur 0, l'adresse est indifférente pour la carte; s'il est sur 1, elle veut se trouver dans le secteur allant de \$200000 à \$FFFFFF.
0C/0E	Toujours nul
10/12	Numéro du constructeur, octet supérieur (High Byte)
14/16	Numéro du constructeur, octet inférieur (Low Byte). Le numéro du constructeur est un nombre de 16 bits, que chaque constructeur de cartes d'extension pour l'Amiga peut demander pour que les logiciels puissent faire la différence entre plusieurs cartes (à l'aide de leurs numéros de produit)
18/1A	Numéro de série de la carte, octet 0 (LSB)

1C/1E	Numéro de série de la carte, octet 1
20/22	Numéro de série de la carte, octet 3
24/26	Numéro de série de la carte, octet 4 (MSB). Le numéro de série n'est pas nécessairement utilisé, il concerne seulement le constructeur.
28/2A	Adresse de la ROM, octet supérieur
2C/2E	Adresse de la ROM, octet inférieur. L'adresse de la ROM est celle de la ROM en relation avec l'adresse initiale du circuit. En même temps que cette adresse, on place le bit 4 dans le premier octet (00/02) sur 1, pour faire savoir à l'Amiga que l'adresse de la ROM est valide. S'il n'y a pas de ROM, ces deux octets n'ont pas de signification.
30/32	Accès à l'écriture toujours 0; avec un accès en écriture, on peut effacer l'adresse de base déposée sur la carte.
34/36	Toujours 0
38/3A	Toujours 0
3C/3E	Toujours 0
40/42	Registre de contrôle/d'état (au choix) <ul style="list-style-type: none"> Bit 0 : Autorise l'interruption (interrupt enable) Bit 1 : Non défini, dépend de la carte Bit 2 : Reset de la carte d'extension Bit 3 : Non défini Bits 4-7 non définis en écriture, mais ont les fonctions suivantes à la lecture : Bit 4 : INT2 posé Bit 5 : INT6 posé Bit 6 : INT7 posé Bit 7 : La carte déclenche une interruption
44/46	Toujours 0
48/4A	Adresse de base (bits d'adresse de A23 à A16). A cette adresse, l'Amiga écrit l'adresse de base de la carte d'extension.
4C/4E	shut_up_forever (désactive la carte)
50/52	
à	
7C/7E	Toujours 0

En fonction du hardware utilisé, il faut encore intervertir toutes les valeurs lues, sauf celles qui se trouvent aux adresses 00/02 et 40/42. Les octets désignés par "Toujours 0" sont lus comme \$FF au lieu de 0.

Comment ce schéma est-il alors réalisé sur la carte d'extension au niveau du hardware ? La base de l'ensemble n'est pas une ROM, comme on pourrait le penser, mais un élément PAL. Celui-ci prend en charge simultanément deux fonctions : il contrôle en premier lieu la configuration du circuit, en gérant entre autres les signaux CFGIN et CFGOUT ; en second lieu, il joue le rôle d'une ROM, en emmagasinant les données de la configuration comme par exemple la taille souhaitée en mémoire. Les autres sorties du PAL restent ainsi libres. L'utilisation d'un PAL est également responsable du fait que la plupart des données doivent être intervertis, c'est-à-dire lues comme 1 à la place de 0.

En effet, un PAL se programme précisément à l'inverse d'une ROM. Au lieu d'indiquer un motif de bits à chaque adresse, on définit la combinaison des données d'entrée (donc les adresses, R/W, CFGIN, etc.) pour laquelle un signal de sortie déterminé devra être placé sur 0. Dans le cas normal, les signaux d'entrées sont donc placés sur high. Si l'on considère alors le tableau d'adresses ci-dessus, on constate qu'il possède beaucoup plus de bits de données, qui sont Hi (=1) (ne pas oublier d'intervertir !) que de bits de données sur 0. Cela simplifie la programmation du PAL. L'utilisation d'un PAL limite malheureusement les possibilités de bricolage, car n'importe quel bricoleur d'ordinateurs ne possède pas chez lui un dispositif de programmation PAL. En revanche, les prix des éléments PAL sont tombés entre-temps sous la barre des 50 F.

Comment fonctionne maintenant le codage des adresses ? Chaque carte d'extension possède un élément de mémoire de 8 bits, avec sorties désactivables, par exemple du type 74LS374. Dans cet élément est déposée l'adresse que l'Amiga affecte à la carte (48/4A dans le tableau ci-dessus). Cette adresse est déposée au début de ce qu'on appelle un comparateur d'adresses. Cette puce, par exemple du type 74F688, compare l'adresse sur le bus d'adresse (A23 à A16, si la carte exige 64 Ko de mémoire) avec l'adresse qui se trouve dans le 74LS374. Si les deux adresses sont identiques, le comparateur le remarque, et il signale à la carte que le processeur s'adresse à elle. Les sorties du 74LS374 ne sont toutefois activées que si la carte est configurée. Pour que la carte se trouve dès l'abord à une adresse définie, on dispose en outre de 8 résistances parallèle aux sorties, actives en partie sur +5 volt ou sur la masse, et de telle façon que l'on obtienne l'adresse \$E80000. La carte se trouve donc avant la configuration à \$E80000, et après la configuration à l'adresse écrite par l'Amiga dans le 74SL374 : elle est auto-configurée !

La structure de la ROM

Comme on peut s'en douter d'après le tableau qui précède, on a la possibilité de déposer un logiciel quelconque dans une ROM sur la carte d'extension. Le contenu de cette ROM est copié automatiquement dans la RAM lorsque le 4ème bit est posé dans 00/02. L'Amiga commence alors à regarder à l'adresse qu'il trouve dans 28/2A et 2C/2E, à la recherche d'une ROM. Le constructeur de la carte a le loisir de définir la largeur de

cette ROM sur quatre, sur huit ou sur seize bits, son contenu étant reporté dans tous les cas correctement à l'intérieur de l'ordinateur. Les bits de données utilisés sont soit D12-D15, D8-D15, soit les 16 bits ensemble. La RAM doit avoir le contenu suivant (en mots) :

Adresse	Fonction
0	Bits 15 et 14 définissent la largeur de la ROM :
	00 - Quatre bits de large
	01 - Huit bits de large
	11 - Seize bits de large
	Bits 13 et 12 définissent le moment où le code de bootage existant (éventuellement) dans la ROM est appelé :
	00 - Jamais
	01 - Au moment où l'on installe la carte
	10 - A l'exécution de BindDrivers

Attention : Lorsque le code de bootage doit être appelé, le secteur de configuration de la carte doit être encore accessible après la configuration.

Les autres bits sont pour le moment tous sur 0.

Adresse	Fonction
2	Taille de la ROM en octets
4	Adresse du programme qui doit être appelé lors de la configuration (comme offset à partir du début de la ROM)
6	Adresse du programme qui doit être appelé pour le bootage (cf. ci-dessus)
8	Offset sur le nom de la carte, du constructeur ou autre, ou alors un 0
A	Toujours 0
C	Toujours 0

A partir de là peut commencer le contenu de la ROM, par exemple un driver de disque dur.

Lors de l'appel de l'une des deux adresses, voici les contenus de registre transmis par l'Amiga :

Adresse	Fonction
A7	pointeur de pile d'au moins 2 Ko
A6	ExecBase
A5	ExpansionBase
A3	Structure ConfigDev
A2	Adresse du contenu de la ROM dans la RAM (où il a été copié)
A0	Adresse de base de la carte après la configuration

Adresse	Valeur en retour
D0	Lors du renvoi de la valeur 0 en D0, la mémoire dans laquelle l'Amiga a copié la ROM est de nouveau ajoutée à la mémoire système libre.

1.6.2. La partie logicielle

Pour tout ce qui touche aux cartes d'extension sur l'Amiga, le responsable est une bibliothèque - comment pourrait-il en être autrement ? Il s'agit de la bibliothèque Expansion. Elle est appelée par Exec immédiatement après un reset, et configure ensuite toutes les cartes d'extension qu'elle trouve, leur affecte des adresses de base, et copie le contenu de la ROM éventuelle dans la RAM. Normalement, le programmeur n'intervient pas dans le travail de cette bibliothèque. Même le concepteur d'une carte d'extension n'a pas à se préoccuper de la façon dont sa carte sera reconnue et configurée, le principal est que cela fonctionne.

A une exception près : la bibliothèque Expansion génère quelques données qui sont en relation étroite avec la carte : son adresse de base, le fait que la configuration ait été réalisée correctement ou non, etc. Ces données sont nécessaires à la partie logicielle qui doit gérer ces cartes. Comment pourrait-elle accéder à la carte si elle ne connaît pas son adresse de base ?

La méthode normale pour installer un driver destiné à une carte d'extension utilise la commande "binddrivers" de CLI. Cette commande, appelée normalement dans la startup-sequence, examine le répertoire Expansion pour y retrouver des drivers qui conviennent à la carte reconnue lors de l'autoconfiguration. Ainsi, les seuls drivers installés sont ceux dont le hardware correspondant existe effectivement. Pour qu'un driver soit reconnaissable par binddrivers, il faut générer pour lui un fichier .info. Dans celui-ci, on reporte "PRODUCT" dans le champ Tooltypes. Lorsque binddrivers trouve ce mot, il lit dans le reste de la ligne le numéro du constructeur et celui du produit, pour reconnaître la carte d'extension à laquelle est destiné le driver.

La syntaxe est ici la suivante. D'abord vient "PRODUCT", suivi d'un "=" . Ensuite vient le numéro du constructeur, puis un "/", suivi du numéro de produit. Si un driver doit

travailler avec tous les produits d'un constructeur, on omet simplement le “/” et les numéros de produit. S'il doit fonctionner uniquement avec certains drivers, on fait suivre la première indication simplement par “”, et l'on entre alors la seconde indication. Voici l'aspect que peut présenter la commande :

```
PRODUCT-100/01 :Constructeur 100, numéro de produit 01
PRODUCT-100      :Constructeur 100, toutes les cartes
PRODUCT-100/01|100/02 :Constructeur 100, P. 01 et 02
```

Toutes les indications doivent étre sans espacement !

Binddrivers charge le driver (au moyen de LoadSeg), lorsqu'il a trouvé au moins une carte qui convient, et il appelle ce driver par l'intermédiaire de “InitResident”. Si celui-ci revient avec ZERO, le driver chargé est à nouveau supprimé de la mémoire (UnLoadSeg).

Le driver chargé doit d'abord ouvrir la bibliothèque Expansion. On obtient en retour l'adresse de la structure ExpansionBase. Celle-ci se présente sous la forme suivante :

```
struct ExpansionBase
{
    struct Library LibNode;
    UBYTE Flags;
    UBYTE pad;
    APTR ExecBase;
    APTR SegList;
    struct CurrentBinding CurrentBinding;
    struct List BoardList;
    struct List MountList;
    UBYTE AllocTable[TOTALSLOTS];
    struct SignalSemaphore BindSemaphore;
    struct Interrupt Int2List;
    struct Interrupt Int6List;
    struct Interrupt Int7List;
}; (que l'on trouve dans "Expansion.h")
```

On n'a besoin ici que de la structure CurrentBinding:

```
Struct CurrentBinding
{
    struct ConfigDev *cb_ConfigDev;
    UBYTE           *cb_FileName;
    UBYTE           *cb_ProductString;
    UBYTE           **cb_ToolTypes;
};(que l'on trouve dans "Configvars.h")
```

C'est là qu'on trouve le nom de fichier du driver chargé (cb_FileName), le texte “PRODUCT etc.” pris dans le fichier .info (cb_ProductString), et un pointeur sur le champ Tooltypes (cb_ToolTypes). Mais ce qui importe le plus pour le driver, ce sont les structures ConfigDev. La structure CurrentBinding contient un pointeur sur la première de ces structures. Chaque structure ConfigDev correspond à une carte d'extension dont les caractéristiques sont fournies après PRODUCT. Elle a la forme suivante (également dans “Configvars.h”):

```

struct ConfigDev
{
    struct Node          cd_Node;
    UBYTE                cd_Flags;
    UBYTE                cd_Pad;
    struct ExpansionRom cd_Rom;
    APTR                 cd_BoardAddr;
    APTR                 cd_BoardSize;
    WORD                 cd_SlotAddr;
    WORD                 cd_SlotSize;
    APTR                 cd_Driver;
    struct ConfigDev    *cd_NextCD;
    ULONG                cd_Unused[4];
};


```

Ici le driver trouve enfin tout ce dont il a besoin. L'adresse effective de la carte figure dans cd_BoardAddr. Le fait que le processus d'auto-configuration a été réalisé avec succès ou non figure dans cd_Flags: Bit 0 s'appelle CDF_SHUTUP. S'il est égal à 1, la carte a été réduite au silence avec Shut_up_forever. Le driver fait de même lorsqu'il trouve CDF_SHUTUP posé. Bit 1, c'est-à-dire CDF_CONFIGME, indique que cette carte ne possède pas encore de driver. Ce bit doit être supprimé par le driver lorsque ce dernier a été appelé par "binddrivers". En outre, il faut reporter dans cd_Driver l'adresse du "node" du driver. Le "node" peut par exemple être un Device-Node, un Library-Node ou un Resource-Node, selon le genre de driver.

La structure cd_Rom correspond d'ailleurs au tableau d'adresses pour les cartes d'extension, donné en début de section, à condition de ne prendre en considération que le secteur allant de 00/02 à 3C/3E. Au lieu d'être déposée dans des nibbles comme sur la carte, elle est emmagasinée normalement en mémoire sous forme d'octets.

Cd_NextCD est pointé sur la structure ConfigDev suivante, correspondant au PRODUCT indiqué dans Tooltype.

1.7. La conception Janus - Amiga et PC

"Conception Janus", c'est ainsi que Commodore appelle la combinaison entre un ordinateur compatible PC ou AT et un Amiga 2000.

Evidemment, cette combinaison ne consiste pas à monter des pièces IBM bon marché sur son Amiga, mais en un couplage spécial du matériel des deux ordinateurs. L'ordinateur IBM est déposé ici sur une carte d'extension spéciale, qu'on appelle le bridgeboard. Cette carte lance un pont, au sens le plus vérifique du terme, entre l'Amiga et le PC, puisque c'est la seule carte d'extension sur le A2000 qui puisse être introduite aussi bien dans un connecteur de l'Amiga que dans un connecteur du PC. Cette carte relie donc les deux univers informatiques.

Les avantages de cette combinaison sont évidents : on dispose d'un Amiga à part entière, mais aussi d'un PC, les deux fonctionnant avec la même périphérie. Cela permet d'économiser de la place et de l'argent. Un disque dur de 60 Mo coûte en effet moins

cher que deux disques de 30 Mo. Le PC utilise normalement le clavier et le moniteur de l'Amiga, il réalise ses communications sérielles et parallèles par l'intermédiaire des broches correspondantes de l'Amiga. A l'inverse, l'Amiga profite du disque dur IBM bon marché, qui coûte environ la moitié d'un disque dur Commodore, et qui peut être utilisé sans problème par Amiga DOS. On a ainsi un fonctionnement de base du PC en tous points conforme aux exigences, y compris le graphisme couleur.

Si on le désire, on peut introduire des cartes d'extension PC dans les connecteurs d'extension PC sur l'Amiga. Voyons maintenant quelle est la structure d'une carte PC.

1.8. La structure de la carte-passerelle (bridgeboard) PC

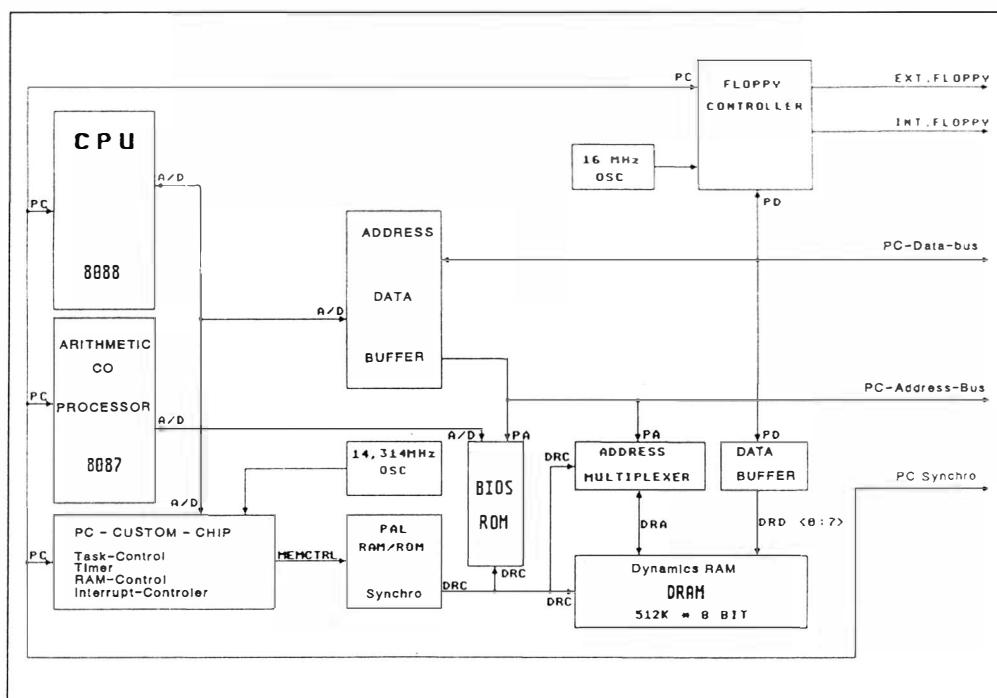


Figure 1 - 46

Les figures 1-46 et 1-47 montrent la structure de base pour l'ensemble de la carte. 1-46 montre le PC. Puisque celui-ci utilise presque la totalité de la périphérie de l'Amiga, on ne trouvera ici qu'un système absolument minimum. Le centre en est le processeur d'un IBM-PC : le processeur 8088 d'Intel. Si vous vous attendiez à trouver ici un processeur moderne et rapide, comme les processeurs 16 ou 32 bits, de plus en plus courants sur les PC, vous serez déçu. Le 8088 est un processeur 8 bits, et de plus c'est un processeur

qui, du fait de la fréquence d'horloge relativement basse de 4,77 MHZ du bridgeboard, ne fonctionne pas plus vite qu'un C64.

Il faut encore mentionner une particularité du 8088. Ses conduits d'adresses et de données ne sont pas disposés de manière séparée, mais sous forme de multiplex. Cela veut dire qu'au début d'un accès à la mémoire, il place les adresses sur le bus; une fois que l'élément adressé les a prises en charge, on passe aux 8 conduits inférieurs du bus de données, destinés aux adresses. D'où le conduit désigné par "A/D" dans la figure 1-46.

Sous le processeur, on a dessiné un chip qui n'existe pas, ou plutôt qui n'est pas livré avec la carte, mais qu'il faut acheter à part si l'on veut en disposer. Seul un socle vide sur le bridgeboard atteste de son existence (en fait de son inexistence). Ce chip mystérieux est un coprocesseur mathématique de type 8087. Presque tous les constructeurs de PC renoncent à ce chip, et laissent à l'acheteur le soin de s'équiper éventuellement lui-même; en effet, tout le monde n'en a pas besoin, et ce n'est pas donné ! (Les seules exceptions sont quelques marques haut de gamme, pour lesquelles le prix d'un 8087 est négligeable devant le prix total de l'installation).

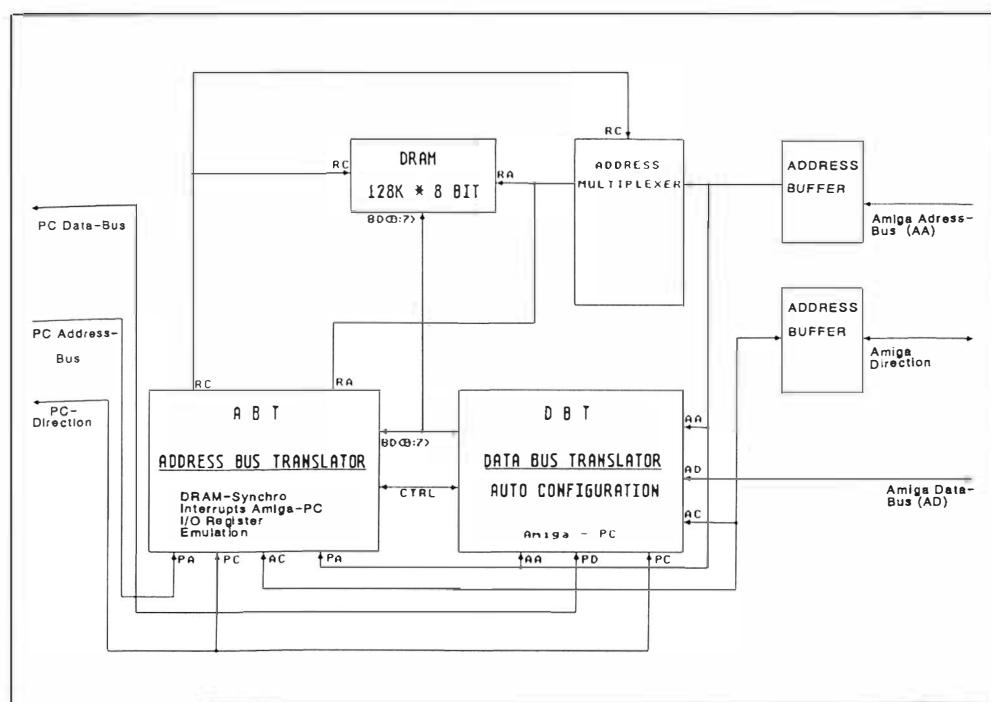


Figure 1 - 47

Presque toute la logique de contrôle d'un IBM-PC, ainsi que l'horloge et les chips d'interruption existant dans le PC initial ont été réunis au fil des ans dans des circuits spéciaux hautement intégrés, par différentes sociétés de semi-conducteurs (ce qui est dû également à l'augmentation exponentielle des clones du PC, puisque c'est ainsi que l'on nomme les innombrables imitations de l'IBM original). Sur le bridgeboard, on n'emploie plus qu'un seul de ces chips, qu'on trouvera dans la figure 1-46 en dessous du 8087, sous le nom "PC-Custom-Chip".

Pour la mémoire principale, l'IBM dispose de 512 Ko de RAM dynamique. Le contrôle de ces chips de RAM est pris en charge par un PAL de type 16L8 développé par Commodore. Le bus de données est relié à la RAM par l'intermédiaire d'un tampon de bus de données 74LS245, les adresses sont reliées par l'intermédiaire de deux éléments multiplex 74HCT257. Il ne s'agit pas toutefois ici des conduits combinées données/adresses du 8088, mais de conduits tout à fait normaux, où adresses et données sont séparées. Cette tâche est exécutée par trois éléments 74LS373.

Dès que le 8088 sort les adresses, celles-ci sont emmagasinées par les trois chips en question, et elles restent disponibles pendant toute la durée de l'accès mémoire par les conduits PA (Adresse PC). Dès que le processeur convertit les conduits d'adresse inférieurs en conduits de données, ils sont reliés au bus de données (PD) du PC par l'intermédiaire d'un tampon 74LS245. A côté du tampon se trouvent, comme on peut le voir sur la figure 1-46, le contrôleur de disquettes, la RAM et toute l'interface Amiga.

Le chip désigné par BIOS-ROM est, comme son nom l'indique, une ROM. Sa taille est de 16 Ko, et il contient ce qu'on appelle le "Basic Input Output System" du PC (BIOS).

Ce BIOS est le niveau le plus bas du système d'exploitation IBM, que l'on peut comparer en quelque sorte (pas entièrement) avec l'EXEC de l'Amiga. Le reste du système d'exploitation, le MS-DOS, est chargé en mémoire principale à partir d'une disquette après un reset. C'est à cela que sert le contrôleur de disquettes qui se trouve en haut à gauche. Il est du type FDC 9268. C'est à lui qu'est relié également le lecteur de disquettes de 5"1/2. Le connecteur du floppy sur la carte PC, exactement comme celui de l'Amiga, est compatible avec le bus Shugart. On peut donc échanger à volonté les lecteurs de disquettes. Si le PC n'utilise pas en retour le floppy-controller de l'Amiga, c'est qu'un nombre considérable de programmes écrits pour le PC accèdent directement au contrôleur de disquettes, et attendent donc évidemment les registres originaux.

L'interface avec l'Amiga

Le schéma de cette interface est donné par la figure 1-47. Elle comprend essentiellement une RAM Dual-Port d'une taille de 128 Ko, et deux circuits spéciaux hautement intégrés provenant de la cuisine Commodore : le Data Bus Translator, et le Address Bus Translator. Rappelons à nouveau les tâches que cette interface doit accomplir :

- ✓ Transmettre les données en provenance du clavier, de l'Amiga au PC.
- ✓ Convertir les données graphiques de l'IBM en un format compatible avec Amiga, et les présenter sur l'écran Amiga.
- ✓ Transférer des données entre les connecteurs sériels et parallèles de l'Amiga et ceux du PC.
- ✓ Transférer des fichiers dans les deux directions.

Du fait des grandes quantités de données, et du traitement rapide des données graphiques, il se pose un problème difficile. Le PC possède deux modes d'affichage principaux : présentation de texte, monochrome ou couleur; et présentation graphique à haute résolution avec 320 ou 640 points par ligne.

L'interface propose la mémoire d'écran nécessaire pour résoudre cette question, à l'aide de la RAM Dual-Port. Dual-Port-RAM signifie que cette RAM peut être utilisée par les deux côtés, Amiga et PC, indépendamment l'un de l'autre. La RAM qui se trouve sur la carte PC n'est pas une RAM Dual-Port "véritable", car les deux processeurs ne peuvent y accéder réellement en simultanéité. Dans ce cas, celui qui vient en second doit attendre jusqu'à ce que l'autre ait terminé. Cette limitation n'est cependant pas très grave pour la tâche que la RAM doit accomplir. Elle remplit parfaitement le rôle qui lui est dévolu.

Lorsque les données graphiques se trouvent dans la RAM Dual-Port, l'Amiga peut les lire et les afficher dans un Screen propre, à l'intérieur du système d'exploitation de l'Amiga. Ou plutôt pourrait le faire, s'il ne surgissait ici un petit problème : le format des données, en effet, n'est pas le même. Du côté de l'Amiga, les données sont distribuées suivant divers bitplanes pour un point donné de l'écran graphique couleur, alors que du côté de l'IBM, deux bits de chaque octet graphique contiennent ensemble les données pour un point. Comme ce serait beaucoup trop long de convertir les données graphiques IBM en format Amiga avec le 68000, c'est l'un des circuits spéciaux qui s'en charge, le Data Bus Translator : celui-ci prend un mot de données graphiques du PC, contenant huit points pour chaque couple de deux bits, et les transforme en deux octets, compatibles avec les bitplanes de l'Amiga. En outre, l'ordre dans lequel les mots et les mots longs du 68000 et du 8088 sont disposés en mémoire n'est pas le même. Avec le 68000, les huit bits supérieurs d'un mot sont déposés en mémoire avant les huit bits inférieurs. La même chose vaut pour les mots longs. Le mot contenant les 16 bits supérieurs précède d'une adresse de mot celui qui contient les 16 bits inférieurs. Sur le 8088, c'est exactement le contraire : le low-byte est placé en mémoire avant le hi-byte. Ce problème est également résolu par le Data Bus Translator.

Pour le transfert des données de l'interface, on a choisi un autre moyen. Tous les registres des interfaces sérielles et parallèles (et aussi les registres de contrôle pour le graphique) ont été déposés dans le second circuit spécial, le Address Bus Translator. Celui-ci ne fait certes pas grand chose avec ces données, mais il permet à l'Amiga de lire les valeurs et de les traiter. En outre, lors d'un accès à certains registres I/O, une interruption est déclenchée dans l'Amiga, de façon à ce que celui-ci puisse traiter les données qui se trouvent dans les registres. Pour le clavier aussi, il existe une émulation tout à fait spéciale. Le circuit spécial du PC que l'on voit dans la figure 1-46 contient entre autres un connecteur de clavier PC. Un tel clavier transfère ses données comme le fait le clavier Amiga, sous forme serielle, au moyen d'un conduit de données et d'un

conduit de fréquence. Au lieu de relier à un clavier ces deux conduits du circuit spécial PC, ils conduisent au chip Address Bus Translator. Si l'Amiga veut transférer au PC un code clavier, il l'écrit simplement dans un registre spécial de ce chip, qui l'envoie alors sérielement au circuit spécial PC, exactement comme le ferait un clavier IBM réel.

L'accès alterné aux mémoires de masse comme les disques durs ou les lecteurs de disquettes s'effectue simplement par l'intermédiaire de la RAM Dual-Port. Les drivers correspondants vont chercher les données non pas dans un espace réel de mémoire, mais dans cette RAM, ou ils les y inscrivent. Les tâches des deux circuits spéciaux d'interface sont donc distribuées de la façon suivante :

Data Bus Translator

- ✓ Autoconfiguration
- ✓ Conversion format PC -> Amiga

Address Bus Translator

- ✓ Contrôle de l'accès à la RAM Dual-Port et aux registres I/O communs.
- ✓ Logique d'interruption entre Amiga et PC - registres I/O et registres de contrôle graphique du PC.
- ✓ Registre de contrôle, pour gérer l'interface à partir de l'Amiga
- ✓ Emulation du clavier

Dépôt en mémoire de la carte PC par l'Amiga

L'adresse de base de la carte est définie par la bibliothèque Expansion. Au total, celle-ci définit 512 Ko d'espace mémoire pour les adresses dans l'Amiga. Si l'on connaît l'adresse de base de la carte, il faut se servir des structures de données de la bibliothèque Expansion (section 1.6).

Les 512 Ko sont partagés en 4 blocs de 128 Ko :

Adresse de base+(\$00000 - \$1FFFF): Accès octet
Adresse de base+(\$20000 - \$3FFFF): Accès mot
Adresse de base+(\$40000 - \$5FFFF): Accès graphique
Adresse de base+(\$60000 - \$7FFFF): Accès registres I/O

Les trois premières sections fournissent chacun la RAM Dual-Port, les données étant toutefois groupées différemment.

LA RAM Dual-Port est partagée comme suit :

\$00000 - \$0FFFF	64 Ko	Mémoire tampon pour disquette/disque dur
\$10000 - \$17FFF	32 Ko	Mémoire graphique couleur
\$18000 - \$1BFFF	16 Ko	Paramètre RAM (pour mode texte)
\$1C000 - \$1CFFF	8 Ko	Mémoire graphique monochrome
\$1E000 - \$1FFFF	8 Ko	Secteur I/O

2. Exec

Dans ce chapitre, nous allons pénétrer dans les profondeurs du système d'exploitation de l'Amiga, dont la connaissance est indispensable pour une programmation correcte.

Comme nous l'avons mentionné précédemment, Exec représente le niveau le plus profond du système d'exploitation, celui qui s'occupe avant tout du fonctionnement Multi-tâches. Les autres tâches effectuées à ce niveau sont la gestion des interruptions, ainsi que l'affichage du message d'effondrement que chacun connaît : le "GURU MEDITATION". Entre autre, la structure ExecBase (la structure de base d'Exec, dont nous aurons à reparler) représente l'accès à presque toutes les structures système.

Avant de nous occuper des fonctions de Exec, nous allons présenter les structures les plus élémentaires, dont se servent pratiquement toutes les structures système.

2.1. Les listes chaînées

Tous les lecteurs de cet ouvrage ne savent sans doute pas ce que sont des "listes" au moment d'aborder ce chapitre. C'est pourquoi il faut dire ici quelques mots du principe et de l'utilité d'une liste.

Supposons que vous ayez besoin dans votre programme de nombres en grande quantité, qui doivent se présenter triés selon leur taille. Dans cet ensemble de nombres, vous aurez à insérer sans cesse de nouveaux nombres ou à en supprimer.

Si l'ensemble de nombres est défini par un tableau de variables, l'insertion d'un nombre nouveau oblige à décaler tous ceux qui le suivent (les plus grands). Lorsque le tableau est volumineux, on perd alors beaucoup de temps, puisqu'il faut copier tous les nombres qui viennent ensuite.

Ce problème peut se résoudre de manière plus élégante et surtout plus rapide à l'aide d'une liste chaînée. Dans ce cas, vous ne placez pas les nombres dans un tableau, chacun d'eux restant ensuite à l'endroit où on l'a placé. Mais il faut adjoindre à chacun de ces nombres des informations supplémentaires, à l'aide desquels ils pourront être triés.

Sur la base de ces informations supplémentaires, vous ne gérez plus des nombres pris pour eux-mêmes, mais un faisceau d'informations (une structure). A l'intérieur de ce faisceau, l'une des informations est le nombre à trier.

Les nouvelles informations dans cette structure (ce faisceau d'informations) concernent la position de la structure suivante, dans laquelle se trouve le nombre qui vient ensuite quant à la taille, ainsi que la position de la structure du nombre précédent.

1ère structure <-> 2ème structure <-> 3ème structure <-> ...

Pour retrouver un nombre donné, il suffit alors de savoir où se trouve la 1ère structure, et vous retrouvez alors toutes les autres grâce au chaînage des structures. Ce système nécessite certes plus de place en mémoire, mais son grand avantage sera visible lorsqu'on introduira ou que l'on supprimera des nombres (structures). Pour introduire en effet un nouveau nombre dans la liste triée, il n'y pas grand chose à copier. Il suffit de définir les deux structures entre lesquelles le nouveau nombre devra être introduit. Il faudra ensuite modifier les pointeurs des structures entre lesquelles s'insère la nouvelle structure.

Considérons ce processus à l'aide d'un exemple.

Supposons que vous vouliez placer un nouveau nombre (et donc une nouvelle structure) entre la 2ème et la 3ème structure. Dans ce cas, il faudra indiquer dans la 2ème structure que le nombre suivant ne se trouve pas dans la structure 3, mais dans la structure nouvellement définie. Dans celle-ci, on indiquera que le nombre immédiatement inférieur se trouve dans la structure 2, et le nombre immédiatement supérieur dans la structure 3. Il ne reste plus qu'à indiquer dans la structure 3 que le nombre immédiatement inférieur ne se trouve pas dans la structure 2, mais dans la nouvelle structure.

Ces trois étapes étant réalisées, le nouveau nombre est désormais intégré dans la liste. Il est clair que ce processus est beaucoup plus rapide que de recopier des centaines de nombres, lorsque le cas se présente. Une liste a cependant un autre avantage. Elle permet de gérer un très grand nombre de données en relation l'une avec l'autre, en fait sans limitation de longueur, et de se contenter pour cela d'une seule donnée : le début de la liste.

Le système d'exploitation de l'Amiga travaille beaucoup avec les listes chaînées, du fait des avantages que nous venons de décrire. Ces listes chaînées sont réalisées au moyen de deux types de structure. On a besoin d'une part d'une structure de base, pour réaliser le chaînage de la liste. D'autre part, il faut pouvoir gérer convenablement l'ensemble de la liste - du début à la fin.

La structure Node

Nous allons parler tout d'abord de la structure à l'aide de laquelle on réalise le chaînage de la liste. Cette structure s'appelle la structure Node (Node = noeud).

La structure Node est placée en général en tête d'une structure plus complexe, chaînée avec d'autres structures. Dans notre exemple, les éléments reliés par la chaîne étaient des nombres.

Elle offre la possibilité de reporter les adresses en mémoire, là où se trouvent la structure précédente et la structure suivante. On peut reporter également d'autres informations.

Voici la structure Node pour les programmeurs en C, à partir du fichier 'Include "exec/nodes.h"' :

```
struct Node {
    struct Node *ln_Succ;          /* Pointeur sur successeur */
    struct Node *ln_Pred;          /* Pointeur sur prédécesseur */
    UBYTE      ln_Type;           /* Type de la structure */
    BYTE       ln_Pri;            /* Priorité */
    char       *ln_Name;          /* Pointeur sur le nom*/
};
```

Les programmeurs en Assembleur trouveront cette structure dans le fichier Include "exec/nodes.i" :

STRUCTURE	LN,0	* Offsets
APTR	LN_SUCC	* 0 \$00
APTR	LN_PRED	* 4 \$04
UBYTE	LN_TYPE	* 8 \$08
BYTE	LN_PRI	* 9 \$09
APTR	LN_NAME	* 10 \$0A
LABEL	LN_SIZE	* 14 \$0E

On peut partager cette structure en deux parties. Elle comprend d'une part un chaînage (ln_Succ et ln_Pred), et d'autre part des données (ln_Type, ln_Pri et ln_Name).

***ln_Succ**

Pointeur sur le noeud suivant à l'intérieur de la liste.

***ln_Pred**

Pointeur sur le noeud précédent à l'intérieur de la liste.

ln_Type

Dans cet octet sont rangés sous forme de codes les différents types de la structure dont le début représente la structure Node.

ln_Pri

Contient la priorité de la structure à l'intérieur de la liste.

***ln_Name**

Dans ce mot long sera conservé un pointeur sur une chaîne de caractères terminée par un 0. On dépose ici le nom de la structure (par exemple "Ma_structure_globale"). Il doit être choisi de telle manière que l'on sache immédiatement de quel noeud il s'agit, ce qui facilite grandement la recherche des erreurs.

Pour beaucoup d'applications, les entrées comme Type, Priorité et Nom de la structure ne sont pas significatives. C'est le cas de l'exemple précédent. C'est pourquoi on dispose

d'une structure supplémentaire, qui renonce à ces entrées. Cette structure "amincie" s'appelle MinNode, et nous la détaillons dans ce qui suit.

Structure MinNode pour les programmeurs en C, dans le fichier Include "exec/nodes.h" :

```
struct MinNode {  
    struct MinNode *mln_Succ; /* Pointeur sur Successeur */  
    struct MinNode *mln_Pred; /* Pointeur sur Prédécesseur */  
};
```

Les programmeurs en assembleur trouveront cette structure dans le fichier Include "exec/nodes.i"

```
STRUCTURE MLN,  
    APTR MLN_SUCC  
    APTR MLN_PRED  
    LABEL MLN_SIZE  
          * Offsets  
          * 0 $00  
          * 4 $04  
          * 8 $08
```

Les entrées de cette structure sont identiques à celles de la structure Node.

Initialisation d'une structure Node

Avant d'insérer une structure Node dans une liste, il faut d'abord entièrement l'initialiser, même si toutes les entrées ne présentent pas d'intérêt pour votre cas concret. L'initialisation complète de la structure facilite la recherche des erreurs, car on retrouve plus facilement ainsi l'origine des structures qui se trouvent en mémoire.

Dans ce but, il faut définir entre autres le type. On a ici le choix entre plusieurs types standardisés, que nous allons détailler dans ce qui suit :

La définition des types est faite dans le fichier Include "exec/nodes.h" ou "exec/nodes.i" pour assembleur.

NT_UNKNOW	00
NT_TASK	01
NT_INTERRUPT	02
NT_DEVICE	03
NT_MSGPORT	04
NT_MESSAGE	05
NT_FREEMSG	06
NT_REPLYMSG	07
NT_RESOURCE	08
NT_LIBRARY	09
NT_MEMORY	10
NT_SOFTINT	11
NT_FONT	12
NT_PROCESS	13
NT_SEMAPHORE	14
NT_SIGNALSEM	15
NT_BOOTNODE	16 (Kick 1.3)

Les types indiqués ici ne concernent évidemment que ceux qui sont utilisés par le système d'exploitation. Si vous désirez dans vos programmes initialiser les structures

correspondantes, vous devez faire usage de ces types définis. Voici maintenant un exemple en assembleur pour l'initialisation d'une structure Node :

```
*****
*           Initialisation d'une structure Node
*           écrite pour l'assembleur Manx 'as' V3.6
*
*           Appel de l'assembleur :
*           as -c -d -o InitNode.o -iinclude InitNode.asm
*           Appel de l'éditeur de liens :
*           ln -o InitNode InitNode.o
*
*****
include "exec/types.i"
include "exec/nodes.i"

    lea      MyNode(pc),a0          ;Pointeur sur Node
    lea      MynodeName(pc),a1      ;Pointeur sur Name
    move.b  #NT_UNKNOWN,LN_TYPE(a0) ;Indique le type de Node
    clr.b   LN_PRI(a0)            ;Priorité du Node
    move.l   a1,LN_NAME(a0)        ;Pointeur sur Name
    clr.l   d0                    ;Message OK
    rts                            ;Retour

MyNode:             ds.b LN_SIZE      ;Emplacement structure
MynodeName:         dc.b 'MaStructureNode',0 ;Nom de la structure Node

END
```

La structure List

Comme nous l'avons déjà dit, il faut une structure supplémentaire pour mieux gérer une liste chaînée. Dans cette seconde structure, on reporte le début et la fin de la liste. De plus, on indique le type de liste. Les types disponibles sont les mêmes que ceux qui sont utilisés avec la structure Node.

Les programmeurs en C trouveront la définition de la structure dans le fichier Include "exec/lists.h" :

```
struct List {
    struct Node *lh_Head;        /* Pointeur sur le début de la liste */
    struct Node *lh_Tail;        /* toujours nul */
    struct Node *lh_Tailpred;    /* Pointeur sur la fin de la liste */
    UBYTE  lh_Type;              /* Type de la liste */
    UBYTE  lh_pad;               /* octet pour la longueur des mots */
};
```

La structure correspondante en assembleur se trouve dans le fichier Include "exec/lists.i" :

```
STRUCTURE LH_0          * Offsets
    APTR  LH_HEAD      * 0 $00
    APTR  LH_TAIL      * 4 $04
    APTR  LH_TAILPRED  * 8 $08
    UBYTE LH_TYPE      * 12 $0C
    UBYTE LH_pad        * 13 $0D
    LABEL LH_SIZE      * 14 $0E
```

lh_Head

Pointeur sur la première entrée (premier noeud) d'une liste.

lh_Tail

Toujours égal à 0.

lh_TailPred

Pointeur sur la dernière entrée valide de la liste.

lh_Type

Donne le type des structures enchaînées dans une liste. *lh_Type* est défini exactement comme les types des noeuds.

lh_pad

lh_pad sert uniquement à ce que la structure ait un nombre pair d'octets.

Pour la structure List, il existe également une version réduite :

```
Struct MinList {
    struct MinNode * mlh_Head;
    struct MinNode * mlh_Tail;
    struct MinNode * mlh_TailPred;
};
```

mlh_Head

est un pointeur sur la première entrée (premier noeud) de la liste.

mlh_Tail

toujours nul.

mlh_TailPred

est un pointeur sur la dernière entrée valide de la liste.

La structure d'une liste

Après avoir passé en revue les deux structures nécessaires pour la réalisation des listes, nous allons voir comment initialiser une telle liste.

Pour pouvoir initialiser une liste, il faut avoir parfaitement compris comment elle est constituée. Nous avons dit que la première entrée de la structure List (lh_Head ou mlh_Head) pointe sur le premier membre de la liste. Ce premier membre est une structure Node ou MinNode avec des paramètres que le programmeur peut définir librement.

En partant de cette structure, on a un pointeur sur la structure suivante au moyen de ln_Succ, l'en-tête de cette structure suivante constituant une structure Node. En outre, on a un pointeur sur la structure List par l'intermédiaire du chaînage en sens inverse (par l'intermédiaire de ln_Pred). La seconde structure Node dans la liste pointe sur la première et sur la troisième structure.

La dernière structure Node de la liste doit évidemment être caractérisée comme telle. On reconnaît la fin d'une liste au fait que le pointeur sur la structure suivante n'existe pas - donc est nul. Cela est vrai dans les deux sens. Dans ce but, il existe dans la structure List l'entrée lh_Tail, qui est toujours nulle. La dernière entrée de la liste pointe sur lh_Tail (entrée nulle dans la structure List).

Si vous parcourez vers l'avant une liste élaborée ainsi, vous rencontrez un membre de la liste dont l'entrée ln_Succ est nulle. Lorsque cette entrée est atteinte, ce n'est plus un node qui est attendu, mais une structure List. Malgré tout, vous trouvez le pointeur attendu à l'endroit où vous avez habituellement le pointeur sur la structure précédente.

Cependant, ce n'est pas ln_Pred, comme pour une structure Node; il s'agit cette fois de l'entrée lh_TailPred de la structure List. Si vous parcourez la liste vers l'arrière, vous parviendrez en un point où le pointeur ln_Pred est nul. En réalité, là encore, il s'agit de l'entrée lh_Tail de la structure List. Grâce à cette forme subtile de la structure List, on économise deux structures Node dans la liste, précisément celles qui devraient signaler les extrémités.

Pour mieux comprendre comment est constituée une liste, considérez la figure suivante :

Exemple d'une liste avec enchaînement de noeuds.

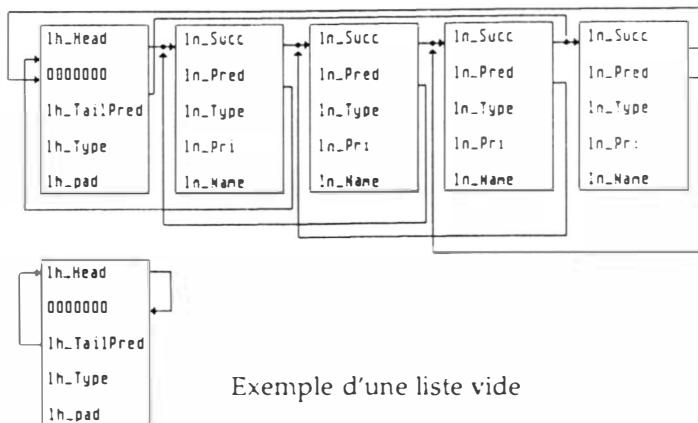


Figure 2 - 48

L'initialisation des listes

Créons d'abord une liste vide. Comme vous pouvez le constater sur la figure, l'entrée `lh_Head` pointe sur `lh_Tail` (celui-ci est toujours nul), et `lh_TailPred` sur `lh_Head`. En C, l'initialisation aurait l'aspect suivant :

```
#include "exec/lists.h"

main ()
{
    struct List liste;

    liste.lh_Head = (struct Node *)&liste.lh_Tail;
    liste.lh_Tail = 0;
    liste.lh_TailPred = (struct Node *)&liste.lh_Head;
    liste.lh_Type = NT_TASK;
}
```

En assembleur, on aurait ceci :

```
include "exec/types.i"
```

```
include "exec/lists.i"
```

```
main :
    lea      List(pc),a0          Pointeur sur List-Struct
    move.l  a0,(a0)
    clr.l   LH_TAIL(a0)          LH_TAIL = Zéro
    move.l  a0,LH_TAILPRED(a0)  Pointeur sur le début de la liste
```

```

add.l    #LH_TAIL,(a0)      Pointeur sur le successeur (LH_TAIL)
rts

List:   ds.b LH_SIZE

END

```

L'insertion de Nods dans la liste sera expliquée ultérieurement. Il faut cependant montrer encore comment une liste vide peut être reconnue comme telle. Il existe pour cela deux méthodes. D'une part, on peut vérifier si lh_Head est nul (NIL), d'autre part si lh_TailPred pointe sur le début de la liste (lh_Head). Si c'est le cas, la liste en question est vide. Cet examen se fait ainsi en C :

```

if (liste.lh_TailPred == &liste) {
    printf ("liste vide");
}

```

et selon l'autre possibilité :

```

if (liste.lh_Head -> ln_Succ == 0) {
    printf ("liste vide");
}

```

2.2. Fonctions Exec pour la gestion des listes

Pour gérer les listes, Exec dispose d'une série de fonctions très utiles, à l'aide desquelles on peut exécuter pratiquement toutes les opérations. Si vous ne savez pas encore comment on utilise la fonction Library, passez dès maintenant au chapitre sur les bibliothèques, et vous reviendrez ensuite à la section présente. Ne nous tenez pas rigueur du fait que, pour conserver la cohérence des chapitres, nous introduisons ici des fonctions sans avoir expliqué leur utilisation. La première fonction est Insert(). Elle sert à insérer des noeuds dans une liste avec indication de la position à laquelle le nouveau noeud (Node) devra être inséré.

Insert

Fonction : Insert (Liste, Noeud, Prédécesseur);
 a0 a1 a2

Offset -234 -\$0EA

Description

Cette fonction sert à insérer un noeud dans une liste.

Paramètres :**Liste**

Pointeur sur la liste où le noeud doit être inséré.

Noeud

Pointeur sur le noeud dans la liste où a lieu l'insertion..

Prédécesseur

Pointeur sur le noeud précédent l'insertion. Si ce pointeur a une valeur différente de 0, le paramètre Liste ne présente plus d'intérêt. Ce ne sera pas le pointeur du noeud qui sera employé pour déterminer la position de l'insertion, mais le paramètre prédécesseur. Si ce dernier est à 0, le noeud sera inséré à la première position. La deuxième possibilité pour placer un noeud en première position est de mettre "prédécesseur" sur lh_Head. Pour la dernière position, il suffit d'initialiser ce paramètre sur lh_TailPred. Pour placer un noeud en première ou dernière position, il existe encore un certain nombre d'autres fonctions.

Remove

Fonction : Remove(Noeud);
a1

Offset : -252 -\$0FA

Description :

Comme le nom l'indique, cette fonction sert à éliminer des noeuds dans une liste.

Paramètre :**Noeud**

Pointeur sur le noeud qui doit être éliminé. S'il ne s'agit pas d'un pointeur sur un noeud, Exec ne le reconnaît pas et 'élimine' tout de même, ce qui risque de conduire à une perte d'informations ou à un effondrement du système.

AddHead

Fonction : AddHead(Liste, Noeud);
a0 a1

Offset : -240 -\$0F0

Description :

Cette fonction est utilisée pour insérer un noeud en tête de liste.

Paramètres :*Liste*

Pointeur sur la liste où le noeud doit être inséré.

Noeud

Pointeur sur le noeud à insérer.

RemHead

Fonction : RemHead(Liste);
a0

Offset : -258 -\$102

Description

Elimine le premier noeud de la liste indiquée.

Paramètre :*Liste*

Pointeur sur la liste où le premier noeud sera éliminé.

AddTail

Fonction : AddTail(Liste, Noeud)
a0 a1

Offset : -246 -\$0F6

Description :

Insérer un noeud comme dernier membre d'une liste

Paramètres :*Liste*

Pointeur sur la liste où le noeud doit être inséré.

Noeud

Pointeur sur le noeud qui doit être inséré.

RemTail

Fonction : RemTail(Liste);
 a0

Offset : -264 - \$108

Description :

RemTail élimine la dernière entrée d'une liste

Paramètre :*Liste*

Pointeur sur la liste, où le dernier enregistrement doit être éliminé.

Enqueue

Fonction : Enqueue(Liste, Noeud);
 a0 a1

Offset -270 - \$10E

Description :

Cette fonction est utilisée pour classer des entrées dans une liste suivant leurs niveaux de priorité. Le noeud possédant le niveau de priorité le plus important sera mis au début de la liste. Si plusieurs noeuds ont la même priorité, le nouveau sera mis derrière ceux qui sont déjà présents.

Paramètres :*Liste*

Pointeur sur la liste, où le noeud doit être inséré.

Noeud

Pointeur sur le noeud qui sera inséré.

FindName

Fonction : Entrée = FindName(Liste,"Nom");
d0 a0 a1

Offset : -276 -\$114

Description :

Avec FindName, on peut chercher dans la liste indiquée un noeud ayant le nom mentionné.

Paramètres :

Liste

Pointeur sur la liste dans laquelle s'effectue la recherche.

Nom

Pointeur sur le nom cherché. Cette chaîne de caractères doit être terminée par un 0. Lors de l'appel d'une fonction C, il ne s'agira pas d'un pointeur sur le nom, mais de la chaîne de caractères elle-même.

Résultat :

Entrée

Ici, la fonction renvoie un pointeur sur le noeud trouvé. Si aucune entrée ne correspond au nom fixé, la fonction renverra un 0.

L'exemple suivant montre comment on peut établir si le nom d'un noeud figure deux fois dans une liste. L'exemple ne peut pas être lancé dans sa forme présente. La liste doit être initialisée avant l'appel de 'FindName()', sous peine d'effondrement du système.

```
#include <exec/lists.h>

main()
{
    struct node *FindName(),*node;
    struct List *liste;

    if ((node = FindName (liste,"testnode"))!=0)
        if ((node = FindName (node,"testnode"))!=0)
            printf ("\n le nom testnode a été trouvé deux fois\n");
}
```

Après avoir présenté les fonctions disponibles pour le traitement des listes, nous devons expliciter leur utilisation.

C'est le rôle de notre exemple, qui montre comment on crée une liste, comment on l'affiche, et comment on efface une de ses entrées.

Exemple de liste :

```
#include <exec/lists.h>

char *name[] = {"Node1","Node2","NODE3"};

struct List liste;
struct Node node[3],*np;

main()
{
    int i;
    char n;
    liste.lh_Head = (struct Node *)&liste.lh_Tail;
    liste.lh_Tail = 0;
    liste.lh_TailPred = (struct Node *)&liste.lh_Head;
    liste.lh_Type = NT_UNKNOWN;

    for (i=0;i<-2;i++) {
        node[i].ln_Type = NT_TASK;
        node[i].ln_Name = name[i];

        AddTail (&liste,&node[i]);
    }

    emission();
    printf ("\n Emission de la liste prête\n");

    np = liste.lh_Head->ln_Succ;
    Remove(np);
    emission();
    printf ("\n Le 2ème noeud sera éliminé\n");
}

emission ()
{
    for (np = liste.lh_Head;
         np != liste.lh_Tail;
         np = np->ln_Succ)
        printf ("\n %s \n",np->ln_Name);
}
```

2.3. Macros en assembleur pour la gestion des listes

Pour les programmeurs en assembleur, il existe dans le fichier Include "exec/lists.i" plusieurs macros utiles pour la gestion des listes.

Si vous n'avez aucune expérience dans l'utilisation des macros, peut-être vaut-il mieux travailler d'abord la section suivante, sur la programmation en assembleur.

```

NEWMEST MACRO * list
MOVE.L \1,(\1)
ADDQ.L #LH_TAIL,(\1)
CLR.L LH_TAIL(\1)
MOVE.L \1,(LH_TAIL+LN_PRED)(\1)
ENDM

```

La macro NEWMEST sera utilisée pour créer une structure de liste. La liste créée sera vide.

Exemple :

```

lea Mylist(pc),a0      Pointeur sur la liste
NEWLIST A0              initialiser la liste

TSTLIST MACRO * [list]
IFC '\1,''
CMP.L LH_TAIL+LN_PRED(A0),A0
ENDC
IFNC '\1,''
CMP.L LH_TAIL+LN_PRED(\1),\1
ENDC
ENDM

```

La macro TSTLIST est utilisée pour vérifier s'il y a des entrées dans une liste. Si l'on ne transmet pas de paramètre (registre) à cette macro, c'est A0 qui est pris comme registre de manière standard.

Exemple :

```

lea liste(pc),a1      Pointeur sur liste
TSTLIST A1            Appel de la macro
beq listeVide         Embranchement si liste vide

SUCC     MACRO * node,succ
MOVE.L (\1),\2
ENDM

```

La macro SUCC sera utilisée pour trouver le successeur d'une entrée dans la liste. On transmet à la macro deux paramètres (registres). Le premier paramètre est un registre d'adresse, dont le contenu reflète les adresses de la structure Node. Dans le second registre, on place le pointeur sur le successeur.

Exemple :

```

lea Node,a0            Pointeur sur la structure Node
SUCC A0,D0             Pointeur sur le successeurs en D0

ou

lea Node,a0            Pointeur sur la structure Node
SUCC A0,Successeur    Garde le successeur en mémoire

```

```
PRED      MACRO    * node,pred
          MOVE.L  LN_PRED(\1),\2
ENDM
```

La macro PRED est l'inverse de la macro SUCC. Elle va chercher le pointeur sur le prédecesseur. On se reportera à l'exemple de la macro SUCC.

```
IFEMPTY   MACRO    * list,label
          CMP.L   LH_TAIL+LN_PRED(\1),\1
          BEQ     \2
ENDM
```

On peut utiliser la macro IFEMPTY pour sauter à un label, lorsqu'une liste est reconnue comme étant vide. On transmet pour paramètres un registre d'adresse avec le pointeur sur la structure de liste, ainsi que le label auquel le saut doit s'effectuer.

Exemple :

```
lea      List,a0  Pointeur sur liste
IFEMPTY A0,Saut1  Embranchement vers le label Saut1, si la liste est vide
```

Saut1:

```
IFNOTEMPTY MACRO    * list,label
          CMP.L   LH_TAIL+LN_PRED(\1),\1
          BNE     \2
ENDM
```

On utilise la macro IFNOTEMPTY comme la précédente, avec cette différence que l'embranchement s'effectue cette fois lorsque la liste n'est pas vide. On se reportera à l'exemple de la macro IFEMPTY.

```
TSTNODE   MACRO    * node,next
          MOVE.L  (\1),\2
          TST.L   (\2)
ENDM
```

La macro TSTNODE sert à vérifier s'il existe un successeur. Le pointeur sur le successeur est déposé dans le registre de données transmis. Les paramètres attendus sont un registre d'adresse avec le pointeur sur la structure Node, et un second registre d'adresse.

Exemple :

```
lea      Node,a0           Pointeur sur la structure Node
TSTNODE A0,A1              Appel de macro
beq     ListeFin           Saut si c'est la fin de la liste
```

ListeFin:

```
ADDHEAD  MACRO    * list, node (A0, A1)
          MOVE.L  (A0),D0
```

```

MOVE.L A1,(A0)
MOVEM.L D0/A0,(A1)
MOVE.L D0,A0
MOVE.L A1,LN_PRED(A0)
ENDM

```

On utilise la macro ADDHEAD pour ajouter une structure Node à l'en-tête d'une liste. Cette macro s'emploie comme la fonction Exec AddHead. Dans le registre A0, on trouve le pointeur sur la liste, et dans le registre A1, le pointeur sur la structure Node.

Exemple :

<pre> lea liste,a0 leaNode,a1 ADDHEAD </pre>	Pointeur sur la liste Pointeur sur Node Insère un noeud dans la liste
ADDTAIL MACRO	
<pre> LEA LH_TAIL(A0),A0 MOVE.L LN_PRED(A0),D0 MOVE.L A1,LN_PRED(A0) MOVE.L A0,(A1) MOVE.L D0,LN_PRED(A1) MOVE.L D0,A0 MOVE.L A1,(A0) ENDM </pre>	

La macro ADDTAIL est utilisée pour ajouter une structure Node à la fin d'une liste. C'est le pendant de la macro ADDHEAD. On peut la remplacer par la fonction Exec AddTail. En A0, on doit avoir le pointeur sur la liste, et en A1 le pointeur sur la structure Node.

Exemple :

<pre> lea liste,a0 leaNode,a1 ADDTAIL </pre>	Pointeur sur liste Pointeur sur Node Insère un noeud dans la liste
REMOVE MACRO	
<pre> MOVE.L (A1),A0 MOVE.L LN_PRED(A1),A1 MOVE.L A0,(A1) MOVE.L A1,LN_PRED(A0) ENDM </pre>	

La macro REMOVE sera employée pour supprimer une structure Node dans une liste. Le paramètre est le même que pour la fonction Remove de Exec. Dans A1 se trouve le pointeur sur la liste.

REMHEAD MACRO	
<pre> MOVE.L (A0),A1 MOVE.L (A1),D0 BEQ.S REMHEAD\@ MOVE.L D0,(A0) EXG.L D0,A1 </pre>	

```
MOVE.L A0,LN_PRED(A1)
REMHEAD\@*
ENDM
```

Comme il fallait s'y attendre, la macro REMHEAD peut être remplacée par la fonction RemHead de Exec. En A0, on trouve le pointeur sur la structure de liste, et en A1 le pointeur dirigé vers la structure Node.

```
REMHEADQ MACRO * head,node,scratchReg
MOVE.L (\1),\2
MOVE.L (\2),\3
MOVE.L \3,(\1)
MOVE.L \1,LN_PRED(\3)
ENDM
```

Lorsque les trois registres sont libres, vous pouvez aussi utiliser la macro REMHEADQ. Grâce au troisième registre libre, il est possible de supprimer plus rapidement une structure Node dans une liste. Comme paramètres, on indique le pointeur sur la liste, le pointeur sur la structure Node, et un troisième registre d'adresse.

Exemple :

lea	List,a0	Pointeur sur la liste
lea	Node,a1	Pointeur sur la structure Node
REMHEADQ	A0,A1,A2	Appel de macro

```
REMTAIL MACRO
MOVE.L LH_TAIL+LN_PRED(A0),A1
MOVE.L LN_PRED(A1),D0
BEO.S REMTAIL\@*
MOVE.L D0,LH_TAIL+LN_PRED(A0)
EXG.L D0,A1
MOVE.L A0,(A1)
ADDQ.L #4,(A1)
REMTAIL\@*
ENDM
```

La dernière macro mentionnée sera REMTAIL. Elle travaille comme la fonction RemTail de Exec, et supprime la structure Node qui se trouve à la fin de la liste.

Exemple :

lea	liste,a0	Pointeur sur la liste
REMTAIL		Appel de macro

2.4. Programmation professionnelle en assembleur

Nous ne répondrons pas ici à la question générale consistant à se demander s'il vaut mieux programmer en assembleur ou dans un langage évolué comme le C. Nous nous proposons plutôt de montrer à ceux qui, de gré ou de force, programment en assembleur, comment se faciliter la tâche.

On peut cependant énoncer quelques généralités sur le conflit mentionné. Le code en assembleur est un peu plus court et plus rapide que celui d'un langage évolué. Mais la recherche des erreurs (et quel programme est exempt d'erreur dès le début ?) est plus difficile en assembleur. En outre, un programme en assembleur peut être difficilement exporté sur un autre ordinateur.

Si l'on veut cependant écrire un programme spécifique à l'Amiga, la dernière objection n'est pas recevable. Pour vous décider entre l'assembleur et un langage évolué, voici quelques exemples :

- ✓ Les interruptions doivent être toujours écrites en assembleur.
- ✓ Les devices sont le plus souvent en assembleur.
- ✓ Les handlers sont en assembleur ou en langage évolué.
- ✓ Il vaut mieux écrire les très longs programmes en langage évolué avec des incises en assembleur.

Mais venons-en maintenant au sujet de cette section.

Pour la programmation professionnelle, on a bien sûr besoin d'un assembleur convenable. Certains assembleurs sont particulièrement propices. Nous citerons "ASSEM", développé par la société Metacomco, fourni avec un pack de développement. Dans la même catégorie (un peu plus performant), on peut également citer "AS" de MANX et le Devpack 2.0 de la société HighSoft, ainsi que l'assembleur du domaine public 'A68K', qui se trouve sur le disque 186 de la série Fish. Les assembleurs disposent de labels locaux, de macros, et de la possibilité d'utiliser les fichiers Include de Commodore. En outre, on peut créer quelques bibliothèque, et les lier à ceux-là.

L'inconvénient de ces quatre assembleurs est le suivant : il faut d'abord créer le programme avec un éditeur, et ensuite assembler et éditer les liens, ce qui n'est pas particulièrement rapide. Une variante plus souple est offerte par le DevPack, qui permet d'assembler rapidement dans la RAM.

Les exemples que nous allons citer maintenant ont été écrits - à quelques exceptions près - en 'ASSEM' ou en 'AS'.

2.4.1. Indications sur l'utilisation des assembleurs

Les indications que nous allons donner se réfèrent à ASSEM, mais on pourra les transférer aux autres assembleurs cités.

Tous les labels doivent se trouver au début d'une ligne. Avant un code de fonction, il faut laisser au moins un espace. Les caractères ";" et "/*" introduisent un commentaire.

Pour affecter une valeur déterminée à une constante, on utilise la commande EQU. La constante doit toujours se trouver au début d'une ligne. Exemple :

```
MaValeur EQU 20
```

Pour affecter de manière provisoire une valeur à une variable, on utilise la commande SET. Exemple :

```
MaValeur SET 20
..
.
MaValeur SET MaValeur + 10
```

L'insertion d'octets dans le code se fait avec dc.x. A la place de x, on peut mettre b (pour bytes), w (pour word) et l (pour mot long). Exemple :

```
dc.b $12,%01001101,10,'hallo'
dc.w $1234,100
dc.l $12345678,1202621,LABEL
```

L'insertion de blocs remplis de 0 s'effectue avec ds.x. Ici encore, à la place de x, on peut mettre b, w ou l. Lorsqu'on insère des mots ou des mots longs, ceux-ci commencent toujours automatiquement à une adresse paire. Voici un exemple :

```
ds.b 10           ; Insertion de 10 octets vides
ds.w 10           ; Insertion de 10 mots vides
ds.l 10           ; Insertion de 10 mots longs vides
ds.w 0            ; Placer le PC sur une adresse paire (Align)
```

Pour insérer des blocs qui ont un contenu déterminé, on se sert de dcb.x. X peut être remplacé par b, w ou l. Exemple :

```
dcb.b 10,$ff      ; Insertion de 10 $FF octets
dcb.w 10,$ffff     ; Insertion de 10 $FF mots
dcb.l 10,$fffffff  ; Insertion de 10 $FF mots longs
```

Labels locaux

Les labels locaux sont caractérisés par un nombre suivi d'un signe \$. Ils se placent entre deux labels globaux. Par exemple :

```
Start:                   ;Label global
$2:    subq.l #1,d1
      move.l #100,d0
$1:    subq.l #1,d0
```

```

bne.s 1$           ;Label local
tst.l d1
bne.s 2$           ;Label local
rts
Fill:              ;Label global
move.l #100,d1
1$:   move.b d0,(a0)+
subq.l #1,d1
bne.s 1$           ;Label local
rts

```

Fonctions arithmétiques

Voici les fonctions arithmétiques disponibles en assembleur :

Add/Soustraction :	p.ex. move.b #(3+5),d0, move.b #(3-5),d0
Mul/Division :	p.ex. move.b #(10*2),d0, move.b #(10/2),d0
Décalage de bits:	p.ex. move.b #(1<<8),d0, move.b #(50>>2),d0
OU Logique:	p.ex. move.b #(1!2!8),d0
ET logique:	p.ex. move.b #(255&26),d0
OU exclusif:	p.ex. move.b #(10^23),d0
NON logique:	p.ex. move.b #((10!3)),d0

Les fonctions XREF et XDEF

De façon générale, la fonction XREF importe des définitions et des adresses en provenance d'autres modules pendant le processus de l'édition de liens. Ces modules sont ou bien des bibliothèques de liens (Linker-Libraries, à ne pas confondre avec les bibliothèques Amiga, comme la bibliothèque Exec), ou d'autres modules objets linkés. A l'aide des exemples qui vont venir ultérieurement, nous préciserons les circonstances d'utilisation de ces bibliothèques.

La fonction XDEF propose à d'autres modules les définitions ou les adresses indiquées, ce qui ne présente une certaine importance que dans le cas où il s'agit de lier ensemble plusieurs modules.

Pour utiliser des fonctions de bibliothèque comme Exec, DOS ou Intuition, il faut connaître leurs offsets. Pour y parvenir, il existe une méthode qui consiste à feuilleter à la fin de ce livre pour retrouver l'offset recherché. L'autre méthode consiste à demander à l'éditeur de liens de rechercher lui-même l'offset. Il faut pour cela cependant faire savoir à cet éditeur de liens quels sont les offsets souhaités. On devra dans ce cas importer les noms de fonction. C'est la commande XREF qui s'en charge. Avant toute fonction de bibliothèque ainsi importée, on devra placer un _LVO, pour que l'éditeur de liens retrouve les offsets dans ses définitions. Exemple :

```

XREF _LVOAllocMem,_LVOFreeMem

jsr _LVOAllocMem(a6)
jsr _LVOFreeMem(a6)

```

L'éditeur de liens se charge d'aller chercher les offsets correspondants. On peut se dispenser de taper plusieurs _LVO, comme vous le verrez lorsque nous décrirons les macros.

Assembler avec ASSEM

La création d'un programme en état de marche se fait en deux étapes. La première étape consiste à assembler le programme qu'on a écrit. Il faut ensuite le lier avec les bibliothèques souhaitées et d'autres fichiers objets.

C'est devenu la règle de donner à la source d'un programme en assembleur l'extension ".asm". Le module objet créé par l'assembleur reçoit l'extension ".o" ou ".obj". Cette extension n'existe plus après l'édition des liens.

Les appels suivants peuvent servir à transformer un programme source en un programme exécutable :

```
assem file.asm -o RAM:file.o -i include  
blink RAM:file.o to file library lib/amiga.lib
```

L'option "-o-" dans la première ligne indique à l'assembleur qu'il doit déposer le fichier objet dans la RAM sous le nom "file.o". Avec "-i", on indique l'endroit où se trouvent les fichiers Include correspondants.

En second lieu, on appelle l'éditeur de liens, auquel on transmet le fichier que l'on vient de créer : "RAM:file.o". Celui-ci se lie avec la bibliothèque Amiga, une bibliothèque personnelle ou une autre bibliothèque, et crée le programme définitif sous le nom de "file".

Lorsque vous importez des bibliothèques avec la fonction XREF, le linkage avec la bibliothèque Amiga est absolument nécessaire.

2.4.2. L'utilisation de macros

Une macro consiste en une ou plusieurs instructions réunies pour former un ensemble (commandes assembleur ou octets de données); ces commandes peuvent être insérées à l'intérieur d'un programme par appel du nom de la macro.

Lorsqu'une succession de commandes toujours identiques revient souvent dans un programme, on peut les rassembler à l'intérieur d'une macro. En introduisant le nom de la macro dans le programme, on insère en fait la suite des commandes au moment de l'assemblage. On se dispense ainsi d'un travail de dactylographie parfois important.

Un exemple : supposons que cette suite de commandes revienne très souvent :

MOVE.L A2,A0	:Premier paramètre vers A0
MOVE.L A4,A1	:Second paramètre vers A1
BSR SeraAinsi	:Appel de fonction
TST.L D0	:Test du résultat

Dans ce cas, il n'est pas inutile de définir une macro. Une macro aura toujours la même forme :

```
Nom de la macro      MACRO
Commandes
ENDM
```

MACRO est le code qui signale à l'assembleur le début d'une macro. ENDM signale la fin de la macro. Exemple :

```
APPELFONCTION MACRO
  MOVE.L A2,A0          ;Premier paramètre vers A0
  MOVE.L A4,A1          ;Second paramètre vers A1
  BSR    SeraAinsi       ;Appel de fonction
  TST.L  00              ;Test du résultat
ENDM
```

Chaque fois que cette succession de commandes doit apparaître dans le programme, on peut la remplacer par le nom de la macro (APPELFONCTION).

Le fait de rassembler plusieurs commandes dans une macro ne représente cependant qu'une petite partie des possibilités offertes par les macros. Grâce à la transmission des paramètres, les macros ont en fait bien plus de pouvoir que notre exemple ne l'a montré. Le nom de la fonction à appeler peut constituer un paramètre de ce type. Le paramètre transmis est adressé avec un "\ ", suivi d'un nombre indiquant le numéro du paramètre. Exemple :

```
APPELFONCTION MACRO      ;1er paramètre - nom de la fonction
  MOVE.L A2,A0
  MOVE.L A4,A1
  JSR    \1                ;Appel de la fonction
  TST.L  00
ENDM
```

Avec

```
APPELFONCTION FonctionXY
```

on reporte "JSR FonctionXY" au lieu de "JSR \1", et l'assemblage se fait en conséquence.

Si l'on indique plusieurs paramètres, ceux-ci doivent être séparés par des virgules dans l'appel de la macro. A l'intérieur de celle-ci, les paramètres sont pris en compte dans l'ordre de la transmission avec "\1", "\2", "\3", etc. S'il y a des boucles à l'intérieur de la macro, il faut faire encore attention à un détail.

Exemple de macro pour remplir un secteur en mémoire :

```
FILL    MACRO      ;Valeur de remplissage, nombre, registre
          move.w  #\1,d0
          move.l  #\2,d1
Loop@   move.w  d0,( \3)+ 
          subq.l #\1,d1
          bne     Loop@ 
ENDM
```

L'utilisation du signe "\@" derrière le label "Loop" est indispensable. Supposons que le label figure dans la macro sans ce signe particulier, et que l'on utilise la macro à trois reprises. A chaque fois, le code macro indiqué serait inséré par l'assembleur dans le code programme. On aurait dans ce cas à trois endroits différents du programme le label "Loop" et la commande "bne Loop". L'assembleur réclamerait aussitôt, puisque le même label a été utilisé plusieurs fois.

C'est pourquoi il faut ajouter "\@" au nom du label. A chaque nouvel appel de la macro, un nouveau nombre sera alors mis en oeuvre, et au premier appel le label s'appellera "Loop0.000", au second appel il s'appellera "Loop0.001", et au troisième "Loop0.002", etc.

Lorsqu'on utilise ce type de macros, il faut prendre garde au fait que le label à l'intérieur de la macro vaut comme label global, et que l'on ne peut donc pas y accéder avec un label local.

Outre les paramètres transmis, on peut effectuer dans une macro un assemblage conditionnel. Ce veut dire que l'on donne certaines conditions, et la macro a un aspect différent au moment d'être intégré dans le programme, selon que ces conditions sont satisfaites ou non.

Une condition commence avec une instruction IF et se termine avec ENDC (End Condition = Fin de la condition). Dans les conditions énoncées, le résultat d'une opération arithmétique est comparé à 0.

Voici les conditions IF disponibles :

IFEQ AG1-AG2

Poursuivre si les arguments sont égaux (equal).

IFNE AG1-AG2

Poursuivre si les arguments sont différents (not equal).

IFGT AG1-AG2

Poursuivre si AG1 > AG2 (greater than).

IFGE AG1-AG2

Poursuivre si AG1 >= AG2 (greater or equal).

IFLE AG1-AG2

Poursuivre si AG1 <= AG2 (lower or equal).

IFC STR1,STR2

Poursuivre si String 1 = String 2 (copy of).

IFNC STR1,STR2

Poursuivre si String 1 \leftrightarrow String 2 (no copy of).

IFD Label

Poursuivre si Label est déjà défini (if defined).

IFND Label

Poursuivre si Label n'est pas défini (not defined).

Les expressions comme "IFGE(2*\1)+10-(3*\2)" sont également autorisées. En rapport avec les conditions, on peut interroger avec NARG le nombre d'arguments transmis. Avec MEXIT, on peut mettre fin à la macro avant terme. Pour finir cette section sur les macros, nous allons encore expliquer leur utilisation en partant de quelques exemples.

2.4.2.1. Exemples de macros

Appel de fonctions de bibliothèque sans indication de _LVO; exécuté par la macro suivante :

```
CALLSYS MACRO
    IFGT NARG-1
        FAIL    !!! CALLSYS MACRO - trop de paramètres!!!
        ENDC
        JSR     _LVO\1(A6)
    ENDM
```

Appel:

```
move.l $4,a6
CALLSYS AllocMem
```

Appel d'une fonction de bibliothèque, le pointeur sur la bibliothèque ne se trouvant pas en A6, Il faut faire savoir ici à la macro où il doit aller chercher l'adresse de base :

```
LINKSYS MACRO
    IFGT NARG-2
        FAIL    !!! LINKSYS MACRO - trop de paramètres!!!
        ENDC
        MOVE.L A6,-(SP)
        MOVE.L \2,A6
        CALLSYS \1
        MOVE.L (SP)+,A6
    ENDM
```

Appel:

```
LINKSYS AllocMem,ExecBase
```

Macro servant à éviter d'écrire le _LVO quand on importe une fonction de bibliothèque :

```
XLIB:    MACRO
        XREF _LVO\1
    ENDM
```

Appel :

```
XLIB AllocMem  
XLIB CreateProc
```

au lieu de :

```
XREF _LVOAllocMem,_LVOCREATEPROC
```

Tests pour voir si une liste est vide ou non. Si l'on transmet un paramètre (un registre) au moment de l'appel de la macro, le pointeur sur la liste est censé se trouver dans ce registre. Si l'on n'indique aucun paramètre, le pointeur sur la liste doit se trouver en A0.

```
TSTLIST MACRO * [list]  
    IFC '\1',''  
        CMP.L LH_TAIL+LN_PRED(A0),A0  
    ENDC  
    IFNC '\1',''  
        CMP.L LH_TAIL+LN_PRED(\1),\1  
    ENDC  
ENDM
```

Appel :

```
TSTLIST
```

ou

```
TSTLIST A3
```

La macro BITDEF

Pour pouvoir poser un bit déterminé dans une constante, il faut entrer la construction suivante :

```
NuméroBit EQU 3  
MyBit_Bit EQU 1 << NuméroBit
```

Le fichier Include "exec/types.i" propose ici une macro utile (BITDEF).
Exemple :

```
BITDEF MyBit,Bit,3
```

et l'on obtient en retour :

```
MyBitB_Bit = 3  
MyBitF_Bit = %00001000
```

```
BITDEF MACRO * prefix,&name,&bitnum  
    BITDEFO \1,\2,B_,\3  
    \@BITDEF SET 1<<\3  
        BITDEFO \1,\2,F_,\@BITDEF  
    ENDM  
  
BITDEFO MACRO * prefix,&name,&type,&value  
    \1\3\2 EQU \4  
    ENDM
```

2.4.3. Utilisation des fichiers Include

Les fichiers Include sont des fichiers dans lesquels ont été déposées des définitions en grand nombre, disponibles par intégration de ces fichiers à l'intérieur des programmes écrits par vos propres soins.

L'utilisation des fichiers Include est une chose acquise pour tous ceux qui ont déjà programmé en C. En assembleur, on dispose des mêmes possibilités. Cela ne veut pas dire, néanmoins, que les fichiers Include du compilateur C peuvent être utilisés pour l'assembleur. On peut cependant faire usage de fichiers comparables.

Les structures prédéfinies que l'on connaît en C présentent un aspect différent dans les fichiers Include en assembleur, mais leur principe est le même. Pour les utiliser, on dispose de quelques macros très utiles, qui se trouvent dans le fichier Include "exec/types.i", et qui sont indispensables pour utiliser les structures.

Comparons d'abord avec la structure "List" en C :

```
struct List {
    struct Node *lh_Head;
    struct Node *lh_Tail;
    struct Node *lh_TailPred;
    UBYTE      lh_Type;
    UBYTE      lh_pad;
};
```

A l'aide de nos macros, nous n'aurons aucun problème pour écrire l'équivalent de cette structure. Voici comme elle se présentera en assembleur :

```
STRUCTURE LH.0
APTR   LH_HEAD
APTR   LH_TAIL
APTR   LH_TAILPRED
UBYTE  LH_TYPE
UBYTE  LH_pad
LABEL  LH_SIZE
```

Comment cela est-il rendu possible ? STRUCTURE, APTR et LABEL sont des appels de macro avec des transmissions de paramètres qui leur correspondent. A l'aide des macros, on affecte une valeur au label adéquat. Cette valeur correspond à l'offset de l'entrée à l'intérieur de cette structure.

Voyons maintenant comment se présentent ces macros :

```
STRUCTURE MACRO
\1      EQU     0
SOFFSET  SET     \2
ENDM
```

On appelle en premier la macro STRUCTURE. On lui transmet deux valeurs. La première valeur est un label (LH) et la seconde un nombre. La macro confère au label LH la valeur nulle. De plus, le label SOFFSET reçoit la valeur du second paramètre

transmis. Puisqu'on utilise ici la commande SET (cf. les indications sur l'utilisations de l'assembleur), on pourra réutiliser ensuite la valeur de SOFFSET.

Arrêtons-nous un instant là-dessus :

```
LH      - 0
SOFFSET - 0
```

On appellera ensuite la macro APTR :

```
APTR      MACRO
\1        EQU      SOFFSET
SOFFSET   SET      SOFFSET+4
          ENDM
```

La macro reçoit le nom de label LH_HEAD.

Comme on peut le voir à la forme de la macro, ce label reçoit à son tour la valeur nulle (SOFFSET = 0). SOFFSET est augmenté de 4 unités.

Lors d'un nouvel appel de APTR avec transmission du label LH_TAIL, celui-ci reçoit la valeur 4. SOFFSET est augmenté à 8. Le label LH_TAILPRED reçoit ensuite la valeur 8 et SOFFSET la valeur 12.

La macro suivante se nomme UBYTE :

```
UBYTE     MACRO
\1        EQU      SOFFSET
SOFFSET   SET      SOFFSET+1
          ENDM
```

Par l'appel de cette macro, LH_TYPE reçoit la valeur 12, et SOFFSET la valeur 13. "LH_pad" reçoit la valeur 13 et SOFFSET la valeur 14.

En dernier lieu, examinons encore la macro LABEL :

```
LABEL     MACRO
\1        EQU      SOFFSET
          ENDM
```

Ici, le label reçoit uniquement la valeur inchangée de SOFFSET. Une fois que toutes les macros ont été appelées, voici ce qu'on obtient :

```
LH      - 0
LH_HEAD - 0
LH_TAIL - 4
LH_TAILPRED - 8
LH_TYPE - 12
LH_pad  - 13
LH_SIZE - 14
```

On comprend donc quel est l'usage de ces macros. Chaque label transmis à la macro a reçu l'offset convenable, celui de l'entrée qui lui correspond dans la structure. La valeur dont SOFFSET est augmentée est à chaque fois la longueur de l'entrée correspondante. Avec APTR (pointeur de 4 octets), la valeur ajoutée est 4, tandis qu'avec Byte (longueur 1), la valeur est 1.

Il ne reste plus qu'à expliquer à quoi sert la macro **LABEL**. Avec cette macro, la valeur SOFFSET actuelle est transférée sur le label indiqué. Si cela se passe à la fin d'une structure ainsi définie, c'est automatiquement la longueur de la structure. L'accès à cette structure ne pose plus de problème. En C, elle aurait l'aspect suivant :

```
#include <exec/lists.h>

main ()

    struct List MyList;
```

```
    MyList.lh_Type = 2;
```

Et donc en assembleur :

```
include "exec/types.i"
include "exec/lists.i"

lea      Listadresse.a0    ;Adresse, à partir de laquelle figure la
structure
move.b #2,LH_TYPE(a0)     ;entrée du type
```

Si l'on veut par exemple définir un secteur en mémoire pour une structure, on n'a pas besoin de se fatiguer à rechercher la longueur. On peut s'y prendre désormais très simplement :

```
move.l #LH_SIZE,d0          ;Longueur de la structure List en D0
CALLSYS AllocMem            ;Cf. exemples de macros
```

Supposons que ce soit une sous-structure d'une autre structure. Ce problème est résolu à son tour très facilement :

```
STRUCTURE MyStruct.0
APTR QuiSaitOu
UBYTE UneValeur
STRUCT MaListe,LH_SIZE
UBYTE Compteur
LABEL MyStruct_SIZE
```

A l'intérieur de la structure, nous avons un nouvel appel de macro :

```
STRUCT    MACRO
\1        EQU      SOFFSET
SOFFSET   SET      SOFFSET+\2
ENDM
```

On peut transmettre deux paramètres à cette macro. Il s'agit d'une part du label auquel on affecte l'offset, et d'autre part de la longueur de la structure à insérer.

SOFFSET est augmenté d'une valeur égale à la longueur de la structure, ce qui signifie que l'on transmettra au label suivant l'offset qui se trouve derrière la structure. Dans notre exemple, on insère une structure de liste, ce qui est reconnaissable au fait que l'on reporte une longueur de 14 octets (LH_SIZE = longueur de la structure de liste = 14). Si l'on veut alors accéder à la structure de liste, voici comment on s'y prend :

```
lea      MaStructure.a0    ;Pointeur sur l'adresse de base  
lea      MaListe(a0),a0    ;Pointeur sur le début de la liste  
move.b #2,LH_TYPE(a0)    ;Reporte le type
```

Ou ainsi:

```
lea      MaStructure+MaListe,a0  
move.b #2,LH_TYPE(a0)
```

Voici les macros disponibles pour constituer des structures :

```
STRUCTURE Label,Offset  
STRUCT   Label,Struct_SIZE  
  
BYTE    Longueur = 1 octet  
UBYTE   Longueur = 1 octet  
WORD    Longueur = 2 octets  
UWORD   Longueur = 2 octets  
SHORT   Longueur = 2 octets  
USHORT  Longueur = 2 octets  
BOOL    Longueur = 2 octets  
LONG    Longueur = 4 octets  
ULONG   Longueur = 4 octets  
FLOAT   Longueur = 4 octets  
APTR    Longueur = 4 octets  
CPTR    Longueur = 4 octets  
RPTR    Longueur = 2 octets (pour offsets)
```

Dans les fichiers Include, toutes les structures utiles pour C sont également converties en structures assembleur. Les noms des entrées changent souvent de graphie (Majuscule/minuscule). Indépendamment des structures, on trouve également dans les fichiers Include d'autres macros et définitions très utiles.

2.4.4. Remarques pour une programmation "élégante"

La plupart des conseils que nous allons donner dans cette section proviennent des programmeurs du système d'exploitation de l'Amiga. On ne pourra certes pas les soupçonner de ne pas être des programmeurs professionnels, et de ne pas avoir d'expérience en la matière.

Lorsqu'on programme, il faut partager les registres du processeur en deux groupes. On prendra d'une part les registres D0, D1, A0 et A1, et on mettra tous les autres registres dans l'autre groupe. Ces groupes entrent en considération lorsqu'on a besoin de passer dans une sous-routine.

Le système d'exploitation (sauf les parties BCPL, par exemple DOS) est toujours programmé de telle manière que les registres du premier groupe (D0, D1, A0, A1) puissent être utilisés comme paramètres à transmettre pour la fonction appelante. Si ces registres ne sont pas suffisants, on en prendra d'autres.

Le programme appelant ne doit pas supposer que ces registres du premier groupe conserveront leur valeur après appel de la fonction. Au contraire, les registres du second

groupe sont ceux dont la valeur reste inchangée. Si le sous-programme utilise les registres du second groupe, il faut sauver leur contenu avant utilisation, et le restaurer lorsqu'on quitte le sous-programme.

De cette façon, on peut élaborer des boucles avec les registres D2 à D7, sans être obligé de se préoccuper du fait que d'éventuelles sous-routines viendront mettre "des bâtons dans les roues".

Si l'on se tient strictement à ce principe, on ne risque pas de détruire des registres encore nécessaires à cause d'une routine écrite il y a longtemps, et dont on ne se rappelle plus bien le schéma de fonctionnement.

Il arrive souvent, lorsqu'on programme, qu'une fonction figurant dans une partie du programme ne puisse être utilisée qu'en partie. Beaucoup de programmeurs ont recours, dans un pareil cas, à une méthode très peu saine. Ils définissent un second label à l'intérieur de la fonction et ils sautent par ce label au milieu de la fonction. On peut bien sûr le faire, mais ce n'est pas du meilleur style. Le programme devient de cette façon vite incompréhensible, non seulement pour les autres, mais aussi pour soi-même. La fonction n'est pas définie clairement.

Exemple :

```

jsr      Fonction principale
.
.
.
jsr      Sous-fonction
.
rts

Fonction principale :      ;Fonction appelée
.
.
.

Sous-fonction :           ;Label inséré
.
.
.

rts                  ;Fin de la fonction

```

Il est bien plus élégant de partager la fonction principale en deux fonctions plus petites, et d'appeler l'une d'elles de manière séparée.

Exemple :

```

jsr      Fonction principale
.
.
.
jsr      Sous-fonction
.
rts

Fonction principale :
.

```

```
.jsr      Sous-fonction
rts

Sous-fonction :
.
.
rts
```

Cette façon de structurer un programme présente le grand avantage de dégager clairement la tâche de chaque fonction, ce qui rend le programme beaucoup plus lisible.

Dans presque tous les programmes en assembleur, il faut déposer des valeurs et des adresses globales. On prendra pour exemples les pointeurs sur les bibliothèques, sur les fenêtres, ou d'autres structures analogues. On peut déposer pour cela des labels à la fin du programme. Si pour l'un de ces labels, le pointeur sur la bibliothèque DOS doit par exemple être conservé en mémoire, le programme ne pourra pas être rendu relocalisable (indépendant de son emplacement en mémoire), ou seulement avec beaucoup de difficulté. Il n'est pas possible d'écrire de cette façon des programmes réentrant, ce qui est souvent important. On appelle "réentrant" un programme dont le code est traité simultanément par plusieurs tâches.

Un bon moyen pour y remédier est de créer une structure (on en a déjà parlé) dont tous les pointeurs sont globaux. Une telle structure peut prendre l'aspect suivant :

```
STRUCTURE Globals,0
  APTR  g1_SysBase
  APTR  g1_DosBase
  APTR  g1_WindowPtr
  WORD   g1_Counter
.
.
LABEL  g1_SIZE
```

Le secteur de mémoire nécessaire pour cette structure ne demande pas en général une place excessive, de telle sorte que la structure peut être placée confortablement sur la pile.

Votre programme ne peut commencer que de la façon suivante :

```
LEA    -g1_SIZE(a7),a7 ;Créer de la place sur la pile
move.l a7,a5           ;Pointeur sur la structure en A5
```

En A5 se trouve maintenant le pointeur sur le début de notre structure. Si l'on va alors chercher le pointeur sur la bibliothèque DOS, il peut être placé en mémoire avec

```
move.l d0,g1_DosBase(a5)
```

Mais le contenu de A5 ne doit plus être modifié par mégarde. Grâce à l'accès aux variables globales indirectement par l'intermédiaire de A5, le code programme sera automatiquement rendu relocalisable et réentrant, puisque les "variables" ne sont pas déposées de manière globale, mais localement dans la pile qui leur est propre.

L'insertion d'une nouvelle variable ne présente pas de difficulté (il suffit de la reporter dans la structure, et c'est tout).

Si l'on veut mettre fin au programme, il faut rétablir la pile dans son état initial. Pour cela, il suffit d'écrire :

```
LEA gl_SIZE(a7),a7
```

Si la structure globale est trop longue, il n'est pas recommandé de la déposer sur la pile. Dans ce cas, il faut définir au début du programme l'emplacement correspondant en mémoire à l'aide de la fonction AllocMem.

Exemple :

```
move.l #MEMF_CLEAR,d1
move.l #gl_SIZE,d0
CALLSYS AllocMem
tst.l d0
beq _AllocMem
move.l d0,a5
```

En A5, on trouve à nouveau le pointeur sur la structure des variables globales.

Le préfixe "gl " a été introduit pour empêcher que certains noms des entrées de la structure n'apparaissent également dans d'autres structures. Il n'est pas nécessaire de le faire intervenir. Vous constaterez néanmoins que cela présente des avantages certains.

Documentation des programmes

Si vous avez programmé longtemps en assembleur, en écrivant déjà des programmes relativement longs, vous serez d'accord avec nous pour dire que les programmes doivent être "documentés".

Je me suis moi-même habitué à accompagner au minimum chaque fonction d'une phrase explicative, et à mentionner les paramètres d'entrée et de sortie. La façon dont ces mentions doivent être effectuées est certes affaire de goût. J'ai pris l'habitude de procéder de la façon suivant :

```
;:- signale un paramètre d'entrée
;- signale un paramètre de sortie
```

Ainsi, un en-tête de fonction peut prendre l'aspect suivant :

```
*****
;Fonction dans tel ou tel but
;:- A0 - Pointeur sur les données XY
;:- D0 - Nombre d'octets
;- D0 - Pointeur sur les nouvelles données/ Zéro -> Erreur
```

Lignes générales d'un programme Commodore

Il ne faut jamais pénétrer directement dans la ROM, comme le font certains programmeurs de virus. Les programmes ne fonctionnent dans ce cas qu'avec une

version déterminée de Kickstart. Or les versions de Kickstart se modifient. Utilisez les vecteurs bibliothèque, device ou resource.

Ne partez pas de l'adresse où se trouvent certaines structures. Cette adresse peut en effet se modifier d'ordinateur en ordinateur. On peut prendre ici encore comme contre-exemple certains programmeurs de virus, qui supposent que la pile système se trouve toujours à la même adresse fixe dans la RAM chip.

Lorsque nous indiquons dans ce livre des adresses fixes pour les structures (la structure ExecBase), cette information ne doit pas servir à autre chose qu'à la recherche des erreurs.

- ✓ N'utilisez pas de commandes assembleur privilégiées
- ✓ N'utilisez pas la commande assembleur TAS.

Les adresses doivent toujours être longues de 32 bits, même si seuls les 24 bits inférieurs sont actuellement utilisés. Sur les modèles futurs, ce ne sera plus le cas.

Ne lancez pas de programme dans votre pile. Sur les processeurs à venir, il est possible que les secteurs de données (et donc également la pile) soient placés à des adresses impaires. Les programmes doivent, eux, se trouver dans tous les cas aux adresses paires.

N'utilisez pas de boucle dépendant du processeur pour contrôler les processus temporels. Pour assurer une attente d'une durée déterminée, il faut utiliser le device Timer.

Nous donnerons pour exemple ici une boucle d'attente concernant le mouvement du moteur sur les lecteurs de disquettes. Beaucoup de programmes de copie exécutent cette attente avec des boucles d'attente, qui se révèlent bien trop courtes lorsqu'on dispose d'une carte turbo.

Comme le hardware se modifie constamment, il n'est pas assuré que des programmes travaillant directement avec le hardware fonctionneront encore sur les versions ultérieures.

2.4.5. Utilisation des bibliothèques de liens

Si l'on veut programmer de manière efficace, on ne doit pas réécrire à chaque fois les sous-fonctions qui reviennent souvent. Il suffit de les lier à des fonctions déjà écrites dans le nouveau programme sous forme de modules tout prêts. On peut le faire par exemple en insérant les fonctions déjà écrites en tant que code source dans le nouveau programme.

Cette méthode est possible, mais elle n'est pas particulièrement élégante. Il vaut bien mieux demander à l'éditeur de liens ('ln' chez Manx, mais le plus souvent 'blink') de s'occuper de l'insertion, en prenant les modules dans une bibliothèque de liens personnelle.

Dans certains des exemples qui suivent, on a effectivement recours à une bibliothèque de liens personnelle. Pour que vous puissiez comprendre et utiliser les programmes donnés en exemple, les modules utilisés dans ces exemples ont été reproduits.

Description des fonctions :

Fonction : _CreatePort

La fonction génère un port de message, qui transmet un signal à la tâche qui lui propre. Le port peut être indiqué comme port global.

Paramètres

Registre A0

Pointeur sur le nom du port. S'il n'y a pas de nom transmis ($A0 = 0$), le port ne figure pas dans la liste globale des ports.

Registre D0

Priorité du port. Importe uniquement si le port figure comme port global.

Résultat

Registre D0

Pointeur sur le port qu'on vient de créer, ou nul si aucun port n'a pu être créé.

```
:Fonction pour créer un port de message :
;=> A0 - Pointeur sur le nom du port (nécessaire uniquement pour les
ports globaux)
;=> D0 - priorité du port (nécessaire uniquement pour les ports globaux)
;=> D0 - Pointeur sur le port. S'il n'y pas de port créé, D0 = Zéro
```

```
include "exec/types.i"
include "exec/ports.i"
include "exec/memory.i"
include "exec/execbase.i"
```

```
CALLSYS MACRO
    jsr _LVO\1(a6)
    ENDM
XLIB MACRO
    XREF _LVO\1
    ENDM
```

:La fonction est mise à la disposition d'autres programmes
:grâce à l'option XDEF.

```
XDEF _CreatePort
XREF _AbsExecBase
```

```

XLIB AllocMem
XLIB FreeMem
XLIB AllocSignal
XLIB AddPort

_CreatePort:
    movem.l d2/a2-a3/a6,-(a7) ;sauver le registre
    move.w d0,d2 ;priorité en D0
    move.l a0,a2 ;Pointeur sur le nom
    move.l _AbsExecBase,a6 ;ExecBase en A6

;les indications sur le mode de la mémoire à définir
;:'MEMF_PUBLIC' et 'MEMF_CLEAR' se trouvent à l'intérieur
;du fichier Include 'exec/memory.i'.

        move.l #MEMF_PUBLIC!MEMF_CLEAR,d1 ;Description de la mémoire

;La taille de la structure de port est déterminée par l'intermédiaire
;de la structure 'MP', définie à l'intérieur
;du fichier Include 'exec/ports.i'

        move.l #MP_SIZE,d0 ;Taille de la structure de port

;CALLSYS est une macro qui a été définie au début du programme

    CALLSYS AllocMem ;définit un emplacement en mémoire
    tst.l d0 ;test: y a-t-il eu de la mémoire définie ?
    beq.s 1$ ;Non, alors erreur
    move.l d0,a3 ;Pointeur sur la mémoire pour le port
    move.l #-1,d0 ;Valeur pour un signal quelconque
    CALLSYS AllocSignal;Définir le signal pour le port
    tst.b d0 ;A-t-on pu définir un signal pour le port?
    bpl.s 2$ ;Oui, tout va bien
    move.l a3,a1 ;Sinon, rechercher le pointeur sur la mémoire
    move.l #MP_SIZE,d0;Taille de la mémoire en d0
    CALLSYS FreeMem ;Libère la mémoire
    clr.l d0 ;supprimer D0 (message d'erreur)
    bra.s 1$ ;Saut à la fin

;Grâce aux fichiers Include indiqués au début, le fichier
;'exec/nodes.i' est lu lui aussi automatiquement. On définit
;ici les entrées 'LN_NAME', 'LN_PRI', 'LN_TYPE' et 'NT_MSGPORT'.

2$:      move.l a2,LN_NAME(a3) ;raporte le pointeur sur le nom
        move.b d2,LN_PRI(a3) ;indique la priorité
        move.b #NT_MSGPORT,LN_TYPE(a3) ;indique le type

;Lorsqu'un message parvient au port, il faut poser un signal
;dans la tâche. Pour cela, le flag MsgPort doit être posé
;en conséquence. PA_

        move.b #PA_SIGNAL,MP_FLAGS(a3)
        move.b d0,MP_SIGBIT(a3)

;Pour obtenir le pointeur sur la tâche propre, on peut d'une part la
;rechercher avec la fonction 'FindTask', mais il est plus rapide de la
;lire dans la structure ExecBase. Pour pouvoir réaliser cette dernière
;opération, on doit chercher l'offset de cette entrée dans le fichier
;Include 'exec/execbase.i'. L'offset est placé dans 'ThisTask'.

```

```

move.l    ThisTask(a6),MP_SIGTASK(a3) ;Chercher pointeur sur
                                         ;la tâche

;Pour que notre port puisse "conserver" des messages sous forme de liste,
;il faut initialiser en lui une telle structure de liste. Ceci
;se fait par l'intermédiaire d'une macro, que l'on fait figurer dans
;le fichier Include 'exec/lists.i'. Ce fichier est automatiquement
;lu à l'ouverture du fichier 'exec/ports.i'. La macro s'appelle
;NEWMEST, et elle initialise une structure de liste. Comme paramètre,
;on doit transmettre le pointeur sur la structure de liste
;dans un registre d'adresse.

      lea      MP_MSGLIST(a3),a1 ;Pointeur sur structure de liste
      NEWLIST   A1                 ;Appel de macro
      move.l    a2,d0              ;Pointeur sur le nom du poste
      beq.s     3$                ;embranchement, s'il n'y a pas de
                               ;nom indiqué
      move.l    a3,a1              ;Pointeur sur la structure de port
      CALLSYS   AddPort           ;fait figurer le port dans la
                               ;liste globale
      3$:      move.l    a3,d0          ;Pointeur sur le port en D0
      1$:      movem.l   (a7)+,d2/a2-a3/a6 ;Recherche le registre
      rts                  ;Saut en arrière

      END

```

Fonction : _DeletePort

La fonction supprime un port de message du système. S'il s'agit d'un port global, il est d'abord supprimé dans la liste des ports globaux.

Paramètres

Registre A0

Pointeur sur le port de message à supprimer.

;Fonction servant à supprimer un port de message:

;=> A0 = Pointeur sur le port

```

include "exec/types.i"
include "exec/ports.i"
include "exec/memory.i"

CALLSYS  MACRO
        jsr _LVO\1(a6)
        ENDM
XLIB     MACRO
        XREF _LVO\1
        ENDM

```

```
;La fonction est mise à la disposition d'autres programmes
;par l'intermédiaire de l'option XDEF

XDEF _DeletePort

XREF _AbsExecBase
XLIB FreeMem
XLIB FreeSignal

_DeletePort:
    movem.l  a2/a6,-(a7)      ;Sauver le registre
    move.l   a0,a2            ;Pointeur sur Port
    move.l   _AbsExecBase,a6 ;ExecBase vers A6

    move.l   LN_NAME(a2),d0  ;fait figurer le nom
    beq.s    1$                ;non, pas global

    move.l   a2,a1

;'REMOVE' est une macro que l'on trouvera dans le fichier Include
;:'exec/lists.i'. Cette macro supprime une structure Node dans
;la liste correspondante. Il s'agit de la même fonction que la fonction
;Exec 'Remove'.

    REMOVE           ;Supprimer le port dans la liste des ports

1$:    move.b  MP_SIGBIT(a2),d0
        ext.w   d0
        ext.l   d0
        CALLSYS FreeSignal     ;Définit le signal pour le port

;La taille de la structure de port est déterminée par l'intermédiaire
;de la structure 'MP', définie à l'intérieur du fichier Include
;:'exec/ports.i'. La taille est déposée dans 'MP_SIZE'.

    move.l   #MP_SIZE,d0      ;Taille de la structure de port
    move.l   a2,a1            ;Pointeur sur le port

;CALLSYS est une macro qui a été créée au début du programme

    CALLSYS FreeMem          ;Libère la mémoire
    movem.l  (a7)+,a2/a6      ;Rétablit le registre
    rts                      ;Saut en arrière

    END
```

Fonction : CreateIO

La fonction crée une structure de message de taille quelconque, et la fait figurer dans le port de message indiqué comme Reply-Port.

Paramètres

Registre A0

On transmet ici le pointeur sur le port qui figure comme Reply-Port.

Registre D0

On indique ici la longueur de la structure à créer.

Résultat

Registre D0

Pointeur sur la structure de message créée, ou Zéro si la structure n'a pas pu être créée.

Fonction : CreateStdIO

La structure correspond à la fonction _CreateIO, mais cette fois la longueur d'une structure IOStdRequest est adoptée automatiquement.

Paramètres

Registre A0

Ici est transmis le pointeur sur le port qui figure comme Reply-Port.

Résultat

Registre D0

Pointeur sur la structure IOStdRequest créée ou sur zéro, si la structure n'a pas pu être créée.

Fonction : _DeleteIO / _DeleteStdIO

Ces fonctions suppriment une structure de message créée précédemment (donc aussi une structure IO). La longueur est reconnue automatiquement.

Paramètres

Registre A0

Pointeur sur la structure de message ou la structure IO.

```

include "exec/types.i"
include "exec/io.i"
include "exec/memory.i"

CALLSYS MACRO
    jsr _LVO\1(a6)
    ENDM
XLIB MACRO
    XREF _LVO\1
    ENDM

;La fonction est mise à la disposition d'autres programmes par
;l'intermédiaire de l'option XDEF.

XDEF _CreateIO
XDEF _CreateStdIO

XDEF _DeleteIO
XDEF _DeleteStdIO

XREF _AbsExecBase
XLIB AllocMem
XLIB FreeMem



---


;<-- D0 - Taille de la structure IO
;<-- A0 - Pointeur sur le port de message

;--> D0 - Pointeur sur la structure IO ou zéro -> erreur

_CreateIO:
    movem.l  d2/a2/a6,-(a7)      ;Sauver le registre
    move.l   a0,a2                ;Pointeur sur le port
    move.l   d0,d2                ;Longueur de la structure
    move.l   _AbsExecBase,a6      ;Pointeur sur ExecBase
    move.l   a0,d0                ;un port est indiqué
    beq.s    1$                   ;Erreur, pas de port
    move.l   d2,d0                ;Transmission de la longueur
    move.l   #MEMF_PUBLIC!MEMF_CLEAR,d1 ;Indication de la mémoire
    CALLSYS AllocMem              ;Définition de la mémoire
    tst.l    d0                   ;La mémoire a été définie
    beq.s    1$                   ;Pas de mémoire définie
    move.l   d0,a1                ;Pointeur sur la mémoire

```

```

move.b    #NT_MESSAGE,LN_TYPE(a1)      ;Fait figurer le type
move.w    d2,MN_LENGTH(a1)            ;Fait figurer la longueur
move.l    a2,MN_REPLYPORT(a1)         ;Fait figurer le pointeur
                                         ;sur le port
1$:      movem.l  (a7)+,d2/a2/a6     ;Rétablit le registre
                                         ;Saut en arrière
                                         ;-----  

;:- A0 - Pointeur sur le port de message

_CreateStdIO:
move.l    #IOSTD_SIZE,d0           ;Longueur de la structure StdIO
bsr      _CreateIO                 ;Créer une structure
rts

                                         ;-----  

;:- A0 - Pointeur sur la structure IORequest

_DeleteIO:
_DeleteStdIO:
move.l    a6,-(a7)
move.l    _AbsExecBase,a6
move.w    MN_LENGTH(a0),d0
ext.i    d0
move.l    a0,a1
CALLSYS  FreeMem
move.l    (a7)+,a6
rts

END

```

2.5. L'utilisation des tables de fonctions

Cette section concerne aussi bien les programmeurs en C que ceux qui emploient l'assembleur. La compréhension de cette section est essentielle si on veut utiliser toutes les possibilités de l'Amiga.

Une Library est une bibliothèque de fonctions, qui pourra être utilisée par le programmeur. Elle est constituée d'une série d'instructions auxquelles on accède par des sauts. Ces bibliothèques sont employées par le programmeur, aussi bien en C qu'en assembleur, lorsque certaines fonctions ou instructions ne sont pas disponibles dans ces langages. Par exemple, la fonction OpenScreen() sert à établir un écran (Screen). En C, cette instruction n'est pas présente et elle ne pourra être exécutée qu'avec l'aide d'une bibliothèque de l'Amiga.

Les bibliothèques de l'Amiga sont subdivisées selon la tâche à accomplir. La plupart d'entre elles se trouvent déjà dans le secteur ROM de l'ordinateur. Certaines doivent cependant être chargées en cas de besoin à partir d'une disquette.

Parmi les bibliothèques, Exec occupe une position spéciale, car ses fonctions sont accessibles tout de suite après un reset. Si l'on veut utiliser une fonction d'une autre

bibliothèque, on doit le faire savoir au système, qui ouvre alors la bibliothèque en question, pour que l'on puisse y accéder. Si cette bibliothèque existe déjà, un message indique au programmeur où il peut la trouver.

La seule chose connue d'une bibliothèque est normalement son adresse de base. "Avant" cette adresse se trouvent les accès à la fonction qui permet de disposer de la bibliothèque. "Après" l'adresse, on trouve le secteur de données de la bibliothèque. Nous n'en parlerons pas ici. Ce secteur n'est intéressant, en effet, que si l'on veut créer une bibliothèque personnelle. C'est pourquoi nous ne l'envisagerons que dans un chapitre ultérieur.

Une bibliothèque a donc, par principe, l'aspect suivant :

```
JMP Fonction 1
JMP Fonction 2
JMP Fonction 3
Adresse de base
Secteur des données de la bibliothèque
```

La fonction souhaitée est atteinte à partir de l'adresse de base avec des offsets négatifs. Pour appeler une fonction, il suffit de connaître l'offset et l'adresse de base. L'avantage des bibliothèques est immédiatement sensible. Les développeurs du système d'exploitation peuvent l'étendre à volonté, et modifier la position de la fonction à l'intérieur de la ROM, sans que l'utilisateur de la bibliothèque en soit incommodé, car les offsets pour les fonctions de la bibliothèque demeurent identiques.

Un appel de bibliothèque, valide pour toutes les bibliothèques, se présente ainsi :

```
move.l LibBasis,a6      ;adresse de base de la bibliothèque
jsr Offset(a6)          ;appel d'une fonction avec un offset négatif
```

L'offset de la fonction correspondante doit être prélevé dans un tableau, que vous pourrez trouver à la fin de ce livre, en annexe. La meilleure méthode est cependant de demander à l'éditeur de liens de faire lui-même ce travail. Avant l'appel de la fonction, il faut penser à transmettre les paramètres (pour autant qu'il en existe) aux registres correspondants. L'adresse de base de la bibliothèque doit se trouver dans le registre A6, ce qui se fait automatiquement lorsqu'on fait usage d'un langage évolué.

Comme cela a déjà été dit, Exec occupe une position spéciale parmi les bibliothèques. L'adresse de base de la bibliothèque n'a pas besoin d'être communiquée par le système, on peut la lire immédiatement à l'emplacement \$4 en mémoire. En C, on y accède par lecture de la variable standard 'SysBase'.

La sélection d'une fonction Exec-Library en C, est assez simple : voici pour exemple l'appel de la routine FindName.

```
#include <exec/execbase.h>
struct ExecBase *SysBase;
struct Library *FindName(), *Library;

main()
{
```

```

library = FindName(&(SysBase->LibList), "dos.library");
printf ("\n %x \n", Library);
}

```

Ce petit programme C explore la liste des bibliothèques existantes à la recherche de la dos.library. Si elle est trouvée, son adresse sera émise. Si elle ne l'est pas, c'est un 0 qui sera émis.

Vous voyez qu'il n'y a rien de particulier à signaler, à part peut-être le risque de confondre les fonctions Exec lors de la sélection. Dans le chapitre sur la gestion des listes, on a vu qu'il faut communiquer à la fonction FindName un pointeur sur une liste et le nom du noeud (Node). La raison pour laquelle l'affectation de la liste présente cet aspect ne sera pas discuté ici. Nous y reviendrons dans la section consacrée à la structure ExecBase.

Nous allons voir maintenant comment le compilateur C traduit ce programme en assembleur. Lorsqu'on se limite à l'essentiel, le programme compilé a l'allure suivante :

```

global _SysBase,4
global _Library,4

_main:

pea *Name          ;déplacer le pointeur sur le nom dans la pile
move.l _SysBase,a6  ;pointeur sur ExecBase (à partir de $4)
pea 378(a6)        ;Déplacer le pointeur sur la liste dans la pile
jsr _FindName       ;sélection de la fonction FindName
move.l d0._Library  ;conserver la réponse dans la bibliothèque
move.l _library,-(a7) ;et la transmettre dans la pile
jsr _printf          ;à la routine _printf
rts

_FindName
movem.l 4(sp),a0/a1 ;charger les paramètres de FindName dans
                     ;les registres correspondants à partir de la pile
move.l _SysBase,a6  ;chercher l'adresse de base de la bibliothèque Exec
move.l -276(a6)     ;appel de la fonction Findname

```

Le listing complet en assembleur est un peu plus long, tout ce qui ne concernait pas l'appel des fonctions de bibliothèques ayant été éliminé.

_SysBase est un pointeur sur la bibliothèque Exec, ses fonctions étant accessibles par des offsets négatifs. Lorsqu'on accède à la mémoire avec des offsets positifs à partir de _SysBase, on obtient les valeurs du secteur de données d'ExecBase (structure générale du système d'exploitation), laquelle sera détaillée plus loin. Parmi les entrées d'ExecBase, on trouve également une structure de liste où les bibliothèques sont indiquées. Le résultat de la compilation est un peu compliqué, mais néanmoins facile à comprendre.

Le programme principal met les pointeurs de la liste et le nom dans la pile, et appelle ensuite le sous-programme. C'est par ce dernier que les paramètres seront chargés dans les registres prévus. Puis l'adresse de base de la bibliothèque sera écrite dans a6. Tout sera alors prêt pour la sélection de la fonction FindName de la bibliothèque Exec. L'offset de cette fonction est -276 (comme on l'a vu dans la section sur les listes). L'appel de la fonction se fera donc par l'instruction 'JMP -276(a6)'. Le paramètre obtenu en réponse

sera stocké dans 'Library', décalé dans la pile et affiché à l'écran avec l'instruction 'printf'. Le système et le compilateur C connaissent l'emplacement de la bibliothèque Exec, et ses fonctions peuvent donc être sélectionnées sans problème. Si on désire accéder à une autre bibliothèque, ni le système d'exploitation, ni le compilateur C ne savent au premier abord où elle se trouve.

2.5.1. Ouverture et fermeture d'une bibliothèque

Pour que le programmeur ne soit pas limité aux fonctions de la bibliothèque Exec, celle-ci dispose d'une fonction permettant d'obtenir les adresses de base des autres bibliothèques. Cette fonction se nomme 'OpenLibrary'. On transmet à la fonction le nom de la bibliothèque recherchée et le numéro de version. Il faut faire attention à la graphie (majuscules/minuscules). La règle veut que les noms qui interviennent dans le système d'exploitation soient écrits en minuscules.

Chaque bibliothèque est disponible dans une version bien définie. Pour que l'on puisse distinguer les versions, chaque bibliothèque dispose donc d'un numéro de version. Si vous écrivez un programme qui utilise les fonctions d'une bibliothèque existant seulement à partir d'une version déterminée, le numéro de version est transmis au moment de l'ouverture, et vous êtes sûr de cette façon que la seule bibliothèque ouverte sera celle qui possède ce numéro de version, ou un numéro supérieur. En général, la version n'intervient pas, et c'est alors un zéro qui est transmis comme numéro. La fonction OpenLibrary fournit l'adresse de base de la bibliothèque que vous voulez utiliser.

Si vous n'avez plus besoin de la bibliothèque, il faut le faire savoir au système. Vous le ferez par l'intermédiaire de la fonction CloseLibrary de Exec. Le pointeur sur la bibliothèque ouverte sera transmis à cette fonction. Il importe de fermer les bibliothèques, pour que le système puisse les supprimer de la mémoire dès qu'on ne s'en sert plus.

En assembleur, il est facile d'utiliser la bibliothèque que l'on vient d'ouvrir. Le pointeur sur la bibliothèque (l'adresse de base) s'écrit dans le registre A6, et l'on passe à la fonction correspondante par l'intermédiaire de l'offset.

En C, l'appel d'une fonction de la bibliothèque s'obtient simplement en utilisant le nom de la fonction. Au moment de l'édition des liens, la fonction souhaitée est prise dans la bibliothèque de linkage, et insérée dans le programme (la fonction Amiga est évidemment inconnue au langage C). Dans la fonction en question, les différents paramètres sont placés dans les registres correspondants, et la fonction de la bibliothèque Amiga est appelée. Pour que l'appel réussisse, il faut que le pointeur sur la bibliothèque Amiga se trouve dans le registre A6. Ce pointeur n'est cependant pas transmis avec l'appel de fonction. La fonction provenant de la bibliothèque de linkage utilise une variable standardisée pour l'initialisation du registre A6. Le pointeur sur la bibliothèque doit se trouver dans cette variable. Pour que la fonction de linkage puisse utiliser la variable, celle-ci doit auparavant avoir été définie dans le programme qu'on a écrit, sous forme de variable globale.

Liste des variables définies en C

clist.lib	CListBase
diskfont.lib	DiskFontBase
dos.lib	DosBase
expansion.lib	ExpansionBase
exec.lib	SysBase
graphics.lib	GfxBase
icon.lib	IconBase
intuition.lib	IntuitionBase
layers.lib	LayersBase
mathffp.lib	MathBase
mathieeedoubbas.lib	MathIeeeDoubBasBase
mathtrans.lib	MathtransBase
translator.lib	TranslatorBase

Avant d'illustrer l'ouverture et la fermeture d'une bibliothèque à l'aide d'exemples, nous allons expliciter les fonctions Exec correspondantes.

OpenLibrary

Fonction LibPtr = OpenLibrary(LibName, Version);
 d0 a1 d0

Offset -522 -\$228

Paramètres

LibName

Pointeur sur le nom de la bibliothèque qui doit être ouverte, par exemple "intuition.library" (le nom sera toujours terminé par un 0)

Version

Donne la version de la bibliothèque que l'utilisateur veut ouvrir. Si plusieurs des bibliothèques disponibles portent le même nom, elles seront distinguées par leurs numéros de version. Si une nouvelle version est exigée, alors qu'elle n'est pas encore disponible, la fonction 'OpenLibrary' échouera. Si l'on transmet un zéro pour numéro de version, la version n'est pas acceptée pour l'ouverture.

LibPtr

Contient, après l'appel de la fonction, l'adresse de base de la bibliothèque, si cette dernière a été trouvée. Sinon, c'est un 0 qui sera renvoyé.

CloseLibrary

Fonction CloseLibrary(library)
 A1

Offset -414 -\$19E

Paramètres

library

Pointeur sur la bibliothèque ouverte que l'on désire fermer.

Nous allons présenter un exemple en C pour l'ouverture et la fermeture d'une bibliothèque. Une fois que la bibliothèque a été ouverte, ses fonctions sont accessibles.

```
#include "exec/types.h"
#include "exec/libraries.h"

struct Library *DosBase, *OpenLibrary();

main()
{
    DosBase = (struct Library *) OpenLibrary("dos.library",OL);
    if (DosBase == 0)
    {
        printf ("\n Dos-Library introuvable \n");
        exit(0);
    }
    CloseLibrary(DosBase);
}
```

Comme nous venons de le dire, il faut définir en C une variable globale déterminée, pour que les fonctions de la bibliothèque puissent être utilisées. Il y a plusieurs façons de faire pour déclarer ces variables globales. Voici l'une des méthodes sur l'exemple de la Dos-library :

```
struct Library *DosBase;
```

Pour que le compilateur sache comment est faite la structure de la bibliothèque, il faut importer celle-ci par l'intermédiaire du fichier Include "exec/libraries.h". Etant donné que la plupart des programmeurs en assembleur, en début de carrière, ne travaillent pas avec les assembleurs que nous avons mentionnés, mais le plus souvent avec l'assembleur Seka, certainement plus simple à manipuler, nous allons d'abord donner un exemple d'ouverture immédiatement utilisable sur n'importe quel assembleur (et donc en particulier sur le Seka). Nous montrerons ensuite quelle forme on peut donner à

l'ouverture de la bibliothèque, en mettant en oeuvre les capacités des assembleurs que nous avons recommandés.

```

OpenLibrary      EQU -552
CloseLibrary    EQU -414

main:
    move.l $4,a6          ;Pointeur sur Exec-Library
    lea    LibName(pc).a1   ;Pointeur sur le nom de la bibliothèque
    clr.l d0                ;version 0 → pas de version
    jsr    OpenLibrary(a6)  ;Ouvrir Library
    move.l d0.LibBase        ;Entreposer le pointeur sur Library
    beq    main_NoLib        ;Fin, Library introuvable

;Insérer un programme personnel, par exemple.

;      move.l LibBase,a6    ;Pointeur sur Dos-Library vers A6
;      jsr    ?Offset(a6)    ;Appel de la fonction DOS

    move.l $4,a6          ;Pointeur sur Exec-Library vers A6
    move.l LibBase,a1        ;Pointeur sur Dos-Library vers A1
    jsr    CloseLibrary(a6);Fermer Library
main_NoLib:
    clr.l d0                ;Supprimer le message d'erreur pour DOS
    rts                  ;Fin
*****
LibBase: dc.l 0            ;Place pour pointeur sur Library
LibName: dc.b 'dos.library',0 ;Nom de la Library

END

```

Voici maintenant le même programme, mais cette fois avec les fonctions de l'assembleur que nous avons recommandé. Au premier abord, le programme est plus long et plus complexe.

Les macros utilisées dans le programme ne doivent pas être redéfinies avant chaque programme, mais une fois pour toutes dans un fichier qui sera intégré au moyen de la fonction Include de l'assembleur dans le programme personnel.

L'avantage de cette utilisation des fichiers Include n'est peut-être pas évident dans ce petit exemple, mais il est indéniable lorsqu'on travaille avec des structures système.

```

include "exec/types.i"

* Les macros sont normalement définies une fois pour toutes
* et intégrées ensuite au moyen de Include
* par exemple include "mymacro.i"

* Pour notre exemple, elles sont définies à l'intérieur du programme

CALLSYS MACRO
    IFGT NARG-1
        FAIL    !!! CALLSYS MACRO - trop de paramètres !!!
    ENDC
        JSR     _LVO\1(A6)
    ENDM

```

```

LINKSYS MACRO
    IFGT NARG-2
        FAIL !!! LINKSYS MACRO - trop de paramètres !!!
    ENDC
        MOVE.L A6,-(SP)
        MOVE.L \2,A6
        CALLSYS \1
        MOVE.L (SP)+,A6

    ENDM
XLIB MACRO
    XREF _LVO\1
ENDM
*****
; structure personnelle

STRUCTURE global.0
    APTR     gl_DosLib
    LABEL    gl_size

*****
;Définition recherchée par l'éditeur de liens

XREF _AbsExecBase

*****
;Exec (Faire rechercher les offsets par l'éditeur de liens)

XLIB OpenLibrary
XLIB CloseLibrary
*****

main:
    move.l  _AbsExecBase,a6      Pointeur sur Exec-Library
    lea     -gl_size(a7),a7      Créer de la place dans la pile
    move.l  a7,a5                Pointeur sur structure personnelle
    lea     DosName(pc),a1      Pointeur sur nom de Library
    clr.l   d0                  Numéro de version sans importance (0)
    CALLSYS OpenLibrary         Appel de la fonction Exec
    move.l  d0,gl_DosLib(a5)    Entreposer le pointeur sur Library
    beq    main_NoDosLib       Fin, bibliothèque introuvable

* Insérer programme personnel. par exemple

*      LINKSYS ?Fonction,gl_DosLib(a5) Appel d'une fonction DOS

    move.l  gl_DosLib(a5),a1    Cherche pointeur sur Dos-Library
    CALLSYS CloseLibrary        Appel d'une fonction Exec
main_NoDosLib:
    lea     gl_size(a7),a7      Libérer à nouveau la pile
    clr.l   d0                  pas de message en retour dans CLI
    rts
*****
DosName: dc.b 'dos.library',0          Nom de la bibliothèque

END

```

La mémoire pour les variables (pointeur sur la bibliothèque DOS) est prise dans les piles. Il n'y a pas de variables globales, permettant de rendre un programme réentrant. Rappelons qu'on appelle réentrant un programme qui peut être lancé par plusieurs tâches à la fois, sans que celles-ci se gênent l'une l'autre. L'avantage des programmes réentrants n'est pas encore sensible à cette étape.

2.5.2. Autres fonctions sur les bibliothèques

RemLibrary

Fonction Error = RemLibrary(library)
D0 A1

Offset -402 -\$192

Description

La fonction supprime une structure de bibliothèque dans la liste des bibliothèques de la structure ExecBase.

Library

Pointeur sur la structure de bibliothèque.

Error

Indique si une erreur est survenue dans la fonction. Si c'est le cas, on obtient le message d'erreur en D0, sinon Error prend la valeur 0.

OldOpenLibrary

Fonction Library = OldOpenLibrary(libName)
D0 A1

Offset -408

Description

Cette fonction est un vestige de la version Kickstart 1.0. Elle sert à ouvrir une bibliothèque, sans tester cependant le numéro de version de la bibliothèque à ouvrir. Cette fonction a été conservée dans la bibliothèque Exec afin que les programmes conçus avec la version Kickstart 1.0 puissent tourner également avec les versions ultérieures.

2.6. Le fonctionnement Multi-tâches

Le fonctionnement Multi-tâches est l'une des caractéristiques les plus remarquables de Exec. On entend par ce terme de "Multi-tâches" la capacité du système d'exploitation à exécuter simultanément plusieurs programmes, chacun d'entre eux représentant alors une tâche. Comme l'Amiga ne dispose que d'un seul processeur, il ne peut en réalité exécuter évidemment qu'une seule tâche à la fois. En fait, le 68000 partage son temps de calcul entre les différentes tâches, ce qui nous donne l'impression qu'il les exécute toutes à la fois.

Ceci n'est possible qu'en organisant les temps d'accès, ce qui signifie que chaque tâche dispose du processeur pendant un cours laps de temps donné. On peut dire que les tâches se partagent le processeur suivant un procédé de multiplexage du temps.

Toutes les tâches se trouvant en mémoire n'ont pas besoin de tourner en même temps. Beaucoup d'entre elles ne sont activées qu'en cas de besoin lorsqu'une touche déterminée est pressée ou lorsque le bouton de la souris est activé. Pour cette raison, les tâches sont classées en différentes catégories (Task States en anglais).

Running :

Ceci correspond à la tâche exécutée à l'instant même par le processeur. Il n'y a qu'une seule tâche de cette catégorie au même moment.

Ready :

Toutes les tâches prêtes à être exécutées se trouvent dans cet état, le processeur ne leur consacrant à l'instant même aucun temps d'exécution.

Waiting :

Les tâches de cette catégorie correspondent à celles qui ne doivent pas être traitées à l'instant mais qui attendent un événement déterminé.

Une tâche peut se trouver de plus dans l'un des trois états suivants :

Added :

une telle tâche vient d'être rajoutée au système et ne se trouve pas encore dans l'un des trois états cités ci-dessus.

Removed :

cette tâche vient de se terminer. Elle n'appartient plus à l'une des trois catégories citées ci-dessus et va être écartée du système.

Exception :

une Exception est un état dans lequel la tâche peut être placée à cause d'un événement particulier. Après traitement de cette exception, la tâche retourne à l'un des trois états cités ci-dessus.

Une tâche se compose de deux éléments essentiels : le programme et la structure de tâche (ou structure Task). Cette dernière comprend toutes les informations concernant la tâche, qui sont nécessaires à Exec. On comprendra mieux les multiples possibilités d'une tâche en examinant plus attentivement la structure Task.

2.6.1. La structure Task

Cette structure telle qu'elle apparaît dans le fichier Include "exec/tasks.h" d'un compilateur C est de la forme suivante (les nombres entre parenthèses correspondent à l'écart d'un élément par rapport à l'adresse de base de la structure) :

```

extern struct Task {
    struct Node tc_Node;
    UBYTE      tc_Flags;        /* (14) */
    UBYTE      tc_State;        /* (15) état de la tâche */
    BYTE       tc_IDNestCnt;    /* (16) compteur pour Disable() */
    BYTE       tc_TDNCnt;       /* (17) compteur pour Forbid() */
    ULONG      tc_SigAlloc;     /* (18) bits de signal occupés */
    ULONG      tc_SigWait;      /* (22) attente de ces derniers */
    ULONG      tc_SigRecvD;    /* (26) signal de réception */
    ULONG      tc_SigExcept;    /* (30) exceptions déclenchées */
    UWORLD     tc_TrapAlloc;    /* (34) instruction Trap occupée */
    UWORLD     tc_TrapAble;     /* (36) instruction Trap autorisée */
    APTR      tc_ExceptData;   /* (38) données pour exceptions */
    APTR      tc_ExceptCode;   /* (42) code pour exceptions */
    APTR      tc_TrapData;     /* (46) données pour TrapHandler */
    APTR      tc_TrapCode;     /* (50) code pour TrapHandler */
    APTR      tc_SPReg;        /* (54) Mémoire pour SP */
    APTR      tc_SPLower;      /* (58) limite inférieure de la pile */
    APTR      tc_SPUpper;      /* (62) limite supérieure de la pile +2 */
    VOID      (*tc_Switch)();  /* (66) perte du CPU */
    VOID      (*tc_Launch)();  /* (70) obtention du CPU */
    struct List tc_MemEntry;   /* (74) mémoire occupée */
    APTR      tc_UserData;     /* (88) pointeur sur les données Task */
};

/*-- Flag-Bits -----*/
#define TB_PROCTIME      0
#define TB_STACKCHK      4
#define TB_EXCEPT        5
#define TB_SWITCH        6
#define TB_LAUNCH        7

#define TF_PROCTIME      (1<<0)
#define TF_STACKCHK      (1<<4)
#define TF_EXCEPT        (1<<5)
#define TF_SWITCH        (1<<6)
#define TF_LAUNCH        (1<<7)

```

```
/*--- Task Status -----*/
#define TS_INVALID      0
#define TS_ADDED        1
#define TS_RUN          2
#define TS_READY         3
#define TS_WAIT          4
#define TS_EXCEPT        5
#define TS_REMOVED       6

/*--- Signaux prédéfinis-----*/
#define SIGB_ABORT      0
#define SIGB_CHILD        1
#define SIGB_BLIT         4
#define SIGB_SINGLE        4
#define SIGB_DOS          8

#define SIGF_ABORT      (1<<0)
#define SIGF_CHILD        (1<<1)
#define SIGF_BLIT         (1<<4)
#define SIGF_SINGLE        (1<<4)
#define SIGF_DOS          (1<<8)
```

Comme on peut le constater, l'en-tête d'une structure Task est constitué par une structure Node, dont il a été question au chapitre précédent. La raison en est que Exec gère les tâches dans deux listes différentes : l'une correspond aux tâches de type Ready, l'autre correspond aux tâches de type Waiting. On peut donc appliquer par principe à une liste Task les fonctions telles que Insert(), Remove() ou FindName().

Il existe évidemment des fonctions de gestion des listes de tâches. Comme on le sait, chaque liste possède une tête de liste. Dans le cas des listes de tâches, ces têtes de liste se trouvent dans la structure ExecBase. Nous ne dirons rien de plus ici sur cette structure. Nous aurons l'occasion d'y revenir plus tard. Les pages suivantes vont expliquer l'accès aux listes de tâches.

```
#include <exec/execbase.h>

extern ExecBase *SysBase;

main()
{
    struct Task *waiting, *ready, *running;

    waiting=(struct Task *) SysBase -> TaskWait.lh_Head;
    ready=(struct Task *) SysBase -> TaskReady.lh_Head;
    running=(struct Task *) SysBase -> ThisTask;

    etc..
}
```

Ce programme transfère à l'état Waiting et Ready chaque pointeur sur la première structure Task de la liste correspondante. Lors de l'exécution, il s'agit d'un pointeur sur la tâche exécutée. TaskWait et TaskReady sont donc des têtes de listes qui ne

correspondent à un pointeur que pour ThisTask. Comme il ne peut y avoir qu'une seule tâche Running à la fois, il n'existe pas ici de liste.

Task-Switching

Avant de poursuivre, il faut dire ici quelques mots sur le processus qui permet l'utilisation du processeur par plusieurs tâches : le Task-Switching. Il s'agit de la commutation entre différentes tâches. Si une tâche se trouve dans l'une des deux listes Task, elle sera automatiquement insérée dans ce processus. La première question que l'on se pose est la suivante :

Comment se passe la commutation entre tâches ?

Une tâche ne sait pas quand elle aura accès au processeur, ni à quel moment son accès prendra fin. Si elle teste le champ ThisTask d'ExecBase, elle y trouve toujours son propre pointeur, étant donné que ce champ ne lui est accessible que lorsque le CPU lui est attribué. Une tâche peut être interrompue à tout moment. Le processus est déclenché par une interruption qui apparaît lorsque la tâche a épuisé son temps d'accès ou lorsqu'une autre tâche importante doit être traitée en priorité. La routine Switch du système d'exploitation rend possible toutes ces commutations. Contrairement à la tâche qui fonctionne en mode Utilisateur du 68000, la routine Switch est toujours traitée en mode Superviseur. En premier lieu, les registres processeur D0-D7 et A0-A6 sont mis en sécurité sur la pile Task. Le pointeur de pile Utilisateur est transféré dans le champ tc_SPReg de la structure Task, le registre d'état et le compteur programme venant en dernier sur la pile Utilisateur. Ils sont transférés par le 68000 lors de l'interruption automatiquement sur la pile Superviseur. Puis la tâche est reçue dans la liste Ready.

La nouvelle tâche provient soit de la liste Ready, soit de la liste Waiting. Elle y est supprimée pour être transférée dans le champ ThisTask. Son pointeur de pile est alors retiré du champ tc_SPReg, et ses registres de la pile. Exec abandonne la routine Switch à la fin avec une instruction RTE. Après cette dernière, une nouvelle tâche est traitée automatiquement.

Ceci n'est évidemment qu'un résumé du déroulement de la routine Switch. En vérité, il y a encore des échanges de données et des examens de cas spéciaux.

Si la tâche se trouve dans l'état Exception, il peut arriver que la routine Switch, avant ou après la commutation, appelle des routines dont les adresses de la structure Task correspondent à tc_Launch et tc_Switch. Mais, comme cela a été dit, l'ensemble du processus se déroule automatiquement. Vous n'avez pas à intervenir.

Quand a lieu la commutation entre les tâches ?

On peut formuler la question d'une autre façon : quelle fraction de temps revient à une tâche déterminée ? Le partage du temps de calcul se nomme : Task-Scheduling. L'élément de base utilisé par Exec est le champ ln_Pri de la structure Node au début de la structure Task. Plus simplement, on peut dire que plus la priorité d'une tâche est

importante, plus elle aura de temps de calcul à sa disposition, et plus vite elle sera traitée. Exec commence toujours avec la tâche de priorité supérieure. Cette dernière se voit attribuer un temps processeur. Le temps de calcul déjà utilisé est retiré de sa priorité relative par rapport aux autres tâches Ready. A ce moment là, la tâche qui était prioritaire ne le sera plus, et une nouvelle tâche pourra être exécutée par le processeur. Ceci permet à d'autres tâches de priorité inférieure d'être traitées elles aussi.

Le niveau de priorité peut prendre des valeurs comprises entre -128 et +127, une tâche normale (CLJ) ayant la priorité 0. Les tâches du système d'exploitation s'étalent entre des valeurs de -20 à +20. Il n'est donc pas nécessaire de mettre des valeurs extrêmes dans le champ `tc_Node.In_Pri` d'une structure Task.

Un autre champ de la structure Task correspond à `tc_Node.In_Name`. Il renferme un pointeur sur le nom de la tâche. La recherche de cette dernière est ainsi grandement facilitée. Le champ `tc_State` contient l'état de la tâche. Voici les valeurs affectées aux différents états :

<code>TS_INVALID</code>	- 0
<code>TS_added</code>	- 1
<code>TS_running</code>	- 2
<code>TS_ready</code>	- 3
<code>TS_waiting</code>	- 4
<code>TS_exception</code>	- 5
<code>TS_removed</code>	- 6

La pile de tâche

En dehors de la structure Task, une tâche nécessite aussi une pile. Celle-ci est de type Utilisateur. `tc_SPLower` correspond à la limite inférieure de la pile alors que `tc_SPUpper` correspond à la limite supérieure. (Rappelons qu'une pile croît toujours du bas vers le haut, c'est-à-dire de l'adresse la plus élevée à la plus faible).

L'adresse se trouvant dans `tc_SPReg` est utilisée comme mémoire pour le pointeur de pile par le registre de la pile. C'est ici que sont déposés les registres, ainsi que l'adresse de retour de la tâche, lorsqu'elle se trouve en mode Waiting.

En temps normal, `tc_SPReg` est égal à `tc_SPUpper`. On peut aussi mettre `tc_SPReg` à n'importe quelle adresse comprise entre `tc_SPUpper` et `tc_SPLower`. On peut alors utiliser la zone entre `tc_SPReg` et `tc_SPUpper` comme mémoire pour les variables globales ou autres.

Encore un mot sur `tc_SPUpper`

`tc_SPUpper` pointe sur la limite supérieure de la pile, signalant ainsi la première adresse de la zone pile. Le mot se trouvant à la dernière position de la pile est d'ailleurs à l'adresse `tc_SPUpper` moins 2.

Etant donné qu'Exec met en mémoire les registres sur la pile Task lors du Task-Switching, cette dernière doit avoir une taille minimale de 70 octets. Mais la tâche ne peut déposer sur la pile ni des variables, ni des adresses de retour. Puisque les

programmes en C en particulier font un usage intensif de la pile, il vaut mieux lui accorder un Ko.

Comme une tâche se compose de différents éléments, structure Task, pile, programme, etc... on doit lui réservé plus ou moins de mémoire. Il y a d'ailleurs la possibilité de rajouter à la structure Task une liste où figurent tous les emplacements occupés en mémoire par la tâche. Le champ tc_MemEntry comprend la tête de liste correspondante. Nous n'en dirons pas plus sur cette possibilité dans le présent chapitre, car la structure d'une telle liste de mémoire, ou MemList, sera expliquée plus tard. Examinons maintenant la pile des registres.

La structure de la pile des registres est claire si l'on considère la fonction Dispatch de Exec. Il faut noter que Dispatch fonctionne en mode Superviseur, et est donc conclu par un RTE.

Lorsque les commandes suivantes seront en cours de fonctionnement, on trouvera dans A5 le pointeur sur la pile de registres (tc_SPRReg). Avec le RTE, on lance la tâche dont les registres sont utilisés.

lea	66(A5),A2	Prend le pointeur de pile pour la tâche
move	A2,USP	Pose le pointeur de pile
move.l	(A5)+,-(A7)	Définit l'adresse de retour
move.w	(A5)+,-(A7)	Définit le registre d'état
movem.l	(A5),A6-A0/D7-D0	Prend le registre de tâche
rte		Saut dans la tâche

Voici la définition de la pile que l'on obtient :

Offset	Explication
00	Adresse de retour
04	Registre d'état
06 à 37	Registres D0 à D7
38 à 65	Registres A0 à A6

Autorisation et interdiction du Task-Switching

Il peut être fâcheux qu'une tâche perde l'accès du processeur de manière imprévue. Vous voulez afficher la liste des tâches de type Waiting à l'écran mais pendant que votre programme lit entrée après entrée, une tâche de type Ready risque de passer en mode Waiting ou inversement.

Les valeurs lues seront alors fausses. C'est pourquoi il faut mettre en oeuvre le principe suivant : lorsqu'une tâche accède à des structures de données qui sont à la disposition du système entier ou d'autres tâches, le Task-Switching doit être désactivé afin que les données ne soient pas modifiées par une autre tâche ou par le système d'exploitation.

Forbid() et Permit()

Ces deux routines représentent le premier échelon de désinitialisation. Forbid() désactive le Task-Switching, Permit() l'autorise à nouveau. Les deux routines sont appelées sans paramètre et ne donnent pas de valeur en retour.

Disable() et Enable()

Souvent, il ne suffit pas de désactiver le Task-Switching. Beaucoup de structures de données du système peuvent être en effet modifiées par Exec pendant les interruptions. On peut interdire ces dernières avec Disable() et les autoriser à nouveau avec Enable(). Mais attention : alors qu'une interdiction du Task-Switching, même pendant un long moment, ne change rien, il en va tout autrement avec les interruptions : leurs apparitions régulières sont nécessaires à Exec (par exemple pour le device Keyboard). Si on désactive les interruptions pendant un long moment, cela pourra amener un effondrement du système lorsque l'on voudra revenir en mode de fonctionnement Multi-tâches. Pendant un Disable(), la tâche se déroule sans être dérangée le moins du monde, car la désactivation des interruptions a pour conséquence la mise hors-jeu du task-switching.

Il est aussi possible d'imbriquer plusieurs Forbid() ou Disable() les uns dans les autres.

Il existe deux compteurs : TDNestCnt et IDNestCnt (Task Disable Nesting Counter et Interrupt Disable Nesting Counter). A chaque appel de Forbid() TD_NestCnt est incrémenté de 1, et il est décrémenté de 1 avec Permit(). La commutation des tâches n'est possible que lorsque TDNestCnt < 0. Ceci signifie que le nombre d'appels Permit() doit être égal au nombre de Forbid() avant que le Task-Switching ne soit autorisé.

La même chose est vraie pour Enable() et Disable().

Le programme suivant montre l'utilisation des fonctions Enable() et Disable(). Il lit le pointeur de l'ensemble des structures Task des tâches Ready et Waiting dans un champ pendant que les interruptions sont désactivées au moyen de Disable(). Ces dernières sont à nouveau autorisées avec Enable() et les champs importants des structures Task, comme le nom, la pile et la priorité sont affichés à l'écran au moyen des pointeurs déposés dans ces champs. Le programme montre aussi quelle est la tâche exécutée. Vous pourrez l'expérimenter avec des instructions CLI telles que NEWCLI, RUN ou STATUS.

La tâche de type Running est aussi affichée. Celle-ci est normalement celle qui permet l'exécution du programme.

```
#include <exec/execbase.h>

extern struct ExecBase *SysBase;

main()
{
    register struct Task *a_task;
    APTR run, tnodes[50], wtask, ltask;
    void o1(),o2();
```

```

register APTR Anode;
Disable();
Anode = tnodes;
run = (APTR)SysBase->ThisTask;

for(a_task=(struct Task *)SysBase->TaskReady.lh_Head;
    a_task->tc_Node.ln_Succ;
    *Anode=(APTR)a_task,Anode++,
    a_task=a_task->tc_Node.ln_Succ);
wtask=Anode;

for(a_task=(struct Task *)SysBase->TaskWait.lh_Head;
    a_task->tc_Node.ln_Succ;
    *Anode=(APTR)a_task,Anode++,
    a_task=a_task->tc_Node.ln_Succ);
ltask=Anode;

Enable();

printf("\nTask in the running state:\n");o1();o2(run);

printf("\nTask(s) in the ready state:\n");o1();
for(Anode=tnodes;Anode!=wtask;o2(*Anode),Anode++);

printf("\nTask(s) in the waiting state:\n");o1();
for(;Anode!=ltask;o2(*Anode),Anode++);

}

void o1()
{
printf
("Standarde Stacksize Priority Signals Name\n");
printf
("-----\n");
}

void o2(at)
register struct Task *at;
{
printf("%10lx%10lx%8ld%11lx %s\n",at->tc_SPLower,
      (ULONG)at->tc_SPUpper - (ULONG)at->tc_SPLower -2L,
      (LONG)at->tc_Node.ln_Pri,
      at->tc_SigWait,at->tc_Node.ln_Name);
}

```

Etant donné que ce programme sauvegarde le pointeur sur les structures Task et non le contenu de ces structures, il existe encore une possibilité d'erreur potentielle comme par exemple lorsqu'une tâche est effacée lors de l'affichage des valeurs. Mais comme on peut dire que ceci n'arrive pratiquement jamais, et puisqu'il aurait fallu un énorme travail pour sauvegarder temporairement toutes les structures Task, nous y avons renoncé dans ce programme. Par modification des routines o1() et o2(), on peut afficher d'autres champs des structures Task.

Générer de nouvelles tâches

Après avoir décrit et expliqué les tâches et les structures de tâche, nous allons aborder ce qui concerne la création d'une nouvelle tâche. Voici ce qui est nécessaire à cet effet : en premier lieu, il faut une structure Task, puis une pile, puis un nom de tâche car la structure de tâche contient déjà un pointeur sur le nom lui-même. En dernier lieu, il faut avoir un programme qui constitue la tâche proprement dite. Ici apparaissent quelques problèmes : il est nécessaire de disposer d'une certaine quantité de mémoire libre pour la tâche, car celle-ci reste en mémoire à la fin du programme qui l'a engendrée. Afin de ne pas trop compliquer ce programme, la structure Task, le nom de la tâche et la pile seront rassemblés dans un nouveau type de structure "alltask" pour laquelle un bloc mémoire conséquent est réservé.

La tâche proprement dite est écrite comme une fonction C normale portant le nom "Code". Elle ne fait rien d'autre qu'incrémenter un compteur jusqu'à la valeur \$FFFFFF. Avant qu'elle y parvienne, il faut plusieurs minutes. Son exécution se remarque à la lenteur soudaine de l'Amiga, car notre tâche utilise une partie du temps de calcul du processeur. On peut aussi utiliser le programme précédent qui permet de lister toutes les tâches. Notre tâche présente apparaîtra en tant que tâche Ready avec le nom Exemple.

Afin de bien copier la fonction "code" à sa position définitive en mémoire, vous pourrez employer son nom comme pointeur d'adresses. La fonction "End" ne sert qu'à contenir l'adresse de fin de la fonction code. Comme il n'y a aucune possibilité en C de communiquer le besoin en mémoire d'une fonction, celui-ci sera calculé par la différence entre l'adresse de départ et celle de fin.

```
***** générer une tâche *****

#include <exec/types.h>
#include <exec/Tasks.h>
#include <exec/memory.h>

#define STACK_SIZE 500 /* taille de la pile */

main()
{
    void code(),end();
    APTR mycode,AllocMem();
    Static char Taskname[] = "module d'exemple"
    register APTR c1,c2;

    struct alltask {
        struct Task tc;
        char Name[sizeof(Taskname)], Stack[STACK_SIZE];
    } *mytask;

    mytask = AllocMem((ULONG)sizeof(*mytask),
                      MEMF_PUBLIC/MEMF_CLEAR);
    if(mytask==0)
    {printf("Pas de mémoire pour la structure AllTask!\n");
     return(0);
    }
```

```

mycode = AllocMem((ULONG)end-(ULONG)code, MEMF_PUBLIC);
if(mycode==0)
{FreeMem(mytask,(ULONG)sizeof(*mytask));
printf("Pas de mémoire pour le code Task!\n");
return(0);
}

strcpy(mytask->Name,Taskname);

mytask->tc.tc_SPLower=mytask->Stack;
mytask->tc.tc_SPUpper=mytask->Stack+STACK_SIZE;
mytask->tc.tc_SPReg=mytask->tc.tc_SPUpper;

mytask->tc.tc_Node.ln_Type=NT_TASK;
mytask->tc.tc_Node.ln_Name=mytask->Name;

for(c1=code,c2=mycode;c1<=end;*c2++=*c1++);
AddTask(mytask,mycode,0L);
}

/** La fonction "code" correspond au module **/
```

```

void code()
{
ULONG count;

for(count=0;count>0xFFFFFFF;count++);
}

void end(){}

```

Comme on peut le remarquer, il y a peu de champs de la structure Task à initialiser :

tc_SPLower avec la limite de pile inférieure
 tc_SPUpper et tc_SPReg avec la limite de pile supérieure
 tc_Node.ln_Type avec le type de liste NT_TASK

On peut aussi se passer d'un nom. Une fonction essentielle est utilisée dans ce programme :

```
AddTask(task,initialPC,finalPc);
```

Cette fonction insère une nouvelle tâche dans le système. Normalement, cette dernière sera ajoutée directement à la liste Ready. Cette fonction nécessite les paramètres suivants :

Task

Pointeur sur la structure Task dans laquelle doivent être initialisés au minimum les 4 champs cités plus haut.

InitialPC

Adresse où le traitement de la tâche doit débuter. Dans notre exemple, l'adresse de départ de la fonction code, dont la position définitive se trouve dans mycode.

FinalPC

FinalPC renferme l'adresse à laquelle on effectue un saut lorsque le module exécute l'instruction RTS. On peut mettre en place l'adresse d'une routine qui rétablit la mémoire, ferme les fichiers ouverts etc... Si l'on donne simplement la valeur 0 à FinalPC (comme dans notre exemple), Exec utilisera sa routine FinalPC standard. Celle-ci libère la mémoire pointée par le champ MemEntry, puis élimine ce module des listes système.

Fin d'une tâche

La description de FinalPC nous amène au sujet suivant. Il existe plusieurs possibilités pour mettre fin à une tâche :

- ① La tâche atteint une instruction RTS, qui ne représente pas un saut de retour JSR ou BSR propre à la tâche. Le processus sera alors celui qui a été décrit dans FinalPC.
- ② Une exception du 68000 est déclenchée, sans aucun rapport avec la tâche en elle-même, comme par exemple, une erreur de bus ou d'adresse, une division par zéro... Exec engendre alors un message "Software Error - Task Held" ou une méditation du gourou. Nous expliquerons à la fin de ce chapitre comment intercepter ces erreurs.
- ③ Appel de la fonction RemTask, qui supprime la tâche du système.

2.6.2. Fonctions pour les tâches

Il existe dans Exec plusieurs fonctions servant à générer ou à éliminer des tâches, ainsi qu'à gérer les listes de tâches.

AddTask

Fonction : AddTask(task, initialPC, finalPC)
 a0 a1 a2

Offset : -282 -\$11A

Description :

Addtask insère une nouvelle tâche dans le système.

Paramètres :***task***

Pointeur sur une structure de tâche. Les champs tc_SPUpper, tc_SPLower, tc_SPRReg et tc_Node.In_Type doivent être initialisés correctement.

InitialPC

Adresse de début de traitement du programme du module.

finalPC

Adresse de retour qui, au début de la tâche, est mise sur la pile. Si la tâche exécute un RTS en surnombre, on saute à cette adresse. Si finalPC est à 0, Exec met en œuvre sa routine standard finalPC.

FindTask

Fonction : Task = FindTask(Nom)
d0 a1

Offset : -294 - \$126

Description :

FindTask cherche une tâche portant un nom déterminé dans la liste des tâches. Si cette tâche est trouvée, un pointeur sur cette structure de tâche sera délivré. Si on indique 0 pour le nom, on obtiendra un pointeur sur la structure Task de la tâche en cours.

Paramètre :***Nom***

Pointeur sur le nom de la tâche recherchée ou 0.

Résultat :***Task***

Pointeur sur la structure Task de la tâche recherchée.

RemTask

Fonction : RemTask(task)
A1

Offset : -288 - \$120

Description :

RemTask sert à éliminer une tâche du système. Si le champ tc_MemEntry pointe sur une liste MemEntry, cette dernière sera libérée. Toutes les autres ressources système occupées par la tâche doivent avoir été restituées auparavant au système.

Paramètre :

Task

Pointeur sur la structure Task de la tâche à écarter. Si Task = 0, la tâche en cours sera éliminée. Une fois la tâche supprimée, on revient dans la fonction Switch de Exec.

SetTaskPri

Fonction : ancpriorité -SetTaskPri(Task, nouvpriorité)
D0 A1 D0

Offset : -300 -\$12C

Description :

SetTaskPri renvoie l'ancienne priorité d'une tâche et donne à la tâche sa nouvelle priorité. On a en plus l'exécution d'un Rescheduling, ce qui signifie que le temps de calcul de chaque tâche est redécoupé suivant les nouvelles priorités. Si on donne une haute priorité à une tâche avec cette fonction, cette dernière aura accès immédiatement au processeur.

Paramètres :

Task

Correspond au pointeur sur la structure Task de la tâche.

Nouvpriorité

Correspond à la nouvelle priorité de la tâche (dans les 8 bits inférieurs de D0).

AncPriorité

Correspond à l'ancienne priorité de la tâche (dans les 8 bits inférieurs de D0).

Forbid

Fonction : Forbid()

Offset : -132 - \$084

Description :

Interdit le Task-Switching et incrémente TDNestCnt.

Permit

Fonction : Permit()

Offset : -138 - \$08A

Description :

Décrémentation de TDNestCnt et Task-Switching autorisé à nouveau lorsque TDNestCnt < 0.

Disable

Fonction : Disable()

Offset : -120 - \$078

Description :

Interdiction de toutes les interruptions et incrémentations de IDNestCnt.

Enable

Fonction : Enable()

Offset : -126 - \$07E

Description :

Décrémentation de IDNestCnt et interruptions à nouveau autorisées lorsque IDNestCnt < 0.

2.6.3. Communication entre tâches

Toutes les tâches ne traitent pas que leurs propres données. Elles peuvent communiquer leurs données avec d'autres tâches. La plupart de ces communications sont des processus de type entrée/sortie car les routines qui gèrent les différents périphériques d'entrée/sortie tels que le clavier, l'écran ou les disquettes peuvent être considérées comme des tâches au sens propre.

Il a déjà été dit que certaines tâches attendent un signal d'autres tâches avant d'entrer en action. Ces signaux sont décrits dans le paragraphe suivant.

2.6.3.1. Les signaux de tâches

Chaque tâche possède 32 bits de signal afin d'avoir la possibilité de différencier divers événements. Chaque tâche peut utiliser ces signaux à sa guise. Un signal déterminé peut avoir des significations entièrement différentes pour deux tâches distinctes. Certaines fonctions du système utilisent toutefois des signaux déterminés pour leurs communications. Si une tâche veut utiliser un de ces signaux, elle doit tout d'abord l'occuper. Ceci se passe avec la fonction AllocSignal(). Normalement, les 16 bits inférieurs sont réservés aux fonctions du système, ce qui laisse 16 signaux à utiliser librement.

On peut avec AllocSignal() allouer à une tâche un signal déterminé en donnant le numéro du signal souhaité en tant qu'argument ou en faisant en sorte qu'AllocSignal cherche lui-même le prochain signal libre lorsque l'on entre -1.

Comme résultat, nous obtenons toujours le numéro du signal souhaité s'il n'a pas encore été occupé, ou -1 si une erreur s'est produite. AllocSignal(-1) retourne la valeur -1 lorsqu'aucun signal n'est libre. Le programme C suivant alloue le premier signal trouvé avec la fonction AllocSignal :

```
Signal=AllocSignal(-1L);
Signal<0 ?
printf("Pas de signaux libres"!!);
printf("Le signal numéro %ld est occupé!", (long)Signal);
```

Il y a deux façons d'indiquer un signal déterminé : on peut en premier lieu indiquer son numéro qui est un nombre compris entre 0 et 31 correspondant au numéro de bit; la deuxième façon est d'indiquer le mot long du signal en entier. Dans ce masque de signaux, l'état d'un bit reflète alors l'état du signal correspondant, l'avantage ici étant la possibilité d'indiquer plusieurs signaux en une seule fois. AllocSignal retourne le numéro du signal. Pour revenir de là au masque de signaux, il faut poser le bit dans la position désignée par le numéro de signal. Ceci peut se faire en C de la manière suivante :

```
MasqueSignal=1<<NuméroSignal;
```

En langage machine :

```

MOVE.W NumeroSignal, D0
MOVE.L MasqueSignal, D1
BSET D0, D1

```

où les variables `MasqueSignal` et `NumeroSignal` représentent à chaque fois l'adresse de la valeur, et non pas la valeur elle-même.

Les signaux occupés par une tâche sont mis en mémoire dans le champ `tc_SigAlloc` de la structure de tâche sous la forme d'un masque de signaux.

Attente de signaux

La fonction essentielle d'un signal réside dans le fait qu'une tâche peut l'attendre. Examinons le cas où une tâche attend la pression d'une touche déterminée.

Elle peut le faire en principe sous la forme d'une boucle. Mais elle utiliserait alors inutilement le temps de calcul qui lui est imparti. Pour empêcher cela, on peut faire en sorte qu'une tâche attende un événement précis, par exemple la pression d'une touche, à l'aide des signaux de tâche. Chaque tâche peut en effet attendre les signaux qui la concernent. Pendant l'attente, la tâche est transférée dans la liste `Waiting`, et ainsi elle n'utilise pas de temps de calcul.

Pour faire en sorte qu'une tâche attende l'un des signaux qui lui sont destinés, on utilise la fonction `Wait()`. Cette dernière ne nécessite qu'un seul paramètre, précisément un masque de signaux, comprenant tous les signaux qui doivent être attendus. Il est ainsi possible d'attendre plusieurs signaux en même temps. Dès qu'un des signaux indiqués est occupé par une autre tâche, la fonction `Wait()` revient et transmet comme résultat un masque de signaux contenant le signal (ou les signaux) en question. Au moyen de l'opération ET logique entre les signaux souhaités et le masque de signaux retourné par la fonction `Wait()`, on peut déterminer quel est le signal apparu.

Le programme suivant, certes hypothétique, vous en donne un exemple :

```

unsigned long Signal;
Signal=Wait(Touche|Bouton souris|Menu);
if(Signal & Touche)      { /* Touche enfoncée */ }
if(Signal & BoutonSouris) { /* Bouton souris activé */ }
if(Signal & Menu)         { /* Point menu activé */ }

```

Si l'un des signaux souhaités est déjà occupé avant l'appel, la fonction `Wait()` retourne tout de suite au programme. Les signaux attendus par une tâche sont mis en mémoire dans son champ `tc_SigWait`, les signaux qu'il a reçus étant mis en mémoire dans `tc_SigRecv`.

Si avant l'appel d'une fonction `Wait()`, le Task-Switching ou les interruptions sont désactivés, la fonction les autorisera à nouveau. Lorsque `Wait()` retourne au programme, l'état d'interdiction est à nouveau mis en place.

On réalise ceci en transférant lors d'un appel de `Wait()` les contenus des deux compteurs `TDNestCnt` et `IDNestCnt` dans les champs correspondants de la structure de tâche :

tc_TDnestCnt et tc_IDNestCnt

Il existe cinq routines en relation avec les signaux de tâche. Ici, SetSignals() n'est pas nécessaire à l'utilisateur normal.

Les fonctions de signal

AllocSignal

Fonction : NumeroSignal = AllocSignal(NumeroSignal)
 D0 D0

Offset : -330 -\$14A

Description :

Au moyen de cette fonction, on peut allouer un des signaux de tâche. Si on donne comme numéro de signal -1, et non pas le numéro de signal à allouer, AllocSignal() cherche le premier signal libre et l'alloue. Si le signal souhaité est déjà occupé, AllocSignal() retourne la valeur -1.

Cette fonction ne peut être appelée que pour les signaux correspondant à leur propre tâche, et elle ne doit pas être appelée à l'intérieur d'une exception.

Paramètre :

NumeroSignal

Numéro du signal à occuper (0-31) ou -1 pour le prochain signal libre.

Résultat :

NumeroSignal

Numéro du signal occupé ou -1 lorsque le signal demandé est déjà occupé (ou tous les signaux avec AllocSignal(-1)).

FreeSignal

Fonction : FreeSignal(NumeroSignal)
 D0

Offset : -336 -\$150

Description :

`FreeSignal()` est la fonction inverse de `AllocSignal()`. Le signal correspondant au numéro est libéré. De la même manière que `AllocSignal()`, `FreeSignal()` ne doit pas être employé à partir d'une exception.

Paramètre :*NumeroSIGNAL*

Numéro du signal à libérer (0-31).

SetSignals

Fonction : `Signal précédent = SetSignals(NouveauxSignaux, Masque)`

D0	D0	D1
----	----	----

Offset : -306 -\$132

Description :

`SetSignals()` transfère l'état de chaque signal dont les bits correspondants sont posés dans le masque, en les prenant dans `NouveauxSignaux` et en les portant dans les signaux de la tâche (`tc_SigRecv()`). Si un bit de `NouveauxSignaux` et de `Masque` est égal à 1, le signal correspondant sera activé. Si un bit de `NouveauxSignaux` = 0 alors que celui de `Masque` est égal à 1, le signal sera supprimé. Si un bit du `Masque` est égal à 0, le signal correspondant reste inchangé.

Paramètres :*NouveauxSignaux*

Contient le nouvel état des signaux.

Masque

Détermine le bit Signal à modifier.

Résultat :*Signal précédent*

Indique l'état précédent des signaux de tâche.

SetSignals(OL, OL)

Fournit les signaux sans les modifier.

Signal

Fonction : Signal(Task, Signaux)
A1 D0

Offset : -324 -\$144

Description :

Avec cette fonction, on peut allouer un signal d'une autre tâche. Cette fonction est le centre du système des signaux car elle permet de communiquer entre tâches. Lorsqu'une tâche reçoit un signal attendu, elle retourne automatiquement dans le mode Ready ou Running. Cette fonction est utilisée par les messages système qui seront décrits dans le chapitre suivant.

Paramètres :

Task

Pointeur sur la structure Task de la tâche réceptrice.

Signaux

Masque de signaux contenant le bits Signal à communiquer.

Wait

Fonction : Signaux=Wait(MasqueSignal)
D0 D0

Offset : -318 -\$13E

Description :

Wait attend un signal du masque de signaux indiqué. Ceci signifie qu'une tâche restera dans l'état Waiting aussi longtemps que le signal ne sera pas occupé par une autre tâche ou une autre interruption. Si un des signaux est activé avant l'appel de la fonction Wait, Wait retourne au programme. Le résultat se présente sous la forme d'un masque de signaux contenant les signaux attendus.

Attention :

Cette fonction doit être appelée uniquement en mode utilisateur.

Paramètre :

MasqueSignal

Ce sont les signaux contenus dans ce masque qui sont attendus par la tâche.

Résultat :

Signaux

Signaux réceptionnés issus du masque.

2.6.3.2. Le système des messages

Les signaux de tâche forment l'élément de base pour le système de communication entre tâches, ce qu'on appelle aussi le système des messages. Celui-ci autorise non seulement le transfert des signaux mais aussi l'envoi et la réception des communications pouvant contenir n'importe quelle donnée. On trouve aussi, implémentée dans ce système, la constitution automatique des files d'attente, dans le cas où le destinataire ne peut réagir assez vite à l'envoi d'un message.

La base de ce type de communication est ce qu'on appelle le port de message (Message-Port). Celui-ci est à nouveau une structure de données. En C, cette structure a le format suivant :

```
struct MsgPort {
    struct Node mp_Node;
    UBYTE mp_Flags;
    UBYTE mp_SigBit;
    struct Task *mp_SigTask;
    struct List mp_MsgList;
};
```

Les programmeurs en assembleur trouveront la structure dans le fichier Include "exec/ports.i" :

	Offsets
STRUCTURE MP_LN_SIZE	00 \$00
UBYTE MP_FLAGS	14 \$0E
UBYTE MP_SIGBIT	15 \$0F
APTR MP_SIGTASK	16 \$10
STRUCT MP_MSGLIST, LH_SIZE	20 \$14
LABEL MP_SIZE	34 \$22

Un port de message est un emplacement qui rassemble les messages destinés à une tâche. Chaque tâche peut envoyer des données à un port de message mais une seule tâche peut les recevoir. Les champs d'une structure Message-Port ont la signification suivante :

mp_Node

Ceci est une structure Node qui nous est familière. Dans le champ ln_Name, on trouve un pointeur sur le nom du port de message. Ceci facilite la recherche d'un port de message déterminé.

Le type Node dans ln_Type est toujours NT_MSGPORT pour un port de message. Les autres champs de la structure Node ne seront utilisés que lorsque l'on cherchera à trouver

un port de message dans une liste. Cette dernière pourra être de type personnel ou appartenir à la liste des ports publics, liste de tous les ports de message connus par Exec.

mp_Flags

Les deux bits inférieurs de ce champ déterminent ce qui se passe lorsque le port de message reçoit un message. Il existe plusieurs possibilités de sélection (les combinaisons de bits correspondantes se trouvent elles aussi dans "exec/ports.h") :

PA_IGNORE (2)

Cette combinaison de bits signifie qu'il ne se passe rien lorsqu'un message est reçu.

PA_SIGNAL (0)

A chaque fois que le port de message reçoit un message, le signal en provenance du champ *mp_SigBit* est envoyé à la tâche destinataire.

PA_SOFTINT (1)

Déclenche une interruption logicielle chaque fois qu'un message est reçu. Vous trouverez d'autres détails sur les interruptions logicielles dans le chapitre qui leur est consacré.

mp_SigBit

Dans ce champ, on trouve le numéro du bit signal envoyé à la tâche lorsque *mp_Flags* = PA_SIGNAL. Il s'agit d'un numéro de signal compris entre 0 et 31. On ne peut jouer que sur un bit de signal à la fois.

mp_SigTask

Ce champ doit contenir un pointeur sur la structure de tâche de la tâche à laquelle le signal présent dans *mp_SigBit* doit être communiqué.

Si le mode PA_SOFTINT est sélectionné, *mp_SigTask* contient un pointeur sur la structure Interrupt de l'interruption Software correspondante.

mp_MsgList

En-tête de la liste de tous les messages. Chaque message reçu sera rajouté à la fin de cette liste, ce qui - selon le mode indiqué dans *mp_Flags* -, ne déclenche rien (PA_IGNORE), déclenche une interruption software (PA_SOFTINT), ou un signal à la tâche réceptrice (PA_SIGNAL). Cet en-tête de la liste doit être initialisé en conséquence.

Structure d'un message

Chaque message consiste en une structure de message et en un message quelconque, dont la longueur ne doit pas dépasser 64 Koctets. Le message est rajouté directement à la structure de message.

Cette structure est déposée elle aussi dans le fichier Include "exec/ports.h" :

```
struct Message {
    struct Node mn_Node;
    struct MsgPort *mn_ReplyPort;
    ULONG mn_Length;
};
```

La structure en assembleur se trouve dans le fichier Include "exec/ports.i" :

	Offsets
STRUCTURE MN, LN_SIZE	00 \$00
APTR MN_REPLYPORT	14 \$0E
ULONG MN_LENGTH	18 \$12
LABEL MN_SIZE	20 \$14

mn_Node

Ceci est une structure Node normale. Elle sert à lier le message à la liste des messages reçus par le port de message. Le champ ln_Typ est à initialiser avec le type Node NT_MESSAGE. On décidera soi-même si on veut donner un nom au message.

mn_Length

Ce champ renferme la longueur du message en octets. Comme nous l'avons dit, le message suit immédiatement le champ mn_Length.

Envoi d'un message

Un message est envoyé au moyen de la fonction PutMsg à n'importe quel port de message.

PutMsg(MessagePort, Message);

Cette fonction envoie le message au port de message. Les deux paramètres doivent être indiqués sous la forme de pointeurs sur les structures correspondantes. L'exemple suivant envoie un texte comme message à un port de message hypothétique.

```
{
    extern APTL Port; /* Pointeur sur le port de message */
    static char texte[] = "Ceci est un exemple de message !";
    static struct {
        struct Message msg;
        char contenu[sizeof(texte)];
    } message;
```

```
message.msg.mn_Node.ln_Type=NT_MESSAGE;
strcpy(message.contenu, texte);
PutMsg(Port,&message);
}
```

Lors de l'envoi d'un message, celui-ci est rajouté à la liste des messages reçus, mp_MsgList. Ceci se passe comme avec les champs ln_Succ et ln_Pred dans la structure Node du message, mn_Node. Le message n'est pas copié. Ceci signifie que le message entier continue à faire partie de la tâche qui a envoyé le message. L'envoi d'un message autorise donc la tâche réceptrice à utiliser une zone mémoire de la tâche émettrice.

Réception d'un message

La réception se fait en deux étapes. En premier lieu, il s'agit d'attendre un signal du port de message et, lorsque ce dernier signale qu'il a reçu un message, d'aller le chercher.

En supposant que le port de message soit initialisé de façon correcte, la tâche a deux façons pour attendre l'arrivée d'un message sur le port de message :

Elle utilise soit la fonction Wait(), soit la fonction WaitPort().

Message = WaitPort(Port);

Cette fonction attend l'arrivée d'un message au port de message "Port". Si un ou plusieurs messages sont annoncés, la fonction retourne au programme. Sinon elle fait passer la tâche en mode Waiting de la même manière que la fonction Wait. Le résultat est un pointeur sur la structure de message du premier message, sans que celui-ci soit supprimé du port de message.

Quand doit-on utiliser l'une ou l'autre des deux fonctions ?

Wait() a l'avantage de rendre possible l'attente de plusieurs messages. C'est donc le meilleur choix dans tous les cas où l'on attend plusieurs événements à la fois. Si on attend en revanche un message unique à un port de message déterminé, WaitPort() est plus approprié car cette fonction n'attend que lorsque le port de message est vide. Wait au contraire se conforme aux signaux. Si par exemple le port de message a déjà reçu deux messages avant l'emploi de la fonction Wait(), on rencontre le problème suivant :

Le premier appel de Wait() retourne tout de suite au programme, car les deux messages ont déjà posé le signal voulu. Si on veut attendre le message suivant avec la même fonction, l'attente peut se prolonger indéfiniment, car le message souhaité est arrivé depuis un moment. La raison de ce problème tient au fait qu'un signal ne peut être posé qu'une fois, même si plusieurs messages ont été réceptionnés. On est obligé alors, avant l'emploi du deuxième Wait(), de tester si le message suivant est déjà arrivé ou pas. Il est donc préférable d'utiliser WaitPort() lorsqu'on n'attend qu'à un seul port de message.

La prise en compte d'un message est réalisée par la fonction GetMsg().

Message = GetMsg(Port);

Cette fonction prend en compte le premier message arrivé au port et transfère un pointeur dirigé sur sa structure de message. Le message est alors supprimé de la liste du port de message. La fonction retire le message qui se trouve en première position dans la liste. Etant donné que les nouveaux messages viennent se placer à la fin, la liste des messages reçus représente une file d'attente de type FIFO. FIFO signifie en anglais "First In First Out", ce qui veut dire que l'élément arrivé en premier dans la liste sera aussi le premier à être transmis.

On peut ainsi avec GetMsg() retirer une série de messages du port dans l'ordre d'arrivée. Si aucun message n'apparaît, GetMsg() retourne la valeur 0.

L'exemple suivant utilise WaitPort() et GetMsg() pour retirer un message d'un port hypothétique :

```
extern struct MsgPort *Port;
struct Message *GetMsg();
int signal;

if((signal=AllocSignal(-1L))<0)
{printf("il ne reste plus aucun signal libre !");return(0);}

Port->mp_Flags=PA_SIGNAL;
Port->mp_SigBit=signal;
Port->mp_SigTask=FindTask(0); /* tâche propre */

WaitPort(Port);
message = GetMsg(Port);
```

Réponses à un message

Lorsqu'une tâche envoie un message, elle voudra naturellement savoir si le message est arrivé. La raison en est que le message (y compris la structure de message) appartient à la tâche émettrice. En envoyant un message, celle-ci donne à la tâche réceptrice l'autorisation de lire dans la zone mémoire où se trouve le message. De plus, la tâche cible peut fort bien déposer des messages en retour ou des résultats dans le message. Comme la tâche émettrice voudra utiliser tôt ou tard ce secteur de mémoire dans un autre but, par exemple pour un nouveau message, elle doit savoir si le message a été ou non reçu et traité par la tâche réceptrice. Elle ne peut tout de même pas l'effacer purement et simplement sans savoir s'il a déjà été lu ou non. Pour cette raison, il existe un port Reply (port de réponse).

Ce dernier peut être n'importe quel port de la tâche émettrice. Afin de communiquer l'adresse de ce port à la tâche cible, il existe dans la structure de message un champ que nous n'avons pas encore mentionné :

mn_ReplyPort

Ce champ contient l'adresse du port Reply de la tâche émettrice. Pour répondre à un message, la tâche cible la renvoie à ce port réponse après l'avoir lue ou éventuellement traitée. La source sait ainsi que le message a été reçu. Elle peut alors utiliser l'emplacement du message en mémoire, ou le rendre simplement au système.

ReplyMsg(Message);

Se charge du message de retour.

Créer un nouveau Message-Port

Pour créer un nouveau Message-Port (port de message), il suffit de déposer en mémoire une structure Message-Port correctement initialisée. Le plus simple est de le faire au moyen d'une structure C avec le type mémoire static. Toutefois, la mémoire peut aussi être obtenue et initialisée avec la routine AllocMem() du système.

Lorsqu'on en a terminé avec la structure Message-Port, on doit encore décider si on veut la rajouter à la liste des ports publics de message. Ceci se fera au moyen de la fonction AddPort. Cette dernière prend aussi en charge l'initialisation du champ mp_MsgList de la structure de message, rendant superflu l'appel de la routine NewList(). AddPort nécessite comme seul paramètre l'adresse de la structure de message. Le fait qu'il existe un port de message dans la liste des ports publics présente l'avantage suivant : une autre tâche pourra retrouver rapidement ce port grâce à son nom. Si on ne le rajoute pas à cette liste, on devra communiquer l'adresse du Message-Port à toutes les tâches voulant communiquer avec lui.

Pour trouver un port dans la liste lorsqu'on connaît son nom, il existe la fonction FindPort.

Port = FindPort(nom);

Cette fonction cherche un Message-Port au moyen du paramètre nom, et donne son adresse lorsqu'il le trouve. Si l'on ajoute au système un port avec la routine AddPort(), on doit tout d'abord tester si un port ne possède pas déjà ce nom.

Lorsqu'on n'a plus besoin d'un port, on peut l'éliminer simplement, après avoir attendu les messages restants et y avoir répondu avec la routine ReplyMsg().

Si le Message-Port fait partie des ports publics, on doit l'écartier du système avec la fonction RemPort(Port), avant de l'éliminer (libérer la mémoire).

Pour simplifier la création de nouveaux ports de message, il y a la fonction CreatePort(). Celle-ci n'appartient pas au système d'exploitation mais se trouve dans la librairie du compilateur C Amiga (amiga.lib). Son code source C se présente de la manière suivante :

```
#include <exec/exec.h>

extern APTR AllocMem();
extern UBYTE AllocSignal();
extern struct Task *FindTask();

struct MsgPort *CreatePort (Name, Pri)
    char *Name;
    BYTE Pri;
{
    BYTE Signal;
    struct MsgPort *Port;

    if ((Signal=AllocSignal(-1)) == -1)
        return((struct MsgPort *)0);

    Port=AllocMem((ULONG)sizeof(*Port),MEMF_CLEAR|MEMF_PUBLIC);

    if (Port==0) {
        FreeSignal (Signal);
        return((struct MsgPort *)0);
    }

    Port->mp_Node.ln_Name = Name;
    Port->mp_Node.ln_Pri = Pri;
    Port->mp_Node.ln_Type = NT_MSGPORT;

    Port->mp_Flags = PA_SIGNAL;
    Port->mp_SigBit = Signal;
    Port->mp_SigTask = FindTask(0);

    if (name != 0)
        AddPort(Port);
    else
        NewList (&(Port->mp_MsgList));

    return(Port);
}
```

Cette fonction crée un Message-Port avec un nom et une priorité donnés. Le résultat contient l'adresse du nouveau port ou 0 lorsqu'il n'y a plus de mémoire ou de signal libre. Si le pointeur sur le nom est différent de zéro, le port sera inséré dans la liste des ports publics. Avec cette fonction, il est possible, par exemple, d'établir rapidement et simplement un ReplyPort.

Il existe aussi dans Amiga.Lib, une fonction permettant d'éliminer un port :

```
DeletePort(Port)
    struct MsgPort *Port;
{
    if ((Port->mp_Node.ln_Name) != 0)
        RemPort(Port);

    Port->mp_Node.ln_Type = 0xFF;
    Port->mp_MsgList.lh_Head=(struct Node *)-1;

    FreeSignal(Port->mp_SigBit);
```

```
    FreeMem(Port, (ULONG) sizeof(*Port));
}
```

DeletePort() élimine le port indiqué. Si cette fonction connaît le nom de ce dernier, elle pourra l'effacer de la liste des ports publics.

Task-Exceptions (*états d'exception*)

Il peut être gênant, lors de l'attente d'un signal, que la tâche ne puisse pas se poursuivre. Prenons par exemple un module qui dessine une fonction mathématique. Comme l'opération risque de durer assez longtemps, on doit avoir la possibilité d'interrompre le processus en appuyant sur une touche. Ceci doit être transmis par un Message-Port. On doit cependant tester sans cesse le signal à l'intérieur de la boucle de dessin, pour savoir si le port a reçu un message. Ce test va, malheureusement, ralentir la boucle. Cette situation peut être évitée sur l'Amiga au moyen d'une Task-Exception (état d'exception d'une tâche). Il s'agit d'une état d'exception de la tâche, qui est déclenché par un signal, comme on pouvait s'y attendre.

De même que pour un interrupt, la tâche sera interrompue par le déclenchement d'un signal. Ceci peut se passer à n'importe quel endroit. Puis le handler d'exception sera appelé. Celui-ci fait partie de la tâche d'origine, et a donc accès à ces données (en supposant qu'elles ne sont pas locales). Dans notre exemple, il peut affecter à la variable de boucle la valeur finale, mettant ainsi fin au dessin. En langage machine, il existe beaucoup plus de possibilités. On peut, par exemple, manipuler directement la tâche.

Pour rendre possible une Task-Exception, on doit passer par les étapes suivantes :

- ① L'adresse de départ du handler Exception doit être déposée dans le champ correspondant de la structure de tâche, tc_ExceptCode. On peut encore écrire un pointeur sur les données communes dans tc_ExceptData.
- ② On doit définir les signaux qui peuvent libérer une exception. C'est le champ tc_SigExcept de la structure de tâche qui s'en charge. Chaque signal qui y est activé déclenche une exception lorsqu'il est reçu par une tâche. Pour simplifier l'initialisation ou l'élimination d'un bit de ce champ, il existe une fonction spéciale : SetExcept. La description exacte de cette fonction se trouve dans le récapitulatif, à la fin de ce chapitre.

Si une exception apparaît, Exec met le contenu actuel des registres processeurs (PC, SR, D0-7 et A0-6) sur la pile Task, afin de permettre la suite du déroulement de la tâche à la fin de l'exception.

Puis un masque de signaux, renfermant tous les signaux d'exception permis, vient en D0. L'adresse se trouvant dans tc_ExceptData sera copiée dans A1. Ensuite commence le traitement des codes d'exception à l'adresse présente dans tc_ExceptCode.

La fin d'une exception doit être signalée par RTS. A ce moment-là, Exec reprend l'ancien contenu des registres dans la pile Task et continue le déroulement de la tâche.

Lors d'une exception, Exec empêche le déclenchement d'autres exceptions. Pour leur permettre de se déclencher à nouveau à la fin de l'exception, il faut rétablir en D0 la même valeur que celle qui s'y trouvait au début, et donc conserver le contenu de D0.

Si un signal de déclenchement d'une exception surgit pendant une exception, la nouvelle exception sera exécutée à la fin du déroulement de l'exception présente. Si un signal est déjà activé, avant qu'on autorise, par SetExcept, l'exécution d'une exception, celle-ci se déclenchera automatiquement.

Traps du processeur

L'autre état d'exception correspond aux Traps. C'est ainsi que l'on appelle les exceptions du processeur 68000. Elles ne doivent pas être confondues avec les exceptions Task, décrites plus haut, même si Motorola (la société qui fabrique le 68000) leur a attribué le même nom d'exception. Les exceptions 68000 suivantes seront caractérisées par Exec en tant que Traps :

Trap :

2	Erreur de Bus
3	Erreur d'adresse
4	Erreur d'instruction illégale
5	Division par zéro
6	Erreur CHK
7	Erreur TRAPV
8	Violation de privilège
9	Trace
10	Erreur avec 1010 au départ (line 1010 emulator)
11	Erreur avec 1111 au départ (line 1111 emulator)
32-37	Erreur TRAP

Un Trap est toujours la suite directe d'une déclaration dans un programme. Ceci peut être voulu (CHK, TRAPV, TRAP, 1010, 1111 ou TRACE), mais peut aussi provenir d'une erreur de programmation (Erreur de Bus, d'adresse, division par zéro, violation de privilège).

Si un Trap processeur se déclenche, Exec saute à un handler Trap. L'adresse du handler se trouve dans le champ tc_Trapcode. Normalement, on y trouve un pointeur sur le handler Trap standard d'Exec. Ceci engendre le message bien connu (malheureusement) "Software error - Task held", ou bien une "méditation du gourou".

On peut aussi dériver l'adresse présente dans tc_Trapcode vers un handler personnel, celui-ci pouvant réagir soit à tous les Traps, soit à un seul déterminé, puis sauter au handler standard. Un handler Trap est accessible presque directement. Exec se contente de mettre les numéros de Trap (Cf liste ci-dessus) sur la pile. En voici les conséquences :

En premier lieu, on se trouve en mode superviseur et on travaille avec la pile superviseur. Pendant le déroulement du handler Trap, le Task-Switching est donc désactivé. Deuxièmement, le contenu de la pile peut varier. Le format normal est le suivant :

Stackpointer	(SSP)	numéro de Trap (mot long)
Stackpointer	+4	registre d'état(mot)
Stackpointer	+2	adresse de retour (mot long)

Lors d'une erreur d'adresse ou de bus, d'autres messages sont mis sur la pile. Le déroulement sera différent suivant le processeur employé (68000, 68010, 68020). Pour que la compatibilité soit totale, il faut donc veiller à certains détails. Pour plus d'informations, reportez-vous à la littérature concernant le 68000.

Il se peut aussi que, lors d'une erreur d'adresse ou de bus, on saute au handler de trap de Exec, car ces exceptions n'offrent souvent aucun autre recours.

Pour libérer le handler Trap, il suffit de prendre le numéro de trap qui se trouve sur la pile (attention ! mot long), et d'y ajouter une instruction RTE. Comme aucun des registres d'Exec n'est sauvegardé sur la pile, et il ne faudra pas modifier de manière durable le contenu des registres processeurs.

L'exemple suivant utilise un handler Trap pour intercepter une division par zéro. Comme on peut difficilement écrire en C un tel handler Trap, il a été directement intégré dans le code source en langage machine au moyen de #asm et de #endasm. Etant donné que beaucoup de compilateurs C ne connaissent pas ce genre de déclarations du préprocesseur, il est conseillé de compiler et d'assembler séparément la partie C et la partie en langage machine, et de les linker par la suite. Ce programme a été conçu avec AZTEC C.

```
***** Interception d'une exception du 68000 ****/  
  
#include <exec/execbase.h>  
  
extern struct ExecBase *SysBase;  
  
main()  
{  
/** Rajout du handler Trap ***/  
  
extern APTR Trap;  
APTR oldtrap;  
USHORT Nombre1,Nombre2;  
struct Task *ThisTask;  
  
oldtrap=SysBase->ThisTask->tc_TrapCode;  
SysBase->ThisTask->tc_TrapCode=&Trap;  
SysBase->ThisTask->tc_TrapData=(APTR)0;  
  
/** Déclenchement d'un Trap "Division par zéro" **/
```

```

Nombre1 = 10; Nombre2 = 0;
Nombre1 = Nombre1/Nombre2;

if((ULONG)SysBase->ThisTask->tc_TrapData==0)
    printf("Cet emplacement n'est jamais atteint!\n");
else
    printf ("Exception reconnue, tc_TrapData = Numéro-de-Trap:%ld\n",
    SysBase->ThisTask->tc_TrapData);

/** Handler Trap désactivé **/
SysBase->ThisTask->tc_TrapCode=oldtrap;

}

/** Handler Trap **/
/** Seul Aztec C en assure le fonctionnement sûr **/
/** TAB devant Opcode nécessaire! **/

#asm

_trap move.1 a0,-(sp)
        move.1 4,a0 :SysBase
        move.1 276(a0),a0 ;SysBase->ThisTask
        move.1 4(sp),46(a0) ;Numéro de Trap dans
                            ;SysBase->ThisTask->tc_TrapData
        move.1 (sp),a0
        add.1 #8,sp
rte
#endasm

```

Sans le handler Trap, ce programme se serait effondré avec un "Software Error - Task held", étant donné qu'il y a division par zéro. La preuve qu'il y a eu déclenchement d'un Trap nous est donnée par l'instruction IF. En effet, seul le Trap peut donner à tc_TrapData une valeur différente de 0, lorsqu'il vient d'être effacé par la tâche. Directement après la division illégale, Exec saute au handler Trap. Celui-ci prend le numéro du Trap sur la pile superviseur et l'écrit dans le champ tc_TrapData. Puis l'instruction printf() affiche à l'écran le nombre 5 qui correspond en fait au numéro de trap d'une division par zéro.

Les instructions Trap

Un Trap déclenché par l'une des 16 instructions Trap (numéro 32 à 47) saute au handler de trap. On peut aussi déterminer à l'avance certains numéros de Trap, comme on le faisait pour les signaux, avec les fonctions AllocTrap et FreeTrap.(on trouvera l'explication précise dans la liste de fonctions qui va venir).

L'allocation et la libération des traps ne servent toutefois qu'à faire savoir quels sont les traps utilisés et ceux qui ne le sont pas. Si une instruction Trap apparaît, elle sera toujours dirigée vers le handler de trap actuel, et cela indépendamment du fait qu'une instruction Trap ait été définie avec AllocTrap ou non.

Fonctions du système de messages, des traps et des exceptions

AddPort

Fonction : AddPort(*Port*)
 a1

Offset : - 354 -\$162

Description :

AddPort() insère la structure indiquée de port de message dans la liste des ports publics. Elle sera classée suivant son niveau de priorité. On peut adresser la tête de liste par SysBase -> PortList. AddPort initialise aussi la structure mp_MsgList à l'intérieur du Message-Port.

Paramètre :

Port

Pointeur sur la structure de port de message.

AllocTrap

Fonction : Numtrap = AllocTrap(Numtrap)
 d0 *d0*

Offset : - 342 -\$156

Description :

AllocTrap permet d'allouer une des instructions Trap du 68000. Le numéro Trap peut être compris entre 0 et 15, initialisant ainsi l'instruction Trap correspondante. Avec le numéro -1, AllocTrap recherche la première instruction Trap libre.

Paramètre :

Numtrap

Nombre compris entre 0 et 15 pour une instruction trap déterminée (ou -1 pour la première instruction libre).

Résultat :

Numtrap

Contient l'instruction Trap allouée. Si numtrap = -1, c'est que l'instruction souhaitée n'est pas libre ou qu'il n'y a plus d'instruction libre.

FindPort

Fonction : Port = FindPort(Nom)
d0 a1

Offset : - 390 -\$186

Description :

FindPort() cherche dans la liste des ports publics le prochain Message-Port qui porte le nom indiqué. Si ce port existe, la fonction renvoie un pointeur sur ce port.

Paramètre :

Nom

Nom du port cherché.

Résultat :

Port

Pointeur sur le Message-Port. Lorsque ce dernier n'existe pas, Port = 0.

FreeTrap

Fonction : FreeTrap(Numtrap)
d0

Offset : - 348 -\$15C

Description :

FreeTrap libère l'instruction Trap caractérisée par le numéro donné.

Paramètre :

Numtrap

Numéro de l'instruction Trap (0-15).

GetMsg

Fonction : Message=GetMsg(Port)
D0 A0

Offset : -372 -\$174

Description :

GetMsg permet de recevoir un message à un port. Passe alors à la liste des messages et élimine le premier message de la liste.

Paramètre*Port*

Pointeur sur le port de message où le message a été reçu.

Résultat :*Message*

Pointeur sur le message qui se trouve en première place dans la liste. S'il n'y a pas de message, on obtient en retour un 0.

PutMsg

Fonction : PutMsg(*Port*, *message*)
 a0 a1

Offset : - 366 -\$16E

Description :

PutMsg envoie un message à un port de message. Dans ce dernier, il sera rajouté à la liste des messages et l'action correspondant au contenu du champ mp_Flags sera déclenchée.

Paramètres :*Port*

Adresse de la structure de Message-Port du port de destination.

Message

Pointeur sur la structure de message du message.

RemPort

Fonction : RemPort(*Port*)
 A1

Offset : - 360 -\$168

Description :

Cette fonction élimine un port de message de la liste des ports publics. Dans ce cas, il ne sera plus possible de l'adresser au moyen de FindPort.

Paramètre :

Port

Pointeur sur le port de message.

ReplyMsg

Fonction : ReplyMsg(message)
A1

Offset : - 378 -\$17A

Description :

ReplyMsg() envoie un message de retour à son Reply-Port. Si le champ mn_ReplyPort de la structure de message est égal à 0, cette fonction sera ignorée.

Paramètre :

Message

Pointeur sur la structure de message.

SetExcept

Fonction : AncSignaux = SetExcept(NouvSignaux, masque)
d0 d0 d1

Offset : - 312 -\$138

Description :

SetExcept détermine les signaux qui peuvent être déclenchés par une exception. Le comportement de cette fonction est identique à celui de SetSignal().

Paramètres :

NouvSignaux

Nouveaux états des signaux Exception.

Masque

Le masque établit quels signaux Exception doivent être influencés.

Résultat :**AncSignaux**

Etat des signaux Exception avant la modification.

WaitPort

Fonction : Message = WaitPort(Port)
 D0 a0

Offset : - 384 -\$180

Description :

WaitPort attend la réception d'un message à un Port déterminé. Si un message y parvient ou s'il y en avait déjà un présent avant l'appel de la fonction, WaitPort() retournera l'adresse de ce message. Le message ne sera pas éliminé de la liste des messages reçus. Pour ce faire, on devra utiliser GetMsg().

Paramètre :*Port*

Adresse du Port.

Résultat :*Message*

Pointeur sur le premier message de la liste.

2.7. Fonction de la mémoire de l'Amiga

L'Amiga dispose d'une gestion dynamique de la mémoire. Cela signifie que la mémoire d'écran, la mémoire de disque, etc, ainsi que les programmes à charger ne sont pas placés dans un secteur de mémoire déterminé et invariable, mais peuvent se voir affecter un nouvel emplacement à chaque fois. Cette gestion dynamique de la mémoire permet le traitement simultané de plusieurs programmes en mémoire, étant donné qu'aucun d'entre eux ne se verra attribuer une zone mémoire prédéfinie, comme c'est le cas, par exemple, avec le C64.

Le système doit être, par contre, informé de la mémoire nécessaire à un programme, afin d'en attribuer une quantité suffisante à l'utilisateur. Le système ne sait pas que tel programme (tâche) occupe telle zone mémoire, mais il sait qu'il n'en a plus à sa disposition pour de nouvelles tâches. Pour dire les choses plus précisément, le système ne connaît pas les emplacements occupés, mais uniquement les emplacements libres.

Lorsqu'une tâche n'a plus besoin d'une zone mémoire déterminée, elle avertit le système, afin que cette zone puisse être attribuée à une autre tâche. Si la mémoire dont elle n'a plus besoin n'est pas rendue au système, cette zone restera occupée jusqu'au prochain RESET, provoquant ainsi une diminution rédhibitoire de la capacité mémoire.

L'occupation de la mémoire n'est possible que par blocs de 8 octets. Autrement, le système arrondira la taille mémoire donnée au prochain multiple de 8 octets. La zone mémoire minimale disponible est donc de 8 octets.

Pour occuper ou libérer une zone mémoire particulière, il existe dans la librairie Exec plusieurs fonctions, dont deux sont particulièrement importantes. Il s'agit des fonctions AllocMem() et FreeMem().

Pour définir des conditions de mémoire, il faut communiquer au système le type de mémoire à attribuer, ainsi que certaines caractéristiques particulières. Les conditions sont définies dans le fichier Include "exec.memory.x". x représente ici la lettre h (fichiers Include C) ou la lettre i (fichiers Include assembleur).

Ces conditions que l'on peut définir sont les suivantes :

MEMF_CHIP

Indique que la mémoire affectée doit être de type CHIP-MEMORY. Chip-Memory est la mémoire à laquelle peuvent accéder les chips (éléments DMA comme le blitter). Il s'agit du bloc de mémoire situé le "plus bas" dans l'Amiga. Le graphisme et le son doivent toujours être stockés dans cette partie. Même si, à l'heure actuelle, cette zone ne possède que 512 Koctets (à moins de posséder un super fat Agnus et 1 Mo de RAM...) et que cette condition est toujours réalisée, il vaut mieux ne pas renoncer à une détermination exacte, pour des raisons de compatibilité avec les appareils ayant une mémoire plus grande. Le code de cette condition est \$02 et il est défini dans le fichier Include "exec/memory.h" en C ou "exec/memory.i" en assembleur, comme pour toutes les autres conditions.

MEMF_FAST

Indique que la mémoire affectée doit se trouver en dehors de la zone inférieure de 512 Ko (CHIP-MEMORY). L'affectation ne sera possible que si on dispose d'une extension RAM. Le code de cette condition est \$04.

MEMF_PUBLIC

La mémoire à affecter est nécessaire aux structures qui utilisent plusieurs tâches (par exemple les messages et les paquets). Le code de cette condition est de \$01.

MEMF_CLEAR

Indique que la mémoire affectée doit être remplie par des 0. Le code de cette condition est \$10000.

MEMF_LARGEST

Indique que la mémoire affectée doit être un bloc mémoire de la plus grande taille disponible. Le code est \$20000.

Si plusieurs conditions doivent être réalisées, comme par exemple CHIP et CLEAR, les codes devront être couplés avec un OU logique.

Si la condition "CHIP OU FAST-MEMORY" doit être réalisée, le système cherchera d'abord à exécuter FAST-MEMORY. En cas d'échec, c'est CHIP-MEMORY qui sera exécuté.

Attention : On doit éviter d'occuper ou de libérer de la mémoire lors d'une interruption, étant donné que les routines ont été conçues pour ne pas bloquer les interruptions. Si une tâche se trouve dans une routine de gestion de mémoire, et s'il intervient une interruption qui travaille également avec des routines mémoire, le système risque de se trouver en grande difficulté.

2.7.1. Les fonctions AllocMem() et FreeMem()

AllocMem

Fonction : Mémoire = AllocMem(taillemem, conditions)
d0 d0 d1

Offset : - 198 -\$0C6

Description :

La fonction cherche une zone mémoire libre, qui correspond aux conditions données, puis la caractérise comme zone allouée. L'adresse de départ de la zone trouvée sera donnée par d0. S'il n'est pas possible d'occuper la zone mémoire souhaitée, le système retournera un 0 dans d0 pour signaler l'erreur.

Paramètres :*Taillemem*

Indique la taille de la mémoire qui doit être attribuée.

Conditions

Correspond aux conditions énoncées précédemment, qui sont transmises à la fonction AllocMem(), et qui sont recherchées dans la zone de mémoire (par exemple Chip-Memory).

Avec la fonction AllocMem(), il n'est pas possible d'attribuer une zone mémoire déterminée, mais il est possible d'en recevoir une qui soit disponible.

De même qu'il existe une fonction permettant d'attribuer et d'occuper de la mémoire, il y a aussi une fonction permettant de libérer de la mémoire.

FreeMem

Fonction : FreeMem(blocmémoire, taille)
 a1 d0

Offset : - 210 -\$002

Description :

La fonction rend au système la zone mémoire précédemment occupée, permettant ainsi une nouvelle affectation pour une autre tâche. Les paramètres pris en charge par cette fonction seront arrondis.

Paramètres :*Blocmémoire*

Pointeur sur le début de la zone mémoire, qui doit être rendue au système. Le pointeur sera arrondi à un multiple de 8.

Taille

Indique la quantité de mémoire à libérer. La taille sera arrondie à un multiple de 8.

Attention :

Si on cherche à libérer de la mémoire non occupée, le système s'effondrera avec le numéro de gourou suivant : 81000009. Le programme C qui suit montre comment on peut disposer de la mémoire, puis à nouveau la libérer.

```
#include <exec/memory.h>
#include <exec/types.h>
```

```
#define SIZE 1000

main()
{
ULONG Memoire;

Memoire = AllocMem (SIZE, MEMF_CHIP | MEMF_PUBLIC);

if (Memoire == 0) {
printf ("\n Mémoire non attribuée\n");
exit (0);
}
printf ("\n Mémoire attribuée\n");

FreeMem (SIZE,Memoire);
}
```

Dans "MEMOIRE" sera conservée l'adresse de départ de la mémoire attribuée.

2.7.2. La structure Memory-List

Il est souvent nécessaire d'allouer plusieurs zones mémoire. Pour cette raison, on pourrait appeler, pour chaque zone, la fonction AllocMem. Mais on voit tout suite que c'est une méthode coûteuse. C'est pourquoi la librairie Exec dispose de deux fonctions pour nous faciliter la tâche. Ces fonctions se nomment AllocEntry() et FreeEntry(). Pour pouvoir les appeler, on doit en premier lieu initialiser une structure, dans laquelle les fonctions prendront leurs valeurs. Cette structure s'appelle MemList, et elle se trouve dans le fichier "exec/memory.h", ou dans le fichier "exec/memory.i" en assembleur. Elle est de la forme suivante :

```
struct MemList {
    struct Node ml_Node;
    UWWORD ml_NumEntries;
    struct MemEntry ml_ME [1];
};

STRUCTURE ML_LN_SIZE      00 $00
    UWWORD   ML_NUMENTRIES 14 $0E
    LABEL    ML_ME          16 $10
    LABEL    ML_SIZE         16 $10
```

ml_Node

C'est une structure de type Node, capable de coupler plusieurs structures MemList.

ml_NumEntries

Indique la quantité de mémoire qu'on pense occuper.

ml_ME[1]

C'est une structure isolée, qui contient les conditions de la mémoire à réserver ainsi que la taille de la zone. La structure est de la forme suivante :

```
struct MemEntry {
    union {
        ULONG meu_Req;
        APTR meu_Addr;
    } me_Un;
    ULONG me_Length;
};

#define me_un me_Un
#define me_Req me_Un.meu_Req
#define me_Addr me_Un.meu_Addr

Offsets
STRUCTURE ME,0
LABEL ME_REQS      00      $00
APTR ME_ADDR       00      $00
ULONG ME_LENGTH    04      $04
LABEL ME_SIZE      08      $08
```

Cette structure se partage selon la condition 'union'. Elle peut contenir soit les conditions (Requests) de l'affectation de la mémoire, soit le pointeur sur la mémoire réservée. Lors de l'installation de la structure MemEntry pour l'appel de la fonction AllocEntry(), on y trouve les conditions d'affectation (par exemple : MEMF_CHIP) et après l'appel de la fonction, l'adresse de départ de la zone mémoire affectée.

me_Length

Donne la longueur de la mémoire réservée.

AllocEntry

Fonction : Liste = AllocEntry(MemList)
d0 a0

Offset : - 222 -\$0DE

Description :

Cette fonction transmet un pointeur sur la structure MemList dans a0. Elle cherche ensuite à occuper toutes les zones mémoire insérées dans la structure. Dès qu'une zone est occupée, le pointeur dirigé vers cette zone sera mis à la place des conditions.

Paramètre :*Liste*

Pointeur sur la structure MemList, nouvellement créée. Si une erreur apparaît pendant l'affectation, on obtiendra en retour dans d0 la condition de la mémoire non attribuée, où le bit de poids fort (bit 31) sera activé. Dans ce cas, il n'y aura pas de mémoire occupée, même si d'autres Entrées (Entries) auraient pu être exécutées.

FreeEntry

Fonction : FreeEntry(liste)
a0

Offset : - 228 -\$0E4

Description :

Cette fonction retourne au système toutes les zones mémoire conservées dans la structure MemList. Il n'est pas possible de traiter simultanément plusieurs structures MemList avec la fonction FreeEntry(), pas plus qu'avec AllocEntry().

Paramètre :*Liste*

Pointeur sur la structure MemList qui a été transmise par la fonction AllocEntry().

Vous vous demandez peut-être comment il est possible d'initialiser plusieurs Entries avec une structure MemList, alors que cette dernière n'est reliée qu'à une structure MemEntry(ml_ME [1]). Pour initialiser plusieurs Entries, on doit utiliser une astuce. On crée pour cela une structure isolée, qui est par exemple de la forme suivante :

```
struct {
    struct MemList me_tete;
    struct MemEntry me_rajout [3];
} MaListe;
```

On ne transmet pas à la fonction AllocEntry un pointeur sur la structure MemList, mais un pointeur sur sa propre structure initialisée, avec laquelle il sera possible d'initialiser plusieurs Entries. Dans notre exemple, 4 zones mémoire seront occupées par la fonction AllocEntry, étant donné que la structure MemList dispose encore d'une structure MemEntry qui lui est propre :

Voici l'utilisation de la fonction AllocEntry() sur un exemple :

```
#include <exec/memory.h>
#include <exec/types.h>

struct MemListe {
    struct MemList me_Tete;
```

```

    struct MemEntry me_Rajout[2]; }

main()
{
    struct MemList *Listememoire, *AllocEntry();

    struct MemListe MaListe;

    MaListe.me_Tete.ml_NumEntries = 3;
    MaListe.me_Tete.ml_me[0].me_Req = MEMF_CLEAR;
    MaListe.me_Tete.ml_me[0].me_Length = 100;
    MaListe.me_Tete.ml_me[1].me_Req = MEMF_CLEAR | MEMF_FAST;
    MaListe.me_Tete.ml_me[1].me_Length = 1900;
    MaListe.me_Tete.ml_me[2].me_Req = MEMF_PUBLIC | MEMF_CHIP;
    MaListe.me_Tete.ml_me[2].me_Length = 300;

    Listememoire = AllocEntry (&MaListe);

    if ( ((ULONG)Listememoire) >> 30) {
        printf("\n Toutes les entrées ne peuvent être attribuées\n");
        exit (0);
    };
}

```

2.7.3. L'affectation de mémoire et les tâches

Quand une partie de la mémoire doit être occupée par une tâche, il est recommandé de le faire avec la fonction AllocEntry(). Dans la structure de tâche est intégrée en effet une structure de liste (tc_MemEntry), dans laquelle la mémoire occupée sous la forme d'une structure MemList peut être reliée à une liste.

La mémoire qui est reliée à ces listes peut alors facilement être rendue au système par le biais des routines que la tâche déclenche.

Un autre avantage de l'insertion de la mémoire utilisée dans la liste est que la tâche peut reconnaître à tout moment les zones qu'elle occupe.

Il est naturellement aussi possible d'allouer une zone mémoire avec la fonction AllocMem() et de l'insérer dans une telle liste.

2.7.4. La gestion interne de la mémoire

Puisqu'on connaît maintenant la façon de réservé un emplacement en mémoire, nous allons aborder la gestion interne de la mémoire.

Comme on peut l'imaginer, la gestion de la mémoire se fait à nouveau au moyen d'une structure. Celle-ci se présente ainsi :

```

struct MemHeader {
    struct Node mh_Node;
    UWORLD mh_Attributes;
}

```

```
    struct MemChunk *mh_First;
    APTR mh_Lower;
    APTR mh_Upper;
    ULONG mh_Free;
};

#define MEMF_PUBLIC (1L<<0)
#define MEMF_CHIP (1L<<1)
#define MEMF_FAST (1L<<2)
#define MEMF_CLEAR (1L<<16)
#define MEMF_LARGEST (1L<<17)
#define MEM_BLOCKSIZE 8L
#define MEM_BLOCKMASK 7L

          Offsets
STRUCTURE MH_LN_SIZE    00  $00
UWORD   MH_ATTRIBUTES   14  $0E
PAR     MH_FIRST        16  $10
APTR    MH_LOWER        20  $14
APTR    MH_UPPER        24  $18
ULONG   MH_FREE         28  $1C
LABEL   MH_SIZE         32  $20
```

mh_Node

Structure de noeud, permettant de coupler la structure MemHeader à une structure List.

mh_Attributes

Indique les conditions de la mémoire à gérer (ex. MEMF_FAST).

***mh_First**

Pointeur sur la première structure MemChunk. L'architecture et le sens de cette structure seront détaillés plus loin.

mh_Lower

Pointeur sur le début de la mémoire gérée par le Header.

mh_Upper

Pointeur sur la fin de la mémoire gérée par le Header.

mh_Free

Indique la quantité de mémoire disponible à l'aide du Header.

Comme nous l'avons déjà dit, le système ne sait pas quelle est la mémoire allouée, mais connaît au contraire les emplacements qui ne sont pas encore alloués. Les zones mémoire inaccessibles sont reliées entre elles à l'aide d'une structure MemChunk. Par cette dernière, le système détermine sans difficulté la taille et la position des blocs de mémoires libres. Cette structure est de la forme suivante :

```
struct MemChunk {
    struct MemChunk *mc_Next;
    ULONG mc_Bytes;
};

STRUCTURE offsets
APTR MC_0
APTR MC_NEXT 00 $00
ULONG MC_BYTES 04 $04
APTR MC_SIZE 08 $08
```

* **mc_Next**

Pointeur sur la prochaine structure MemChunk.

mc_Bytes

Indique le nombre d'octets libres dans ce bloc mémoire.

L'entrée mh_First de la structure MemHeader pointe sur la première structure MemChunk qui se trouve au début des premiers blocs mémoire libres. Les 4 premiers octets de ces blocs libres constituent le pointeur sur la zone libre suivante (la structure MemChunk suivante). Les 4 octets suivants indiquent la taille de la zone mémoire actuelle. Dans la dernière structure MemChunk, le pointeur mc_Next est mis à zéro et mc_Bytes indique le nombre d'octets entre la position mémoire présente et la valeur qui est conservée dans mh_Upper.

Une structure MemHeader gère l'ensemble de la Chip-Memory et une autre structure MemHeader gère la Fast-Memory, s'il y en a une. Ces structures sont chaînées dans une liste qui se trouve dans la structure ExecBase. La liste porte le nom "MemList" et se trouve à l'Offset 322.

La priorité de MemHeader pour la zone Fast-Memory est 0, et pour la zone Chip-Memory -10. C'est la raison pour laquelle le système cherchera d'abord à allouer la Fast-Memory, puis seulement après la Chip-Memory.

Si un emplacement en mémoire doit être alloué, la MemList de la structure ExecBase sera consultée afin de voir si les conditions données dans la fonction AllocMem() correspondent à celles de la structure MemHeader. En deuxième lieu, la mémoire libre indiquée dans mh_Free est testée, pour voir si elle suffit, pour que l'on puisse réservé la quantité de mémoire indiquée. Si l'une des deux conditions n'est pas remplie, le système se demandera s'il existe encore une structure MemHead disponible. S'il n'en existe pas, un message négatif sera retourné par la fonction AllocMem(). Sinon, le

système va chercher le pointeur `mh_First` et se demande si la première structure `MemChunk` donnée par la mémoire suffit.

Si ce n'est pas le cas, on passe à la structure `MemChunk` suivante (c'est-à-dire au prochain bloc de mémoire libre), et une nouvelle comparaison est effectuée. Si l'on ne rencontre pas de mémoire suffisante en un seul bloc, on reçoit en retour un message négatif.

Si l'on rencontre au contraire une zone de mémoire de taille suffisante, le système calcule la quantité encore libre dans ce bloc de mémoire, et insère une structure `MemChunk` à l'endroit où commence la mémoire libre. La zone ainsi affectée est supprimée de la chaîne, et le nombre d'octets alloués est retiré du nombre total d'octets disponibles.

2.7.5. Les fonctions `Allocate`, `Deallocate` et `AddMem`

Il est possible de créer soi-même une structure `MemHead` et de gérer une zone mémoire à part avec les fonctions `Allocate()` et `Deallocate()`. Ces dernières permettent en tout et pour tout d'occuper ou de libérer une zone mémoire sans pouvoir faire appel à quelque condition que ce soit.

Allocate

Fonction : Mémoire = Allocate(`MemHeader`, `Bytesize`)
d0 a0 d0

Offset : - 186 -\$0BA

Description :

Cette fonction alloue la quantité de mémoire indiquée, gérée par la structure `MemHead` donnée.

Paramètres :

MemHeader

Pointeur sur la structure `MemHead`.

ByteSize

Indique la quantité de mémoire à allouer.

Mémoire

Pointeur sur la mémoire allouée. Si l'on n'a pas trouvé de bloc mémoire, on obtient en retour un 0.

Deallocate

Fonction : DeAllocate(MemHeader, Memoire, ByteSize)
 a0 a1 d0

Offset : -192 -\$0C0

Description :

Cette fonction retourne la mémoire allouée à la structure MemHeader.

Paramètres :

MemHeader

Pointeur sur la structure MemHeader.

Mémoire

Pointeur sur le début de la mémoire à libérer.

ByteSize

Indique la quantité de mémoire à libérer.

AddMEmList

Fonction : error = AddMemList(size,req,pri,basis,name)
 D0 D0 D1 D2 A0 A1

Offset : -618 -\$26A

Description :

La fonction crée un Memory-Header, et elle l'insère dans la liste Exec-Memory.

Paramètres :

size

Taille de la mémoire à gérer.

req

Indique le type de mémoire qui doit être gérée par l'internédaire de la structure MemHeader. Exemple :

MEMF_PUBLICMEMF_FAST

pri

Indique quelle priorité doit avoir la structure. La mémoire gérée par l'intermédiaire d'une structure MemHeader avec une priorité élevée est affectée avant la mémoire de priorité inférieure.

La Fast-Memory, si une structure MemHeader a été créée par le système d'exploitation, a une priorité 0. La Chip-Memory a une priorité égale à -10.

basis

Pointeur sur le début de la mémoire.

name

Pointeur sur le nom de la mémoire (par exemple hyprafast.memory).

2.7.6. Description des autres fonctions

AvailMem

Fonction : AvailMem(conditions)
d1

Offset : - 216 -\$0D8

Description :

Cette fonction indique la taille de la mémoire caractérisée par les conditions (par exemple MEMF_CHIP).

AllocAbs

Fonction : Mémoire = AllocAbs(ByteSize, Position)
d0 d0 a1

Offset : -204 -\$0CC

Description :

Cette fonction permet d'allouer une zone mémoire déterminée, qui ne sera pas recherchée par Exec, mais indiquée par le programmeur.

Paramètres :***ByteSize***

Indique la taille de la mémoire à allouer.

Position

Pointeur sur la mémoire à allouer.

Mémoire

Pointeur sur la mémoire allouée, correspondant à celle qui est indiquée.

S'il n'est pas possible d'allouer la mémoire souhaitée, on obtient en retour un 0 comme message d'erreur.

2.8. La structure interne des librairies

On définit la structure de librairie comme suit dans le fichier Include "exec/libraries.h" :

```
extern struct Library {
    struct Node lib_Node;
    UBYTE   lib_Flags;
    UBYTE   lib_pad;
    UWORLD  lib_NegSize;
    UWORLD  lib_PosSize;
    UWORLD  lib_Version;
    UWORLD  lib_Revision;
    APTR    lib_IdString;
    ULONG   lib_Sum;
    UWORLD  lib_OpenCnt;
};
```

En assembleur, la structure est définie dans le fichier Include "exec/libraries.i".

Offsets			
STRUCTURE	LIB.LN_SIZE	00	\$00
UBYTE	LIB_FLAGS	14	\$0E
UBYTE	LIB_pad	15	\$0F
UWORD	LIB_NEGSIZE	16	\$10
UWORD	LIB_POSSIZE	18	\$12
UWORD	LIB_VERSION	20	\$14
UWORD	LIB_REVISION	22	\$16
APTR	LIB_IDSTRING	24	\$18
ULONG	LIB_SUM	28	\$1C
UWORD	LIB_OPENCNT	32	\$20
LABEL	LIB_SIZE	34	\$22

Flags internes de librairie:

```
BITDEF LIB_SUMMING,0
BITDEF LIB_CHANGED,1
BITDEF LIB_SUMUSED,2
BITDEF LIB_DELEXP,3
```

Explication de la structure :

Lib_Node

Structure de type noeud. A l'aide de cette structure, les librairies sont reliées pour former une liste. Le type de librairie est dans ce cas, naturellement, 'NT_LIBRARY' et le nom du noeud correspond au nom de la librairie.

lib_Flags, lib_pad

Octet inutilisé, qui sert à mettre les mots ou les mots longs suivants de la structure à nouveau sur une adresse paire.

lib_NegSize

Indique la taille de la zone pour l'offset négatif.

lib_PosSize

Indique la taille de la zone occupée par la librairie à partir de l'adresse de base. Cette valeur est importante, autant que l'indication concernant la taille du secteur négatif, pour la suppression de la librairie, car elles permettent de déterminer la longueur de cette librairie.

lib_Version

Indique la version de la librairie sélectionnée.

lib_Revision

Indique la quantité de remaniements d'une librairie sélectionnée.

lib_IdString

Pointeur sur un texte, contenant des informations supplémentaires sur la librairie.

lib_Summ

Somme de contrôle pour une librairie donnée. Si cette dernière est modifiée, la somme devra être recalculée.

lib_OpenCnt

Indique le nombre de tâches utilisant cette librairie. Lorsqu'aucune tâche n'accède à la librairie, la possibilité de fermer la librairie dépend de son type.

Les fonctions d'une librairie sont disponibles pour l'utilisateur à partir de l'offset -30, même si elles peuvent théoriquement débuter à -6. Lorsqu'on examine attentivement les premiers offsets, on remarque qu'ils pointent sur des fonctions utilisées par Exec pour gérer les librairies.

Il s'agit en fait de fonctions nécessaires à l'ouverture et à la fermeture d'une librairie, et auxquelles accèdent les fonctions Exec. Chaque librairie possède donc ses propres routines d'ouverture et de fermeture. C'est dans ces dernières que se décide si une librairie non utilisée doit être fermée ou non. Comme on peut le voir à partir de la définition des fichiers Include, les 4 offsets dont il a été question ont les significations suivantes :

LIB_OPEN	-6	ouverture d'une librairie
LIB_CLOSE	-12	fermeture d'une librairie
LIB_EXPUNGE	-18	suppression d'une librairie
LIB_EXTFUNC	-24	ouverte pour extension

Les librairies non éliminées lorsqu'elles ne sont plus nécessaires n'utilisent pas LIB_EXPUNGE. Tous les offsets non utilisés doivent pointer sur une routine qui désactive D0 et revient ensuite à son travail.

2.8.1. Modifier une librairie

Pour modifier une librairie existante, on utilise une fonction, présente dans la librairie Exec et permettant la modification de certains offsets. Cette fonction est de la forme suivante :

SetFunction

Fonction : SetFunction(Librairie, offset, saut)
 a1 a0 d0

Offset : -420 -\$1A4

Description :

Modifie le saut d'offset d'une fonction définie avec un offset négatif, de façon à ce que l'offset de la fonction pointe désormais sur la nouvelle routine. La somme de contrôle de la librairie est recalculée.

Paramètres :*Librairie*

Pointeur sur la librairie à modifier.

Offset

Indique l'offset de la fonction, qui doit être modifié.

Saut

Pointeur sur la routine.

2.8.2. La création d'une librairie

Nous avons expliqué comment on utilise les librairies. Nous allons maintenant montrer comment s'y prendre pour créer une librairie.

La création d'une librairie est utile lorsqu'on a l'intention d'employer plusieurs tâches fonctionnant en parallèle et utilisant un certain nombre de fonctions communes. Il est également conseillé de créer un librairie dans laquelle on placera des fonctions que l'on utilise très souvent.

Une fois qu'une librairie a été créée, il est possible de l'ouvrir à volonté pour utiliser les fonctions qu'elle contient. Mais il faut néanmoins qu'elle se trouve pour cela dans le répertoire LIBS de la disquette système.

Pour créer une librairie, Exec met à votre disposition plusieurs fonctions, que nous allons expliciter avant de passer aux explications sur la création d'une librairie :.

InitStruct

Fonction : `InitStruct(initTable, Mémoire, Taille)`
 A1 A2 D0

Offset : -78 -\$04E

Description :

La fonction initialise une structure à partir du secteur indiqué en mémoire, et selon une table données.

Paramètres :*InitTable*

Pointeur sur la table avec laquelle on crée la structure.

Mémoire

Pointeur sur la mémoire allouée.

Taille

Indique la taille de la structure à initialiser. La mémoire où la structure est créée n'a pas besoin d'être vidée, car la fonction InitStruct se charge de cette opération.

La table à partir de laquelle on crée la structure a un aspect quelque peu déconcertant. Elle se compose en effet d'un octet de commande, suivi de données qui seront traitées de manières différentes suivant l'aspect de l'octet de commande. Après ces données, on trouve à nouveau un octet de commandes et des données. La longueur des données dépend à nouveau de l'octet de commande. La fin de la table est marquée par un octet de commande de valeur nulle.

L'octet de commande est divisé en deux niveaux (nibbles), niveau inférieur et niveau supérieur. Les combinaisons de bits du niveau supérieur (4 bits de plus fort poids) correspondent à la commande, alors que le niveau inférieur correspond au nombre d'exécutions de la commande.

Nous allons nous consacrer en premier lieu au nibble supérieur. Ce dernier est à nouveau divisé en deux bits supérieurs et deux bits inférieurs. Les deux bits de poids fort correspondent à la commande proprement dite. Les deux bits de poids faible donnent la taille des données avec lesquelles l'octet de commande opérera. Les différentes tailles de données disponibles sont le mot long, le mot et l'octet.

On pourra donc obtenir 4 commandes différentes avec les 2 bits de poids fort.

Combinaison 00

Cette combinaison indique que les données débutant après l'octet de commande sont transférées dans la structure créée. Le type de données (mot long, mot, octet) à traiter est défini par les deux bits suivants

Combinaison 01

Cette combinaison indique que la donnée de longueur indiquée sera copiée autant de fois dans la structure créée.

Combinaison 10

Cette combinaison indique que l'octet qui suit le mot de commande sert d'offset dans la structure créée. L'offset est additionné à l'adresse de départ de la structure et les données situées après cet octet sont copiées dans la structure créée.

Combinaison 11

Cette combinaison indique que les trois octets suivant l'octet de commande sont utilisés comme offset 24 bits. En dehors de ce fait, la commande est identique à la précédente.

Les deux bits suivants de l'octet de commande correspondent au type des données à traiter (bits 4 et 5).

Combinaison 00 :	mot long	(adresse paire uniquement)
Combinaison 01 :	mot	(adresse paire uniquement)
Combinaison 10 :	octet	
Combinaison 11 :	ne pas utiliser,	sous peine d'effondrement

Le nibble inférieur de l'octet de commande indique le nombre d'exécutions d'une fonction donnée; il sert de compteur. Comme ce compteur peut être décrémenté jusqu'à -1, la fonction est toujours exécutée une fois de plus que le nombre indiqué par le compteur. L'octet de commande doit toujours être à une adresse paire.

Voici deux exemples, qui expliciteront la situation :

```
dc.b %00010010,$00  
dc.w $FFFF,$FFFF,$1234
```

L'octet de commande correspond à \$12 (%00010010) et établit donc que les trois mots suivants seront copiés dans la structure. L'octet nul suivant l'octet de commande est nécessaire, car les mots doivent débuter à une adresse paire.

```
dc.b %10000001,$10  
dc.l $12341234,$FFFF1111
```

L'octet de commande indique que deux mots longs doivent être copiés à partir de la position 16 dans la structure.

Vous admettrez sans doute que la création d'une telle table demande bien plus de temps que l'initialisation d'une structure faite "à la main". C'est pourquoi, les fichiers Include de Commodore mettent à votre disposition quatre macros très utiles, grâce auxquelles la création d'une telle table devient une partie de plaisir. Les macros se trouvent dans le fichier "exec/initializers.i". INITBYTE sert à faire figurer dans la structure un octet avec l'offset indiqué.

```
INITBYTE      MACRO    * &offset,&value  
DC.B        $e0  
DC.B        0  
DC.W        \1  
DC.B        \2  
DC.B        0  
ENDM
```

La macro suivante a les mêmes effets que INITBYTE, mais se réfère à un mot :

```
INITWORD     MACRO    * &offset,&value  
DC.B        $d0  
DC.B        0  
DC.W        \1
```

```
DC.W    \2
ENDM
```

La troisième macro ressemble à la macro INITBYTE, mais se réfère aux mots longs :

```
INITLONG MACRO * &offset,&value
DC.B   $c0
DC.B   0
DC.W   \1
DC.L   \2
ENDM
```

Enfin, la dernière macro peut être utilisée pour copier plusieurs valeurs dans la structure. Elle copie les données suivantes la commandes à partir de l'offset indiqué. Le nombre de valeurs copiées est transmis à la macro.

On indique en premier lieu la taille des valeurs à traiter. Les valeurs permises sont :

0	pour les mots longs
1	pour les mots
2	pour les octets

L'indication suivante, par laquelle débutent les entrées qui figurent dans la structure, définit l'offset. La macro reconnaît s'il s'agit d'un octet ou d'un offset 24 bits.

La troisième valeur a visiblement été oubliée. En effet, cette macro exige un troisième paramètre, même si sa valeur n'a pas d'importance (elle n'est pas utilisée).

Le quatrième paramètre indique le nombre de valeur qui doivent être transférées dans la structure. Le nombre maximum est 15.

```
INITSTRUCT MACRO * &size,&offset,&value,&count
DS.W  0
IFC  '\4',''
COUNT@ SET 0
ENDC
IFNC  '\4',''
COUNT@ SET \4
ENDC
CMD@ SET (((\1)<<4)!COUNT@)
IFLE (\2)-255
DC.B  (CMD@)!$80
DC.B  \2
MEXIT
ENDC
DC.B  CMD@!$0C0
DC.B  (((\2)>>16)&$0FF)
DC.W  ((\2)&$FFFF)
ENDM
```

Cette macro, plus complexe que les autres, peut être utilisée de la façon suivante :

```
INITSTRUCT 1.10.0.5
ds.w 0
```

```
dc.w $1111,$2222,$3333,$4444,$5555
INITBYTE ....
```

Exemple de programme :

Dans cet exemple, on réserve un emplacement en mémoire pour une structure de librairie, et on l'initialise partiellement :

```
include "exec/types.i"
include "exec/nodes.i"
include "exec/libraries.i"
include "exec/initializers.i"
include "exec/memory.i"
STRUCTURE MyLib,LIB_SIZE
    STRUCT      myl_champ,10
    LABEL MyLib_SIZE
XREF _AbsExecBase
XREF _LVOInitStruct
XREF _LVOAllocMem

        move.l      _AbsExecBase,a6
        move.l      #MEMF_CLEAR!MEMF_PUBLIC,d1
        move.l      #MyLib_SIZE,d0
        jsr         _LVOAllocMem(a6)
        tst.l      d0
        beq.s      Erreur_Alloc
        move.l      d0,a2
        lea          Tableau,a1
        move.l      #MyLib_SIZE,d0
        jsr         _LVOInitStruct(a6)
Erreur_Alloc :
        rts

Byte      EQU      2

Tableau:
        INITBYTE    LN_TYPE,NT_LIBRARY
        INITLONG    LN_NAME,MyName
        INITSTRUCT  Byte,myl_champ,0,10

;Valeurs copiées.
;Les valeurs fournies ici n'ont aucune signification.
;Elles servent seulement d'exemple.

        dc.b 11,22,33,44,55,66,77,88,99,00
        ds.w        0
        dc.l        0           ;Fin du tableau

MyName:    dc.b 'malibrary.library',0
END
```

MakeLibrary

Fonction : Library = MakeLibrary(Vecteur,Structure,Init,Taille,SegListe);
d0 a0 a1 a2 d0 d1

Offset : -84 -\$054

Description :

Il est possible, avec cette fonction, de créer une structure de librairie ou des structures voisines (device, resource). La place en mémoire nécessitée par la structure de librairie est allouée par la fonction elle-même ; lib_MemSize et lib_PosSize sont correctement initialisées par celle-ci.

Paramètres :

Vecteur

Pointeur sur la table des vecteurs de la librairie. Dans ce tableau, on trouve ou bien directement les pointeurs sur les différentes fonctions, ou bien les offsets qui seront additionnés à l'adresse de base pour obtenir le saut des fonctions. Si vous avez stocké des offsets dans le tableau, celui-ci devra débuter à \$FFFF (-1). La marque de fin de tableau est à nouveau -1 avec la même longueur pour les entrées dans le tableau (longueur du mot pour le tableau des offsets, mot long pour les pointeurs de fonctions).

Structure

Pointeur sur une table d'initialisation, qui a été détaillée pour la fonction InitStruct. La structure de librairie n'est élaborée avec cette table qu'à la fin de la fonction MakeLibrary. Les entrées lib_NegSize et lib_PosSize ne doivent pas être initialisées à l'aide de cette table, sinon les valeurs calculées par la fonction déjà insérées seront remplacées. Si le paramètre Structure n'est pas indiqué dans la fonction, aucune table ne sera utilisée pour l'initialisation, et la création de la structure devra se faire 'à la main'.

Init

Pointeur sur le programme qui sera appelé à la fin de la fonction MakeLibrary(), à condition que le pointeur soit initialisé. On peut donc initialiser une structure de librairie dans une routine personnelle, si cela n'a pas été déjà fait à l'aide d'un tableau d'initialisation. Le pointeur sur la structure de librairie à créer est transmis en D0, celui de liste segment l'est en A0. Si l'on veut modifier la routine Init, cette modification sera transmise en tant que paramètre à la fin de la fonction.

Taille

Longueur du secteur de données nécessité par la structure de librairie (par exemple MyLib_SIZE dans InitStruct, pour l'exemple de programme).

SegListe

Pointeur sur une liste de segments, transmise en A0 lorsqu'on utilise la routine Init (si la librairie a été chargée à partir du répertoire LIBS).

Paramètres en retour**Library**

Pointeur en retour sur la structure de librairie, qu'il ne faut pas confondre avec le début de la mémoire allouée pour la librairie.

Avec cette fonction, il est possible de créer une librairie personnelle. Celle-ci n'est cependant pas encore insérée dans la LibListe de la structure ExecBase. La somme de contrôle de la librairie n'est pas encore calculée non plus. Pour ces opérations, la librairie Exec propose la fonction suivante :

AddLibrary

Fonction : AddLibrary(Library)
a1

Offset : -396 -\$18C

Paramètre :**Library**

Pointeur sur la structure de librairie créée précédemment par la fonction MakeLibrary().

La librairie personnelle

Nous venons de passer en revue les fonctions nécessaires. Nous allons maintenant passer à la pratique, et écrire une librairie personnelle.

Notre librairie devra être copiée dans le répertoire LIBS de la disquette système après assemblage, et pourra être appelée avec "OpenLibrary ("test.library")". Elle ne représente en fait qu'un squelette de librairie, dans laquelle vous devez encore intégrer les fonctions qui devront y figurer. Pour servir d'exemple, nous prenons deux fonctions qui ne prétendent nullement être utiles, et qui sont là uniquement pour expliciter la méthode.

Nous avons écrit ce programme de création d'une librairie personnelle avec l'assembleur Metacomco "ASSEM".

:Remarques sur les commentaires du programme :
:Les caractères '>-' signalent que le registre qui suit
:dont le contenu est décrit, est un paramètre d'entrée.
:Les caractères '->' désignent un paramètre de sortie.

```
include "exec/types.i"
```

```

include "exec/initializers.i"
include "exec/libraries.i"
include "exec/lists.i"
include "exec/nodes.i"
include "exec/resident.i"
include "libraries/dos.i"
include "exec/alerts.i"
CALLSYS MACRO
    jsr _LVO\1(a6)
    ENDM
XLIB MACRO
    XREF _LVO\1
    ENDM
;Structure d'une librairie personnelle

STRUCTURE MyLib.LIB_SIZE
    ULONG m1_SysLib
    ULONG m1_DosLib
    ULONG m1_SegList
    UBYTE m1_Flags
    UBYTE m1_pad
    LABEL MyLib_Sizeof

XLIB OpenLibrary
XLIB CloseLibrary
XLIB FreeMem
XLIB Remove
XLIB Alert

Version equ 1           ;Version de la Library
Revision equ 0          ;Révision de la Library
Pri equ 0               ;Pri. de la Library (sans importance)

;Pour éviter tout effondrement du système, si jamais l'on
;charge par mégarde la librairie en tant que programme, on dispose
;des deux commandes suivantes :
Start :
    moveq #0,d0
    rts
;La structure Resident est utilisée par la fonction InitResident,
;pour élaborer notre librairie. La fonction InitResident de Exec
;est appelée par la routine pour le chargement d'une librairie
;à partir de la librairie RAM.
Resident :
    dc.w RTC_MATCHWORD;Code pour Resident
    dc.l Resident      ;Pointeur sur le début de la structure
    dc.l CodeEnde     ;Pointeur sur la fin de la structure
    dc.b RTF_AUTOINIT :Flag pour l'appel automatique
    dc.b Version       ;Version de la Library
    dc.b NT_LIBRARY    ;Type de la structure Resident=Library
    dc.b Pri           ;Priorité de la str.Resident
    dc.l LibName       ;Pointeur sur le nom de la Library
    dc.l idString      ;Chaîne d'id. pour la Library
    dc.l Init           ;Pointeur sur le tableau d'initialisation
LibName:  dc.b 'test.library',0
idString: dc.b 'test.library 1.0 (10 Mai 88)',13,10,0
DosName: dc.b 'dos.library',0

```

```

        ds.w 0
;Fin de la structure Resident
FinCode:
;Tableau d'initialisation, utilisée par la fonction InitResident
;pour transmettre les paramètres correspondants à la fonction

;MakeLibrary.
Init:      dc.l MyLib_Sizeof ;Taille de la structure de librairie
           dc.l FuncTable    ;Pointeur sur le tableau

           dc.l DataTable    ;des fonctions Lib
           dc.l InitRoutine   ;Pointeur sur tableau pour InitFunktion
           dc.l Null          ;Pointeur sur routine propre
           dc.l Null          ;de création (appel de MakeLibrary())
FuncTable:
;----- Routines système

           dc.l Open
           dc.l Close
           dc.l Expunge
           dc.l Null
;----- Routines personnelles

           dc.l BildBlink
           dc.l LEDBlink
;----- Marque de fin

           dc.l -1
;tableau transmis à la fonction InitStruct
;INITBYTE, INITWORD et INITLONG sont des macros contenus dans
;le fichier Include "exec/initializers.i".
;(cf. la description dans la section présente)
DataTable: INITBYTE LH_TYPE,NT_LIBRARY
           INITLONG LN_NAME,LibName
           INITBYTE LIB_FLAGS,LIBF_SUMUSED!LIBF_CHANGED
           INITWORD LIB_VERSION,Version
           INITWORD LIB_REVISION,Revision
           INITLONG LIB_IDSTRING,idString
           dc.l 0
;La routine suivante est appelée par la fonction MakeLibrary.
;Elle sert à l'initialisation d'autres entrées de la librairie,
;que l'on ne peut pas faire par l'intermédiaire de DataTable.
;:- D0 - Pointeur sur la structure de librairie
;:- A0 - Pointeur sur la liste des segments de la Library chargée
;:- A6 - Pointeur sur Execbase
;:- D0 - Pointeur sur la structure de librairie
InitRoutine:
           move.l a5,-(a7)      ;sauver A5
           move.l d0,a5          ;Pointeur sur MyLib
           move.l a6,m1_SysLib(a5);Pointeur sur ExecLib
           move.l a0,m1_SegList(a5);introduit liste des segments
           lea DosName(pc),a1      ;Pointeur sur le nom DOS
           move.l #Version,d0      ;numéro de version= 0
           CALLSYS OpenLibrary      ;Open Library
           move.l d0,m1_DosLib(a5);Entrée de l'adresse
           bne.s 1$                ;Ok, Lib trouvé
;ALERT est une macro qui se trouve dans "exec/alerts.i"
;Cette macro affiche un gourou (Alert), lorsque la

```

```

;DOS-Library est introuvable.

        ALERT AG_OpenLib!AO_DOSLib ;Sort Alert
1$:
;Vous placerez ici votre propre routine d'initialisation
;Vous placerez ici votre propre routine d'initialisation
;Vous placerez ici votre propre routine d'initialisation

        move.l a5,d0          ;Pointeur sur Mylib

        move.l (a7)+,a5          ;Recherche les registres
        rts                      ;Retour
;La routine suivante est appelée par la fonction OpenLibrary()
;une fois que la librairie a été créée complètement.
;- A6 - Pointeur sur la structure personnelle de librairie

;>- D0 - Pointeur sur la structure personnelle de librairie
Open:
        addq.w #1,LIB_OPENCNT(a6)      ;Incrémente le compteur pour
                                    ;le nombre des accès à la Library
        bclr #LIBB_DELEXP,m1_Flags(a6) ;Flag pour supprimer
                                    ;la Library
        move.l a6,d0          ;définir les paramètres de retour
        rts                      ;Retour
;Lorsqu'une tâche n'a plus besoin d'une librairie, elle la dérime,
;pour permettre au système d'exploitation d'enlever la librairie
;du système. La librairie n'est supprimée que lorsque la fonction
;AllocMem constate qu'il n'y a pas assez de mémoire disponible
;et lorsqu'il n'y a pas de tâche accédant à la librairie.
;Vous pouvez demander la suppression de la librairie, en posant
;le flag LIBB_DELEXP avant l'appel de la fonction CloseLibrary.
;(Include-File "exec/libraries.i").
Close:
        clr.l d0          ;Supprime le pointeur
                            ;sur liste de segments
(important)

        subq.w #1,LIB_OPENCNT(a6)      ;compteur pour ouverture de la
                                    ;Library -1
        bne.s 1$              ;saut si Library
                            ;encore utilisée
        btst #LIBB_DELEXP,m1_Flags(a5) ;est-ce que le flag
                                    ; LIBB_DELEXP est posé ?
        beq.s 1$              ;Fin, s'il ne l'est pas
        bsr Expunge            ;supprime Library

1$:           rts          ;retour
;Routine pour supprimer la librairie de la mémoire.
;- A6 - Pointeur sur Library
;- D0 - Pointeur sur liste des segments de la Library chargée
Expunge:

        movem.l d1/a5-a6,-(a7)    ;sauver les registres
        move.l a6,a5          ;Pointeur sur Library vers A5
        move.l m1_SysLib(a5),a6  ;ExecBase vers A6
        tst.w LIB_OPENCNT(A5)   ;Library encore utilisée ?
        beq 1$                  ;Saut si non utilisée

```

```

        bset #LIBB_DELEXP,m1_Flags(a5) :on souhaite supprimer
                                         ;la Library
        clr.l d0                      ;supprime le pointeur sur
                                         ;liste des segments
        bra.s Expunge_end             ;saut inconditionnel
1$:      move.l m1_SegList(a5),d2   ;Pointeur sur liste des segments
                                         ;vers D2
        move.l a5,a1                  ;Pointeur sur Library vers A1
        CALLSYS Remove                ;supprimer Library
                                         ;de la liste Exec-Lib

        move.l m1_DosLib(a5),a1       ;Pointeur sur DOS-Library
        CALLSYS CloseLibrary          ;Fermer Library
        clr.l d0                      ;effacer D0
        move.l a5,a1                  ;Pointeur sur Library
        move.w LIB_NEGSIZE(a5),d0     ;cherche pointeur sur
                                         ;début de la mémoire
        sub.l d0,a1                  ;occupée par la librairie
                                         ;obtenir longueur de la
                                         ;mémoire occupée
        add.w LIB_POSSIZE(a5),d0     ;libère la mémoire
                                         ;Pointeur sur liste des segments
                                         ;vers D0
        CALLSYS FreeMem
        move.l d2,d0

Expunge_end:
        movem.l (a7)+,d2/a5-a6       ;restaurer les registres
        rts                          ;retour
;la fonction suivante peut être atteinte avec offset -24.
;Elle n'est pas utilisée dans la version Kickstart actuelle.
Zéro:
        moveq #0,d0                  ;efface D0
        rts                          ;retour
;Ici commencent les fonctions personnelles de librairie.
;Les fonctions reproduites ici sont destinées uniquement à servir
;d'exemple pour la créations de librairies personnelles et ne jouent
;aucun rôle particulier
;Scintillement de l'écran
BildBlink:
        move.l #$20000,d0            ;compteur pour boucle vers D0
1$:      move.w d0,$dff180         ;écrire valeur dans
                                         ;registre des couleurs
        subq.l #1,d0                 ;décrémenter
        bne 1$                      ;saut si compteur <> 0
        rts                          ;retour
;Scintillement de la LED
LEDBlink:
        bchg #1,$bfe001              ;Scintillement de la LED
        rts                          ;retour

        END

```

2.9. Structure interne d'un device Exec

Dans ce chapitre, nous n'allons pas entrer dans le détail de l'utilisation des devices; nous parlerons uniquement de leur structure et de leur programmation. Les pages qui suivent demandent donc certaines connaissances en ce qui concerne l'utilisation proprement dite. Vous trouverez ces connaissances dans le chapitre où il est question de chacun des devices disponibles dans l'Amiga. Un device a deux fonctions distinctes. Il sert d'une part à échanger des données avec un élément quelconque du hardware; mais d'autre part on doit pouvoir le gérer. Considérons ces deux tâches séparément, en nous tournant d'abord vers la tâche de gestion.

2.9.1. L'intégration du device dans le système

Les informations nécessaires pour la gestion d'un device se trouvent dans la structure Device. La gestion d'un device est presque identique à celle d'une librairie. Il faut par exemple pouvoir décider si le device est nécessaire, et si c'est le cas, connaître le nombre de tâches qui en ont besoin. Puisque chaque device est initialisé de façon différente, on dispose ici encore de fonctions qui servent à initialiser le device, et aussi au besoin à l'enlever du système.

Puisque les tâches de gestion pour une librairie et pour un device sont si proches, les programmeurs se sont décidé à utiliser le même en-tête de structure pour les deux. Un device utilise donc comme base la structure définie dans le fichier Include "exec/device.h" ou en assembleur "exec/device.i".

```
struct Device {
    struct Libray dd_Library;
};
```

Assembleur:

```
STRUCTURE DD_LIB_LIB      00  $00
          DD_SIZE        34  $22
```

Les fonctions fournies pour la gestion sont obtenues comme pour une librairie par l'intermédiaire d'offsets négatifs, qui ne peuvent évidemment pas être définis librement, mais sont prescrits par le système. Si vous désirez ajouter des fonctions personnelles, vous ne pouvez le faire qu'à la suite de celles qui sont nécessaires au système. Les fonctions qui permettent de disposer d'un device sont les suivantes :

Offset	Fonction
-06	Open
-12	Close
-18	Expunge
-24	Non utilisé actuellement
-30	BeginIO
-36	AbortIO

Fonction : Open

Cette fonction est appelée par le système d'exploitation à partir de la fonction OpenDevice de Exec. Elle permet de créer la structure Unit et de transmettre au device la tâche appelante. Si la structure a déjà été créée, elle est simplement transmise. Le compteur pour le nombre d'ouvertures de l'Unit est ici incrémenté, de même que celui du device.

Fonction : Close

La fonction est appelée à partir de la fonction CloseDevice de Exec. Le compteur pour le nombre d'ouvertures du device est ici décrémenté. S'il se révèle que le device n'est plus nécessaire, on peut sauter à la fonction Expunge.

Fonction : Expunge

Cette fonction est appelée à partir de la fonction Close du device, ou encore à partir de la librairie RAM pour supprimer le device dans le système.

Fonction : BeginIO

La fonction BeginIO est appelée à partir de la fonction DoIO de Exec, ou de la fonction SendIO, et elle sert à gérer les commandes envoyées par le device. C'est ici que se décide si la commande rencontrée sera traitée ou non. Nous y reviendrons.

Fonction : AbortIO

La fonction est appelée par la fonction AbortIO de Exec. On l'utilise pour interrompre des processus de devices. Comme chaque device propose des fonctions différentes, on ne peut pas créer de fonction d'interruption ou d'annulation générale. La fonction en question se prépare par l'intermédiaire du présent vecteur.

Après avoir ainsi explicité schématiquement l'appareil de gestion d'un device, nous allons nous tourner vers les tâches du device.

2.9.2. Division interne du device

Un device est en mesure d'adresser des périphériques d'un même type. On peut prendre pour exemple le device Trackdisk, qui peut gérer jusqu'à 4 lecteurs de disquettes. Ces lecteurs sont tous adressés par l'intermédiaire des fonctions du device Trackdisk. La gestion des différents lecteurs ne se fait pas par l'intermédiaire d'une structure device, mais d'une structure spéciale. Cette structure est appelée "structure Unit". C'est ici qu'est conservé par exemple le numéro de la piste sur lequel se trouve la tête du lecteur.

On peut donner à la structure de device un rôle prépondérant, et confier la gestion du hardware à la structure Unit, que l'on atteint à son tour par l'intermédiaire de la structure de device.

Puisque chaque élément de hardware est conçu différemment, on aura également besoin d'entrées différentes dans la structure Unit pour s'occuper de leur gestion. L'en-tête de la structure Unit est cependant normalisée, et c'est pourquoi elle est également définie dans les fichiers Include. Puisque la structure Unit est utilisée uniquement par les routines personnelles, on n'est pas obligé de se limiter à l'en-tête défini, mais il faut tout de même réfléchir pour voir si un écart par rapport à la norme présente vraiment des avantages.

Les programmeurs en C trouveront la définition de la structure Unit dans le fichier Include "exec/devices.h" :

```
struct Unit {
    struct MsgPort unit_MsgPort;
    UBYTE          unit_flags;
    UBYTE          unit_pad;
    WORD           unit_OpenCnt;
};

#define UNITF_ACTIVE (1L<<0)
#define UNITF_INTASK (1L<<1)
```

Les programmeurs en assembleur trouveront cette structure dans le fichier Include "exec/devices.i".

	Offsets
STRUCTURE UNIT,MP_SIZE	00 \$00
UBYTE UNIT_FLAGS	34 \$22
UBYTE UNIT_pad	35 \$23
WORD UNIT_OPENCNT	36 \$24
LABEL UNIT_SIZE	38 \$26
 BITDEF UNIT,ACTIVE,0	
BITDEF UNIT,INTASK,1	
 unit_MsgPort UNIT	

Il s'agit du port de message où une commande (une structure IORequest) doit être envoyée, si elle doit être traitée par la tâche de device (device-task). Nous en reparlerons plus tard.

unit_flags | UNIT_FLAGS
Flags internes des Devices.

unit_pad | UNIT_pad

Octet de remplissage, pour que la structure soit remplacée à une adresse de mot.

unit_OpenCnt | UNIT_OPENCNT

Pointeur pour le nombre d'ouvertures de l'Unit.

Normalement, il vaut mieux laisser fonctionner en tâche de fond les tâches qui sont transmises à un device. On parvient à ce comportement asynchrone d'un device en transférant l'ordre à une tâche personnelle, responsable d'une unité (Unit) déterminée. Lorsque la tâche d'unité (Unit-Task) a réalisé ce qu'on lui demandait, elle le fait savoir au donneur d'ordre qui, entre-temps, s'est occupé d'autre chose. Si l'on ne veut pas de ce fonctionnement asynchrone, on peut attendre explicitement le message OK de la tâche d'unité.

Lorsqu'une commande est envoyée à un device et que son traitement ne demande pas de temps, il n'est pas utile de la faire traiter par la tâche d'unité de manière asynchrone, car la gestion de la tâche demandera plus de temps que la commande elle-même.

Pour distinguer correctement entre les commandes, il faut sauter - comme nous l'avons dit plus haut - au vecteur BeginIO (offset -30) interne au device, avec les fonctions DoIO et SendIO de Exec. Il faut distinguer ici entre plusieurs cas. La commande peut être traitée directement (QUICK), ou conduite à la tâche d'unité correspondante.

2.9.3. Les commandes Device

Le programmeur d'un device ne peut pas donner à chacune de ses fonctions personnelles un code de commande arbitraire. Certains codes sont déjà affectés, et ils doivent être soutenus par le device personnel. Si certaines fonctions prédéfinies ne sont pas utiles dans le cas concret qui se présente, elles doivent être "intégrées" dans le sens où elles pourront quitter le device sans aucune opération. Les codes de commande déjà affectés se trouvent dans le fichier Include "exec/io.h" ou "exec/io.i".

```
#define CMD_INVALID 0L
#define CMD_RESET 1L
#define CMD_READ 2L
#define CMD_WRITE 3L
#define CMD_UPDATE 4L
#define CMD_CLEAR 5L
#define CMD_STOP 6L
#define CMD_START 7L
#define CMD_FLUSH 8L
```

CMD_INVALID

Commande illégale. Elle n'exécute aucune fonction.

CMD_RESET

Rétablissement l'Unit dans l'état précédent, comme si elle venait d'être initialisée.

CMD_READ

Lit des données à partir de l'Unit indiqué.

CMD_WRITE

Ecrit des données dans l'Unit indiqué.

CMD_UPDATE

Sauvegarde le contenu du tampon (par exemple pour le device Trackdisk sur disquette).

CMD_CLEAR

Efface tous les tampons.

CMD_STOP

Ne traite plus les commandes non encore prises en compte, et se place en position d'attente.

CMD_START

A l'effet opposé à celui de CMD_STOP. Reprend le traitement des commandes.

CMD_FLUSH

Efface dans la liste des commandes à traiter toutes les commandes qui se trouvaient jusque là en position d'attente.

Si vous voulez écrire un device qui collabore avec le système des fichiers et les commandes CLI, vous devez envisager encore d'autres commandes.

Il s'agit de toutes les commandes dont on dispose à partir du device Trackdisk. On peut prendre pour exemple le device Harddisk, et la commande de device "Format", appelée par la commande FORMAT du CLI, avec le code de commande correspondant. Comme exemple de "commande dummy", on peut prendre avec le device de disque dur la commande "Remove", utilisée par le système des fichiers pour reconnaître un changement de disque. Pour un disque dur, un "changement de disque" n'a pas beaucoup de sens, mais le code de la commande "Remove" ne peut cependant pas être utilisé pour une commande personnelle. Ce n'est qu'à partir du code de commande 22 que l'on peut être sûr que le système de fichiers ou la commande CLI n'exécuteront pas par mégarde des commandes définies personnellement.

2.9.4. L'intialisation des devices

Si vous voulez insérer un device personnel dans le système, on le fait en général avec la commande MOUNT de CLI. On charge ensuite le device à partir du répertoire Devs, par l'intermédiaire de la fonction LoadSeg() de DOS (par la librairie RAM). Une fois le chargement effectué, on saute à une routine d'initialisation, qui affecte les ressources pour le device. L'intervention de la routine d'initialisation s'effectue par l'analyse d'une structure Resident, qui se trouve à une adresse fixe à l'intérieur du device. La traitement de la structure Resident s'effectue à son tour par l'intermédiaire de la routine InitResident de Exec, que l'on appelle à partir de la librairie RAM.

Si l'on ouvre ultérieurement le device par l'intermédiaire de la fonction OpenDevice de Exec, avec un numéro d'Unit déterminé (numéro de périphérique), on passe à une seconde routine d'initialisation, à partir de laquelle la structure Unit est créée, de même que la tâche d'Unit (si l'on en a besoin).

2.9.5. Un exemple de device

Pour empêcher le chargement intempestif du device sous forme de programme exécutable, et donc l'effondrement du système qui en résulterait, les premiers octets présentent les commandes suivantes :

```
MOVEQ #0.00  
RTS
```

Si le device n'est pas chargé comme tel, mais comme programme, il est aussitôt supprimé sans message d'erreur.

Après ces deux commandes, commence une structure Resident, qui permet de créer le device avec la fonction InitResident.

La tâche créée ne connaît pas les adresses de sa structure Unit et de sa structure de device; il faut les lui transmettre ultérieurement. On le fait par l'intermédiaire d'un message qu'on lui envoie. Pour ne pas trop compliquer, nous avons créé dans notre exemple non pas une structure de tâche proprement dite, mais une structure de processus (cf. Amiga-DOS) à l'aide de la fonction CreateProc() de DOS. Cette structure présente également l'avantage de disposer déjà d'une structure de port de message, à laquelle on peut transmettre le message dont il vient d'être question.

Les commandes qui agissent sur certains registres ou certaines structures en écrivant dedans doivent être envoyées à la tâche, pour ne pas se gêner réciproquement. Elles y seront placées à la fin d'une liste. La tâche les traite alors systématiquement dans l'ordre de leur arrivée.

Pour les commandes à traiter directement, on a introduit le bit Quick, utilisé uniquement pour la fonction DoIO. Si la commande transmise doit faire partie de celles-là, le bit Quick posé par la fonction DoIO est à nouveau effacé, pour signaler que - après retour de la

fonction BeginIO - la commande a déjà été exécutée, et qu'on n'est donc pas obligé d'attendre qu'elle ait pris fin.

L'exemple qui suit est celui d'un device personnel qui n'a pas d'utilité pratique. Il sert simplement de squelette pour que vous puissiez intégrer vos fonctions personnelles de device, et remplacer celles qui existent déjà.

Le device a été écrit avec l'assembleur "ASSEM" de Metacomco, et utilise donc des fichiers Include et des macros.

Pour tester un device, il faut le copier dans le répertoire DEVS de la disquette système, et l'appeler avec son nom. Le nom utilisé dans le cas présent est :

```

;"mydevice.device"
;Dans les pages qui suivent, vous trouverez un exemple de device :
;Remarque sur la documentation:
;Les caractères '>' à l'intérieur de la documentation signifient
;que le registre qui suit est posé conformément à l'explication
;lors d'un saut dans une sous-routine (paramètre d'entrée).
;
;Les caractères '>', indiquent un paramètre de sortie.

include "exec/types.i"
include "exec/initializers.i"
include "exec/libraries.i"
include "exec/lists.i"
include "exec/nodes.i"
include "exec/resident.i"
include "exec/alerts.i"
include "exec/ables.i"
include "exec/devices.i"
include "exec/io.i"
include "exec/memory.i"
include "exec/errors.i"
include "libraries/dos.i"
include "libraries/dosextens.i"

CALLSYS    MACRO
          jsr _LVO\1(a6)
          ENDM
LINKSYS   MACRO
          move.l a6,-(a7)
          move.l \2,a6
          jsr _LVO\1(a6)
          move.l (a7)+,a6
          ENDM
XLIB       MACRO
          XREF _LVO\1
          ENDM

;Nombre de Units traités par ce device
MD_NUMUNITS EQU 4

;La structure Device personnelle pour gérer le device

STRUCTURE MyDev,LIB_SIZE
  ULONG      md_SysLib
  ULONG      md_DosLib
  ULONG      md_SegList
  UBYTE     md_Flags

```

```

UBYTE      md_pad
STRUCT     md_Units,MD_NUMUNITS*4
LABEL      MyDev_SIZE

;Le message se trouve dans la structure MyDevUnit
;et il est envoyé à la tâche d'Unit, lors de l'initialisation de celle-ci

STRUCTURE MyDevMsg,MN_SIZE
    APTR      mdm_Device
    APTR      mdm_Unit
    LABEL     MyDevMsg_SIZE

;La structure MyDevUnit sert à gérer une unité (Unit)
;du device

STRUCTURE MyDevUnit,UNIT_SIZE
    UBYTE     mdu_UnitNum
    UBYTE     mdu_pad
    STRUCT   mdu_Msg,MyDevMsg_SIZE
    APTR     mdu_Process
    LABEL     MyDevUnit_SIZE

;Bit posé dans UNIT_FLAGS pour indiquer que la
;tâche d'unité est bloquée (cf. MyStop et Start).

BITDEF     MDU_STOPPED,2

MYDEVNAME  MACRO
    DC.B 'mydevice.device',0
    ENDM

;Les fonctions de librairie qui suivent doivent être
;importées pour l'éditeur de liens.

XREF      _AbsExecBase

XLIB      OpenLibrary
XLIB      CloseLibrary
XLIB      Alert
XLIB      FreeMem
XLIB      Remove
XLIB      FindTask
XLIB      AllocMem
XLIB      CreateProc
XLIB      PutMsg
XLIB      RemTask
XLIB      ReplyMsg
XLIB      Signal
XLIB      GetMsg
XLIB      Wait
XLIB      WaitPort
XLIB      AllocSignal
XLIB      SetTaskPri
XLIB      Permit

INT_ABLES           ;MACRO pris dans le fichier exec/ables.i

;Pour qu'un device ne génère pas un effondrement, s'il est
;chargé par erreur comme module objet, on dispose de ces deux commandes:

START:
    moveq #0,d0          ;Pas de message d'erreur
    rts                  ;Saut en retour

;La priorité n'importe que s'il s'agit d'un device
;résistant au reset, sinon la valeur n'a pas d'importance.

```

```

MYPRI      EQU 0
VERSION    EQU 1           ;Version du device
REVISION   EQU 0           ;Numéro de la révision

;Voici maintenant une structure Resident, permettant
;d'élaborer le device.

InitTable:
    DC.W      RTC_MATCHWORD :Mot code pour Resident
    DC.L      InitTable     :Pointeur sur début de la structure
    DC.L      EndCode       :Pointeur sur fin de la structure
    DC.B      RTF_AUTOINIT :Flag pour initialisation
                  ;automatique
    DC.B      VERSION       ;Version du device
    DC.B      NT_DEVICE    ;type de la structure
                  ;à créer (DEVICE)
    DC.B      MYPRI        ;priorité
    DC.L      MyName       :Pointeur sur nom du device
    DC.L      idString     :Pointeur sur chaîne

                  ;des explications
    DC.L      Init         :Pointeur sur la table
                  ;d'initialisation

MyName:      MYDEVNAME          ;Nom du device

;La chaîne des explications n'est indispensable, mais elle est utile
;pour déterminer la version du device.
;Si l'on veut s'en tenir au standard, il faut écrire comme suit:
;'Nom Version.Révision (Jour Mois Année)'CRLF0

idString:    dc.b 'mydevice 1.0 (20 JUN 1988)',13,10,0
dosName:    DOSNAME          ;Nom de la DOS-Library

        ds.w 0                 ;Décalage pour parvenir
                  ;à une adresse paire

;Le label EndCode indique la fin de la structure Resident.

EndCode:

;Ici commence la table d'initialisation, utilisée par
;la fonction "InitResident()".

Init:
    dc.l MyDev_SIZE  ;Taille de la structure Device
    dc.l funcTable   ;Pointeur sur la table des fonctions
    dc.l dataTable   ;Pointeur sur la table pour créer
                  ;la structure Device (pour la
                  ;fonction "InitStruct()")
    dc.l initRoutine ;Pointeur sur la routine servant à
                  ;mettre en place la structure Device

;Table dans laquelle sont reportées les fonctions accessibles
;par le device avec des offsets négatifs, pour la gestion du device

funcTable:
    dc.l Open        ;fonction pour ouvrir le device
    dc.l Close       ;fonction pour fermer le device
    dc.l Expunge    ;fonction pour supprimer le device
                  ;de la mémoire
    dc.l Null        ;fonction inutilisée (pour extensions)

    dc.l BeginIO    ;fonction à laquelle on saute
                  ;lorsqu'une commande est envoyée
                  ;au device

```

```

        dc.l AbortIO      ;fonction pour interrompre une commande

        dc.l -1           ;caractères pour fin de la table

;table pour la création de la structure Device. La table sert
;à la fonction Exec "InitStruct()". Les mots INITBYTE, INITWORD
;et INITLONG sont des appels de macros, qui simplifient beaucoup
;la création de cette table un peu compliquée.
;Les macros se trouvent dans le fichier Include "exec/initializers.i".

;La table sert à:

;Faire figurer le type.
;Faire figurer le pointeur sur le nom du device.
;Faire figurer les flags de librairie.
;Faire figurer la version du device.
;Faire figurer la révision.
;Faire figurer le pointeur sur la chaîne d'explications

dataTable:
    INITBYTE   LH_TYPE,NT_DEVICE
    INITLONG   LN_NAME,MyName
    INITBYTE   LIB_FLAGS,LIBF_SUMUSED!LIBF_CHANGED
    INITWORD   LIB_VERSION,VERSION
    INITWORD   LIB_REVISION,REVISION
    INITLONG   LIB_IDSTRING,idString
    dc.l 0

;Ici commence la routine appelée par la fonction Exec "MakeLibrary"
;pour initialiser le reste de la structure Device.

;=> D0 = Pointeur sur la structure Device
;=> A0 = Pointeur sur la liste des segments du device chargé
;=> A6 = Pointeur sur ExecBase

;=> D0 = Pointeur sur la structure Device

initRoutine:
    move.l    a5,-(a7)          ;sauver A5
    move.l    d0,a5             ;Pointeur sur device vers A5
    move.l    a6,md_SysLib(a5)  ;reporter le pointeur sur ExecBase
                                ;dans la structure Device
    move.l    a0,md_SegList(a5) ;reporter le pointeur sur la liste des
                                ;segments
    lea       dosName(pc),a1    ;Pointeur sur le nom de la
                                ;DOS-Library
    moveq    #0,d0              ;Version pour OpenLib()
    CALLSYS  OpenLibrary        ;ouvrir DOS-Lib

    move.l    d0,md_DosLib(a5)  ;reporter le pointeur sur DOS-Lib
    bne.s    Dos_OK             ;poursuivre si DOS-Lib a été trouvé

;Au cas où la librairie DOS ne peut pas être ouverte, affichage
;d'un "gourou". L'affichage se fait à l'aide d'une macro
;qui se trouve dans le fichier Include "exec/alerts.i".

        ALERT     AG_OpenLib!A0_DOSLib

Dos_OK:
;Insérer ici une initialisation, si nécessaire

    move.l    a5,d0              ;Pointeur sur device vers D0
    move.l    (a7)+,a5            ;restaurer A5
    rts                  ;retour

```

;Une fois que la structure Device a été initialisée
;après chargement à partir du disque, elle doit encore être
;ouverte. On crée ainsi la tâche destinée à gérer l'unité, ainsi que
;la structure Unit elle-même. Lorsque plusieurs Units sont autorisés
;il faut vérifier si le numéro d'unité souhaité est autorisé.

-> D0 = Numéro de l'Unit
;)-> D1 = Flags
;)-> A1 = Pointeur sur IORequest
;)-> A6 = Pointeur sur le device

Open:

```

    movem.l    d2/a2-a4,-(a7)      ;Sauver le registre
    move.l     a1,a2                ;Sauver le IORequest
    moveq     #MD_NUMUNITS,d2      ;Nombre des Units vers D2
    cmp.l     d2,d0                ;Numéro de l'Unit autorisé?
    bcc.s     Open_error          ;Non, Numéro trop élevé
    move.l     d0,d2                ;Numéro de l'Unit vers D2
    lsl.1     #2,d0                ;Offset pour pointeur sur
                                    ;Unit
    lea        md_Units(a6,d0.L),a4 ;Pointeur sur Unit vers A4
    move.l     (a4),d0                ;Unité déjà créée?
    bne.s     Open_ok              ;Oui, déjà créée
    bsr       InitUnit             ;Créer unité
    move.l     (a4),d0                ;Unité créée
    beq.s     Open_error          ;Non, erreur
Open_ok:   move.l     d0,a3                ;Pointeur sur Unit vers A3
    move.l     d0,IO_UNIT(a2)       ;Report dans IORequest
    addq.w    #1,LIB_OPENCNT(a6)   ;Compteur pour nombre des
                                    ;accès au device +1
    addq.w    #1,UNIT_OPENCNT(a3);Compteur pour nombre des
                                    ;accès à l'unité+1
    bclr     #LIBB_DELEXP,md_Flags(a6) ;Supprimer flag pour
                                    ;library
Open_end:  movem.l    (a7)+,d2/a2-a4      ;Rétablir le registre
    rts                  ;retour

```

;La suite n'intervient que si une erreur a survécu lors de la
;création de l'unité

```

Open_error: move.b     #IOERR_OPENFAIL,IO_ERROR(a2) ;reporter erreur dans IORequest
                                    ;
    bra.s     Open_end            ;retour

```

;Lorsqu'une tâche ne veut plus accéder à ce device, il faut le fermer,
;pour avoir ainsi la possibilité de restituer la mémoire
;au système

-> A1 = Pointeur sur la structure IORequest
;)-> A6 = Pointeur sur la structure Device
-> D0 = Pointeur sur la liste des segments, si le segment (le device
;entier) doit être supprimé, sinon 0

Close:

```

    movem.l    a2-a3,-(a7)      ;Sauver le registre
    move.l     a1,a2                ;Sauver le IORequest
    move.l     IO_UNIT(a2),a3      ;Chercher le pointeur sur Unit
    moveq.l    #-1,d0                ;D0 = -1
    move.l     d0,IO_UNIT(a2)       ;Supprimer le pointeur sur Unit
    move.l     d0,IO_DEVICE(a2)      ;Supprimer le pointeur sur Device
    subq.w    #1,UNIT_OPENCNT(a3);Compteur pour nombre des
                                    ;accès à l'unité-1
    bne.s     CloseDevice         ;Embranchement, sinon nul

```

;Si l'on ne désire plus accéder à l'unité, on peut libérer
;l'espace qu'elle occupait en mémoire.

```

        bsr      ExpungeUnit ;libère la mémoire de l'unité
CloseDevice:clr.1    d0      ;Supprime les paramètres de retour
                      ;(important)

        subq.w   #1,LIB_OPENCNT(a6) ;Compteur pour nombre des
                                ;accès -1
        bne.s   Close_end       ;embranchement si non nul

;S'il n'y a plus de tâche qui accède au device, on peut le supprimer
;du système. Mais comme il arrive souvent que le device soit utilisé
;peu de temps après, on a placé ici en plus une interrogation.
;qui demande s'il faut absolument supprimer le device.
;Si la réponse est négative, le device reste dans le système
;jusqu'à ce que la place en mémoire vienne à manquer. Dans ce dernier cas,
;il est supprimé automatiquement de la librairie RAM.

;Vous pouvez exiger la suppression du device - lorsqu'aucune
;tâche n'en a plus besoin -, en posant le bit 'LIBB_DELEXP'
;dans la ligne de flag du device, avant l'appel de "CloseDevice()".

        btst     #LIBB_DELEXP,md_Flags(a6) ;Est-ce que le bit est posé?
        beq.s   Close_end       ;Non, fin
        bsr      Expunge        ;Supprimer le device
Close_end: movem.1   (a7)+,a2-a3 ;rétablir le registre
        rts      ;retour

;Voici la routine qui supprime le device et rend la mémoire
;allouée.

;=> A6 = Pointeur sur device

;=> D0 = Pointeur pris dans la liste des segments du device chargé

Expunge:
        movem.1   d2/a5-a6,-(a7)  ;Sauver le registre
        move.1    a6,a5          ;Pointeur sur device vers A5
        move.1    md_SysLib(a5),a6 ;ExecBase vers A6
        tst.w    LIB_OPENCNT(a5) ;Compteur pour nombre des
                                ;ouvertures
        beq     1$                ;sauter quand le device est supprimé
        bset     #LIBB_DELEXP,md_Flags(a5) ;Poser le bit
                                ;montrer que le device
                                ;doit être supprimé
        clr.1    d0      ;Supprimer le pointeur sur segment
        bra.s   Expunge_end ;retour
1$      move.1    md_SegList(a5).d2 ;Chercher pointeur sur liste des segments
        move.1    a5,a1          ;Pointeur sur device vers A1
CALLSYS Remove           ;Supprimer le device pris dans la liste
        move.1    md_DosLib(a5),a1 ;Pointeur sur DOS_Lib
CALLSYS CloseLibrary     ;Supprimer librairie DOS
        move.1    a5,a1          ;Pointeur sur Device
        clr.1    d0      ;Supprimer D0
        move.w   LIB_NEGSIZE(a5),d0 ;Chercher grandeur négative
        sub.w    d0,a1          ;Définir le début
                                ;de la mémoire pour device
        add.w    LIB_POSSIZE(a5),d0 ;définir la longueur du device
        CALLSYS FreeMem         ;Libérer la mémoire

        move.1    d2,d0          ;Pointeur sur liste des segments
Expunge_end:movem.1   (a7)+,d2/a5-a6 ;rétablir le registre
        rts      ;retour

;La fonction qui suit n'est pas utilisable. Elle le sera peut-être dans les versions
;ultérieures de Kickstart.

Null:    clr.1    d0          ;Effacer D0

```

```

        rts                      ;retour

;La routine qui suit est appelée pour initialiser une unité ouverte
;pour la première fois. On élabore pour cela un nouveau processus,
;qui gère la structure d'unité qui vient elle aussi d'être créée.
;Pour que le nouveau processus "sache" où il trouvera ses structures
;d'unité et de device, on les lui envoie sous forme de messages. La
;structure de message est intégrés dans la structure d'unité.

;=> D2 = Numéro de l'Unit
;=> A6 = Pointeur sur device

MYPROCSTACK EQU $200      ;longueur de la pile pour la Unit-Task
MYPROC PRI    EQU 0        ;priorité de la tâche d'unité

InitUnit:
    movem.l  d2-d4,-(a7)      ;Sauver le registre
    move.l   #MyDevUnit_SIZE,d0 ;longueur de la structure Unit
    move.l   #MEMF_PUBLIC!MEMF_CLEAR,d1 ;Type de mémoire

```

;L'instruction LINKSYS est une macro définie au début du programme

```

LINKSYS      AllocMem,md_SysLib(a6) ;AllocMem()
tst.l        d0                  ;mémoire disponible
beq          InitUnit_end       ;Non, erreur
move.l        d0,a3              ;Pointeur sur nouvelle structure Unit
move.b        d2,mdu_UnitNum(a3) ;reporter numéro de l'unité

move.l        #MYPROCSTACK,d4    ;longueur de la pile pour
                                ;le processus
move.l        #myproc_seglist,d3 ;Pointeur sur la liste des segments
lsl.r        #2,d3              ;BPTR -> APTR
moveq        #MYPROC_PRI,d2     ;priorité du processus
move.l        #MyName,d1         ;Pointeur sur nom
LINKSYS      CreateProc,md_DosLib(a6) ;Créer un processus
tst.l        d0                  ;Succès?
beq          InitUnit_FreeUnit ;Non, libérer mémoire pour Unit
move.l        d0,mdu_Process(a3) ;reporter pointeur sur processus
move.l        d0,a0              ;Pointeur vers A0
lea           -pr_MsgPort(a0),a0 ;Chercher pointeur sur structure Task
                                ;à l'intérieur du processus
move.l        a0,MP_SIGTASK(a3) ;Task comme Signal-Task pour Unit
move.b        #PA_IGNORE,MP_FLAGS(a3) ;Poser les bits de Message-Port

lea           MP_MSGLIST(a3),a1  ;Pointeur sur la liste Message-Port

;NEWMEST est une macro prise dans le fichier Include "exec/lists.i", et créant
;une liste vide à l'endroit indiqué (A1).

NEWMEST      A1                  ;Créer une liste vide
lea           mdu_Msg(a3),a1    ;Pointeur sur structure Message
                                ;à l'intérieur de la structure Unit
move.l        a3,mdm_Unit(a1)   ;reporter pointeur sur Unit
move.l        a6,mdm_Device(a1) ;reporter pointeur sur Device
move.l        d0,a0              ;Message-Port pour processus
LINKSYS      PutMsg,md_SysLib(a6) ;Envoyer message au processus
clr.l        d0                  ;Effacer D0
move.b        mdu_UnitNum(a3),d0 ;Numéro de l'Unit vers d0
lsl.l        #2,d0              ;Offset, pour faire figurer pointeur sur
                                ; Unit
                                ;dans le device
move.l        a3,md_Units(a6,d0,L) ;reporter pointeur sur Unit
InitUnit_end:
    movem.l  (a7)+,d2-d4        ;rétablir le registre
    rts                      ;retour

```

;Les deux commandes suivantes ne sont exécutées que s'il se produit
;une erreur lors de la création du processus.

```
InitUnit_FreeUnit:
    bsr      FreeUnit      ;restituer mémoire pour Unit
    bra.s   InitUnit_end   ;retour
```

;Rendre au système la mémoire pour structure Unit

```
;=> A3 = Pointeur sur structure Unit

FreeUnit: move.l   a3,a1      ;Pointeur sur Unit vers A1
          move.l   #MyDevUnit_SIZE,d0 ;longueur de la structure
          LINKSYS  FreeMem.md_SysLib(a6) ;FreeMem()
          rts       ;retour
```

;Lorsqu'une unité déterminée n'est plus nécessaire à aucune tâche,
;la mémoire qu'elle occupait est restituée, et le processus
;est supprimé de la mémoire. La routine est appelée par Close.

;=> A3 = Pointeur sur Unit
;=> A6 = Pointeur sur Device

```
ExpungeUnit:
    move.l   d2,-(a7)      ;sauvegarder d2
    move.l   mdu_Process(a3),a1 ;Pointeur sur processus
    lea      -pr_MsgPort(a1),a1 ;Chercher pointeur sur début du
                                ;processus.
    LINKSYS RemTask.md_SysLib(a6) ;Supprimer la tâche
    clr.l   d2              ;Effacer D2
    move.b   mdu_UnitNum(a3),d2 ;Numéro de l'Unit vers D2

    bsr.s   FreeUnit      ;Libérer mémoire pour Unit
    lsl.l   #2,d2          ;Offset pour ligne Unit
                                ;dans la structure Device
    clr.l   md_Units(a6,d2.L) ;Effacer l'entrée
    move.l   (a7)+,d2        ;Rétablir D2
    rts       ;retour
```

;Ensuite vient la table avec les commandes du device.
;L'ordre des fonctions dans la table est défini à l'avance.
;Pour chaque ligne de la table, on dispose d'un bit déterminé.
;Pour l'entrée 0, c'est le bit 0; pour l'entrée 1, c'est le bit 1.
;et ainsi de suite. A l'aide de ces bits, on définit la commande
;qui doit être exécutée directement, ou envoyée au processus
(cf. explication avant le programme).

;Les commandes MyReset et myStop n'ont pas été nommées Reset et Stop,
;car l'assembleur les auraient confondues avec des commandes assembleur.

;Les commandes Invalid à Flush dans la table doivent être représentées
;dans cet ordre. Si toutes les commandes ne sont pas utiles pour votre
;device, faites-les sauter à la fonction Invalid (cf.Update et Clear).

cmdtable:

dc.l	Invalid	;\$001
dc.l	MyReset	;\$002
dc.l	Read	;\$004
dc.l	Write	;\$008
dc.l	Update	;\$010
dc.l	Clear	;\$020
dc.l	MyStop	;\$040
dc.l	Start	;\$080
dc.l	Flush	;\$100

;A partir d'ici, vous pouvez faire figurer vos propres fonctions.
;Nous avons ici deux nouvelles fonctions: Statut et Fonc2

```
dc.l      Statut      ;$200
dc.l      Fonc2       ;$400
cmdtable_end:
```

;Les bits liés avec 'OU' représentent les commandes
;non envoyées au processus, et exécutées directement

```
CommandesDirectes      EQU $1!$2!$40!$80!$100!$200
```

;Voici les commandes exécutées directement:
;Invalid, MyReset, Start, Flush, Statut

```
NOMBFONC      EQU 11      ;Nombre des commandes possibles
```

;Ici commence la routine, appelée chaque fois qu'une commande
;est envoyée au device. Elle indique si la commande est autorisée,
;et si elle est envoyée au processus ou exécutée directement

;=> A1 = Pointeur sur la structure IORequest
;=> A6 = Pointeur sur la structure Device

BeginIO:

```
move.l    a3,-(a7)      ;sauver A3
move.l    IO_UNIT(a1),a3  ;Pointeur sur Unit vers A3
move.w    IO_COMMAND(a1),d0 ;Commande vers D0
cmp.w    #NOMBFONC,d0    ;Commande autorisée?
bcc     BeginIO_NoCmd   ;Saut si non autorisée
```

;DISABLE est une macro qui se trouve dans le fichier Include "exec/ables.i".
;Elle bloque toutes les interruptions comme la fonction Disable.
;Pour que les interruptions puissent être bloquées, il faut écrire
;dans le registre hardware \$DFF09A. L'adresse du registre n'est pas
;indiquée directement dans la macro, mais par l'intermédiaire d'un label.
;Pour que l'éditeur de liens reconnaisse le label, il faut l'indiquer avec
;XREF. Il existe une macro correspondante, appelée INT_ABLES, dans le fichier
;Include "exec/ables.i", et que l'on appelle à partir du programme
;selon les définitions des offsets de la librairie.

```
DISABLE      A0          ;Bloquer les interrupts
move.l    #CommandesDirectes,d1 ;Mot long pour commande directe
btst      d0,d1           ;La commande doit-elle être
                          ;exécutée directement?
bne.s    BeginIO_Immediate ;Sauter, si direct
```

;Sinon, la structure IORequest (la commande) est reconduite à la tâche d'unité
;(processus correct)

BeginIO_QueueMsg:

;le bit UNITB_INTASK indique que la commande est traitée à l'intérieur
;de la tâche d'unité.

```
bset      #UNITB_INTASK,UNIT_FLAGS(a3) ;Poser le bit Unit
```

;En supprimant le bit Quick, on explicite le fait que la tâche
;qui a envoyé la commande passe à l'état d'attente lors de l'utilisation
;de la fonction DoIO, et qu'un message Reply doit être envoyé
;dès que la commande sera terminée.

```
bclr      #IOB_QUICK,IO_FLAGS(a1) ;Effacer IOB_QUICK
```

;Enable est à nouveau une macro du fichier "exec/ables.i".

```

ENABLE      A0          ;Libération des interrupts
move.l    a3,a0          ;Pointeur sur Unit vers A0
LINKSYS    PutMsg.md_SysLib(a6) ;structure IORequest
            ;à déplacer vers le processus
bra.s     BeginIO_end    ;fin

;Offset pour les commandes directes

BeginIO_Immediate:

        ENABLE      A0          ;Libération des interrupts
        bsr        PerformIO    ;Exécuter la commande
BeginIO_end:move.l   (a7)+,a3  ;restaurer A3
            rts         ;retour

;Saut quand le code de commande envoyé est invalide

BeginIO_NoCmd:
        move.b     #IOERR_NOCMD,IO_ERROR(a1)
                    ;Transmet le message d'erreur
        bra.s     BeginIO_end    ;retour

;Aller chercher l'adresse de la fonction pour la commande correspondante et
;passer à cette adresse

;>= A1 - Pointeur sur la structure IORequest
;>= A3 - Pointeur sur la structure Unit
;>= A6 - Pointeur sur la structure Device

PerformIO:
        move.l    a2,-(a7)       ;sauver A2
        move.l    a1,a2          ;IORequest vers A2
        move.w    IO_COMMAND(a2),d0 ;prendre la commande
        lsl.w    #2,d0           ;Offset pour table des
                                ;adresses des fonctions
        lea      cmdtable(pc),a0  ;Pointeur sur table
        move.l    00(a0,d0.W),a0  ;Pointeur sur fonction
        jsr      (a0)             ;Exécuter fonction
        move.l    (a7)+,a2        ;restaurer A2
        rts         ;retour

;La routine suivante doit être appelée à la fin de toutes les fonctions.
;Elle envoie le message Reply à la tâche en attente, à condition que la
;commande ait été exécutée par l'intermédiaire de la tâche d'unité.

TermIO:   move.w    IO_COMMAND(a1),d0 ;Chercher la commande
        move.l    #CommandesDirectes,d1 ;Valeur de masque pour commandes
directes
        btst      d0,d1          ;Commande directe?
        bne.s    TermIO_Direkt   ;Saut si commande directe
        btst      #UNITB_INTASK,UNIT_FLAGS(a3) ;La commande est-elle
                                ;passée par la tâche?
        bne.s    TermIO_Direct   ;Non, donc commande directe
        bclr      #UNITB_ACTIVE,UNIT_FLAGS(a3) ;Libérer la tâche
TermIO_Direct:
        btst      #IOB_QUICK,IO_FLAGS(a1) ;Le bit Quick est-il posé?
        bne.s    TermIO_end       ;Oui, alors commande directe
        LINKSYS  ReplyMsg.md_SysLib(a6) ;sinon envoyer ReplyMsg
TermIO_end: rts         ;retour

;Routine pour interrompre une commande en cours.

;Cette routine dépend du device, et elle n'est donc pas exécutée ici.
;Avec la plupart des devices, il n'est d'ailleurs pas possible d'interrompre
;une action en cours.

;La fonction AbortIO ne fait pas partie des fonctions à prendre dans la

```

;table des commandes. Elle est du même type que BeginIO. C'est pourquoi ;on ne peut pas la conclure avec TermIO.

AbortIO:
 moveq #0,d0
 rts

;Ici commencent les fonctions appelées par l'intermédiaire des commandes ;dans la structure IORequest. Elles doivent toutes être conclues ;par TermIO. A chacune des fonctions, on transmet les mêmes paramètres.

;>= A1 = Pointeur sur la structure IORequest
;>= A3 = Pointeur sur la structure Unit
;>= A6 = Pointeur sur la structure Device

;Commande invalide

Invalid:
 move.b IOERR_NOCMD,IO_ERROR(a1) ;Transmet message d'erreur
 bsr TermIO ;Met fin à la fonction
 rts ;retour

;Commande Reset

;Ce qui doit subir le reset dépend toujours du device, et c'est pourquoi ;il n'est pas possible d'insérer ici un code utile.

MyReset:

; Mettre en place une fonction personnelle
; Mettre en place une fonction personnelle
; Mettre en place une fonction personnelle

 bsr TermIO ;Mettre fin à la fonction
 rts ;retour

;Commande Read

;Comme cela a déjà été dit, nous explicitons ici uniquement le principe ;d'un device personnel. C'est pourquoi notre fonction Read est d'un ;usage pratique très réduit. Elle remplit le secteur mémoire indiqué dans ;IO_DATA avec des octets, dont la valeur représente le numéro de l'unité ;par l'intermédiaire d'une longueur indiquée dans IO_LENGTH. La longueur ;est de plus transférée dans IO_ACTUAL, ce que l'utilisateur peut retirer ;comme valeur de retour dans sa structure IORequest, lorsque le remplissage ;prend fin.

Read:
 move.l IO_DATA(a1).a0 ;chercher le début de la mémoire
 move.l IO_LENGTH(a1),d0 ;chercher la longueur
 move.l d0,IO_ACTUAL(a1) ;longueur comme valeur de retour
 beq.s Read_end ;fin, si longueur = 0
 move.b mdu_UnitNum(a3),d1 ;chercher numéro de l'Unit
Read_Loop: move.b d1,(a0)+ ;remplir le secteur
 subq.l #1,d0 ;diminuer compteur
 bne.s Read_Loop ;poursuivre la copie
Read_end: bsr TermIO ;mettre fin à la fonction
 rts ;retour

;commande Write

;La troisième fonction est traitée dans notre exemple de manière encore ;plus succincte que la fonction Read. Le lecteur doit faire preuve ici de ;créativité. Notre fonction ne transmet que la longueur comme valeur de ;retour.

```

Write:
    move.l      IO_LENGTH(a1),IO_ACTUAL(a1) ;transmet bergeben
    bsr         TermIO                  ;met fin à la fonction
    rts         ;retour

;Les fonctions Update et Clear ne sont pas utiles pour notre "device",
;elles donnent un message d'erreur.

Update:
Clear:   bra      Invalid          ;erreur, fin

;commande Stop

;Avec cette fonction, il est possible d'arrêter toutes les commandes
;entrées après la commande Stop et envoyées à la tâche d'unité. Elles sont
;certes insérées dans la liste des messages, mais elles ne sont pas
;traitées.

MyStop:
    bset      #MDUB_STOPPED,UNIT_FLAGS(a3) ;poser le bit de stop
    bsr       TermIO                  ;mettre fin à la fonction
    rts       ;retour

;commande Start

;Cette commande redonne vie à la tâche précédemment arrêtée, et elle
;traite en même temps toutes les commandes entrées entre-temps. Pour cela,
;il ne suffit pas d'effacer à nouveau le bit de stop, la tâche doit en
;plus attendre qu'on lui dise de poursuivre.

;Pour y parvenir, on fait croire à la tâche qu'une nouvelle commande est
;arrivée (un nouveau message), et qu'elle doit l'exécuter tout de suite.
;En fait, seul le bit correspondant de tâche est posé. La tâche commence par
;examiner la liste des messages, et traite les commandes qu'elle contient.

Start:
    bsr      InternalStart        ;Lancer la tâche
    move.l   a2,a1                ;IORquest vers A1
    bsr      TermIO                ;mettre fin à la fonction
    rts       ;retour

InternalStart:
    bclr      #MDUB_STOPPED,UNIT_FLAGS(a3) ;effacer Stop-Bit
    move.l   MP_SIGTASK(a3),a1  ;cherche pointeur sur Task
    clr.l   d0                   ;D0 à effacer
    move.b   MP_SIGBIT(a3),d1  ;cherche Signal-Bit
    bset      d1,d0                ;Pose bit dans masque
    LINKSYS  Signal,md_SysLib(a6) ;Envoie le signal
    rts       ;retour

;commande Flush

;Avec cette commande, tous les messages (commandes) qui se trouvent dans
;la liste des messages de la tâche sont renvoyés avec un message
;d'annulation.

Flush:
    movem.l  d2/a6,-(a7)        ;Sauver le registre
    move.l   md_SysLib(a6),a6    ;ExecBase vers A6

;FORBID est une macro du fichier "exec/ables.i" et elle fait la même chose que
;la fonction Forbid().

FORBID           ;Forbid()

```

```

Flush_loop:
    move.l      a3,a0          ;Pointeur sur Unit
    CALLSYS    GetMsg           ;Effacer message dans la liste
    tst.l       d0              ;Existe-t-il encore un Msg?
    beq.s       Flush_end       ;Non, liste vide, fin
    move.l      d0,a1           ;Pointeur sur message vers A1
    move.b      #IOERR_ABORTED,IO_ERROR(a1) ;Reporter le message
    CALLSYS    ReplyMsg          ;ReplyMsg()
    bra.s       Flush_loop       ;Saut inconditionnel
    PERMIT
Flush_end:
    movem.l    (a7)+,d2/a6      ;rétablir le registre
    move.l      a2,a1           ;IORequest vers A1
    bsr        TermIO            ;mettre fin à la fonction
    rts         :retour

```

:commande Statut

Cette commande n'appartient pas aux commandes standard dont dispose tout device

;Cet exemple
;est uniquement destiné à montrer comment on crée des fonctions
;personnelles. La fonction lit exclusivement les flags de la structure
;Unit, et les transmet par l'intermédiaire IO_ACTUAL à l'utilisateur

Statut:

```

    clr.l      d0
    move.b    UNIT_FLAGS(a3),d0
    move.l      d0,IO_ACTUAL(a1)
    bsr        TermIO
    rts

```

;Notre dernière fonction est également un fonction personnelle, qui n'est cependant
;pas allouée.

Fonc2:

```

    bsr        TermIO            ;mettre fin à la fonction
    rts         ;retour

```

;Ici commence le segment transmis à la fonction CreateProc(), pour lancer
;le processus Unit.

```

    CNOP      0.4                ;venir sur l'adresse de mot long
myproc_seclist:
    dc.l      0                  ;Pas d'autre segment

```

;Le processus commence à travailler avec Proc_begin

Proc_Begin:

```

    move.l      _AbsExecBase,a6   ;Execbase vers A6
    sub.l       a1,a1           ;Zéro vers A1
    CALLSYS    FindTask          ;Chercher pointeur sur la tâche personnelle
    move.l      d0,a0           ;pointeur vers A0
    move.l      d0,a4           ;et A4
    lea         pr_MsgPort(a0),a0 ;pointeur sur port de messages
                                ;du processus.

```

;Le message attendu est envoyé par la fonction Init_Unit au processus que
;l'on vient de créer. La structure de message se trouve à l'intérieur de
;la structure Unit.

```

    CALLSYS    WaitPort          ;Attendre le message
    move.l      d0,a1           ;Pointeur sur message vers A1
    move.l      d0,a2           ;et vers A2
    CALLSYS    Remove             ;Effacer le message de la liste
    move.l      mdm_Device(a2),a5 ;chercher pointeur sur Device

```

```

move.l    mdm_Unit(a2),a3      ;chercher pointeur sur Unit
moveq    #-1,d0                ;D0 = -1
CALLSYS AllocSignal           ;fonction AllocSignal()
move.b    d0,MP_SIGBIT(a3)     ;Reporter bit de signal
move.b    #PA_SIGNAL,MP_FLAGS(a3) ;S'il est rencontré
                                ;le message doit déclencher
                                ;un signal
clr.l    d7                  ;effacer D7

bset      d0,d7                ;Poser bit de masque pour signal
bra.s    Proc_CheckStatus     ;Interroger le port

;La boucle qui suit est la boucle principale de la tâche d'unité. On attend
;jusqu'à ce qu'un message arrive et soit traité. Après le traitement, on
;saute dans la boucle.

Proc_MainLoop:
move.l    d7,d0                ;Chercher masque bit de signal
CALLSYS Wait                 ;Attendre message
Proc_CheckStatus:
btst      #MDUB_STOPPED,UNIT_FLAGS(a3);Tâche arrêtée?
bne.s    Proc_MainLoop        ;Oui, alors ne pas traiter
                                ;le message
bset      #UNITB_ACTIVE,UNIT_FLAGS(a3) ;Tâche déjà active?
bne.s    Proc_MainLoop        ;Oui, attendre qu'elle soit libre

Proc_NextMsg:
move.l    a3,a0                ;Pointeur sur Port vers A0
CALLSYS GetMsg               ;Chercher message
tst.l    d0                  ;Message existe?
beq.s    Proc_Unlock         ;Non, fin
move.l    d0,a1                ;Pointeur sur message vers A1
exg      a5,a6                ;Pointeur sur Device vers A6
bsr      PerformIO            ;interpréter la commande
exg      a5,a6                ;Pointeur sur ExecBase vers A6
bra     Proc_NextMsg         ;chercher nouveau message

;Il n'y a plus de message (commande), la tâche peut être libérée.

Proc_Unlock:
bclr      #UNITB_ACTIVE,UNIT_FLAGS(a3) ;Effacer bit Active
bclr      #UNITB_INTASK,UNIT_FLAGS(a3) ;Effacer bit Intask
bra     Proc_MainLoop          ;Retour dans la boucle principale

END

```

2.10. Manipulations des interruptions

Dans cette section, nous allons voir comment l'Amiga utilise ses sept niveaux d'interruption et surtout comment nous allons pouvoir les employer. Pour une meilleure compréhension de ce chapitre, il est préférable d'avoir déjà lu la partie concernant les interruptions dans le chapitre sur le hardware. L'interruption ne sera envisagée ici qu'après que le processeur ait reçu le signal correspondant de la logique d'interruption de la puce 4703.

Dès que le processeur a reçu un signal d'interruption non saturé, il charge l'adresse de l'interruption dans son pointeur programme suivant le niveau de priorité, et insère l'interruption à la place correspondante. Les vecteurs pour la suite de l'interruption

commencent à l'adresse \$0064 pour se poursuivre jusqu'à l'adresse \$007F. Les 4 premiers octets contiennent le vecteur pour l'interruption de niveau 1, alors que les octets allant de \$007C à \$007F contiennent le vecteur pour l'interruption de niveau 7.

Etant donné que l'Amiga a besoin de plus d'interruptions que le processeur ne peut en proposer, tous les interrupts interviennent d'abord par l'intermédiaire d'un registre, qui permet de gérer 15 interruptions différentes. Les interruptions introduites sont vérifiées grâce à un registre Interrupt-Enable, ce qui permet d'autoriser ou non le passage d'une interruption vers le processeur. Etant donné que pour les 15 interruptions, seules 7 priorités sont disponibles, plusieurs interruptions possèdent le même ordre de priorité.

En raison des priorités identiques, certaines interruptions différentes auront la même issue, et le tri s'opère par software. Le tableau suivant montre la correspondance entre les interruptions et leur niveau de priorité.

Pseudo-priorité	Nom	Priorité-processeur	Fonction
14	INTEN	(6)	Autorisation d'interruption
13	EXTER	6	Interruption du CIA-B ou du port extension
12	DSKSYN	5	Valeur de synchronisation disquette
11	RBF	5	Tampon d'entrée du port-série plein
10	AUD3	4	Canal audio 3 occupé
9	AUD2	4	Canal audio 2 occupé
8	AUD1	4	Canal audio 1 occupé
7	AUDIO	4	Canal audio 0 occupé
6	BLIT	3	Accès blitter terminé
5	VERTB	3	Début du temps mort vertical
4	COPPER	3	Réservé pour interruptions Copper
3	PORTS	2	Interruption du CIA-A ou du port d'extension
2	SOFT	1	Réservé pour les interruptions du Software
1	DSKBLK	1	Transfert disquette DMA terminé
0	TBE	1	Tampon de sortie du port-série video

La pseudo-priorité permet de gérer par software des interruptions possédant le même niveau de priorité. Les chiffres correspondent, en fait, au numéro de bit des interruptions dans le registre Interrupt-Request.

Pour faciliter la compréhension, voici un exemple :

L'interruption qui sera libérée, lors du passage à la ligne zéro de l'écran du faisceau du raster, possède la même priorité processeur que l'interruption signifiant la fin de l'activité du blitter. Ces deux interruptions sont portées par le bit 5 et 6 du registre Interrupt-Request. Au niveau du Software, l'interruption portant le numéro de bit le plus haut sera traitée en premier. Dans ce cas, l'interruption portant sur le blitter sera exécutée avant celle portant sur le raster. Le tableau montre les 15 interruptions de l'Amiga avec leurs priorités processeur et leurs pseudo-priorités.

Les vecteurs pour les interruptions (\$64-\$7F) sont définis par le système d'exploitation pendant le reset, et ils ne doivent pas être modifiés par le programmeur, car la situation des vecteurs peut se modifier lors des processus ultérieurs. Lors d'une interruption, on passe dans le système d'exploitation par l'intermédiaire des vecteurs d'interruption.

On utilise ici, à partir de la structure ExecBase, la structure IntVector propre à l'interruption, pour poursuivre celle-ci. Il existe ici deux possibilités, qui dépendent de l'initialisation de la structure IntVector.

D'une part, on peut avoir le traitement d'un programme correspondant, par l'intermédiaire d'une interruption, ce traitement se faisant par le moyen d'un handler d'interruption. D'autre part, il peut se faire que plusieurs programmes d'interruption soient traités en même temps.

Cela se réalise par l'intermédiaire du serveur d'interruption. Le système d'exploitation possède quelques-unes des interruptions possibles, prêtes pour la gestion par les serveurs d'interruption. Les autres ne peuvent être utilisées que comme des handlers d'interruption. Voici un tableau qui montre quels sont les interruptions qui doivent être utilisées sous forme de handlers, et quels celles qui doivent l'être sous forme de serveurs :

Pri	Fonction	Type
0	Sortie sérielle	Handler
1	Disk-Block lu	Handler
2	Soft-Interrupts	Handler
3	CIAA-Interrupt	Server
4	Copper-Interrupt	Server
5	Raster-Interrupt	Server
6	Blitter prêt	Handler
7	Audio-0-Interrupt	Handler
8	Audio-1-Interrupt	Handler
9	Audio-2-Interrupt	Handler
10	Audio-3-Interrupt	Handler
11	Entrée sérielle	Handler

Pri	Fonction	Type
12	Disk-Sync-Interrupt	Handler
13	CIAB-Interrupt	Server
14	Interrupt interne (non affecté)	Handler
15	Level-7-Interrupt (NMI)	Server

Le programmeur ne peut utiliser à ce niveau que les interruptions par serveur et l'interruption par handler non affectée. Les autres interruptions sont gérées par le système d'exploitation.

Pour la gestion des interruptions à l'intérieur de la structure ExecBase, on utilise les structures IntVector, définies dans les fichiers Include "exec/interrupts.h" et "exec/interrupts.i". Pour les programmeurs, ces structures n'ont pas d'intérêt particulier. Elles sont simplement utilisées pour les besoins internes du système. C'est pourquoi nous ne les avons pas détaillées ici. Les lecteurs intéressés peuvent les retrouver à l'intérieur de ces fichiers Include.

2.10.1. Utilisation des interruptions

Pour intégrer des interruptions personnelles, on a besoin d'une structure d'interruption. Cette structure est transmise à la fonction que Exec met à votre disposition.

Pour les programmeurs en C, la structure d'interruption est définie dans le fichier Include "exec/interrupts.h" et pour les programmeurs en assembleur dans le fichier Include "exec/interrupts.i".

```

struct Interrupt {
    struct Node is_Node;
    APTR is_Data;
    VOID (*is_Code)();
};

STRUCTURE IS_LN_SIZE 00 $00
APTR     IS_DATA      14 $0E
APTR     IS_CODE      18 $12
LABEL    IS_SIZE      22 $16

```

is_Node / IS

Structure de noeud (Node) pour le chaînage dans une liste lors de l'utilisation de serveurs d'interruptions.

is_Data / IS_DATA

Ceci est un pointeur sur les données en mémoire, géré par l'interruption. Le pointeur est transmis au départ au programme d'interruption. La transmission se fait par l'intermédiaire du registre A1.

is_Code / IS_CODE

Pointeur sur le véritable programme d'interruption.

Une fois la structure d'interruption initialisée, vous pouvez l'insérer dans le système.

2.10.1.1. Handler d'interruption

Envisageons tout d'abord la fonction Exec servant à utiliser un handler d'interruption.

SetIntVector

Fonction `Interrupt = SetIntVector(intNum, int);`
 D0 D0 A1

Offset : -162 -\$0A2

Description :

Cette fonction initialise la structure IntVector à l'intérieur de la structure ExecBase.

Paramètres :

IntNum

Indique le numéro de l'interruption à initialiser. Les numéros correspondent à la priorité (0-15) de l'interruption soutenue par l'Amiga. Avec le numéro 14, vous pouvez redéfinir le vecteur pour une interruption interne.

Int

Pointeur sur la structure Interrupt initialisée. Si le pointeur est nul, cette interruption est considérée comme invalide dans la structure IntVector.

Résultat :

Interrupt

Il s'agit du pointeur sur la structure d'interruption, auparavant gérée par l'intermédiaire de la structure IntVector. A l'aide de ce pointeur, il est possible de restaurer l'état précédent, si l'on doit quitter le programme personnel.

Lorsque l'interruption personnelle est appelée, on trouve dans le registre A1 le pointeur sur l'emplacement en mémoire conservé dans `is_Data`.

2.10.1.2. Les serveurs d'interruptions

On sait qu'il est possible ici de traiter non pas une seule interruption, mais des interruptions en nombre quelconque. Toutes les interruptions exécutées se trouvent dans une liste. La liste est triée d'après la priorité qui figure dans la structure d'interruption.

Pour insérer dans cette structure une interruption personnelle, et pour la supprimer ensuite, Exec propose deux fonctions :

AddIntServer

Fonction `AddIntServer(intNum, Int);`
 D0 A1

Offset : -168 -\$0A8

Description

Cette fonction insère la structure `Interrupt` donnée dans une liste, pour qu'elle puisse être exécutée en cas d'interruption. La priorité donnée dans la structure `Node` correspond à la place de l'interruption dans la liste. Une interruption de haute priorité sera donc insérée dans cette liste avant une interruption de moindre priorité. De plus, l'interruption correspondante sera déclenchée dans le registre `Interrupt-Enable`.

Paramètres :

IntNum

Numéro de l'interruption qui déclenche le programme personnel. Les numéros correspondent à la priorité de l'interruption soutenue par l'Amiga. Avec le numéro 5, vous placeriez par exemple dans la liste une structure d'interruption traitée lors d'une interruption raster.

Int

Pointeur sur la structure d'interruption initialisée.

RemIntServer

Fonction `RemIntServer(intNum, Int);`
 D0 A1

Offset -174 -\$0AE

Description

La structure Interrupt donnée sera supprimée de la liste des serveurs.

Paramètres :

IntNum

Indique le numéro de l'interruption qui gère la liste.

Int

Pointeur sur la structure Interrupt initialisée.

Lorsqu'on quitte l'interruption, on reçoit en retour une valeur d'erreur dans D0. Si celle-ci est nulle, toutes les autres interruptions figurant dans la liste ne sont pas traitées.

2.10.1.3. Interruptions - Soft

Comme le titre peut le laisser entrevoir, cette section va porter sur les interruptions déclenchées (à la main) au niveau du software. Ces interruptions ont une priorité supérieure à celle des tâches, mais inférieure à la plupart des interruptions du hardware. La seule exception est l'interruption de niveau 14, à laquelle nous allons revenir ultérieurement.

Ces interruptions soft peuvent être utilisées pour traiter n'importe quel processus asynchrone.

Pour déclencher une telle interruption, on se sert de la fonction `Cause()` de `Exec`. Celle-ci insère la structure d'interruption qui lui est transmise dans l'une des cinq listes se trouvant à l'intérieur de la structure `ExecBase`. La liste dans laquelle la structure d'interruption est insérée dépend de sa priorité. Puisqu'il existe cinq listes, on ne dispose que de cinq niveaux de priorité. Ces priorités sont les suivantes : -32, -16, 0, 16, 32.

Lorsqu'on utilise des interruptions soft, il importe de définir le type de la structure de noeud de l'interruption comme `NT_INTERRUPT` (2). Pour empêcher qu'une interruption soft figure deux fois dans les listes d'interruptions soft, son type est modifié lorsqu'il s'introduit dans l'une des listes : il n'est plus `NT_INTERRUPT`, mais `NT_SOFTINT`. Avant toute introduction dans les listes, il y a un examen du type. Si le type est `NT_SOFTINT`, la structure ne peut pas être intégrée. Lorsque l'interruption prend fin, son type redevient `NT_INTERRUPT`. La définition de `NT_INTERRUPT` et de `NT_SOFTINT` est réalisée dans le fichier `Include "exec/nodes"`.

Lorsqu'on a introduit une structure d'interruption personnelle dans l'une des listes, un bit correspondant est défini dans la structure ExecBase et le registre Interrupt-Request. Ceci déclenche l'interruption. Le système d'exploitation passe en revue les liste d'interruptions soft, et exécute les interruptions qui y figurent.

Pour les interruptions soft également, le pointeur sur le secteur de données figurant dans is_Data est transmis dans le registre A1.

Cause

Fonction Cause(interrupt)
A1

Offset -180 - \$0B4

Description

Déclenche, à partir d'une tâche ou d'une interruption, une autre interruption de priorité moindre.

Ports de message et interruptions soft

Les interruptions soft sont également utilisées en relation avec le système de messages de l'Amiga. Un port de message peut être initialisé de telle façon qu'une interruption soft soit déclenchée par un message qui arrive. Il faut pour cela réaliser l'initialisation suivante :

```
mp_Flags -> PA_SOFTINT - 1
mp_SigTask -> Pointeur sur la structure d'interruption
```

L'interruption interne de niveau 14

Le bit 14 du registre Interrupt-Enable est utilisé comme "master-bit" pour toutes les interruptions. Si ce bit n'est pas posé, aucune interruption n'est traitée. Ce blocage de l'interruption n'est cependant pas réalisé au niveau du hardware, mais du software. Avant le traitement d'une interruption, le système d'exploitation vérifie si le bit 14 du registre Enable (\$DFF09A) est posé, et l'interruption qui se présente n'est exécutée que si c'est le cas.

L'interruption de priorité 14 ne peut pas être générée par l'intermédiaire du hardware. Elle ne peut pas être autorisée ou bloquée sans l'influence des autres interruptions, à cause du bit 14 dont nous venons de décrire l'effet.

Pour utiliser l'interruption de niveau 14, il faut auparavant initialiser la structure IntVector à l'intérieur de la structure ExecBase par l'intermédiaire de la fonction SetIntVector de Exec (cf. Handler d'interruption). Ensuite, en posant le bit 14 dans le registre Interrupt-Request (\$DFF09C), on peut déclencher l'interruption. Cette interruption a la priorité la plus élevée parmi toutes les interruptions qui peuvent être masquées. Le

blocage et la libération des interruptions s'effectuent au moyen des fonctions Disable() et Enable(), qui influencent simplement le "master-bit" qui a été décrit (bit 14).

2.10.1.4. Les interruptions CIA

Comme vous le savez sans doute, l'Amiga dispose de deux éléments d'entrée/sortie : les CIA. Ces éléments sont en mesure de déclencher des interruptions, au moyen de différents événements. Ces interruptions du CIA sont conduits par CIAA au bit 3 de la logique d'interruption, et par CIAB au bit 13.

Puisque les CIA peuvent déclencher des interruptions avec cinq événements différents, on a introduit pour leur gestion la structure CIA-Resource. Chacune des deux CIA dispose d'une structure nommée "ciaa.resource" ou "ciab.resource".

Voici les interruptions gérées :

Numéro	Interruption
0	Timer A en route
1	Time B en route
2	Alarme de l'horloge en temps réel
3	Arrivée de données sérielles
4	Interruption par un conduit de flag

Quelques interruptions sont utilisées par le système :

CIAA	Interruption Timer B
CIAB	Alarme de l'horloge en temps réel
CIAA	Arrivée des données sérielles (données clavier)
CIAB	Interruption par flag (indication index dans disque)

Les interruptions autorisées sont conservées à l'intérieur de la structure Resource. Chaque interruption est représentée par un bit dont le numéro correspond au numéro de l'interruption (de 0 à 4). Un bit posé à l'intérieur du registre de masque correspond à une interruption autorisée. Pour chacune des cinq interruptions, il existe une structure IntVector, par l'intermédiaire de laquelle on passe dans le programme d'interruption. A l'aide de la fonction CIA, il est possible d'insérer de nouvelles interruptions, pour modifier le registre du masque, et ainsi pour rendre possible ou au contraire pour bloquer des interruptions.

Les cinq interruptions sont traitées dans l'ordre de leur numérotation. Pour appeler une fonction de la ressource CIA, le pointeur sur la structure de ressource CIA doit se trouver dans le registre A6. Dans ce qui suit, nous allons expliciter les différentes fonctions Resource.

AbleICR

Fonction AncMasque = AbleICR(masque)
D0 D0

Offset -18 -\$12 (pointeur sur Resource dans A6)

Description

Avec cette fonction, il est possible d'accéder au registre de masque, et donc de bloquer ou d'autoriser une interruption CIA.

Paramètre

masque

Les bits posés dans le masque sont modifiés dans la structure Resource. Le bit 7 indique la forme sous laquelle les bits seront modifiés. Si le bit 7 est posé, toutes les interruptions dont les bits sont posés dans le masque seront autorisées. Si le bit 7 du registre de masque n'est pas posé, toutes les interruptions indiquées dans la Resource CIA, ainsi que dans le CIA seront bloquées. Voici un exemple pour expliciter la situation :

```
masque = $81 -> autorise l'interruption Timer A
masque = $03 -> Bloque interruptions Timer A et Timer B
```

ancMasque

Le masque précédent est transmis en D0.

AddICRVector

Fonction AncInterrupt = AddICRVector(intNum, Interrupt)
D0 D0 A1

Offset -6 -\$06 (pointeur sur Resource en A6)

Description

Avec cette fonction, on fait figurer une nouvelle interruption dans la structure de ressource CIA, et cette interruption est enregistrée comme étant autorisée dans le registre de masque. Mais ce n'est possible que si l'interruption n'a pas déjà été affectée.

Paramètres

IntNum

On transmet ici le numéro de l'interruption qui doit être nouvellement affectée. Avec le numéro, on peut insérer une nouvelle interruption par flag.

Interrupt

C'est le pointeur sur une structure d'interruption auparavant initialisée, qui a déjà été décrite.

AncInterrupt

On obtient ici en retour le pointeur sur la structure d'interruption que cette interruption de CIA occupait auparavant. Si la valeur de retour est nulle, l'interruption était déjà affectée, et elle n'a pas été remplacée par l'interruption personnelle. Pour intégrer l'interruption personnelle, il faut d'abord supprimer la précédente, ce que l'on fait avec la fonction Resource RemICRVector.

RemICRVector

Fonction RemICRVector(numéro)
 D0

Offset -12 -\$0C (pointeur sur Resource en A6)

Description

Efface l'interruption indiquée dans la structure Resource CIA, et la bloque en même temps.

Paramètre

Numéro

On transmet ici le numéro de l'interruption qui doit être supprimée (0 à 4).

SetICR

Fonction	AncMasque = SetICR(masque)
	DO
Offset	-24 - \$18 (pointeur sur Resource en A6)
Description	
La fonction modifie le registre Interrupt-Request de la structure CIA-Resource. Il est ainsi possible de supprimer des interruptions survenues à partir du registre Request, et donc d'empêcher leur exécution, ou encore de déclencher "artificiellement" les interruptions indiquées dans le masque.	

masque

On indique ici quels sont les bits qui doivent être modifiés dans le registre Request. Si le bit 7 est posé dans le masque, les bits correspondants dans le registre Request de la structure CIA-Resource sont posés, et une interruption est déclenchée. Si le bit n'est pas posé, les bits indiqués dans le registre Request ne le sont pas non plus. Si c'est un zéro qui est transmis, il n'y a pas de modification du registre Request. La fonction travaille en son principe comme AbleICR.

ancMasque

On obtient ici en retour la valeur du registre Request avant toute modification.

2.10.2. Exemples de programmes

Le programme suivant montre comment on utilise un serveur d'interruption sur l'exemple de l'interruption raster. Le programme intègre une interruption dans la liste des interruptions du retour vertical de faisceau. Si l'interruption est appelée, un signal est envoyé à la tâche en attente, et celle-ci supprime ensuite l'interruption et elle-même.

Le programme peut être assemblé avec l'assembleur "AS" de la société MANX et d'autres du même type.

```

include "exec/types.i"
include "exec/memory.i"
include "exec/interrupts.i"
include "mymacro.i" ; (Cf. la description des macros)
;-----
:Exec
    XLIB AllocMem
    XLIB FreeMem
    XLIB AddIntServer
    XLIB RemIntServer
    XLIB Signal
    XLIB AllocSignal
    XLIB FindTask
    XLIB Wait
;-----
```

```

XREF _AbsExecBase
STRUCTURE global.0
    APTR      gl_MyTask
    LONG      gl_SignalBit
    STRUCT    gl_Interrupt.IS_SIZE
    LABEL    gl_size
MyIntPri EQU 0
;*****
;XDEF main
main:
    move.l  _AbsExecBase,a6
    move.l  #gl_size,d0
    move.l  #MEMF_PUBLIC!MEMF_CLEAR,d1
    CALLSYS AllocMem
    tst.l   d0
    beq    main_NoGlobalMem
    move.l  d0,a5          ;Pointeur sur la structure globale
    sub.l   a1,a1          ;A1 = 0
    CALLSYS FindTask        ;chercher pointeur sur la tâche
                            ;personnelle
    move.l  d0,gl_MyTask(a5) ;pointeur sur la tâche personnelle
    move.l  #-1,d0          ;chercher un bit de signal quelconque
    CALLSYS AllocSignal      ;allouer un bit de signal
    tst.b   d0              ;Bit de signal alloué?
    bmi    main_NoFreeSignal ;non. erreur -> fin
    clr.l   d1
    bset   d0,d1
    move.l  d1,gl_SignalBit(a5) ;déposer le masque de signaux
;* Initialiser les interruptions
    lea    gl_Interrupt(a5).a2 ;pointeur sur Int-Struktur
    move.b  #NT_INTERRUPT.LN_TYPE(a2) ;Reporte le type
    move.b  #MyIntPri.LN_PRI(a2) ;Reporter la priorité
    lea    IntName(pc),a0 ;pointeur sur nom
    move.l  a0,LN_NAME(a2) ;reporter pointeur sur nom
    move.l  a5,IS_DATA(a2) ;pointeur sur structure globale
    lea    IntProg(pc),a0 ;pointeur sur Int-Programm
    move.l  a0,IS_CODE(a2) ;reporter pointeur sur programme
    move.l  #5,d0          ;numéro d'interruption
    lea    gl_Interrupt(a5).a1 ;pointeur sur Interrupt
    CALLSYS AddIntServer    ;intégrer le serveur
    move.l  gl_SignalBit(a5).d0 ;chercher les bits de signal
    CALLSYS Wait             ;Attendre une interruption
    move.l  #5,d0          ;Numéro d'interruption
    lea    gl_Interrupt(a5).a1 ;pointeur sur la structure Int
    CALLSYS RemIntServer    ;Supprimer l'interruption
main_NoFreeSignal:
    move.l  a5,a1
    move.l  #gl_size,d0
    CALLSYS FreeMem
main_NoGlobalMem:
    clr.l   d0
    rts
;*****
;Interrupt Programm
;=> A1 = pointeur sur structure globale
IntProg:
    move.l  gl_SignalBit(a1).d0 ;Bits de signal vers D0
    move.l  gl_MyTask(a1).a1 ;pointeur sur Task personnel
    LINKSYS Signal._AbsExecBase ;lancement de la tâche personnelle
    clr.l   d0                 ;Message OK pour l'interruption suivante
    rts
;*****
IntName: dc.b 'test.interrupt',0
;*****
END

```

Le programme suivant montre comment on utilise des interruptions soft. Il crée un port de message, un message et une interruption. Le message est envoyé au port, qui déclenche une interruption soft. L'interruption envoie un signal à la tâche, qui supprime ensuite le tout du système.

```

include "exec/types.i"
include "exec/interrupts.i"
include "exec/ports.i"
include "exec/memory.i"

;*****
CALLSYS MACRO
    IFGT NARG-1
        FAIL
    ENDC
        JSR    _LVO\1(A6)
    ENDM

LINKSYS MACRO
    IFGT NARG-2
        FAIL
    ENDC
        MOVE.L A6,-(SP)
        MOVE.L \2,A6
        CALLSYS \1
        MOVE.L (SP)+,A6
    ENDM

XLIB MACRO
    XREF _LVO\1
    ENDM
;*****

STRUCTURE global.0
    APTR      gl_MsgPort
    APTR      gl_Interrupt
    APTR      gl_Message
    APTR      gl_Task
    BYTE      gl_SigBit
    BYTE      gl_pad
    LABEL     gl_size

STRUCTURE MyPort,MP_SIZE
    STRUCT    mp_SoftInt,IS_SIZE
    STRUCT    mp_Message,MN_SIZE
    LABEL     mp_size

IntPri EQU 0

;*****
XREF _AbsExecBase
;*****
;exec
XLIB AllocMem
XLIB FreeMem
XLIB AllocSignal
XLIB PutMsg
XLIB GetMsg
XLIB Wait
XLIB FindTask
XLIB Signal
;*****

```

```

main:
    move.l  _AbsExecBase,a6
    lea     -gl_size(a7),a7          ;Place dans la pile
    move.l  a7,a5

    move.l  #mp_size,d0
    move.l  #MEMF_PUBLIC!MEMF_CLEAR,d1
    CALLSYS AllocMem               ;allocation de mémoire pour le port
    move.l  d0,gl_MsgPort(a5)
    beq    main_NoMem

    move.l  d0,a2                  ;pointeur sur le Port
    move.b  #NT_MSGPORT,LN_TYPE(a2) ;report du type
    lea     PortName(pc),a0
    move.l  a0,LN_NAME(a2)         ;report du nom

    move.b  #PA_SOFTINT,MP_FLAGS(a2) ;report du flag
    lea     mp_SoftInt(a2),a0
    move.l  a0,MP_SIGTASK(a2)       ;Interrupt -> SignalTask
    lea     MP_MSGLIST(a2),a0

* NEWLIST est une macro de exec/lists.i destinée à initialiser une structure de liste

    NEWLIST  A0                      ;initialise une structure de liste

    lea     mp_SoftInt(a2),a1        ;pointeur sur structure Interrupt
    move.l  a2,gl_Interrupt(a5)      ;raporte le pointeur sur Int
    move.b  #NT_INTERRUPT,LN_TYPE(a1) ;raporte le type
    move.b  #IntPri,LN_PRI(a2)      ;raporte la priorité
    lea     IntName(pc),a0
    move.l  a0,LN_NAME(a1)          ;raporte le nom
    move.l  a5,IS_DATA(a1)          ;pointeur sur structure globale.
    lea     InterruptPrg(pc),a0
    move.l  a0,IS_CODE(a1)          ;raporte le pointeur sur le programme

    lea     mp_Message(a2),a1        ;pointeur sur Message
    move.l  a1,gl_Message(a5)        ;raporte le pointeur sur Message
    move.b  #NT_MESSAGE,LN_TYPE(a1)
    lea     MessageName(pc),a0
    move.l  a0,LN_NAME(a1)          ;raporte le pointeur sur le nom
    move.l  gl_MsgPort(a5),MN_REPLYPORT(a1) ;raporte le pointeur sur le port
    move.w  #MN_SIZE,MN_LENGTH(a1)   ;raporte la longueur

    sub.l   a1,a1
    CALLSYS FindTask               ;cherche le pointeur sur la tâche
personnelle
    move.l  d0,gl_Task(a5)          ;raporte le pointeur sur la tâche

    move.l  #-1,d0                 ;bit quelconque
    CALLSYS AllocSignal             ;cherche le bit de signal
    move.b  d0,gl_SigBit(a5)        ;sauvegarde le numéro de bit
    bmi    main_NoSignal            ;erreur, pas de bit libre

;-----

    move.l  gl_MsgPort(a5),a0
    move.l  gl_Message(a5),a1
    CALLSYS PutMsg                 ;Déclenche une interruption

    clr.l   d0
    move.b  gl_SigBit(a5).d1
    bset   d1,d0
    CALLSYS Wait                     ;pose un bit alloué dans le masque
                                      ;attend la fin de l'interruption

;-----
```

main_NoSignal:

```

move.l    gl_MsgPort(a5),a1
move.l    #mp_size,d0
CALLSYS  FreeMem                      ;libère de la mémoire pour Port et Int
main_NoMem:
    lea     gl_size(a7),a7              ;libère la pile
    clr.l   d0
    rts
*****
InterruptPrg:
    movem.l  a5-a6,-(a7)
    move.l   _AbsExecBase,a6
    move.l   a1,a5                  ;pointeur sur globls
    move.l   gl_MsgPort(a5),a0
    CALLSYS  GetMsg                ;cherche Message

* Intégrer le programme personnel
* Intégrer le programme personnel
* Intégrer le programme personnel

    clr.l   d0
    move.b   gl_SigBit(a5),d1
    bset    d1,d0
    move.l   gl_Task(a5),a1
    CALLSYS  Signal                 ;Envoyer message de fin à la tâche
    movem.l  (a7)+,a5-a6
    rts
*****
PortName:      dc.b 'DemoPort',0
IntName:       dc.b 'DemoInterrupt',0
MessageName:   dc.b 'DemoMessage',0
;*****
END

```

2.11. Sémaphores

Dans les chapitres précédents, nous avons déjà parlé de la communication entre les tâches. Il faut maintenant compléter le tableau.

Nous avons appris que la collaboration entre les différentes parties du système d'exploitation se fait par l'intermédiaire de ports de message et de messages envoyés par les ports en question. A l'aide de ces ports, on maîtrise tous les problèmes qui se posent du fait du fonctionnement multitâche.

Il y a des difficultés dans le fonctionnement multitâche que l'on peut certes résoudre avec les ports de message, mais seulement de manière très complexe. C'est pourquoi de nouvelles structures ont été introduites pour faciliter le travail. Ces structures s'appellent des sémaphores.

Les sémaphores sont utilisés lorsque plusieurs tâches doivent accéder à une unité déterminée, chacun de ces accès devant se réaliser sans intervention des autres tâches. Il existe un premier moyen pour y parvenir : empêcher l'accès des autres tâches avec `Forbid()`, ce qui n'est cependant concevable que pendant un très court laps de temps. Si l'accès dure un certain temps, le blocage des autres tâches risque de provoquer des difficultés dans le système.

Un exemple pour un tel accès est l'utilisation du blitter lors de l'utilisation de OwnBlitter(). Une seule tâche peut accéder au blitter dans le même temps, et l'accès doit donc être interdit aux autres tâches pendant le temps de l'accès.

Un second exemple est le traitement des structures publiques comme la structure Expansion-Library. Si une tâche veut accéder à cette structure, elle envoie une interrogation au sémaphore appartenant à la structure (fonction : ObtainSemaphore()). Ceci permet de voir si une autre tâche accède actuellement à la structure. Si c'est le cas, l'interrogation est placée à la fin d'une liste, et la tâche est placée sur "Wait". Lorsque la tâche qui a actuellement accès à la structure a terminé son travail, elle libère la structure pour les autres tâches (fonction : ReleaseSemaphore()). Grâce à cette libération, la première tâche qui se trouve en liste d'attente est autorisée à accéder à la structure. S'il y a une tâche moins importante que les autres, qui peut donc accéder plus tard à la structure, on peut utiliser la fonction AttemptSemaphore().

Cette fois, la fonction vérifie si le sémaphore est occupé, et elle transmet un message en conséquence. La tâche qui a envoyé l'interrogation n'est pas placée en liste d'attente, contrairement à ce qui se passe avec la fonction ObtainSemaphore().

Exec propose aux programmeurs deux types de sémaphores. Il s'agit du Semaphore-Port et du Semaphore-Signal. Tous deux servent en fin de compte à la même chose : à assurer l'accès exclusif à un périphérique, à une structure ou à un objet analogue.

Le moyen le plus rapide pour obtenir un accès exclusif est de passer par le semaphore-signal, car dans ce cas l'utilisateur n'a pas à initialiser de structures propres, et possède tout un choix de fonctions performantes. Cette méthode simple se réalise cependant aux dépens de la souplesse d'utilisation, car il est uniquement possible de placer sa propre tâche en état de repos (wait) jusqu'à ce que l'accès soit autorisé, ou de tester la possibilité d'accès (gaspillage de temps).

En utilisant les ports de semaphore, il est possible de faire savoir au système d'exploitation que l'on désire accéder à une unité déterminée, et le système d'exploitation vous informe alors automatiquement, dès que l'accès est autorisé. La tâche personnelle peut vaquer entre-temps à d'autres occupations, et attendre de cette façon plusieurs accès à la fois, et exécuter des tâches diverses, selon l'accès qui lui est accordé.

Vous vous doutez sans doute que la communication automatique d'un laissez-passer a pour conséquence un port de message pour la tâche personnelle. En outre, il est préférable de distinguer diverses autorisations d'accès, et donc de créer un message destiné à chacune des interrogations. Vous voyez que l'effort de programmation dans le cas des sémaphores-Ports est infiniment plus grand que dans le cas des sémaphores-signaux.

Ce que vous devez observer en détail pour utiliser des ports de sémaphores est indiqué dans la description des fonctions Procure() et Vacate(). Les autres fonctions décrites se réfèrent aux sémaphores-signaux.

Remarques importantes sur les sémaphores

Pour éviter des "dead-locks", on ne doit pas affecter plusieurs sémaphores simultanément, sans respecter les mécanismes de protection afférents. Pour affecter plusieurs sémaphores, on dispose de la fonction ObtainSemaphoreList(), qui contient ce mécanisme de protection. Un court instant de réflexion permettra de comprendre pourquoi on ne peut pas le faire avec ObtainSemaphore(). Qu'on se représente deux sémaphores (Sem1 et Sem2) et deux tâches (Task1 et Task2). Task1 affecte Sem1 et Task2 affecte Sem2. Sans libérer les sémaphores affectés, Task1 veut également affecter Sem2 et Task2 Sem1. Les deux tâches se bloquent alors mutuellement, car elles attendent toutes deux la libération des sémaphores occupés par l'autre.

En second lieu, il faut noter que les mécanismes qui permettent un accès exclusif ne doivent pas être mélangés. Nous faisons allusion au mélange de Forbid ou Disable avec des sémaphores. Si l'on a prévu un sémaphore pour régler l'accès à une structure, on ne peut pas travailler avec Forbid à partir d'une autre tâche, pour assurer l'accès, car la première tâche travaillant par l'intermédiaire d'un sémaphore risque d'être interrompue par la seconde en plein milieu de son travail avec la structure.

2.11.1. Les structures de sémaphores

Toutes les structures suivantes se trouvent dans le fichier Include "exec/semaphors.h" ou exec/semaphors.i" en assembleur.

```
struct SignalSemaphore {
    struct Node          ss_Link;
    SHORT                ss_NestCount;
    struct MinList       ss_WaitQueue;
    struct SemaphoreRequest ss_MultipleLink;
    struct Task           *ss_Owner;
    SHORT                ss_QueueCount;
};

                                     Offsets
STRUCTURE SS_LN_SIZE             00 $00
SHORT     SS_NESTCOUNT           14 $0E
STRUCT    SS_WAITQUEUE_MLH_SIZE   16 $10
STRUCT    SS_MULTIPLELINK_SSR_SIZE 28 $1A
APTR      SS_OWNER               40 $28
SHORT     SS_QUEUECOUNT          44 $2C
LABEL    SS_SIZE                 46 $2E
```

ss_Link

Il s'agit ici d'une structure de noeud qui nous est familière, et qui a pour fonction d'enchaîner dans une liste plusieurs structures de sémaphores-signaux. Une structure de liste pour l'enchaînement de la structure de sémaphore existe déjà dans la structure ExecBase sous le nom de SemaphoreList, avec l'offset 532.

ss_NestCount

C'est un compteur qui permet de décider si une structure de séaphore-signal est exigée, quand elle doit l'être, et même combien de fois. A chaque demande d'une telle structure, le compteur est incrémenté d'une unité, et il est au contraire décrémenté lorsque l'on prend congé de la structure. Si le compteur est à 0, le séaphore est considéré comme libre.

ss_WaitQueue

On utilise cette structure MinList pour lier l'une à l'autre plusieurs structures SemaphoreRequest (dont nous aurons à reparler). Chacune de ces structures représente une tâche, qui attend d'être affectée à un séaphore-signal. Lorsqu'une nouvelle tâche vient s'ajouter aux précédentes, sa demande (structure SignalRequest) est placée en fin de liste.

ss_MultipleLink

Cette structure (dont nous allons reparler) sert uniquement aux besoins internes du système d'exploitation, en relation avec la fonction ObtainSemaphoreList(), et elle ne concerne pas l'utilisateur.

**ss_Owner*

Pointeur sur la tâche qui occupe actuellement le séaphore.

ss_QueueCount

Ce compteur sert à reconnaître le nombre de tâches qui attendent d'occuper le séaphore. Si le séaphore n'est pas utilisé, le compteur se trouve sur -1. Lorsqu'une tâche utilise le séaphore, et qu'il n'y a pas d'autre tâche en attente, le compteur se trouve sur 0.

Une autre structure nécessaire en relation avec l'utilisation des sémaphores est la structure SemaphoreRequest. Elle intervient dans ss_WaitQueue, et se trouve en relation directe avec la tâche en attente d'un accès. Cette structure est moins intéressante pour l'utilisateur, car on en a besoin uniquement à l'intérieur de la fonction décrite ci-dessous, où elle intervient sans initialisation :

```
struct SemaphoreRequest {
    struct MinNode sr_Link;
    struct Task *sr_Waiter;
};

STRUCTURE      Offsets
SSR,MLN_SIZE  00  $00
APTR          SSR_WAITER   08  $08
LABEL         SSR_SIZE    12  $0C
```

sr_Link

Structure MinNode, grâce à laquelle la structure SemaphoreRequest intervient dans la liste ss_WaitQueue.

*sr_Waiter

pointeur dirigé sur la tâche (la structure de tâche) attendant l'utilisation du sémaphore.

On doit encore mentionner la structure de sémaphore (Semaphore-Port) qui procure les libertés dont il vient d'être question, mais ne peut pas être utilisée sans préparation. Elle consiste en un port de message, étendu d'un MOT, et c'est pourquoi nous l'appellerons aussi port de sémaphore dans ce qui suit, pour bien la distinguer des sémaphores-signaux.

```
struct Semaphore {
    struct MsgPort sm_MsgPort;
    WORD           sm_Bids;
};

#define sm_LockMsg mp_SigTask

STRUCTURE  SM_MP_SIZE      Offsets
WORD        SM_BIDS          00 $00
LABEL       SM_SIZE          34 $22
                  36 $24

SM_LOCKMSG   EQU  MP_SIGTASK
```

Nous allons maintenant décrire les fonctions mises à votre disposition par Exec, pour traiter les sémaphores-signaux, à partir desquelles il est possible de retirer des informations plus précises pour le traitement des sémaphores.

2.11.2. Les fonctions de sémaphores-signaux

Dans cette section, les sémaphores seront toujours des sémaphores-signaux.

initSemaphore

Fonction	InitSemaphore(SignalSemaphore)
	A0

Offset	-558 -\$22E \$FDD2
---------------	--------------------

Description

Pour créer un sémaphore-signal, il faut initialiser ses entrées. Cette tâche est prise en charge par la fonction présente. Elle crée la structure MinList, supprime les entrées ss_NestCount et ss_QueueCount, ainsi que le pointeur sur la tâche. Il ne vous reste plus

qu'à définir le type et le nom dans la structure de noeud, ce qui importe peu pour la fonction du sémaphore, mais est recommandé pour respecter un style de programmation correct.

Si votre sémaphore est un sémaphore global, il faut faire figurer un nom, car sinon les tâches correspondantes ne retrouveraient pas le sémaphore. Il faut veiller à ce que plusieurs sémaphores globaux différents ne portent pas le même nom, car dans une telle situation un seul d'entre eux pourrait être retrouvé. Les sémaphores globaux sont ceux qui figurent dans la liste des sémaphores de la structure ExecBase.

Paramètre

SignalSemaphore

Pointeur sur l'emplacement en mémoire prévu pour le sémaphore-signal.

Obtain Semaphore

Fonction	ObtainSemaphore(<i>signalSemaphore</i>) A0
-----------------	---

Offset	-564 -234 \$FDCC
---------------	------------------

Description

La fonction examine ss_QueueCounter pour voir si le sémaphore est utilisé actuellement par une tâche. Si ce n'est pas le cas, la fonction prend fin, tout en conservant en mémoire le fait que le sémaphore n'est pas utilisé.

Si le sémaphore est déjà occupé, la fonction vérifie si l'utilisateur actuel est la même tâche. Si c'est le cas, cela signifie que la tâche veut attendre à nouveau une "autorisation d'accès", bien qu'elle l'ait déjà. Sa nouvelle demande n'est pas placée dans la liste, mais traitée immédiatement. Il s'agit d'une mesure de sécurité pour éviter un effondrement qui se produirait immanquablement, puisqu'on est alors dans le cas où une tâche attend pour se libérer elle-même (Dead Lock).

En dernier lieu, il reste à déterminer si le sémaphore est utilisé ou non par une autre tâche. Dans ce cas, une structure SemaphoreRequest est créée dans la pile, et placée à la fin de la liste ss_WaitQueue. Puis le bit qui, dans tc_SigRecv de la structure de tâche, représente la réception d'un signal de sémaphore, est supprimé. En fin de compte, la tâche est placée en situation Wait, et elle n'est "réveillée" que si un signal correspondant lui est envoyé (cf. ReleaseSemaphore()).

Paramètre

SignalSemaphore

Pointeur sur une structure de sémaphore-signal initialisée.

ReleaseSemaphore

Fonction ReleaseSemaphore(*SignalSemaphore*)
 A0

Offset -570 -\$23A \$FDC6

Description

Lorsqu'une tâche a reçu, à l'aide de ObtainSemaphore(), l'autorisation d'accéder à une unité correspondant au sémaphore, et donc revendique le sémaphore comme sien, elle doit le libérer quand elle a terminé son travail. A ce moment, le sémaphore peut être confié à une autre tâche.

Au moment où le sémaphore est libéré, il faut se demander si d'autres tâches attendent sa libération par l'intermédiaire de la structure SemaphoreRequest dans la liste d'attente (Wait-Queue). Si c'est le cas, la tâche dont la demande est arrivée en premier figure comme Owner-Task, et sa structure de sémaphore est retirée de la liste. Pour que la nouvelle tâche puisse commencer son travail, on lui envoie un signal en conséquence.

Paramètre

SignalSemaphore

Pointeur sur une structure de sémaphore-signal.

AttemptSemaphore

Fonction Reply = AttemptSemaphore(*SignalSemaphore*)
 D0 A0

Offset -576 -\$240 \$FDC0

Description

On a besoin de cette fonction lorsque l'accès à l'unité gérée par le sémaphore n'est pas absolument nécessaire pour pouvoir poursuivre. La fonction examine si le sémaphore est occupé ou non, et renvoie un message en conséquence. S'il n'est pas occupé, la fonction s'en charge.

Paramètre

SignalSemaphore

Pointeur sur la structure de sémaphore-signal.

Reply

Grâce à la valeur de retour de la fonction, on peut savoir si le sémaphore correspondant est libre ou occupé.

Reply = 0 → sémaphore occupé
 Reply = 1 → sémaphore libre

ObtainSemaphoreList

Fonction ObtainSemaphoreList(List)
 A0

Offset -582 -\$246 \$FDBA

Description

La fonction travaille comme ObtainSemaphore, mais elle se réfère à toute une liste de sémaphores. Elle ne revient dans le programme principal que lorsque tous les sémaphores de la liste ont été occupés pour la tâche. La liste est donc passée en revue, et tous les sémaphores libres sont alloués. Les sémaphores occupés sont attendus avec la fonction Wait, grâce au fait que la structure SignalRequest qui se trouve à l'intérieur de la structure de sémaphore-signal est placée dans la liste ss_WaitQueue du sémaphore propre. Cette astuce permet d'éviter la création d'une structure SignalRequest par l'utilisateur.

Paramètre*List*

Pointeur sur une structure de liste, au moyen de laquelle certaines structures de sémaphores-signaux sont enchaînées.

ReleaseSemaphoreList

Fonction ReleaseSemaphoreList(List)
 A0

Offset -588 -\$24C \$FDB4

Description

La fonction travaille comme ReleaseSemaphore, mais se réfère à une liste de sémaphores.

Paramètre

List

Pointeur sur une structure de liste dans laquelle figurent les sémaphores précédemment occupés.

AddSemaphore

Fonction

AddSemaphore(SignalSemaphore) Erroné!!

Offset -600 -\$258 \$FDA8

Description

Attention : Cette fonction est inutilisable à cause d'une erreur!

Elle avait été conçue pour initialiser une structure de semaphore-signal, et pour la placer dans la liste des sémaphores de la structure ExecBase. Elle est inutilisable, parce que le pointeur sur le semaphore est transmis à la fonction InitSemaphore() en A0, et à la fonction EnQueue en A1. Il n'est pas possible de transmettre le pointeur sur la structure de semaphore aussi bien en A0 qu'en A1, et de l'utiliser en même temps, puisque InitSemaphore() modifie A1.

La fonction correcte doit avoir l'aspect suivant :

```

ExecBase      EQU 4
TDNestCnt    EQU 295
SemaphoreList EQU 532

Permit        EQU -138
Enqueue       EQU -270
InitSemaphore EQU -558
;Paramètre:
;A0 - Pointeur sur la structure de semaphore-signal

move.l ExecBase,a6
move.l a0,-(a7)           ;sauver le pointeur sur le semaphore
jsr InitSemaphore(a6)     ;initialiser le semaphore
lea SemaphoreList(a6),a0  ;Pointeur sur SemaphoreList
move.l (a7)+,a1            ;Prendre le pointeur sur le semaphore
add.b #1,TDNestCnt(a6)    ;Forbid()
jsr Enqueue(a6)          ;Faire figure le semaphore dans la liste
jsr Permit(a6)           ;Permit()
rts                      ;Retour

```

RemSemaphore

Fonction RemSemaphore(SignalSemaphore)
A1

Offset -606 -\$25E \$FDA2

Description

La fonction correspond à Remove(), avec cette différence que les autres tâches sont bloquées avant l'appel de Remove, puis libérées. Pour le reste, cf. Remove().

Paramètre

A1 est le pointeur sur la structure de sémaphore-signal.

FindSemaphore

Fonction Semaphore = FindSemaphore(SignalSemaphore)
D0 A1

Offset -594 -\$252 \$FD9C

Description

Cette fonction recherche un sémaphore-signal à l'intérieur de la SemaphoreList (ExecBase Offset 532), à l'aide du nom indiqué.

Paramètre

SignalSemaphore

est un pointeur sur le nom de la structure de sémaphore-signal.

2.11.3. Utilisation des ports de sémaphores

Pour utiliser les avantages des ports de sémaphores signalés au début, par rapport aux sémaphores -signaux, il faut procéder à une préparation.

On devra créer un port de message (Reply-Port), au moyen duquel la tâche propre pourra être informée d'une autorisation d'accès. De plus, il faut créer un message, dont la mention dans le Reply-Port devra pointer sur le port de message dont il vient d'être question. Lorsqu'on attend un accès de plusieurs sémaphores, il peut être utile de ne pas faire usage de la structure de message, et d'étendre celle-ci avec une caractérisation, permettant de distinguer facilement les diverses autorisations d'accès.

L'initialisation d'un port de sémaphore personnel

Comme on peut le voir sur la structure présentée plus haut, la structure de sémaphore est constituée essentiellement d'un port de message. Celui-ci doit être initialisé de telle façon que son entrée mp_Flags se situe sur PA_IGNORE (2), puisque mp_SigTask ne pointe pas sur une tâche lorsque l'on utilise le sémaphore, mais sur le message actuel. sm_Bids dans la structure de sémaphore doit être sur -1.

Si le port de sémaphore est un port public, il est reporté avec la fonction AddPort() de Exec dans la liste des ports publics, et il faut pour cela lui donner auparavant un nom et une priorité.

Description des fonctions

Procure

Fonction `message = Procure(semaphore, replymessage)`
 D0 A0 A1

Offset -540 -\$21C

Description

Cette fonction examine si le port de sémaphore est occupé. S'il est libre, la fonction l'alloue, et transmet un message de retour positif. S'il n'est pas libre, le message transmis est placé dans la liste des messages en attente, et c'est une réponse négative qui est transmise. Lorsque, par la suite, le port de sémaphore se libère, le message transmis auparavant à la fonction Procure est envoyé au port de message qui figure dans la fonction, et le port de sémaphore est à nouveau occupé.

Paramètres

semaphore

Pointeur sur la structure de sémaphore qui doit être occupée.

replymessage

C'est le message personnel, au moyen duquel la tâche personnelle est informée du fait que le port de sémaphore a été libéré.

Résultat

message

Après l'appel de la fonction, une valeur de vérité est transmise, permettant de savoir si le port de sémaphore était libre ou non. Si la valeur est 1 (TRUE), on n'a pas besoin d'attendre la libération du port. Si la valeur est 0 (FALSE), le port était occupé, et l'on est obligé d'attendre qu'il soit libéré.

Vacate

Fonction Vacate(semaphore)
A0

Offset -546 -\$222

Description

Cette fonction libère un port de sémaphore préalablement alloué par Procure(), et fait savoir à la tâche suivante que le port est libéré.

2.11.4. Exemple d'utilisation des sémaphores-signaux

Cet exemple comprend deux programmes. Le premier génère une structure, dont l'en-tête est fait d'un sémaphore-signal. Cette structure est reportée dans la liste des sémaphores publics. Puisque nous devons assurer une communication minimale entre les deux, pour quitter correctement les programmes, on déposera de plus dans la structure créée le pointeur sur la tâche personnelle, ainsi que le masque pour un signal de tâche alloué précédemment.

Une fois que la structure dont l'en-tête constitue un sémaphore-signal a été insérée dans la liste globale, la tâche passe à "wait", jusqu'à ce qu'elle reçoive le signal précédemment occupé. Ensuite, le sémaphore est retiré de la liste globale, et l'emplacement qu'il occupait en mémoire est libéré.

Le second programme, qui doit être lancé comme processus personnel (à partir d'un nouveauCLI), cherche le sémaphore-signal créé par le premier programme dans la liste des sémaphores globaux. Lorsqu'il l'a trouvé, il l'alloue, et il lit le pointeur sur la tâche de création, qui figure dans la structure; il lit également le masque de signal. Ensuite, il libère le sémaphore. La dernière étape consiste en l'envoi du signal à la tâche qui est en attente, pour que celle-ci puisse retirer les sémaphores de la mémoire.

Voici le programme pour créer un sémaphore-signal :

```
include "exec/types.i"
include "exec/memory.i"
include "exec/execbase.i"
include "exec/semaphores.i"
include "exec/ables.i"
include "mymacro.i" (Cf. la description des macros)

TASK_ABLES Macro de "exec/ables.i"

;-----
:Exec
XLIB AllocMem
XLIB FreeMem
XLIB InitSemaphore
XLIB RemSemaphore
XLIB Enqueue
XLIB FindTask
```

```

XLIB AllocSignal
XLIB Wait
;-----

XREF _AbsExecBase

* Structure destinée à déposer des variables globales

STRUCTURE global1,0
    APTR      gl_MySem
    LABEL     gl_size

* Structure à laquelle on doit accéder par l'intermédiaire de sémaphores signaux.

STRUCTURE MySemaphore,0
    STRUCT    ms_GlobalSem,SS_SIZE
        APTR    ms_Task
        LONG   ms_TaskSigMask
    LABEL    ms_size

*****



XDEF main

main:
    move.l  _AbsExecBase,a6
    move.l  #gl_size,d0
    move.l  #MEMF_CLEAR,d1
    CALLSYS AllocMem           ;alloquer une mémoire globale
    tst.l   d0
    beq    main_NoGlobalMem
    move.l  d0,a5               ;pointeur sur une structure globale

    move.l  #ms_size,d0          ;Longueur de la structure globale
    move.l  #MEMF_PUBLIC!MEMF_CLEAR,d1 ;Type de mémoire
    CALLSYS AllocMem           ;Allouer la mémoire
    move.l  d0,gl_MySem(a5)       ;conserver le pointeur sur la structure
    beq    main_NoSemaphoreMem  ;fin, pas de mémoire pour la structure
    move.l  gl_MySem(a5),a2       ;pointeur sur structure globale

    sub.l   a1,a1
    CALLSYS FindTask            ;chercher pointeur sur tâche personnelle

    move.l  d0,ms_Task(a2)
    move.l  #-1,d0               ;Signal quelconque
    CALLSYS AllocSignal          ;alloquer le signal
    tst.b   d0
    bmi    main_NoSignal         ;Tester le numéro
    clr.l   d1
    bset    d0,d1               ;fin, pas de signal libre
    move.l  d1,ms_TaskSigMask(a2) ;Numéro de signal-> masque signal
    move.l  d1,ms_TaskSigMask(a2) ;conserver le masque

    move.b  #NT_SIGNALSEM,LN_TYPE(a2) ;rapporter le type
    move.b  #0,LN_PRI(a2)           ;rapporter la priorité
    lea    NomStructure(pc),a0
    move.l  a0,LN_NAME(a2)          ;rapporter le nom
    move.l  a2,a0               ;pointeur sur structure (Semaphore)
    CALLSYS InitSemaphore        ;initialiser la structure

* Macro de "exec/ables.i"

FORBID
    lea    SemaphoreList(a6),a0      ;pointeur sur liste de sémaphores
    move.l  a2,a1               ;pointeur sur structure propre
                                    ;(Semaphore)

```

```

CALLSYS Enqueue

* Macro de "exec/ables.i"

PERMIT

move.l ms_TaskSigMask(a2),d0 ;Attendre signal d'une autre tâche
CALLSYS Wait

move.l gl_MySem(a5),a1 ;pointeur sur structure personnelle
;:(Semaphore)
CALLSYS RemSemaphore ;supprimer le sémaphore du système

main_NoSignal:
move.l gl_MySem(a5),a1 ;pointeur sur structure personnelle
move.l #ms_size,d0 ;Longueur de la structure
CALLSYS FreeMem ;libérer la mémoire

main_NoSemaphoreMem:
move.l a5,a1
move.l #gl_size,d0
CALLSYS FreeMem ;Libérer la mémoire globale
main_NoGlobalMem:
clr.l d0
rts
*****
NomStructure: dc.b 'TestStructureAvecSemaphore',0
END

```

Programme servant à allouer les sémaphores-signaux créés avec le programme ci-dessus.

```

include "exec/types.i"
include "exec/memory.i"
include "exec/semmaphores.i"
include "mymacro.i" (Cf. la description des macros)

;-----
;Exec
XLIB AllocMem
XLIB FreeMem
XLIB FindSemaphore
XLIB ObtainSemaphore
XLIB ReleaseSemaphore
XLIB Signal
;-----

XREF _AbsExecBase

STRUCTURE global,0
    APTR     gl_Semaphore
    LABEL    gl_size

* Structure à laquelle on doit accéder par l'intermédiaire de sémaphores-signaux.

STRUCTURE MySemaphore,0
    STRUCT   ms_GlobalSem,SS_SIZE
    APTR    ms_MyTask
    LONG    ms_TaskSigMask
    LABEL   ms_size
;-----

XDEF main

```

```

main:
    move.l  _AbsExecBase,a6
    move.l  #g1_size,d0
    move.l  #MEMF_PUBLIC!MEMF_CLEAR,d1
    CALLSYS AllocMem
    tst.l   d0
    beq    main_NoGlobalMem
    move.l  d0,a5                      ;pointeur sur structure globale

    lea     StrukturName(pc),a1
    CALLSYS FindSemaphore               ;recherche du sémaphore
    move.l  d0,g1_Semaphore(a5)
    beq    main_NoSemaphore            ;fin, sémaphore non trouvé
    move.l  g1_Semaphore(a5),a0
    CALLSYS ObtainSemaphore           ;allouer sémaphore

* On peut maintenant accéder sans restriction à la structure

    move.l  g1_Semaphore(a5),a0
    move.l  ms_MyTask(a0),a2
    move.l  ms_TaskSigMask(a0),d2

    move.l  g1_Semaphore(a5),a0
    CALLSYS ReleaseSemaphore          ;Libérer le sémaphore

    move.l  a2,a1
    move.l  d2,d0
    CALLSYS Signal                   ;Envoyer signal à la première tâche

main_NoSemaphore:
    move.l  a5,a1
    move.l  #g1_size,d0
    CALLSYS FreeMem                 ;Libérer la mémoire pour la structure
                                    ;globale

main_NoGlobalMem:
    clr.l   d0
    rts

*****
NomStructure: dc.b 'TestStructureAvecSemaphore',0
END

```

2.11.5. Exemple d'utilisation des ports de sémaphores

Comme dans l'exemple précédent, nous avons besoin de deux programmes. Le premier initialise une structure de sémaphore, et la place dans la liste des ports globaux. De plus, il dépose dans la structure le pointeur sur la tâche personnelle, ainsi que le masque pour le bit-signal alloué. Avant de quitter la tâche, il faut lui faire savoir que le sémaphore n'est plus utilisé. On retire ensuite le sémaphore du système.

Le second programme cherche le sémaphore au moyen de son nom dans la liste des ports globaux. Le sémaphore est ensuite alloué. Si le sémaphore n'a pas pu être alloué, on attend jusqu'à ce que ce soit possible. Ensuite, le programme lit l'adresse de la tâche génératrice ainsi que le masque de signal. Le sémaphore est libéré, et le programme fait savoir à la tâche génératrice que la libération s'est passée correctement, et donc que le sémaphore ne figure plus dans le système.

Programme pour la création d'un port de sémaphore :

```

include "exec/types.i"
include "exec/memory.i"
include "exec/semaphores.i"
include "exec/ports.i"
include "mymacro.i" (Cf. la description des macros)

;-----
:Exec
XLIB AllocMem
XLIB FreeMem
XLIB FindTask
XLIB AllocSignal
XLIB Wait
XLIB AddPort
XLIB RemPort
;-----

XREF _AbsExecBase

STRUCTURE global,0
    APTR      gl_SemPort
    LABEL     gl_size

STRUCTURE MySemPort,SM_SIZE
    APTR      msp_Task
    LONG     msp_SignalMask
    LABEL     msp_size

;*****



XDEF main

main:
    move.l   _AbsExecBase,a6
    move.l   #gl_size,d0
    move.l   #MEMF_CLEAR,d1
    CALLSYS AllocMem
    tst.l    d0
    beq     main_NoGlobalMem
    move.l   d0,a5           ;pointeur sur structure globale

    bsr     InitSemaPort
    move.l   d0,gl_SemPort(a5) ;pointeur sur port créé
    beq     main_NoSemPort   ;fin, pas de port créé
    move.l   d0,a2

    suba.l  a1,a1
    CALLSYS FindTask
    move.l   d0,msp_Task(a2) ;reporter la tâche
    move.l   #-1,d0
    CALLSYS AllocSignal
    tst.b   d0
    bmi     main_NoSignal
    clr.l   d1
    bset    d0,d1
    move.l   d1,msp_SignalMask(a2) ;reporter le masque signal

    move.l   a2,a1
    CALLSYS AddPort           ;reporter le sémaphore dans la liste

    move.l   d1,d0
    CALLSYS Wait                ;Attendre un message pour mettre fin

main_NoSignal:
    move.l   gl_SemPort(a5),a0
    bsr     FreeSemaPort        ;Supprimer sémaphore

```

```

main_NoSemPort:
    move.l  a5,a1
    move.l  #gl_size,d0
    CALLSYS FreeMem           ;libérer mémoire globale

main_NoGlobalMem:
    clr.l  d0
    rts
;*****;Création d'une structure de port de sémaphore

;=> D0 = pointeur sur la structure créée ou 0 => pas de mémoire

InitSemaPort:
    move.l  #msp_size,d0
    move.l  #MEMF_PUBLIC!MEMF_CLEAR,d1
    CALLSYS AllocMem          ;allouer de la mémoire pour la structure
    tst.l  d0
    beq   1$                  ;fin, mémoire non allouée
    move.l  d0,a0
    move.b  #NT_SEMAPHORE,LN_TYPE(a0);reporter le type
    move.b  #0,LN_PRI(a0)      ;reporter la priorité
    lea    MySemaName(pc),a1
    move.l  a1,LN_NAME(a0)     ;reporter le nom
    move.b  #PA_IGNORE_MP_FLAGS(a0) ;reporter les flags
    lea    MP_MSGLIST(a0),a1    ;pointeur sur liste
    NEWLIST A1                ;créer une liste
                                ;(Macro de "exec/lists.i")
    move.w  #-1,SM_BIDS(a0)   ;nombre de lignes = -1
                                ;(pas d'entrée)
    move.l  a0,d0              ;message en retour
1$:   rts
;*****;Supprimer les sémaphores globaux

;=> A0 = pointeur sur Sémaphore

FreeSemaPort:
    movem.l a2,-(a7)
    move.l  a0,a2              ;pointeur sur Sémaphore
    move.l  a2,a1
    CALLSYS RemPort            ;pointeur sur Sémaphore
    move.l  a2,a1
    move.l  #msp_size,d0       ;Longueur
    CALLSYS FreeMem            ;Libérer la mémoire
    movem.l (a7)+,a2
    rts
;*****;MySemaName: dc.b 'Semaphore-Port_test',0
;END

```

Programme pour allouer et libérer un port de sémaphore créé avec le programme ci-dessus :

```

include "exec/types.i"
include "exec/memory.i"
include "exec/semaphores.i"
include "mymacro.i" (Cf. la description des macros)

;-----
;Exec
XLIB AllocMem
XLIB FreeMem

```

```

XLIB FindPort
XLIB WaitPort
XLIB GetMsg
XLIB Signal
XLIB Procure
XLIB Vacate
;-----
XREF _CreatePort
XREF _DeletePort
XREF _CreateIO
XREF _DeleteIO
XREF _AbsExecBase

STRUCTURE global1.0
    APTR    gl_ReplyPort
    APTR    gl_ReplyMsg
    APTR    gl_Semaphore
    LABEL   gl_size

STRUCTURE MySemPort,SM_SIZE
    APTR    msp_Task
    LONG   msp_SignalMask
    LABEL   msp_size

;*****
;XDEF main

main:
move.l  _AbsExecBase,a6
move.l  #gl_size,d0
move.l  #MEMF_CLEAR,d1
CALLSYS AllocMem
tst.l  d0
beq    main_NoGlobalMem
move.l  d0,a5          ;pointeur sur structure globale

sub.l   a0,a0
clr.l   d0
jsr    _CreatePort      ;Créer ReplyPort
move.l  d0,gl_ReplyPort(a5) ;reporter pointeur sur Port
beq    main_NoPort
move.l  d0,a0          ;pointeur sur Port
move.l  #MN_SIZE,d0     ;Longueur de la structure
jsr    _CreateIO         ;Créer un message
move.l  d0,gl_ReplyMsg(a5)
beq    main_NoMessage

lea    MySemaName(pc),a1
CALLSYS FindPort        ;Chercher un port de sémaphore
move.l  d0,gl_Semaphore(a5) ;conserver le pointeur sur Séma
beq    main_NoSemaphore ;fin, séma non trouvé

move.l  d0,a0
move.l  gl_ReplyMsg(a5),a1

CALLSYS Procure          ;sémaphore alloué
tst.l  d0                ;sémaphore alloué?
bne    1$                ;Oui, poursuivre

2$:   move.l  gl_ReplyPort(a5),a0
CALLSYS WaitPort         ;Attendre que le séma soit alloué
move.l  gl_ReplyPort(a5),a0
CALLSYS GetMsg           ;Chercher message Reply
tst.l  d0                ;Message arrivé ?

```

```

beq      2$                      ;Non, attendre encore

1$:    move.l   gl_Semaphore(a5),a0
       move.l   msp_Task(a0),a2
       move.l   msp_SignalMask(a0),d2
                           ;Chercher valeurs pour la fonction signal

       move.l   gl_Semaphore(a5),a0
CALLSYS Vacate                  ;libérer sémaphore

       move.l   d2,d0
       move.l   a2,a1
CALLSYS Signal                 ;Envoyer signal pour la suppression
                               ;du sémaphore

main_NoSemaphore:
       move.l   gl_ReplyMsg(a5),a0
       jsr      _DeleteIO           ;Supprimer message

main_NoMessage:
       move.l   gl_ReplyPort(a5),a0
       jsr      _DeletePort         ;Supprimer Port

main_NoPort:
       move.l   a5,a1
       move.l   #gl_size,d0
CALLSYS FreeMem                ;libérer mémoire globale

main_NoGlobalMem:
       clr.l   d0
       rts

;*****
```

MySemaName: dc.b 'Semaphore-Port_test',0

END

2.12. La librairie RAM

Nous pouvons déjà dire que la librairie RAM ne peut pas utiliser des projets personnels. Elle est importante uniquement pour les besoins internes du système d'exploitation. Mais si vous voulez réellement connaître votre Amiga, vous devez également vous familiariser avec la signification de cette librairie particulière.

Certains parmi vous auront sans doute examiné de près les vecteurs de la librairie Exec, et auront remarqué que tous les offsets ne pointent pas vers la ROM, mais en partie vers le secteur de RAM de l'Amiga. Puisque la librairie Exec est entièrement déposée en ROM, et ne peut donc pas être chargée, les pointeurs sur la RAM sont à première vue très étonnantes.

Comme on le constatera facilement, le "séjour" dans le secteur de RAM n'est pas de très longue durée, car on revient ensuite assez rapidement dans la ROM.

Comme on l'aura sans doute deviné en lisant le titre de cette section, ces passages par la RAM sont en relation avec la librairie RAM.

Exec propose des fonctions pour ouvrir et fermer les librairies et les devices. Ces fonctions Exec se réfèrent uniquement aux librairies et devices déjà existants dans la RAM. Mais si l'on utilise les fonctions correspondantes, comme par exemple OpenLibrary, on voit

que ces fonctions permettent également d'ouvrir des librairies qui se trouvent dans le répertoire LIBS : de la disquette de bootage (par exemple la librairie Diskfont).

Lorsqu'on charge des librairies, il s'agit d'une extension de la fonction Exec élémentaire. De telles extensions sont implémentées à l'aide de la librairie RAM. La librairie RAM représente une interface entre les routines de base de Exec et les extensions.

Par l'intermédiaire de Exec, on passe d'abord dans la librairie RAM, puis on reçoit le pointeur sur la librairie RAM, et on passe dans l'extension, à partir de laquelle on passe dans la routine de base Exec grâce aux vecteurs de la librairie RAM.

La librairie RAM gère les fonctions EXEC suivantes :

Offset de la librairie RAM	Fonction Exec
-30, -\$1E, \$FFE2	AllocMem
36, -\$24, \$FFDC	OpenLibrary
42, -\$2A, \$FFD6	OpenDevice
48, -\$30, \$FFD0	CloseLibrary
54, -\$36, \$FFCA	CloseDevice
60, -\$3C, \$FFC4	RemLibrary
66, -\$42, \$FFBE	RemDevice

Extensions

AllocMem

Lorsque cette fonction constate qu'il ne reste pas assez de place en mémoire, l'extension tente de supprimer les librairies non utilisées, et examine à nouveau s'il y a assez de place en mémoire.

OpenLibrary

Outre les fonctions normales de OpenLibrary, cette extension examine le répertoire LIBS :, pour voir si la librairie souhaitée existe, et si elle est éventuellement chargée et ouverte.

OpenDevice

La fonction de la librairie RAM travaille comme la fonction OpenLibrary, avec cependant cette différence qu'il s'agit ici d'un device, et donc qu'il faut chercher dans le répertoire DEVS : pour les deux fonctions, c'est la même routine qui est utilisée, mais avec des paramètres différents.

CloseLibrary

Lorsqu'une librairie a été fragmentée en plusieurs segments au moment d'être chargée à partir d'un disque, ces segments sont libérés au moment où l'on supprime la librairie.

CloseDevice

La fonction correspond à *CloseLibrary*, mais se réfère aux devices chargés à partir du disque.

RemLibrary

La fonction travaille comme *CloseLibrary*. S'il existe plusieurs segments, ils sont tous libérés.

RemDevice

Comme *RemLibrary*, mais pour les devices.

2.13. La structure ExecBase

La structure ExecBase est la structure fondamentale de Exec, dans laquelle on retrouve tous les paramètres importants nécessaires, par exemple, à la gestion d'une tâche. L'adresse de base de la structure correspond à l'adresse de base de la librairie Exec et on peut facilement la calculer en C au moyen de la variable SysBase. Sysbase est une variable standard permettant de communiquer la position de la librairie Exec.

Pour accéder à cette structure en assembleur, il faut tout d'abord déterminer l'adresse de base qui se trouve à l'emplacement mémoire \$000004.

L'instruction

```
move.l $4,A6
```

permet de transmettre l'adresse de base de la librairie Exec ou de la structure ExecBase dans A6.

Comme cette structure est initialisée dès qu'une routine Reset est activée, son adresse de base est toujours la même. Il n'y a décalage que lorsque la taille ou la place de la mémoire RAM est modifiée. Après cette modification, sa position reste à nouveau constante.

Un Amiga comportant 512 Koctets de RAM possède une structure ExecBase à la position \$676 (jusqu'à Kick 1.3). La carte passerelle PC-XT décale la structure ExecBase à l'adresse \$C00276, qui n'est cependant à prendre en compte que lorsque l'extension de RAM commence à l'adresse \$C00000.

Les adresses absolues des éléments de la structure ne doivent en aucun cas servir à accéder de manière absolue à la structure (croyez-moi, surtout pas!). Elles servent simplement à faciliter la recherche des erreurs dans les programmes personnels.

La structure est de la forme suivante :

```

322 $142 $7B8 $C003B8 struct List MemList;
336 $150 $7C6 $C003C6 struct List ResourceList;
350 $15E $7D4 $C003D4 struct List DeviceList;
364 $16C $7E2 $C003E2 struct List IntrList;
378 $17A $7F0 $C003F0 struct List Liblist;
392 $188 $7FE $C003FE struct List Portlist;
406 $196 $80C $C0040C struct List TaskReady;
420 $1A4 $81A $C0041A struct List TaskWait;
434 $1B2 $828 $C00428 struct SoftIntList
    SoftInts[5];
514 $202 $878 $C00478 LONG LastAlert[4];
530 $212 $888 $C00488 UBYTE VBlankFrequency;
531 $213 $889 $C00489 UBYTE PowerSupplyFrequency;
532 $214 $88A $C0048A struct List SemaphoreList;
546 $222 $898 $C00498 APTR KickMemPtr;
550 $226 $89C $C0049C APTR KickTagPtr;
554 $22A $8A0 $C004AO APTR KickCheckSum;
558 $22E $8A4 $C004A4 UBYTE ExecBaseReserved[10];
568 $22F $8AE $C004AE UBYTE ExecBaseNew-
    Reserved[20];
};

#define SYSBASESIZE ((long)sizeof(struct ExecBase))

#define AFB_68010 0L
#define AFB_68020 1L
#define AFB_68881 4L
#define AFF_68010 (1L<<0)
#define AFF_68020 (1L<<1)
#define AFF_68881 (1L<<4)
#define AFB_RESERVED8 8L
#define AFB_RESERVED9 9L

```

LibNode - Offset 0

Structure de base de toute librairie, à partir du fichier Include "exec/libraries.h" ou en assembleur "exec/libraries.i".

SoftVer - Offset 34

Numéro de version du Kickstart.

LowMemChkSum - Offset 36

Peut être utilisé par le programmeur pour comparer la somme de contrôle qui sera calculée dans la zone comprise entre l'offset 34 et l'offset 78 dans le cas d'une insertion de vecteur. La façon dont cette somme est calculée peut être observée à l'offset 82 (ChkSum).

ChkBase - Offset 38

Sera utilisé pour tester la position de ExecBase lors d'un Reset. La position de ExecBase sera additionnée avec ChkBase, le résultat devant atteindre \$FFFFFF. Si ce n'est pas le cas, c'est que l'on se trouve en présence d'une erreur importante et il est préférable de réinstaller la structure ExecBase; sinon la structure ne sera pas réinitialisée entièrement, le temps étant ainsi économisé.

ColdCapture - Offset 42

Vecteur qui pourra être utilisé par le programmeur lors d'un Reset pour se brancher sur une routine personnelle. Un vecteur non utilisé sera mis à 0. La routine Reset reconnaîtra si ce vecteur a été défini, et si le passage à la routine donnée a été effectué. L'adresse de retour sera déposée dans A5. Avant le branchement à la sous-routine indiquée par le vecteur, ce dernier sera automatiquement mis à 0. Jusqu'à ce moment-là, il ne s'est encore rien passé de remarquable, à part le blocage des interruptions et de l'accès DAM, ainsi que l'adressage d'une carte Eprom. La routine personnelle adressée ne doit comporter aucune opération qui travaille avec la pile étant donné que celle-ci n'est pas encore correctement initialisée. C'est aussi pourquoi le retour de sous-routine se fait en A5 et pourquoi la routine n'est pas appelée avec une instruction JSR.

CoolCapture - Offset 46

Peut aussi être utilisé pour passer à une routine personnelle lors d'un Reset. La différence entre ColdCapture et CoolCapture réside dans le fait que cette dernière n'appelle la routine personnelle qu'un peu plus tard. CoolCapture n'est pas rétabli par une routine de reset. Etant donné que la pile, la mémoire, la table des exceptions et la librairie Exec sont ici déjà initialisées, ce vecteur se prête mieux à la plupart des utilisations que le vecteur ColdCapture. Pour sortir de la routine personnelle, on pourra utiliser l'instruction RTS.

WarmCapture - Offset 50

Vecteur de reset, que l'on utilise seulement lorsqu'il s'est produit une erreur lors de l'initialisation par DOS. Il ne se produit généralement pas d'erreur de ce genre; c'est pourquoi ce vecteur n'intervient pas dans l'usage quotidien.

ysStkUpper - Offset 54

Indique la limite supérieure de la pile Superviseur.

SysStkLower - Offset 58

Indique la limite inférieure de la pile Superviseur.

MaxLocMem - Offset 62

Indique la zone maximum Chip-Memory accessible (512 Koctets ou \$80000 octets).

DebugEntry - Offset 66

Pointeur sur la sous-routine du débogueur de l'AMIGA.

DebugData - Offset 70

Pointeur sur le tampon de données du débogueur (0).

AlertData - Offset 74

Pointeur sur le segment de données lors d'une alarme.

MaxExtMem - Offset 78

Donne la limite supérieure de la mémoire de données disponible.

ChkSum - Offset 82

Résultat de la somme de contrôle sur la zone comprise entre l'offset 34 et l'offset 78. La somme est calculée avant le branchement à ColdCapture. Quand des vecteurs particuliers sont reliés à cette zone, la somme de contrôle doit être recalculée ou validée dans LowChkSum. La somme de contrôle se calcule comme suit :

```

    lea      34(A6),A0      ;Pointeur sur début
    move.w #\$0016,00      ;Nombre de mot -1 dans compteur
    lopp: add.w  (A0)+,D1  ;Mots additionnés
    dbf     D0,loop        ;Compteur décrémenté,
    not.w   D1            ;Négation et
    move.w D1,82(A6)      ;conserver dans ChkSum

```

IntVects[0] - Offset 84

Interruption lors de la sortie sérielle.

IntVects[1] - Offset 96

Interruption lors de la fin du transfert des blocs disquettes.

IntVects[2] - Offset 108

Soft-Interrupt : pour une plus ample description, se reporter à la section sur les interruptions.

IntVects[3] - Offset 120

Interruption CIAA.

IntVects[4] - Offset 132

Interruption Copper.

IntVects[5] - Offset 144

Cette interruption est déclenchée quand le faisceau d'électrons du Raster passe à la ligne Raster 0.

IntVects[6] - Offset 156

Cette interruption est déclenchée lorsque le Blitter a fini son travail (handler d'interruption).

IntVects[7] - Offset 168

Canal audio 0 (handler d'interruption).

IntVects[8] - Offset 180

Canal audio 1 (handler d'interruption).

IntVects[9] - Offset 192

Canal audio 2 (handler d'interruption).

IntVects[10] - Offset 204

Canal audio 3 (handler d'interruption).

IntVects[11] - Offset 216

Cette interruption est déclenchée lors d'une entrée sérielle.

IntVects[12] - Offset 228

Cette interruption est déclenchée par la synchronisation de la disquette (handler d'interruption).

IntVects[13] - Offset 240

Cette interruption est déclenchée en même temps qu'une interruption du CIA-B (Interrupt-Server).

IntVects[14] - Offset 252

Cette interruption ne pourra être déclenchée que par le Software (non initialisée après un Reset).

IntVects[15] - Offset 264

Interruption qu'il n'est pas possible de masquer. Cette interruption n'est pas utilisée, mais elle est initialisée malgré tout comme serveur d'interruption.

***ThisTask - Offset 276**

Pointeur sur une structure Task en cours de traitement. On ne peut pas, avec un programme qui se déroule dans une tâche, lire le pointeur de cette structure Task et obtenir des valeurs cohérentes, puisque l'on reçoit toujours le pointeur sur la structure de tâche personnelle. La seule possibilité qui reste d'obtenir des valeurs cohérentes est de lire ces valeurs à partir d'une interruption. Ce rôle peut être joué par l'interruption 5 (faisceau du Raster).

idleCount - Offset 280

DispCount - Offset 284

Quantum - Offset 288

Elapsed - Offset 290

SysFlags - Offset 292

Dans ce flag, on trouve les bits importants de gestion du système.

Exemple : bit 5 à 0 correspond à une interruption soft non autorisée. Le même bit mis à 1 correspond à une interruption soft autorisée.

IDNestCnt - Offset 294

Indique si les interruptions sont autorisées. Si IDNestCnt se trouve à la valeur \$FF (-1), elles seront bloquées avec la fonction Disable. A chaque fois que cette fonction est appelée, IDNestCnt est incrémenté de 1. Avec la fonction Enable, IDNestCnt est à nouveau décrémenté. Lorsque IDNestCnt se trouve sur -1, les interruptions deviennent à nouveau possibles (position du bit Master).

TDNestCnt - Offset 295

Indique si la fonction Forbid a été appelée. Si c'est le cas, TDNestCnt est incrémenté de 1. La commutation vers une autre tâche est autorisée lorsque TDNestCnt ne renferme pas la valeur -1. Par la fonction Permit, TDNestCnt est à nouveau décrémenté afin de permettre l'exécution d'une autre tâche à condition que TDNestCnt se trouve à nouveau sur -1.

AttnFlags - Offset 296

Indique le type de processeur connecté:

```
#define AFF_68010 (1L<<0)
#define AFF_68020 (1L<<1)
#define AFF_68881 (1L<<4)
```

AttnResched - Offset 298

Resmodules - Offset 300

Pointeurs sur des modules résidents. Ces modules sont appellés par un reset.

TaskTrapCode - Offset 304

Vecteur prévu pour un trap de tâche.

TaskExceptCode - Offset 308

Vecteur prévu pour une exception de tâche.

TaskExitCode - Offset 312

Vecteur prévu lorsqu'une tâche prend fin.

TaskSigAlloc - Offset 316

Masque pour les bits-signaux alloués aux tâches.

TaskTrapAlloc - Offset 320

Masque pour les bits de trap alloués par le système aux tâches.

MemList - Offset 322

Pointeur sur la liste en mémoire qui caractérise les zones occupées et les zones libres.

ResourceList - Offset 336

Structure List dans laquelle sont enchaînées les structures Resource.

DeviceList - Offset 350

Structure List dans laquelle sont enchaînées les structures Device.

IntList - Offset 364

Structure List, dans laquelle peuvent être enchaînées des interruptions globales.

LibList - Offset 378

Structure List dans laquelle sont enchaînées les structures Library.

PortList - Offset 392

Structure List dans laquelle sont enchaînées les structures Port.

TaskReady - Offset 406

Structure List dans laquelle sont enchaînées les structures Task si elles se trouvent actuellement en mode Ready.

TaskWait - Offset 420

Structure List dans laquelle sont enchaînées les structures Task lorsqu'elles se trouvent en mode Wait.

SoftInts[0] - Offset 434

Structure List dans laquelle sont enchaînées les Soft-Interrupts de priorité -32 qui attendent d'être traitées.

SoftInts[1] - Offset 450

Structure List dans laquelle sont enchaînées les Soft-Interrupts de priorité -16 qui attendent d'être traitées.

SoftInts[2] - Offset 466

Structure List dans laquelle sont enchaînées les Soft-Interrupts de priorité 0 qui attendent d'être traitées.

SoftInts[3] - Offset 482

Structure List dans laquelle sont enchaînées les Soft-Interrupts de priorité 16 qui attendent d'être traitées.

SoftInts[4] - Offset 498

Structure List dans laquelle sont enchaînées les Soft-Interrupts de priorité 32 qui attendent d'être traitées.

LastAlert[4] - Offset 514

C'est ici que seront stockées les données lors d'une alerte après avoir été prélevées par la routine de reset.

VBlankFrequency - Offset 530

Indique avec quelle fréquence le faisceau Raster affiche une image (50 Hertz).

PowerSupplyFrequency - Offset 531

Indique la fréquence de la tension d'alimentation de l'Amiga (50 Hertz).

SemaphoreList - Offset 532

Structure List dans laquelle sont enchaînées toutes les structures Semaphore à condition qu'elles soient utilisées.

KickMemPtr - Offset 546

Pointeur sur une structure MemList; cette mémoire est à nouveau occupée après un reset.

KickTagPtr - Offset 550

Pointeur sur une table résidente, qui est ensuite intégrée à la table résidente générale, à la création de celle-ci.

KickCheckSum - Offset 554

Somme de contrôle, calculée par la fonction SumKickData().

ExecBaseReserved[10] - Offset 558

Ceci correspond à 10 octets réservés pour la structure ExecBase afin de donner à Exec la possibilité d'y mettre en mémoire des valeurs quelconques (non utilisé).

ExecBaseNewReserved[20] - Offset 568

Ceci correspond à 20 octets réservés pour la structure ExecBase, afin qu'Exec ait la possibilité d'y mettre en mémoire des valeurs particulières (non utilisé).

2.14. Programmes résistants au reset et secteurs de données

Vous avez sans doute tous entendu parler du fait qu'avec l'Amiga, il est possible d'intégrer des programmes et des secteurs de données dans le système de manière résistante au reset; vous l'avez peut-être vu sous la forme d'un disque RAM résistant au reset, ou encore d'un virus. Dans cette section, nous allons expliciter les mécanismes qui rendent cela possible.

Introduction

Une intégration résistante au reset, pour les programmes et les blocs de mémoire, s'obtient toujours par l'intermédiaire de la structure ExecBase.

Rappel : La structure ExecBase est le secteur de données de la librairie Exec dont l'adresse est déposée dans le mot long qui commence à l'emplacement 4 en mémoire.

Pendant un reset, des sommes de contrôles sont constituées à partir de certains éléments à l'intérieur de la structure ExecBase, ainsi que des données personnelles, pour constater l'exactitude de ces éléments et de ces données. S'il apparaît que la somme de contrôle qui vient d'être constituée coïncide avec celle qui est déposée dans la structure, il s'ensuit un test, permettant de savoir si l'utilisateur désire exécuter des programmes personnels à l'intérieur du reset. Il est également possible de caractériser pendant le reset certains secteurs de mémoire comme étant déjà alloués, pour que le contenu des données ne soit pas détruit par des programmes chargés ultérieurement.

Important : Puisque la constitution de sommes de contrôle sur les données personnelles est réalisée relativement tôt à l'intérieur de la routine de reset, les programmes personnels et les données ne peuvent se trouver que dans la mémoire accessible à ce moment-là au système. Ces secteurs se réduisent à la RAM Chip, et à la RAM Fast de \$C00000 à \$D00000. Puisque l'allocation de la mémoire par l'intermédiaire de la fonction 'AllocMem' de Exec ne donne pas la possibilité d'obtenir uniquement le secteur qui se trouve entre ces limites, seule l'utilisation de la RAM Chip pour les

programmes résistants au reset permet d'être sûr que ceux-ci fonctionneront sur l'ordinateur.

Il existe deux mécanismes différents pour exécuter des programmes à l'intérieur d'un reset. Ces deux mécanismes se différencient par l'instant où le programme personnel est lancé, et par les possibilités mises à la disposition de l'utilisateur.

2.14.1. Intégration dans un reset par les vecteurs Capture

La possibilité la plus simple de traiter des programmes personnels à l'intérieur d'un reset est celle qui passe par les vecteurs 'ColdCapture', 'CoolCapture' et 'WarmCapture' de ExecBase. Malheureusement, les possibilités de ces vecteurs sont relativement limitées; c'est pourquoi ils sont moins souvent utilisés que les autres. Pour mieux comprendre ce qui se passe, voyons de plus près la partie indispensable de la structure ExecBase; elle se présente comme suit:

	Offsets
STRUCTURE ExecBase.LIB_SIZE	0
UWORD SoftVer	34
WORD LowMemChkSum	36
ULONG ChkBase	38
APTR ColdCapture	42
APTR CoolCapture	46
APTR WarmCapture	50
APTR SysStkUpper	54
APTR SysStkLower	58
ULONG MaxLocMem	62
APTR DebugEntry	66
APTR DebugData	70
APTR AlertData	74
APTR MaxExtMem	78
WORD ChkSum	82
.	
.	

Nous avons reproduit ici uniquement la structure en assembleur, car l'insertion de programmes personnels dans le reset s'effectue normalement en assembleur seulement.

Le vecteur ColCapture

Très peu de temps après le démarrage de la routine de reset, une somme de contrôle est constituée sur le secteur allant de SoftVer à ChkSum. Si la somme de contrôle se révèle exacte, il s'ensuit la vérification du fait que le vecteur ColdCapture est ou non utilisé. Toute valeur différente de 0 indique qu'il est utilisé.

Si c'est le cas, le vecteur est supprimé, et on repasse dans le programme, à partir de l'adresse conservée dans ColdCapture.

En ce point, il n'est pas encore possible d'accéder au système d'exploitation. Il n'existe pas non plus de pile, de sorte qu'on ne peut en fait passer à aucun sous-programme au

moyen de 'JSR' ou 'BSR'. C'est pourquoi il faut quitter le programme personnel grâce à un saut par l'intermédiaire du registre A5, défini par la routine de reset. A ce stade, il n'est pas possible de réserver de la place en mémoire pour le programme personnel, car la mémoire n'est pas encore gérée. Le programme ne peut ainsi être exécuté qu'une seule fois.

Vous voyez que l'on est très limité par l'utilisation de ce vecteur. Commodore a prévu ce vecteur pour des besoins internes; il n'est donc pas recommandé de l'utiliser. Il n'existe pratiquement pas d'application pratique, à cause des limitations que nous venons de mentionner. Il vaut mieux renoncer.

Le vecteur CoolCapture

L'allocation du vecteur CoolCapture ne se distingue pas en son principe de l'utilisation du vecteur précédent. CoolCapture est reporté dans la structure et la somme de contrôle n'est calculée qu'ensuite. Mais le moment où se produit le saut au programme personnel se place bien plus tard qu'avec ColdCapture. C'est pourquoi les possibilités sont plus grandes dans ce cas.

Contrairement à ce qui passe lors de l'utilisation du vecteur ColdCapture, CoolCapture n'est pas supprimé automatiquement de la structure ExecBase. Lorsqu'on passe dans le programme personnel par l'intermédiaire de ce vecteur, la mémoire est à nouveau organisée, la structure ExecBase est créée, ainsi que la librairie Exec, les interruptions sont installées, et les exceptions (traps) sont rétablies sur leurs valeurs correctes.

Puisque la mémoire est gérée à nouveau, on peut aussi travailler avec la pile. Il est également possible ici de protéger le programme personnel de reset par l'intermédiaire de la fonction 'AllocAbs' de Exec, contre l'accès par d'autres programmes. C'est pourquoi le vecteur n'est pas rétabli automatiquement par la routine de reset.

On quitte le programme personnel par l'intermédiaire d'un 'RTS'.

Le programme suivant a été écrit avec l'assembleur 'AS' de Aztec:

```
include "exec/types.i"
include "exec/memory.i"
include "exec/execbase.i"

;-----

CALLSYS MACRO
    IFGT NARG-1
        FAIL
    ENDC
    JSR    _LVO\1(A6)
ENDM

LINKSYS MACRO
    IFGT NARG-2
        FAIL
    ENDC
    MOVE.L A6,-(SP)
    MOVE.L \2,A6
    CALLSYS \1
    MOVE.L (SP)+,A6
ENDM
```

```

;-----.
; XREF _AbsExecBase
; XREF _LVOCopyMem
; XREF _LVOAllocMem
; XREF _LVOAllocAbs
;-----.

main:
    move.l   _AbsExecBase,a6
    lea      FinProgramme(pc),a0
    lea      ResetProgramme(pc).a1
    sub.l   a1,a0
    move.l   a0,d0          ;Déterminer le nombre d'octets
    move.l   d0,d2
    move.l   #MEMF_CHIP,d1
    CALLSYS AllocMem           ;Allouer la mémoire pour le programme
    tst.l   d0
    beg     14                ;fin, pas de mémoire libre
    move.l   d0.a1
    move.l   a1.CoolCapture(a6) ;reporter le vecteur
    lea      ResetProgramme(pc).a0
    move.l   a0,d0          ;Source
    move.l   d2,d0          ;Nombre d'octets
    CALLSYS CopyMem            ;Copier le programme en mémoire

;* Constituer la somme de contrôle

    clr.w   ChkSum(a6)        ;effacer l'ancienne somme
    lea      SoftVer(a6),a0
    moveq   #(ChkSum-SoftVer)/2,d0 ;nombre = 24 Bytes
    clr.w   d1
2$:    add.w   (a0)+,d1
    dbf     d0,24              ;Constituer la somme
    not.w   d1
    move.w   d1,ChkSum(a6)      ;reporter la nouvelle somme
1$:    clr.l   d0
    rts

;-----.
;Programme de reset personnel

;=> A5 = pointeur sur la suite du Reset

ResetProgramme:
    lea      ResetProgramme(pc).a1
    lea      FinProgramme(pc),a0
    sub.l   a1,a0          ;définir le nombre d'octets
    move.l   a0,d0
    LINKSYS AllocAbs,_AbsExecBase ;libérer la mémoire
    move.l   #$20000,d0
1$:    move.w   d0,$dff180
    subq.l   #1,d0 Boucle couleur
    bne     1$
    rts

;-----.
FinProgramme:
END

```

Ce petit programme alloue de la mémoire, pour y copier la petite routine de reset personnelle. L'indication de la longueur du secteur en mémoire ne peut pas être réalisée de manière plus élégante, à cause d'une erreur dans 'AS'. Une fois que les données ont été copiées, la somme de contrôle est copiée et reportée par l'intermédiaire des vecteurs.

Après un reset, la mémoire utilisée par le programme est allouée à nouveau, et l'écran clignote un court instant en couleurs. Il faut noter ici aussi que ce programme ne fonctionne sur tous les appareils que s'il se trouve en RAM Chip. Vous voyez vous-même

que l'utilité pratique du programme est très faible. Il sert uniquement à montrer comment on intègre un programme personnel dans le reset.

Le vecteur WarmCapture

Le vecteur WarmCapture n'est utilisable qu'en théorie. On en fait usage pour passer à un programme lorsqu'il s'est produit une erreur dans la structure de la librairie DOS, obligeant à quitter cette routine. Dans le cas habituel, la librairie DOS est d'abord mise en place, et on attend ensuite des entrées dans le CLI avec la même routine. On ne quitte plus ensuite la routine en question.

2.14.2. Les modules résidents

Le second moyen pour intégrer des programmes résistants au reset utilise la liste Resident. C'est une liste dans laquelle figurent des pointeurs sur les structures Resident. Ces structures permettent de lancer des programmes personnels pendant la phase d'initialisation.

A l'intérieur de la routine de reset, la ROM Kickstart est brièvement examinée à la recherche de structures Resident, avant l'examen du vecteur CoolCapture. Durant cette recherche, une table est créée, dans laquelle chaque entrée est un pointeur sur une telle structure. Les entrées sont triées selon leur priorité, le premier pointeur figurant dans le tableau étant celui qui pointe sur la structure Resident dont la priorité est la plus élevée. Un pointeur sur ce tableau est déposé dans la structure ExecBase, dans ResModules (Offset 300 = \$12C). Après examen du vecteur CoolCapture, on passe à une routine qui appelle la fonction InitCode de Exec. Les entrées de la table Resident sont traitées grâce à cette fonction. Les différentes parties du système d'exploitation sont initialisées par l'intermédiaire du tableau ainsi décrit.

Nous pouvons faire en sorte que nos structures Resident propres soient insérées à l'endroit voulu, selon la priorité indiquée, et soient de cette façon traitées lors d'un reset. Vous voyez déjà que cette méthode vous permet de travailler de manière beaucoup plus souple. Le désavantage inévitable est que vous devez envisager l'intégration de vos propres programmes, ce qui représente une opération relativement complexe.

Voyons d'abord comme se présente la structure Resident; cela nous permettra de mieux comprendre les développements ultérieurs.

La structure en C se trouve dans "exec/resident.h":

```
struct Resident {
    UWORLD rt_MatchWord;
    struct Resident *rt_MatchTag;
    APTR rt_EndSkip;
    UBYTE rt_Flags;
    UBYTE rt_Version;
    UBYTE rt_Type;
    BYTE rt_Pri;
    char *rt_Name;
    char *rt_IdString;
```

```

        APTR  rt_Init;
};

#define RTC_MATCHWORD 0x4AFCL

#define RTF_AUTOINIT (1L<<7)
#define RTF_COLDSTART (1L<<0)
#define RTM_WHEN 3L
#define RTW_NEVER 0L
#define RTW_COLDSTART 1L

```

Voici maintenant la même structure en assembleur, prise dans le fichier "exec/resident.i":

```

"exec/resident.i"                                Offsets
STRUCTURE RT,0
    WORD  RT_MATCHWORD      00 $00
    PTR   RT_MATCHTAG       02 $02
    PTR   RT_ENDSKIP        06 $06
    BYTE  RT_FLAGS          10 $0A
    BYTE  RT_VERSION         11 $0B
    BYTE  RT_TYPE            12 $0C
    BYTE  RT_PRI             13 $0D
    PTR   RT_NAME            14 $0E
    PTR   RT_IDSTRING        18 $12
    PTR   RT_INIT             22 $16
    LABEL RT_SIZE            26 $1A

RTC_MATCHWORD EQU     $4AFC

BITDEF RT,COLDSTART,0
BITDEF RT,AUTOINIT,7

RTM_WHEN      EQU     1
RTW_NEVER     EQU     0
RTW_COLDSTART EQU     1

```

rt_MatchWord

est un mot qui permet de reconnaître la structure en mémoire. Lors d'un reset, c'est ce mot qui sera recherché et reconnu dans la ROM Kick. Ce mot doit avoir la valeur \$4AFC (RTC_MATCHWORD), pour que la structure puisse être retrouvée. Ce mot correspond à la commande Illegal en assembleur.

rt_MatchTag

est un pointeur sur la structure elle-même et est employé lui aussi pour la reconnaissance de la structure en mémoire. Une fois le MatchWord trouvé, un test est effectué pour voir si le mot long suivant est un pointeur sur la structure propre. Si c'est le cas, une structure résidente est reconnue.

rt_EndSkip

Pointeur sur la fin d'une structure ou du bloc complet en relation avec elle. Les autres structures sont recherchées à la fin de ce bloc.

rt_Flags

Indique comment doit être traitée la structure Resident. Nous aurons à reparler des diverses possibilités.

rt_Version

Indique la version de la structure.

rt_Type

Indique le type de traitement, à condition qu'il ait été spécifié dans les flags.

rt_Name

Pointeur sur le nom de la structure.

rt_IdString

Pointeur sur une chaîne de caractères qui donne des indications supplémentaires sur la structure.

rt_Init

Pointeur sur le programme personnel, ou sur un tableau traité par l'intermédiaire de la fonction InitResident.

Consacrons-nous d'abord aux possibilités dont nous disposons grâce à ce qui est mentionné dans *rt_Flags*.

Il permet d'abord d'indiquer si notre structure doit être traitée ou non. Si elle doit l'être, il faut poser le bit 0 dans *rt_Flags*. Il existe pour cela la variable prédéfinie 'RTF_COLDSTART'. On peut ensuite indiquer de quelle manière il faut utiliser l'entrée *rt_Init*. S'il n'y a aucune autre indication, *rt_init* est considéré comme pointeur sur le programme à exécuter. Si l'on a posé le bit 7 dans *rt_Flags*, *rt_Init* est considéré au contraire comme pointeur sur une table de mots longs, par l'intermédiaire de laquelle les registres sont définis en conséquence pour l'appel de la fonction MakeLibrary de Exec. Ce bit a été conçu pour permettre l'élaboration automatique d'une structure Device ou Resource à l'intérieur d'un reset. Pour ce bit aussi, il existe une variable spéciale. Elle s'appelle RTF_AUTOINIT.

La table servant à l'initialisation des registres lors de l'appel de la fonction MakeLibrary de Exec se présente de la façon suivante:

dc.l StructSize	-> Taille de la structure
dc.l VectorTab	-> Table contenant les fonctions de la librairie
dc.l DataTab	-> Table pour l'initialisation de la structure au moyen de la fonction InitStruct de Exec
dc.l InitRoutine	-> Pointeur sur une routine propre

Vous trouverez un exemple d'utilisation de ces possibilités avec la programmation d'une librairie personnelle, dans le chapitre portant sur ce sujet.

Si le flag 'RTF_AUTOINIT' est posé, rt_Type indique de quel type il s'agit pour l'initialisation de la structure. Selon le type, la nouvelle structure est reportée dans la liste adéquate.

Voici les types disponibles:

NT_DEVICE	- 3
NT_RESOURCE	- 8
NT_LIBRARY	- 9

Maintenant que la structure a été explicitée, il faut dire comment on fait savoir au système qu'il doit recevoir la structure personnelle dans la liste.

Il existe pour cela deux lignes dans la structure ExecBase:

	Offset
APTR KickTagPtr	550 \$226
APTR KickCheckSum	554 \$22A

KickTagPtr est un pointeur sur une liste. Dans cette liste, on trouve des pointeurs sur des structures Resident devant être insérées lors d'un reset. La fin de la liste est marquée par un zéro de la longueur d'un mot long. Cette liste contient encore une particularité, qui permet de l'étendre comme on veut.

Lorsque, dans la liste, on trouve une entrée dont le bit supérieur (bit 31) est posé, cette entrée n'est pas considérée comme un pointeur sur une structure Resident, mais comme un pointeur sur une autre liste de même forme.

Pour insérer une structure Resident dans la structure personnelle, il faut parcourir les étapes suivantes:

- ✓ Initialiser la structure Resident
- ✓ Créer une table longue de 2 mots longs (8 octets)
- ✓ Faire figurer l'adresse de la structure Resident personnelle dans le premier mot long de la table
- ✓ Supprimer le second mot long de la table (le remplacer par 0)
- ✓ Sauver l'entrée dans KickTapPtr à l'intérieur de la structure ExecBase
- ✓ Ecrire dans le KickTapPtr le pointeur sur la table ainsi créée

Si la ligne sauvée n'est pas nulle, le bit supérieur (bit 31) est posé, et cette valeur est reportée dans le second mot long de la table personnelle.

Une fois que ces opérations ont été effectuées, la structure Resident personnelle est reportée dans la liste prévue à cet effet. Pour que le système d'exploitation puisse savoir si les données contenues dans la table sont encore valides, on forme une somme de contrôle par l'intermédiaire des structures qu'on a fait figurer dans la liste. Cette somme est comparée avec celle qui est conservée dans KickChecksum. Si les deux sommes coïncident, alors la liste commençant dans KickTagPtr est intégrée dans la table avec les modules résidents dont le début se trouve en ResModules. Comme nous l'avons déjà dit, cette table est traitée à l'intérieur de la routine de reset.

Pour que les sommes de contrôle coïncident, on doit d'abord obtenir les sommes correctes, et les reporter dans KickChecksum. Pour réaliser cette tâche, Exec met à notre disposition une fonction qui s'appelle SumKickData, et dont nous parlerons un peu plus tard.

Nous venons de voir comment il est possible d'intégrer des programmes personnels dans la routine de reset. Le secteur de mémoire occupé par ces programmes est cependant rendu au système après un reset, et il doit pour cela être alloué à nouveau, pour qu'un autre programme ne puisse pas détruire le nôtre. Dans ce but, on pourrait songer à allouer ce secteur de mémoire de nouveau à l'intérieur du programme personnel, au moyen de la fonction AllocAbs de Exec. Cette méthode est cependant peu sûre, pour les raisons que nous allons exposer maintenant.

Selon la priorité de votre structure Resident, certaines structures seront traitées par le système d'exploitation avec cette structure Resident. Dans ce cas, vous êtes sûr qu'il y aura de la mémoire allouée, et que des valeurs y seront écrites. La mémoire allouée à votre structure ainsi qu'à votre programme est encore disponible à chacun en ce moment précis. Il est donc tout à fait possible que l'une des structure Resident traitées auparavant occupe la mémoire dans laquelle se trouve vos données. Pour cette raison, il existe une ligne supplémentaire dans la structure ExecBase.

```
Offset
APTR KickMemPtr 546 $222
```

Cette ligne constitue le début d'une liste de structures MemList (cf. la section sur la gestion de la mémoire). Par l'intermédiaire de cette liste, la mémoire indiquée est allouée avant le traitement de la liste ResModules, et avant le saut par l'intermédiaire du vecteur CoolCapture. Ces structures MemList sont cependant un peu différentes des structures habituellement utilisées. Pour ces structures, la mémoire occupée par elles est inscrite dans la structure elle-même. Il est clair que la mémoire occupée par une structure MemList de ce type doit également être protégée contre l'accès par d'autres programmes. Si l'on travaille normalement avec ces structures et avec les fonctions AllocEntry et FreeEntry de Exec, cette tâche est prise en charge par le système d'exploitation. Ici, il faut la réaliser "à la main".

Attention: Si vous voulez supprimer un programme qui est incrusté dans la mémoire sous forme résistante au reset, et donc si vous devez libérer la mémoire allouée au moyen de la structure MemList, ce n'est pas possible par l'intermédiaire de la fonction FreeEntry de Exec. Si vous essayez quand même de le faire, votre Amiga prendra congé avec un message Gourou.

Ce message vous fera savoir que vous avez voulu libérer un secteur de mémoire qui n'était pas occupé. Il s'agit ici d'une mémoire allouée par la structure MemList en personne.

Par l'intermédiaire de la mémoire gérée par la structure MemList, on forme de même une somme de contrôle à l'intérieur du reset. Si cette somme est correcte, le secteur de mémoire correspondant est alloué.

Attention: On ne peut allouer qu'une mémoire accessible au moment actuel. Il s'agit à nouveau - nous l'avions déjà dit - de la RAM Chip et du secteur allant de \$C00000 à \$D00000. Les extensions ne sont reconnues et reliées au système qu'après l'initialisation de la librairie Expansion.

Initialisation de la structure MemList

Pour obtenir la longueur de la structure MemList et allouer la mémoire correspondante:

```

move.w  #NbrBlocs,d0  Nombre de blocs de mémoire
        y compris ceux qui doivent être alloués maintenant
mulu   #ME_SIZE,d0  Emplacement en mémoire pour les structures Entry
add.i  #ML_SIZE,d0  Mémoire pour la structure MemList

move.l #MEMF_PUBLIC!MEMF_CLEAR!MEMF_CHIP,d1
CALLSYS AllocMem      Allouer la mémoire pour MemList

```

Vous trouverez la structure MemList dans le fichier Include "exec/memory.i" ou en C dans "exec/memory.h". Reportez ensuite les différents blocs de mémoire dans la structure. N'oubliez pas, ce faisant, le bloc alloué par la structure MemList elle-même. Le nombre de blocs qui figurent dans la liste doit être de même conservé en mémoire.

Sauvegardez ensuite la ligne en KickMemPtr et reportez-la en tant que premier mot long dans la structure MemList (pointeur sur le successeur). Il faut ensuite écrire le pointeur dirigé sur votre propre structure MemList dans KickMemPtr.

Evidemment, il est également possible de n'exécuter aucun programme pendant le reset, et de demander uniquement l'allocation de secteurs de mémoire déterminés.

Une fois que vous avez inséré votre structure Resident ainsi que la structure MemList dans les listes correspondantes, vous pouvez appeler la fonction SumKickData() de Exec. La somme calculée est conservée dans KickCheckSum.

KickSumData()

Fonction Somme = KickSumData();
DO

Offset -612 -\$264 \$FD9C

Description

La fonction calcule la somme de contrôle par l'intermédiaire de la structure MemList indiquée dans KickMemPtr et la table Resident indiquée dans KickTagPtr. Le résultat de ce calcul est retourné dans D0.

Pour savoir quelles devront être les priorités des structures Resident personnelles, pour être exécutées à l'endroit voulu, voici un tableau indiquant les structures Resident exécutées lors d'un reset, avec leurs priorités :

Pri.	Description
120	créer exec.library. (non autorisé)
110	créer exception.library.
100	créer lotgo.library
80	créer cia.resources
70	créer disk.resource
70	créer misc.resource
70	créer ram.library. (non autorisé)
65	créer graphics.library.
60	créer keyboard.device
60	créer gameport.device
50	créer timer.device
40	créer audio.device
40	créer input.device
31	créer layers.library.
20	créer console.device
20	créer trackdisk.device
10	créer intuition.library.
5	Afficher gourous s'ils existent
0	créer math.library
0	tâche workbench (non autorisé)
0	créer dos.library (non autorisé)
-20	Elaborer la librairie RomBoot (Kick 1.3)
-60	Sauter dans le processus de démarrage

Après la dernière structure Resident, on ne peut plus exécuter d'autre structure personnelle, puisqu'on ne revient plus à la fonction InitCode() après l'appel de cette structure. Une priorité inférieure à -60 n'a donc aucun sens.

Exemple de programme

Pour conclure ce chapitre, vous allez trouver un programme, dont la tâche est de prendre un programme complet quelconque et de le transformer en programme résistant au reset. On passe pour cela en revue la liste des segments, et l'on rend résistants au reset tous les segments qui appartiennent au fichier. Pour que la mémoire ne soit pas restituée au système lorsqu'on quitte le programme, on peut recourir à une astuce.

La liste des segments est constituée de telle façon que l'on trouve au début de chaque segment un pointeur BCPL dirigé sur le segment suivant. S'il n'y a pas de segment suivant, ce pointeur est nul. Après le pointeur sur le second segment (au début du premier), on trouve le programme proprement dit, ou les données. Devant le pointeur sur le segment suivant, on trouve le nombre d'octets à l'intérieur du segment, pour que la mémoire de ce segment puisse être libérée en conséquence. Pour empêcher que la mémoire pour la liste totale des segments (le programme que l'on vient de charger) soit libérée lorsqu'on quitte le programme, on supprime le pointeur sur le segment suivant. Ce pointeur se trouve 4 octets "en dessous" du début du programme. Mais comme on ne veut pas libérer non plus le premier segment, on place sur 0 le nombre des octets qui se trouvent dans le segment. De cette manière, on pourra quitter le programme, sans que la mémoire correspondante soit restituée au système.

Mais comme nous voulons en outre que la mémoire soit libérée pour la routine d'initialisation, mais pas le reste du programme, il faut avoir recours ici à une seconde astuce. Comme nous l'avons décrit précédemment, le pointeur sur le segment suivant est placé sur 0 (effacé). Le nombre d'octets indiqué pour ce segment est la longueur de la routine d'initialisation. De cette manière, seule la mémoire de la routine de départ est libérée lorsqu'on quitte le programme. Pour libérer le reste du programme si on le veut, nous avons créé nous-même un nouveau segment, où figure le pointeur sur le segment suivant, ainsi que sur les octets restants du premier segment.

Le programme est élaboré de façon à ce qu'on puisse le réécrire facilement, pour charger n'importe quel programme, et le rendre résistant au reset. Il faut noter ici encore que ces programmes doivent se trouver dans la RAM Chip ou dans le secteur de mémoire allant de \$C00000 à \$D00000.

Il n'est pas possible de rendre résistants au reset des programmes qui accèdent à la librairie Dos, puisque celle-ci n'est pas encore mise en place au moment du reset.

S'il se produisait une erreur dans la liste des structures Resident définies par l'utilisateur, aucune des lignes de la liste ne serait traitée.

Si plusieurs structures Resident portent le même nom, seulement l'une d'elles sera traitée.

Le programme a été écrit avec l'assembleur Aztec 'AS'.

```

include "exec/types.i"
include "exec/execbase.i"
include "exec/memory.i"
include "exec/resident.i"
include "mymacro.i" (Cf. la description des macros)

;*****  

STRUCTURE InitGlobals.0
    APTR      ig_MemList
    APTR      ig_ResDataStruct
    APTR      ig_Resident
    APTR      ig_Segment
    APTR      ig_NextSegment
    APTR      ig_SegSize
    LABEL     ig_size
;  
-----  

StringSizeMax      EQU 20  

STRUCTURE MyResident.RT_SIZE
    APTR      mr_ResListe
    APTR      mr_ListEnd
    STRUCT    mr_strings.StringSizeMax
    LABEL     mr_size
;  
-----  

BITDEF   MEM,RESET,MEMB_CHIP           ;Type de mémoire atteinte après reset  

ResetPri  EQU 0
;*****  

XREF _AbsExecBase  

;*****  

XLIB AllocMem
XLIB FreeMem
XLIB CopyMem
XLIB SumKickData
;*****  

XDEF InitResedent  

InitResedent:
    move.l  _AbsExecBase,a6
    move.l  #ig_size,d0
    move.l  #MEMF_PUBLIC!MEMF_CLEAR,d1
    CALLSYS AllocMem          ;Chercher mémoire pour structure globale
    tst.l   d0                ;mémoire affectée ?
    beq    Init_NoGlobalMem  ;Non, erreur => fin
    move.l  d0,a5              ;pointeur sur structure nach A5

    move.l  #mr_size,d0
    move.l  #MEMF_PUBLIC!MEMF_CLEAR!MEMF_RESET,d1
    CALLSYS AllocMem
    move.l  d0,ig_Resident(a5)  ;pointeur sur structure Resident
    beq    Init_NoResidentMem ;fin, pas de mémoire pour la structure
                                ;Resident
;  
-----  

    lea     InitResedent-4(pc),a0  ;pointeur sur début de la liste des
                                ;segments
    lea     ResetSegmentStart(pc),a1
    move.l  (a0).ig_NextSegment(a5) ;conserver pointeur sur segment suivant
    move.l  (a0).4(a1)            ;reporter pointeur sur segment suivant
    clr.l   (a0)                 ;effacer pointeur sur segment suivant

```

```

move.l -4(a0),d0           ;longueur de ce segment
move.l d0,ig_SegSize(a5)   ;conserver longueur
move.l #ResetSegmentStart-InitResedent+B,d1
sub.l d1,d0                ;mémoire pour le reste du segment sans
                            ;Init
move.l d0,(a1)              ;reporter longueur de mémoire pour le
                            ;reste
move.l d1,-4(a0)            ;reporter longueur de mémoire pour Init

;-----;

lea      ResetSegmentStart+4(pc),a0 pointeur sur le "nouveau" segment

;* A cet endroit, on peut transmettre un segment quelconque

move.l a0,ig_segment(a5)    ;reporter pointeur sur segment
move.w #1,d0                ;nombre des entrées supplémentaires
                            ;(ResDataStruct)
bsr      MakeMemList         ;Créer structure MemList
move.l d0,ig_MemList(a5)    ;conserver pointeur sur Mem-Entry
beq      Init_NoMemEntry     ;pas de mémoire libre -> fin
move.l ig_Resident(a5),ME_ADDR(a0);pointeur sur structure Resident propre
move.l #mr_size,ME_LENGTH(a0) ;longueur de la structure

move.l ig_Resident(a5),a0      ;pointeur sur mémoire pour structure
                            ;Resident
move.l ig_MemList(a5),a1       ;pointeur sur structure MemList
move.l ig_segment(a5),d0        ;pointeur sur segment
bsr      MakeResetfest        ;rendre résistant au reset la mémoire
                            ;et le programme
bra      Init_fin

Init_NoMemEntry:
    lea      InitResedent-B(pc),a0
    move.l ig_SegSize(a5),(a0)
    move.l ig_Nextsegment(a5),4(a0) ;restaurer les anciennes valeurs

    move.l ig_Resident(a5),a1
    move.l #mr_size,d0
    CALLSYS  FreeMem

Init_NoResidentMem:
    Init_fin:
        move.l #ig_size,d0
        move.l a5,a1
        CALLSYS  FreeMem          ;libérer mémoire globale

Init_NoGlobalMem:
    clr.l  d0
    rts

;*****Memory-List

;=> A0 = pointeur sur liste de segments
;=> D0 = nombre des autres entrées

;=> D0 = pointeur sur structure MemList ou 0 -> pas de mémoire
;=> A0 = place pour entrées personnelles

XDEF MakeMemList

MakeMemList:
    movem.l a2-a3/d2-d4,-(a7)
    move.l a0,a2                  ;sauver pointeur sur liste des segments
                                    ;retenu
    move.w d0,d2                  ;nombre des entrées supplémentaires
    bsr      ScanSegMemList       ;déterminer le nombre de segments
    move.w d0,d3                  ;conserver le nombre des segments

```

```

add.w    d0,d2          ;nombre des segments dans la liste +
add.w    #1,d2          ;supplément
move.w    d2,d0          ;nombre des entrées dans MemList+1
mulu    #ME_SIZE,d0      ;place en mémoire pour les structure
add.l    #ML_SIZE,d0      ;mémoire pour structure MemList
move.l    d0,d4          ;conserver nombre d'octets
move.l    #MEMF_PUBLIC!MEMF_CLEAR!MEMF_RESET,d1
CALLSYS AllocMem          ;mémoire allouée pour MemList
tst.l    d0
beq    14                 ;fin, pas de mémoire libre pour la structure
move.l    d0,a3          ;pointeur sur mémoire pour la structure
move.w    d2,ML_NUMENTRIES(a3)  ;nombre d'entrées
lea     ML_ME(a3),a0
move.l    d0,ME_ADDR(a0)    ;reporter pointeur sur mémoire propre
move.l    d4,ME_LENGTH(a0)   ;reporter longueur
addq.l    #ME_SIZE,a0      ;pointeur sur entrée suivante
bra    2$                  ;pointeur sur longueur pour segment
3$:   lea     -4(a2),a1
move.l    a1,ME_ADDR(a0)    ;reporter pointeur sur début du segment
move.l    (a1),ME_LENGTH(a0) ;conserver longueur
addq.l    #ME_SIZE,a0      ;pointeur sur entrée suivante
move.l    (a2),d0
lsl.l    #2,d0             ;BPTR -> APTR
move.l    d0,a2             ;segment suivant
2$:   dbf    d3,34
move.l    a3,d0             ;pointeur sur structure MemList
1$:   movem.l  (a7)+,a2-a3/d2-d4
rts
;*****déterminer le nombre des segments dans la liste
;>= AO = pointeur sur liste des segments
;=> D0.W = nombre de segments dans la liste

XDEF ScanSegMemList

ScanSegMemList:
    move.w    #1,d0          ;nombre minimal des segments
2$:   move.l    (a0),d1          ;pointeur sur segment suivant
    beq    1$                 ;fin de la liste atteinte
    lsl.l    #2,d1             ;BPTR -> APTR ( *4 )
    move.l    d1,a0
    addq.w    #1,d0          ;nombre des segments +1
    bra    2$
1$:   rts
;*****Rendre résistants au reset la mémoire et le programme
;>= AO = pointeur sur structure Resident

;>= A1 = pointeur sur structure MemoryList
;=> D0 = pointeur sur liste des segments

XDEF MakeResetfest

MakeResetfest:
    movem.l  a2-a4,-(a7)
    move.l    a0,a2             ;pointeur sur mémoire Resident
    move.l    a1,a3             ;pointeur sur structure MemList
    addq.l    #4,d0
    move.l    d0,a4             ;pointeur sur le programme
    move.l    a2,a1             ;pointeur sur mémoire Resident
    lea     Resident(pc),a0      ;pointeur sur source
    move.l    #RT_SIZE,d0

```

```

CALLSYS CopyMem           ;copier Resident
move.l a2,RT_MATCHTAG(a2) ;reporter pointeur sur structure propre
move.l a2,mr_ResListe(a2) ;pointeur sur début du programme
move.l a4,RT_INIT(a2)
lea    ResidentName(pc),a0
lea    mr_strings(a2),a1
move.l #$StringSizeMax-1,d0
CALLSYS CopyMem           ;décaler la chaîne à sa position
                           ;correcte
lea    mr_strings(a2),a0
move.l a0,RT_NAME(a2)     ;reporter pointeur sur le nom
clr.l RT_IDSTRING(a2)    ;pas de ID-String
lea    mr_size(a2),a0
move.l a0,RT_ENDSKIP(a2) ;indiquer fin de la structure
move.l KickTagPtr(a6),d0
beq   1$
bset  #31,d0
1$:   move.l d0,mr_ListEnd(a2) ;pointeur sur structure suivante
lea    mr_ResListe(a2),a0
move.l a0,KickTagPtr(a6)   ;reporter pointeur sur structure propre
move.l KickMemPtr(a6),(a3) ;reporter pointeur sur successeur dans
                           ;Memlist
move.l a3,KickMemPtr(a6)   ;reporter pointeur sur MemList

CALLSYS SumKickData       ;reporter la somme
move.l d0,KickCheckSum(a6)
movem.l (a7)+,a2-a4
rts

```

;*****
;structure Resident, pour rendre le programme résistant au reset

XDEF Resident

```

Resident:
dc.w   RTC_MATCHWORD      ;Code pour Resident
dc.l   0                  ;pointeur sur structure personnelle
dc.l   0                  ;pointeur sur fin du programme
dc.b   RTW_COLDSTART       ;exécuter le programme au moment du
                           ;reset
dc.b   0                  ;Version
dc.b   NT_UNKNOWN          ;pas de type déterminé
dc.b   ResetPri            ;priorité
dc.l   0                  ;pointeur sur le nom
dc.l   0                  ;pointeur sur ID-String

dc.l   0                  ;pointeur sur programme de reset

```

XDEF ResidentListe

```

ResidentListe:
dc.l   Resident,0

```

XDEF ResidentName

```

ResidentName:
dc.b  'Residentname',0
EVEN

```

```
;*****  
;*****  
;Emplacement pour créer une nouvelles liste des segments  
  
XDEF ResetSegmentStart  
  
ResetSegmentStart:  
  
dc.l    0  
dc.l    0  
  
;*****  
;*****  
XDEF ResetProgramme  
  
ResetProgramme:  
    NOP  
    rts  
;*****  
    END
```

3. L'AmigaDOS

Pour l'utilisateur, la partie la plus importante du système d'exploitation de l'Amiga est sans aucun doute possible le DOS, abréviation de "Disk Operating System". Sa tâche est de s'occuper de tous les problèmes d'entrée et de sortie, par exemple des opérations sur disquettes ou des saisies au clavier.

Le chemin emprunté par l'entrée/sortie est indiqué au DOS par l'intermédiaire du nom au moment de l'ouverture du canal : par exemple CON:, PRT: ou DF0:. Ces noms se trouvent dans une liste interne du DOS, la liste device/volume, et grâce à eux l'entrée/sortie est confiée au handler correspondant.

Outre les handlers standard, existant à demeure, on peut lier d'autres handlers à l'aide de la commande MOUNT, à condition qu'ils soient conservés dans la MountList avec leur nom et les paramètres nécessaires. Le handler reçoit ces paramètres qui lui sont communiqués par le DOS, il ouvre si nécessaire le device correspondant (par exemple PRT: = printer.device, SER: = serial.device, etc.), et il traite la fonction voulue.

Ce qui se trame à ce niveau n'intéresse pas nécessairement l'utilisateur et ne concerne pas le programme d'utilisation, car seuls le nom et la fonction sont indiqués. Dans le cadre de ce livre, les éléments internes sont cependant ceux que nous devons expliciter, et nous allons nous en occuper plus en détail.

3.1. Du CLI au hardware : la hiérarchie du DOS

Les tâches de DOS sont multiples. En effet, non seulement il doit fournir des informations à l'utilisateur, mais aussi mettre à la disposition du programme en cours d'exécution les ressources de l'ordinateur. Le fonctionnement Multi-tâches rend cette tâche particulièrement ardue. Nous commencerons malgré tout par le point de vue de l'utilisateur assis devant l'Amiga et se servant du DOS. Ce contact avec l'ordinateur passe par le CLI, si l'on n'utilise pas l'autre partie constitutive du système d'exploitation, à savoir Intuition.

3.1.1. Le premier contact : Le CLI

La nécessité d'un interpréteur de commandes alphanumérique n'est pas évident, lorsqu'on jette un œil sur le Workbench de l'Amiga. Mais on ne tarde pas à s'apercevoir qu'il s'agit d'un élément indispensable, si l'on veut travailler efficacement avec l'Amiga.

Lorsqu'on allume l'ordinateur, on voit d'abord s'ouvrir une fenêtre CLI, dans laquelle on voit s'afficher le message AmigaDOS. Le DOS s'annonce très tôt, bien avant que l'interface graphique ne devienne visible. La fenêtre CLI se présente au démarrage, tant que la commande EndCLI ne se trouve pas dans la séquence Startup. On peut alors commencer à se servir de l'Amiga, ou plutôt de l'AmigaDOS. Tout ce qui va se passer maintenant est fait d'opérations d'entrée/sortie, ce qui constitue justement la tâche du DOS. Depuis l'affichage des textes à l'écran, jusqu'à l'accès aux disquettes par exemple lorsqu'on charge un programme, et en passant par la gestion des entrées au clavier, toutes ces opérations sont exécutées à partir du DOS.

Les opérations que nous venons de mentionner sont d'ailleurs toutes identiques pour le DOS. Il s'agit encore et toujours de l'ouverture d'un canal d'entrée/sortie (écran, clavier, fichier sur disquette), et de la lecture ou de l'écriture dans ce canal. Ces opérations sont les fonctions les plus générales se trouvant dans la bibliothèque DOS.

3.1.2. La bibliothèque DOS

Comme c'était déjà le cas pour la bibliothèque Exec, la bibliothèque DOS (DOS.library) fait partie des bibliothèques intégrées de l'Amiga, contrairement par exemple à la bibliothèque Intuition qui se trouve, quant à elle, sur disquette. Lorsqu'on ouvre cette bibliothèque, elle n'est donc pas chargée à partir d'une disquette, mais copiée de la ROM dans la RAM. Vous trouverez la description exacte de cette bibliothèque plus loin, dans le chapitre de ce livre qui lui est spécialement consacré.

Les fonctions qui s'y trouvent servent en fait uniquement à l'utilisation des canaux d'entrée/sortie, que l'on différencie au moment de leur ouverture grâce à leurs noms. Ensuite, toutes les opérations sont conduites par l'intermédiaire du handler afférent.

3.1.3. Les handlers

Que se passe-t-il lorsqu'un utilisateur demande dans le CLI, au moyen d'une instruction, d'effectuer une sortie quelconque ? Prenons un exemple, et suivons le chemin que parcourt cette instruction dans les profondeurs de l'AmigaDOS. Commençons par une sortie sur un device normal :

```
1>echo "Hello" >ser:
```

Quand on a entré cette ligne sous le CLI, celui-ci la lit tout d'abord en entier. Puis, une double opération est déclenchée : la sortie standard est placée sur SER:, et le programme ECHO est lancé à partir du répertoire C. Celui-ci reçoit comme paramètre le reste de la ligne ou le pointeur qui est dirigé dessus. Enfin, il envoie simplement le texte indiqué "Hello" vers la sortie standard.

Ce n'est pas plus compliqué. Mais que se passe-t-il en détail lorsqu'on dérive une sortie sur SER: ?

Supposons qu'il n'y ait pas encore eu d'entrée/sortie par l'interface sérielle. Il faut donc commencer par ouvrir celle-ci. Voici comment on procède :

Les entrées et sorties de l'AmigaDOS sont réalisées par l'intermédiaire de ce qu'on appelle des handlers. Ceux-ci traitent les canaux d'entrée/sortie connus de l'utilisateur simplement par leurs noms, comme par exemple SER:, PAR:, DFx ou RAM:. Le DOS ne transmet à ces handlers, qui doivent se trouver en mémoire comme des processus propres, que les commandes qui sont nécessaires. Les handlers se chargent du reste. Les commandes de DOS au handler et les messages en retour sont transmis sous forme de paquets (en anglais : packets) DOS, qui se trouvent sur une structure de message. Nous reviendrons plus en détail ultérieurement sur la forme exacte des paquets.

Dans notre exemple, le DOS a donc besoin d'un handler. Il cherche dans sa liste interne des devices, pour voir quel est le handler qui correspond à la caractérisation SER:. S'il ne le trouve pas, il examine ensuite les entrées de la MountList ; si cette recherche s'avère également infructueuse, on voit apparaître le cher requester qui annonce "Please insert volume xxx in any drive".

Mais dans notre exemple, le DOS a déjà trouvé ce qu'il cherchait dans la liste interne, et il sait quel est le handler de port responsable du SER:, c'est-à-dire de l'interface sérielle. Ce handler se trouve sur la disquette SYS: à partir de laquelle l'ordinateur a été booté, dans le répertoire L. Il est chargé et lancé à partir de là.

Après initialisation, le handler reçoit la commande, selon laquelle il doit mettre un canal de sortie à la disposition de l'utilisateur. Ce qui a lieu sous la forme d'un paquet du type ACTION_FIND_OUTPUT. Le handler constate alors qu'il n'y a pas encore de canal disponible. Il charge donc le device appartenant à l'interface sérielle (serial.device), et il l'initialise. Si cette opération s'est déroulée correctement, le device le fait savoir au handler, qui remplit alors le champ de réponse du paquet DOS par un message OK, et le revoie au DOS. Le DOS transmet au processus appelant le statut de l'opération (donc le OK) qu'il a reçu à partir du paquet renvoyé.

Tout ce parcours peut sembler assez complexe, mais il permet de discerner une hiérarchie. Pour un programme qui doit accéder à l'interface sérielle par l'intermédiaire du DOS, il y a plusieurs étapes à franchir :

Software → Programme hardware → DOS → Handler → Device → Chip
I/O (CIA)

L'inconvénient de cette méthode par rapport à la méthode "normale", qui devrait consister en un accès direct aux ressources hardware de l'ordinateur, est évident : elle demande du temps. Mais les avantages sont tout aussi indéniables, par exemple le partage possible des différentes tâches dans plusieurs programmes partiels (handler, device et DOS), ainsi qu'une architecture extraordinairement ouverte de l'Amiga. Imaginez qu'il y ait un nouveau canal disponible, par installation d'un nouveau hardware : il suffit alors d'écrire un device convenable, ou un nouveau handler, et l'objet est déjà compatible !

Venons-en maintenant aux types de handlers qui ne se contentent pas de gérer un simple flux de données : les systèmes de fichiers (File-Systems).

3.1.4. Les systèmes de fichiers

Les file-systems, comme DFx ou RAM se distinguent des handlers normaux par le fait qu'ils font bien plus que gérer un flux séquentiel de données, puisqu'ils offrent également un accès direct aux données. A quoi vient s'ajouter encore la différence suivante : outre le nom du device comme DF0 :, on peut (et l'on doit) indiquer le nom de chemin ou le nom du fichier. Les handlers sont donc en fait un sous-ensemble des systèmes de fichiers.

Le processus d'évaluation des paquets DOS est d'abord le même qu'avec les handlers normaux, puisque la partie du nom qui suit le double point n'est pas pris en compte. Quand le device correspondant est cependant ouvert et initialisé, le nom de fichier est lui aussi envisagé.

Les deux file-systems existants de manière standard, DFx: et RAM: se distinguent l'un de l'autre sur un point important : le handler RAM: ne nécessite pas de device pour fonctionner, car il travaille directement avec la mémoire. Pour le handler DFx: en revanche, on a besoin du device Trackdisk pour établir la relation avec le hardware.

On voit ainsi que le handler RAM:, qui n'est pas installé dans la ROM, mais se trouve dans le répertoire L, est un exemple un peu inhabituel.

3.1.5. Les devices

Une fois que le handler a compris quel type d'entrée/sortie il doit traiter, il appelle le device qui en est responsable. Les devices sont donc les programmes qui peuvent accéder directement au hardware de l'Amiga. Le handler (ou le file-system) transmet le paquet reçu par le programme sous une forme appropriée au device. Lorsque le device a fait son travail, par exemple envoyé quelques caractères par l'intermédiaire de l'interface serielle, le handler le fait savoir en renvoyant le paquet au DOS.

3.2. La bibliothèque DOS

Les multiples fonctions disponibles au niveau supérieur du DOS sont accessibles à l'utilisateur sous la forme d'une bibliothèque semblable à la bibliothèque EXEC.

Comme cette dernière, la bibliothèque ne se trouve pas sur une disquette sous forme de fichier. Elle doit cependant être ouverte avant de pouvoir être utilisée, pour rendre possible l'accès au DOS. Grâce à ce processus, le programme qui ouvre la bibliothèque a accès aux fonctions du DOS par l'intermédiaire d'un tableau de pointeurs.

3.2.1. Chargement de Dos.Library

Lorsqu'un programme quelconque veut utiliser une fonction de la bibliothèque DOS, il doit donc tout d'abord ouvrir cette bibliothèque. On emploie pour cela une fonction de Exec, qui porte le nom de OldOpenLibrary. Cette fonction reçoit un pointeur sur le nom de la bibliothèque, qui doit être écrit en minuscules et se terminer par un octet 0.

On peut aussi utiliser la fonction OpenLibrary, à laquelle il faut cependant communiquer un paramètre supplémentaire, la version désirée de la bibliothèque. Si le numéro de version est plus grand ou égal à ce numéro, la bibliothèque s'ouvre. C'est aussi pourquoi on place en général un 0 en ce point, pour que la version n'intervienne pas.

En langage C, on y parvient très simplement.

La ligne

```
DOSBase = OpenLibrary("dos.library",0);
```

permet d'obtenir d'Exec en DOSBase un pointeur sur la bibliothèque DOS, pointeur grâce auquel on peut ensuite appeler les fonctions DOS. On n'a plus besoin ensuite du pointeur, car le compilateur C se charge de la suite. On peut malgré tout vérifier que le DOS a été correctement ouvert, en examinant la valeur de retour ; cette valeur est 0 s'il s'est produit une erreur. Voici l'instruction correspondante :

```
if (DOSBase == 0) exit(DOS_OPEN_ERROR);
```

En langage machine par contre, les instructions ne sont pas aussi simples, mais néanmoins pas trop longues. L'ouverture de la bibliothèque DOS se programme de la façon suivante :

```
Exec_Base      = 4
OldOpenLibrary = -408

move.l  Exec_Base,a6      ;Pointeur sur base Exec en A6
lea     DOS_Name,a1      ;Pointeur sur le nom de bibliothèque
jsr     OldOpenLibrary(a6) ;Ouvrir la bibliothèque
move.l  d0,DOS_Base       ;Sauver le pointeur sur base DOS
beq    error              ;Une erreur est survenue
...
error:                      ;Traitement de l'erreur
...
DOS_Base: dc.l 0           ;Place pour la base DOS
DOS_Name: dc.b "dos.library",0
```

Le pointeur obtenu dans D0 est nécessaire pour tous les appels ultérieurs d'une fonction DOS. Si l'ouverture n'a pas bien fonctionné, on reçoit en retour un 0 en D0, et le programme passe à la routine de traitement des erreurs 'error'.

La bibliothèque DOS est donc disponible par l'intermédiaire du pointeur, lorsqu'on utilise le programme ci-dessus. Nous avons dit qu'elle avait la même structure que la bibliothèque Exec, et son traitement est donc identique. Les adresses d'entrée des

différentes fonctions se trouvent sous l'adresse de base qui se trouve dans DOS_Base, et elles sont donc appelées avec des offsets négatifs.

3.2.2. Appels d'une fonction et transmission de paramètres

Pour l'appel d'une fonction DOS, on a souvent besoin, outre l'adresse de la fonction, de quelques autres paramètres, qui doivent donc être transmis. Ces paramètres sont communiqués dans les registres de données D1 à D4 du processeur.

Un exemple : pour ouvrir une simple fenêtre, on utilise la fonction DOS Open(). Les paramètres nécessaires sont les suivants :

- ✓ Un pointeur sur le nom du fichier à ouvrir, se terminant par un octet nul, dans le registre D1. Pour notre exemple, nous adopterons comme nom la définition de la fenêtre CON:, suivi des paramètres adéquats.
- ✓ En D2 est exigée l'indication du mode d'accès. Ce mode dit s'il s'agit d'un fichier à installer ou d'un fichier déjà existant. Pour la fenêtre à ouvrir dans notre exemple, nous transmettrons le mode 'ancien', pour que l'on puisse aussi lire dans la fenêtre.

Le programme en langage machine pour cet exemple devrait donc se présenter ainsi:

```

Open      = -30
Mode_old = 1005

...
move.l #FileName,d1      ;Pointeur sur la définition de fichier
move    #Mode_old,d2      ;Mode: ancien
move.l DOS_Base,a6        ;adresse de la base DOS en A6
jsr     Open(a6)          ;Ouvrir fichier (fenêtre)
move.l d0,ConHandle       ;Sauver pointeur sur handle fichier
beq    error              ;Une erreur est survenue!
...
ConHandle: dc.l 0          ;Place pour le handle du fichier
FileName:  dc.b "CON:10/10/620/200/** Fenêtre de test **",0

```

Nous reviendrons dans un chapitre ultérieur sur l'utilisation précise du canal standard CON:.

3.2.3. Les fonctions DOS

Nous présenterons ultérieurement toutes les fonctions DOS une à une. Nous indiquerons aussi bien les offsets que les registres dans lesquels les différents paramètres doivent être transmis.

3.2.3.1. La gestion des processus

Nous allons créer ici une nouvelle structure de processus sous le nom pointé par D1. Ce processus fonctionnera avec la priorité indiqué dans "Pri", et recevra une pile de taille "Pile".

Dans "Segment", on transmet un pointeur sur une liste de segments (cf. aussi LoadSeg), dans laquelle est défini le code programme à lancer. Le programme devrait commencer dans le premier segment de la liste.

Le résultat de la fonction est le nouveau numéro d'identification (ID) du processus, ou un 0 si une erreur est advenue.

Pour montrer comment fonctionne CreateProc, qui représente une fonction très intéressante, voici un petit programme qui charge un autre programme nommé Nom_Programme, et le lance comme processus avec le nom Nouveau_processus (comme processus en tâche de fond !) :

```
;***** CreateProc-Demo S.D. *****

OpenLib    - -408
closeLib   - -414
ExecBase   - 4

Open       - -30
Close      - -36
Read       - -42
Write      - -48
mode_old   - 1005

LoadSeg    - -150
UnLoadSeg  - -156
CreateProc - -138

run:
move.l  execbase,a6           ;Pointeur sur bibliothèque EXEC
lea      dosname(pc),a1
moveq   #0,d0
jsr      openlib(a6)          ;Open DOS-Library
move.l  d0,dosbase
beq     error

move.l  #consolname,d1         ;Consol-Definition
move.l  #mode_old,d2
move.l  dosbase,a6
jsr      open(a6)              ;ouverture consoles
beq     error
move.l  d0,conhandle          ;sauver le Consol-Handle

move.l  #name,d1               ;Charger le programme
jsr      LoadSeg(a6)
tst.l  d0
beq     error
move.l  d0,segment             ;Erreur!
                                ;Sauver le pointeur sur liste de
segments
```

```

move.l #pname,d1          ;Pointeur sur nom dans D1
move.l #0,d2              ;Priorité dans D2
move.l segment,d3         ;Pointeur liste de segments dans D3
move.l #3000,d4           ;Longueur pile des processus dans D4
jsr    CreateProc(a6)     ;Créer/lancer le processus
beq    error               ;Erreur!

move.l conhandle,d1       ;Adresse du buffer
move.l #inbuff,d2         ;1er caractère
move.l #1,d3
move.l dosbase,a6         ;Lire au clavier
jsr    read(a6)

move.l segment,d1
jsr    UnLoadSeg(a6)      ;Nouveau processus supprimé

error:
move.l conhandle,d1       ;Fermer la fenêtre
move.l dosbase,a6
jsr    close(a6)

move.l dosbase,a1          ;Fermer DOS.Lib
move.l execbase,a6
jsr    closelib(a6)

rts                         ;C'est fini

dosbase: dc.l 0
conhandle: dc.l 0
segment: dc.l 0
inbuff: blk.b 8
consolname: dc.b 'RAW:100/50/300/100/** Processus principal **',0
dosname: dc.b 'dos.library',0
name: dc.b 'Nom_Programme',0
pname: dc.b 'Nouveau_Processus',0
even

```

Ce programme génère donc un processus. Dans les chapitres précédents, nous avons déjà beaucoup parlé de processus. Il est temps de se demander ce que c'est.

On peut comparer les processus aux tâches qui ont été créées par l'intermédiaire de Exec. Comme pour les tâches, un processus est une structure qui sert à gérer un programme tournant à l'intérieur du système Multi-tâches. Ces structures de processus figurent, comme les structures de tâche, dans les listes de tâches de Exec.

La différence entre les tâches et les processus tient dans le fait que les processus sont générés par le DOS, et donc qu'ils contiennent non seulement des informations pour Exec, mais aussi des informations concernant le DOS. Il n'est pas étonnant, dans ces conditions, que les structures de processus se trouvent dans les listes de tâches de Exec, puisqu'elles représentent simplement une extension de la structure de tâche qui nous est déjà familière.

On crée un processus lorsqu'un programme est chargé et lancé par l'intermédiaire du DOS (par exemple pour le lancement d'une commande CLI). Vous pouvez aussi générer un processus "à la main", en utilisant les fonctions DOS LoadSeg() et CreateProc().

La structure de processus se présente ainsi :

```
struct Process {           /*      Offsets      */
    struct Task pr_Task;   /* 00          $00 */
    struct MsgPort pr_MsgPort; /* 92 $5C 00 $00 */
    WORD pr_Pad;           /* 126 $7E 34 $22 */
    BPTR pr_SegList;       /* 128 $80 36 $24 */
    LONG pr_StackSize;     /* 132 $84 40 $28 */
    APTR pr_GlobVec;       /* 136 $88 44 $2C */
    LONG pr_TaskNum;       /* 140 $8C 48 $30 */
    BPTR pr_StackBase;     /* 144 $90 52 $34 */
    LONG pr_Result2;       /* 148 $94 56 $38 */
    BPTR pr_CurrentDir;    /* 152 $98 60 $3C */
    BPTR pr_CIS;           /* 156 $9C 64 $40 */
    BPTR pr_COS;           /* 160 $A0 68 $44 */
    APTR pr_ConsoleTask;   /* 164 $A4 72 $48 */
    APTR pr_FileSystemTask; /* 168 $A8 76 $5C */
    BPTR pr_CLI;           /* 172 $AC 80 $60 */
    APTR pr_ReturnAddr;    /* 176 $B0 84 $64 */
    APTR pr_PktWait;       /* 180 $B4 88 $68 */
    APTR pr_WindowPtr;     /* 184 $B8 92 $6C */
};
```

Vous vous étonnerez certainement de voir que dans cette structure, il n'y a pas un offset par ligne, mais deux. Vous devez savoir en conséquence que le pointeur retourné par CreateProc() (le pointeur sur le processus), représente en réalité un pointeur sur le port de message de la structure de processus présentée plus haut.

DOS fonctionne intérieurement avec deux pointeurs. Il s'agit d'une part du pointeur sur le début de la structure (donc du pointeur sur la structure de tâche), et d'autre part du pointeur sur le port de message (offset 92) de la structure de processus. A partir de ces deux pointeurs, on accède à la structure avec les offsets adéquats.

Pour que vous puissiez utiliser plus commodément les accès aux entrées qui se trouvent derrière la structure de tâche, et auxquelles on a accès d'habitude par le pointeur sur le message de port, nous avons indiqué les deux offsets.

Description de la structure

pr_Task

Structure de tâche, déjà décrite dans le chapitre sur Exec. Elle est nécessaire à l'organisation du fonctionnement Multi-tâches.

pr_MsgPort

Structure de port de message, également traitée dans le chapitre sur Exec. Elle sert de port pour le processus, auquel on peut donc envoyer des messages sans être obligé d'effectuer de nouvelles initialisations du côté du processus. C'est essentiellement par ce port que sont envoyés les paquets. Nous allons y revenir.

pr_Pad

Ce mot est introduit pour que les lignes suivantes commencent à des adresses de mot long.

pr_SegList

Pointeur BPTR sur le champ des listes de segments qui sont nécessaires pour le processus.

pr_StackSize

Le mot long déposé ici indique la longueur de la pile disponible pour le processus.

pr_GlobVec

Pointeur sur le tableau global des vecteurs, créé pour le processus s'il s'agit d'un programme BCPL.

pr_TaskNum

Numéro du CLI ouvert. Il s'agit ici du même numéro que celui qui est affiché à l'écran par l'intermédiaire du prompt. Si le processus n'est pas un CLI, cette ligne est nulle.

pr_StackBase

Pointeur BPTR sur l'extrémité supérieure de la pile dont dispose le processus.

pr_Result2

Le second résultat d'une commande DOS, un message d'erreur.

pr_CurrentDir

Pointeur BPTR sur la structure FileLock, par l'intermédiaire de laquelle le répertoire actuel est adressé.

pr_CIS

Pointeur BPTR sur le stream d'entrée CLI actuel (Input-Stream).

pr_COS

Pointeur BPTR sur le stream de sortie CLI actuel (Output-Stream).

pr_ConsoleTask

Pointeur sur le handler-task pour la fenêtre d'affichage - console - actuelle.

pr_FileSystemTask

Pointeur sur le handler-task pour le lecteur actuel.

pr_CLI

Pointeur BPTR sur une autre structure, qui donne des indications plus précises sur l'aspect du CLI. La structure n'est pas explicitée.

*pr_ReturnAddr**pr_PktWait*

C'est ici le pointeur sur une routine propre, attendant un paquet. La routine est appelée par la fonction GetPacket(), à condition que le processus dispose d'une fonction d'attente propre.

pr_WindowPtr

Pointeur sur la fenêtre utilisée. Il est nécessaire pour le cas où le pointeur habituellement utilisé serait perdu à cause d'une erreur. Comme on le voit sur la structure, l'offset 172 porte le pointeur PBTR "pr_CLI". Il pointe sur une autre structure. La structure s'appelle CommandLineInterface, et présente l'aspect suivant :

```
struct CommandLineInterface {      /* Offsets */
    LONG cli_Result2;             /* 00 $00 */
    BSTR cli_SetName;             /* 04 $04 */
    BPTR cli_CommandDir;          /* 08 $08 */
    LONG cli_ReturnCode;           /* 12 $0C */
    BSTR cli_CommandName;          /* 16 $10 */
    LONG cli_FailLevel;            /* 20 $14 */
    BSTR cli_Prompt;               /* 24 $18 */
    BPTR cli_StandardInput;        /* 28 $1C */
    BPTR cli_CurrentInput;         /* 32 $20 */
    BSTR cli_CommandFile;          /* 36 $24 */
    LONG cli_Interactive;          /* 40 $28 */
    LONG cli_Background;            /* 44 $2C */
    BPTR cli_CurrentOutput;         /* 48 $30 */
    LONG cli_DefaultStack;          /* 52 $34 */
    BPTR cli_StandardOutput;        /* 56 $38 */
    BPTR cli_Module;                /* 60 $3C */
};
```

cli_Result2

Message d'erreur du dernier appel CLI.

cli_SetName

Pointeur BPTR sur le nom du répertoires actuel.

cli_CommandDir

Pointeur BPTR sur la structure FileLock, par l'intermédiaire de laquelle on peut accès au répertoire des commandes CLI.

cli_ReturnCode

Valeur de retour de la dernière commande CLI.

cli_CommandName

Pointeur sur le nom de la commande actuellement en fonctionnement.

cli_FailLevel

Valeur indiquée avec FAILAT.

cli_Prompt

Pointeur sur le PROMPT.

cli_StandardInput

Pointeur sur l'entrée standard (clavier).

cli_CurrentInput

Pointeur sur l'entrée actuelle.

cli_CommandFile

Pointeur sur le nom du fichier de commandes en cours de traitement (par exemple la startup-sequence).

cli_Background

A la valeur 1 si la commande CLI est appelée avec RUN.

cli_CurrentOutput

Pointeur sur l'affichage actuel.

cli_DefaultStack

Longueur de la pile disponible, en mots longs.

cli_StandardOutput

Sortie standard (écran).

cli_Module

Pointeur sur les listes de segments pour la commande CLI traitée actuellement.

3.2.4. Messages d'erreur du DOS

Vous trouverez dans la liste suivante les codes d'erreur transmis par IoErr() ou par la commande WHY du CLI, suivis par leur noms et leurs significations.

103	Insufficient free store	Il n'y a pas assez de place libre en mémoire.
104	Task table full	Déjà 20 processus, il n'est pas possible d'en ajouter d'autres.
120	Argument line invalid or too long	La liste des arguments pour cette commande n'est pas correcte ou contient trop d'indications.
121	File is not an object module	Le fichier appelé n'est pas exécutable.
122	Invalid resident library during load	La bibliothèque résidente que vous appelez n'est pas valide.
202	Object in use	Le fichier ou le répertoire indiqué est utilisé en ce moment par un autre programme, et il n'est pas disponible pour d'autres applications.
203	Object already exists	Ce nom de fichier existe déjà.

204	Directory not found	Le répertoire sélectionné n'existe pas.
205	Object not found	Il n'existe pas de fichier de ce nom.
206	Invalid window	Le paramètre indiqué pour la fenêtre à ouvrir n'est pas correct.
209	Packet requested type unknown	La fonction souhaitée ne peut pas être exécutée sur le périphérique indiqué.
210	Invalid stream component name	Le nom de fichier n'est pas valide (trop long ou comportant des caractères non autorisés).
211	Invalid object lock	La structure Lock indiquée n'est pas valide.
212	Object not of required type	Confusion entre nom de fichier et nom de répertoire.
213	Disk not validated	La disquette n'a pas été reconnue par le système, ou alors elle est défectueuse.
214	Disk write-protected	La disquette est protégée contre l'écriture.
215	Rename across devices attempted	La fonction RENAME n'est possible qu'à l'intérieur d'une disquette.
216	Directory not empty	Le répertoire dont on demande la suppression n'est pas vide.
218	Device not mounted	Périphérique non reconnu ou non défini.
219	Seek error	La fonction Seek() est pourvue de paramètres non autorisés.
220	Comment too big	Le commentaire est trop long.
221	Disk full	La disquette est pleine, ou alors elle ne contient pas assez de place pour cette application.
222	File is protected from deletion	Le fichier ne peut pas être supprimé, ou bien il est protégé contre la suppression.
223	File is protected from writing	Le fichier est protégé contre l'écriture.
224	File is protected from reading	Le fichier est protégé contre la lecture. Pour les trois derniers messages d'erreur, vous pouvez examiner le statut des fichiers en question à l'aide de la commande LIST.

225	Not a DOS disk	Cette disquette n'a pas été formatée en format AmigaDOS.
226	No disk in drive	Il n'y a pas de disquette dans le lecteur.
232	No more entries in directory	Il n'y a plus d'entrées dans le répertoire.

3.3. Entrée/Sortie

Une composante essentielle d'un programme est l'échange des données avec le monde extérieur, par l'intermédiaire de l'écran, du clavier, des disquettes ou encore des interfaces. Seule cette entrée/sortie, nommée aussi tout simplement I/O (Input/Output), rend un programme capable de tirer pleinement parti de l'ordinateur sur lequel il fonctionne. Il y a trois manières de réaliser l'entrée/sortie.

La première est l'entrée/sortie au moyen des fonctions DOS correspondantes, comme Open(), Close(), Read() et Write(). Cette méthode est évidemment la plus simple, car elle n'exige pas de gros efforts de programmation de votre part. L'inconvénient est que la fonction appelée doit être entièrement traitée avant que votre programme ne poursuive son travail.

La seconde méthode ne présente pas cet inconvénient. Le mot magique est ici Devices. Grâce à ces Devices, vous pouvez rendre le fonctionnement de l'entrée/sortie entièrement indépendant de la suite de votre programme. L'entrée/sortie fonctionne en arrière-plan, donc parallèlement à votre programme, et ne revendique pratiquement pas de temps propre au processeur. L'inconvénient de cette technique est l'effort bien plus grand exigé de la part du programmeur.

La troisième méthode, enfin, est la programmation directe du hardware de l'Amiga. Ceci suppose néanmoins des connaissances très poussées sur le système et a de plus l'inconvénient de conduire à de grandes complications si on veut un fonctionnement Multi-tâches. Vous trouverez d'autres informations là-dessus dans la partie de cet ouvrage consacrée au hardware.

Commençons notre tour d'horizon sur la programmation de l'entrée/ sortie par la méthode standard : l'utilisation des fonctions DOS.

Comme nous l'avons déjà mentionné, les quatre fonctions DOS responsables de l'entrée/sortie des données sont Open(), Close(), Read() et Write(). Elles permettent d'exécuter la plupart des tâches que nécessite un programme.

Il existe toute une série de canaux d'entrée/sortie que le DOS connaît déjà par leur nom. Ces noms peuvent être utilisés dans une commande Open. Voici quels sont ces canaux standard :

DFn :

Désigne le lecteur de disquettes numéro n. Ce numéro peut être 0, 1, 2 ou 3.

SYS :

Désigne le lecteur de disquettes à partir duquel a été chargé le système.

RAM :

Désigne le disque RAM, toujours disponible, et dont la taille est proportionnelle aux données qu'il contient. Il peut être utilisé comme un lecteur de disquettes, à ceci près que les données sont entreposées non pas sur disquette, mais dans la mémoire vive de l'Amiga.

NIL :

Ce canal est un véritable cimetière de données : les données qui y sont envoyées sont rejetées et ne servent donc à rien. Il peut être parfois utile d'en passer par là, lorsque par exemple un programme veut effectuer des sorties dont vous n'avez rien à faire.

SER :

Désigne l'interface sérielle (RS232), et permet l'entrée/sortie des données par ce port.

PAR :

Désigne l'interface parallèle (port d'imprimante), qui contient en fait 8 canaux d'entrée/sortie. Vous pouvez donc sortir par ce port parallèle des données, ou les charger.

PRT :

Désigne également l'interface parallèle, à cette différence près que cette fois vous pouvez adresser une imprimante. Si cependant l'imprimante est définie pour l'interface sérielle, ce sera quand même celle-ci qui sera adressée. Vous pouvez prescrire à l'avance les définitions pour l'imprimante, à l'aide du programme Preferences.

CON :

Prescrit une fenêtre pour l'entrée/sortie. Cette fenêtre est ouverte automatiquement lors de l'ouverture du canal. Les paramètres de la fenêtre sont indiqués de la façon suivante :

CON:x/y/l/h/Nom

x et *y* représentent les coordonnées du coin supérieur gauche de la fenêtre à l'écran, *l* et *h* la largeur et la hauteur de la fenêtre en pixels ; *Nom* est le titre de la fenêtre qui apparaît dans la ligne de titre. Ainsi

```
CON:20/10/200/100/Fenêtre-Test
```

désigne une fenêtre qui porte le nom "Fenêtre-Test", qui commence à la position *x*=20 et *y*=10, et dont la largeur et la hauteur font respectivement 200 et 100 points.

RAW :

Représente également une fenêtre et transmet les entrées et les sorties à cette fenêtre. Au contraire de ce qui se passe avec CON:, il n'y a pas ici de fonction qui soit traitée à l'avance (par exemple l'édition d'une ligne), si bien que la fenêtre en question ne doit être utilisée que dans des cas très particuliers.

*

Désigne la fenêtre actuelle, par exemple la fenêtre CLI.

Sur le Workbench qui est sorti vers la fin de l'année 1988, on trouve des handlers supplémentaires, contenus dans le répertoire DEVS. Si on les place dans la MountList (donnée comme exemple sur le disque WB), si on lance la commande BindDrivers, on lie les devices standard suivants :

NEWCON :

Ce device correspond en fait à l'ancien CON:. La grande différence entre les deux tient au fait que NEWCON permet l'édition d'une ligne. On peut donc aller et venir dans une ligne avec les touches de direction du curseur, et supprimer ou insérer des caractères. Vous pouvez essayer de le faire avec la commande

```
copy newcon:10/10/400/100/Newcon *
```

Dans la fenêtre que vous verrez s'ouvrir, vous pourrez éditer le texte, qui sera ensuite affiché dans la fenêtre CLI actuelle, lorsque vous presserez la touche Return. Le nec plus ultra est la fonction History, ce qui veut dire que vous pouvez répéter les lignes déjà entrées en déplaçant le curseur du haut vers le bas ! C'est ainsi que travaille également l'AmigaShell du Workbench 1.3.

AUX :

Propose un handler série sans tampon. L'avantage est évident. Avec

```
newcli aux:
```

On peut transformer un terminal lié à l'interface sérielle en second poste de travail, puisque toutes les entrées/sorties de texte de ce nouveau CLI se font sur le terminal. La machine Multi-tâches Amiga devient maintenant une installation multi-utilisateurs.

PIPE :

Offre une communication très simple entre deux processus. Comme le disque RAM, ce device est utilisé pour entreposer temporairement des données. On peut ainsi écrire quelque chose sur PIPE:, et même indiquer un nom de fichier. Un autre processus peut ensuite lire directement dans PIPE: et obtenir les données du processus précédent.

Au contraire du disque RAM, un pipe occupé ne peut pas recevoir un autre texte à la place de celui qu'il contient. Si vous écrivez d'écrire une seconde fois dans un pipe, cela ne fonctionne que si son contenu a déjà été lu. En effet, l'opération de lecture dans un pipe efface son contenu. Si on essaie de lire dans un pipe qui n'a encore rien reçu, le processus chargé de lire attend jusqu'à ce que le pipe soit rempli. Vous n'aurez pas de difficulté à effectuer des essais avec deux fenêtres CLI.

SPEAK :

Ce device propose une ancienne marchandise dans un nouvel emballage. Il correspond en effet à peu de chose près à la commande Say, à ceci près que la sortie de la parole fonctionne ici comme un simple device. Vous pouvez ainsi demander à entendre la lecture à haute voix de la MountList grâce à la commande

```
copy devs:MountList speak:
```

Bien que ce ne soit pas ce qu'il y a de plus aisément...

RAD :

Il s'agit d'un nouveau disque RAM avec deux différences de taille par rapport à l'ancien : il est de longueur fixe et il résiste au reset !

La taille de la RAD peut être définie dans la MountList. La taille prédéfinie est déduite des spécifications empruntées au disque :

```
Surfaces - 2
BlockPerTrack -11
LowCyl -0
HighCyl -21
```

Si vous désirez modifier la taille, il vous suffit de changer la valeur HighCyl. Celle-ci donne le nombre de tracks - 1 (la numérotation commence à 0), chacun d'eux contenant environ 11 Ko.

FAST :

Ce device se réfère à un disque dur, formaté en FFS (Fast Filing System).

Commençons maintenant par l'application probablement la plus importante : la saisie au clavier et l'affichage à l'écran.

3.3.1. Clavier et écran

L'énumération précédente montre que l'AmigaDOS met trois possibilités à votre disposition pour les entrées/sorties à l'écran : CON:, RAW: et *.

Fenêtre CON:/NEWCON:

Pour ouvrir une fenêtre CON:, on utilise la fonction Open() du DOS. La fonction attend pour paramètres un pointeur sur le nom du canal à ouvrir, et le mode sous lequel ce canal doit être ouvert. Les modes possibles sont les suivants :

Mode_new pour un canal dans lequel on peut seulement écrire.

Mode_old pour un canal à partir duquel on peut aussi lire.

Mode_readwrite dans la version DOS 1.2, où on peut aussi bien écrire que lire dans le canal.

Pour ouvrir une fenêtre CON: ou RAW:, on utilise le mode Mode_old, puisque le canal est connu, et que l'on peut aussi lire dans ce canal.

Voici un petit programme de démonstration en langage machine, qui ouvre une fenêtre CON:, qui affiche un texte dans cette fenêtre, attend une saisie, puis referme la fenêtre :

```
;***** Entrée/Sortie simple par CON: *****

OpenLib = -408
closelib = -414
ExecBase = 4

; offsets Amiga DOS

Open    = -30
Close   = -36
Read    = -42
Write   = -48
Exit    = -144

Mode_old = 1005

run:
    move.l execbase,a6      ;Pointeur sur Bibliothèque Exec
    lea    dosname.al
```

```

moveq #0,d0
jsr openlib(a6)      ;Open DOS-Library
move.l d0,dosbase
beq error            ;pas réussi

move.l dosbase,a6    ;Adresse de base DOS en A6

move.l #name,d1       ;Pointeur sur le nom
move.l #mode_old,d2   ;Mode
jsr Open(a6)          ;ouvrir le fenêtre
move.l d0,conhandle   ;sauvegarder le Handle
beq error

move.l conhandle,d1   ;Handle de la fenêtre en D1
move.l #text,d2        ;adresse du texte en D2
move.l #tende-text,d3  ;longueur en D3
jsr Write(a6)          ;affichage du texte
move.l conhandle,d1   ;Handle de la fenêtre
move.l #buffer,d2      ;adresse du tampon
move.l #80,d3           ;longueur max.
jsr Read(a6)           ;attendre saisie

move.l conhandle,d1
jsr Close(a6)          ;fermer la fenêtre
bra fin                ;terminé

error:
move.l #-1,d0          ;Error-Status
fin:
move.l d0,d1

move.l dosbase,a6
jsr Exit(a6)           ;fin du programme

rts                    ;ne se produit pas

dosname: dc.b 'dos.library',0
name:    dc.b 'CON:20/10/200/100/** Fenêtre de test',0
text:    dc.b 'Veuillez écrire votre texte !',0
tende:   blk.b 80
even
dosbase: dc.l 0
conhandle: dc.l 0

```

Fenêtre RAW:

Le programme ci-dessus peut aussi être mis en route avec RAW : au lieu de CON:. Essayez, et vous remarquerez tout de suite la différence ! La version CON: attend en effet que l'on actionne la touche Return après la saisie, tandis que le programme avec RAW: reprend les commandes dès que l'on a appuyé sur une touche, quelle qu'elle soit. Ceci est vrai également pour les touches de fonction et les touches de déplacement du curseur, qui ne sont d'ailleurs pas reconnues par la fenêtre CON:.

Une fenêtre CON: offre donc plus de confort pour la saisie de textes entiers; une fenêtre RAW: met par contre à votre disposition l'ensemble du clavier.

Mais dans les deux cas, la représentation normale des caractères n'est pas la seule possible. Il existe encore la possibilité de représenter d'autres types de caractères, par exemple soulignés et en gras. Il y a encore d'autres fonctions disponibles, grâce auxquelles on peut manipuler les fenêtres. Il est possible par exemple de supprimer l'image, de la descendre ou de la monter, etc... Toutes ces fonctions sont exécutées grâce à des séquences de contrôle, qui doivent être affichées dans la fenêtre, pourvues pour une partie d'entre elles de paramètres.

Voici une liste des caractères de contrôle qui déclenchent l'exécution d'une fonction. Ces signes sont présentés sous la forme de nombres en hexadécimal :

Séquence	Fonction
08	Backspace
0A	Linefeed
0B	curseur vers le haut d'une ligne
0C	Supprimer la fenêtre
0D	Carriage Return
0E	Retour à la présentation normale (OF suppression)
0F	Passage à un style particulier
1B	Escape

Les séquences qui suivent commencent toutes par le signe \$9B, c'est-à-dire le CSI (Control Sequence Introducer). Les caractères qui viennent après ce signe déclenchent tous une fonction. Les valeurs indiquées entre crochets peuvent ne pas figurer. La donnée 'n' doit être indiquée sous forme de caractères ASCII, et doit représenter un nombre décimal à un ou plusieurs chiffres. La valeur adoptée par défaut pour n est placée après le n entre parenthèses.

Séquence	Fonction
9B[n] 40	Indentation de n caractères
9B[n] 41	Curseur de n (1) lignes vers le haut
9B[n] 42	Curseur de n (1) lignes vers le bas
9B[n] 43	Curseur de n (1) lignes vers la droite
9B[n] 44	Curseur de n (1) lignes vers la gauche
9B[n] 45	Curseur de n (1) lignes vers le bas colonne 1
9B[n] 46	Curseur de n (1) lignes vers le haut colonne 1
9B[n] [3B n] 48	Placer le curseur dans la ligne-colonne ...
9B 4A	Supprimer la fenêtre à partir de la position du curseur

Séquence	Fonction
9B 4B	Effacer la ligne à partir de la position du curseur
9B 4C	Insérer une ligne
9B 4D	Effacer la ligne
9B[n] 50	Effacer n caractères à partir de la position du curseur
9B[n] 53	Déplacement vers le haut de n lignes
9B[n] 54	Déplacement vers le bas de n lignes
9B 323068	Désormais : Linefeed → Linefeed+Return
9B 32306C	Désormais : Linefeed → seulement Linefeed
9B 6E	<p>Retourner la position du curseur ! On obtient en retour une chaîne de caractères de la forme :</p> <p>9B (ligne) 3B (colonne) 52</p> <p>9B (style);(coul. premier plan);(coul.arrière-plan) 6D</p> <p>Les trois paramètres sont eux aussi des nombres décimaux en format ASCII. Ils signifient :</p> <p>Style :</p> <p>1=gras</p> <p>3=italique</p> <p>4=souligné</p> <p>7=inverse-vidéo</p> <p>Couleur de premier plan :</p> <p>30-37 : couleurs 0 à 7 pour le texte</p> <p>Couleur d'arrière plan :</p> <p>40-47 : couleurs 0 à 7 pour l'arrière plan</p>
9B (Longueur) 74	Fixe le nbre max. de caractères à présenter
9B (Largeur) 75	Fixe la longueur max. de la ligne
9B (Distance) 78	Définit la distance en pixels à partir du bord gauche de la fenêtre jusqu'au point où l'affichage doit commencer.
9B (Distance) 79	Définit la distance en pixels à partir du bord supérieur de la fenêtre jusqu'au point où l'affichage doit commencer. Les 4 dernières fonctions peuvent placer le point de départ du texte en position normale si l'on ne fait pas figurer le paramètre.
9B 30 20 70	Rendre le curseur invisible
9B 20 70	Rendre le curseur visible

Séquence	Fonction
9B 71	Obtenir la taille de la fenêtre en retour ! On obtient en retour une chaîne de paramètres de la forme suivante :
	9B 31 3B 31 3B (lignes)
	3B (colonnes) 73

Pour voir comment on utilise ces caractères de contrôle, amusez-vous à afficher le texte suivant dans votre fenêtre :

```
text: dc.b $9b,"4;31;40m"
      dc.b "titre"
      dc.b $9b,"3;33;40m",$9b,"5;20H"
      dc.b "*** Hello, le monde ! ***,0
```

Les paramètres pour les séquences de contrôle sont indiqués ici, entre guillemets, comme chaînes de caractères ASCII. Vous voyez que vous avez ainsi un moyen très efficace d'afficher des textes.

Ces séquences peuvent être envoyées, mais elles peuvent aussi être reçues, lorsqu'une touche de fonction du clavier ou une touche de déplacement du curseur vient d'être pressée. Les caractères que l'on reçoit en un pareil cas sont les suivants (CSI est mis pour \$9B) :

Touche	Sans Shift	Avec Shift
F1	<CSI>0~	<CSI>10~
F2	<CSI>1~	<CSI>11~
...		
F9	<CSI>8~	<CSI>18~
F10	<CSI>9~	<CSI>19~
HELP	<CSI>?~	<CSI>?~
haut	<CSI>A	<CSI>T~
bas	<CSI>B	<CSI>S~
gauche	<CSI>C	<CSI> A~
droite	<CSI>D	<CSI> @~

De cette façon, on peut être renseigné sur tout ce qu'entreprend l'utilisateur au clavier. Si ça ne suffit toujours pas, il existe encore une source d'information : le RAW-Input-Events. Ce sont des événements qui sont annoncés par une séquence, si vous le désirez. Le désir de recevoir un message qui renseigne sur ces événements est lui-même communiqué au DOS par une séquence, sous la forme suivante :

<CSI>n{

Le 'n' représente ici un nombre entre 1 et 16, ce nombre correspondant à un événement qui doit être annoncé. Les événements en question sont les suivants :

1	Touche pressée
2	Touche de la souris pressée
3	Fenêtre activée
4	Souris déplacée
5	Non utilisé
6	Timer
7	Gadget sélectionné
8	Gadget abandonné
9	Requester effacé
10	Menu sélectionné
11	Fenêtre fermée (cf. Console-Device)
12	Taille de fenêtre modifiée
13	Nouvelle fenêtre
14	Réglages modifiés
15	Disquette enlevée du lecteur
16	Disquette introduite

Pour certains (10, 11), ces événements ne sont pas disponibles, étant donné qu'une fenêtre ouverte avec DOS ne dispose en fait ni de menus, ni du symbole de fermeture. Ces possibilités commencent à devenir réellement intéressantes lorsqu'on a créé soi-même sa fenêtre-console. Ceci n'est malgré tout possible que par la combinaison de Intuition et de Devices, et c'est pourquoi nous en traiterons un peu plus tard, dans le paragraphe concernant le Console-Device.

Lorsqu'un de ces événements énumérés se produit (par exemple l'introduction d'une disquette), on obtient alors en retour une séquence de la forme suivante :

<CSI><Classe>;<sous-classe>;<touche>;<état>;<x>;<y>;
<secondes>;<microsecondes>

La signification des différents termes est la suivante :

CSI	Control-sequence-introducer \$9B
Classe	Numéro de l'événement (cf.ci-dessus)
Touche	Code clavier de la dernière touche pressée ou de la touche souris
Etat	Etat du clavier : Cf.tableau qui suit
X et Y	Coordonnées du pointeur de la souris lors d'un événement concernant la souris.
Secondes	
Microsecondes	Temps système au moment de l'événement.

Bit	Masque	Signification
0	0001	Touche Shift de gauche
1	0002	Touche Shift de droite
2	0004	Touche CapsLock
3	0008	Control
4	0010	Touche Alternate gauche
5	0020	Touche Alternate droite
6	0040	Touche Amiga gauche
7	0080	Touche Amiga droite
8	0100	Bloc numérique
9	0200	Répétition des touches
10	0400	Interrupt (non utilisé)
11	0800	Fenêtre active
12	1000	Touche souris gauche
13	2000	Touche souris droite
14	4000	Touche souris milieu (non utilisé)
15	8000	Coordonnées relatives souris

Fenêtre *

La plupart des commandes CLI utilisent *, car c'est le moyen le plus simple. Etant donné que l'on obtient de cette façon la fenêtre actuelle, évidemment déjà ouverte, on peut ainsi se dispenser même d'ouvrir et de fermer le canal.

Comme les fonctions Read() et Write() ont tout de même besoin d'un handle du canal dans lequel elles doivent lire ou écrire des données, ce handle doit tout d'abord être transmis.

Les fonctions DOS Input() et Output() sont prévues à cet effet. Elles ne nécessitent pas de paramètre, et elles renvoient le handle du canal standard correspondant. Ce sera la fenêtre CLI si le programme a simplement été appelé à partir de là.

Si l'entrée ou la sortie a cependant été dérivée avec l'une des fonctions <- ou >- du CLI, on obtient le handle actuel par la fonction Input() ou Output().

3.3.2. Fichiers sur disquettes

De la même façon que les fenêtres CON: et RAW:, on peut aussi ouvrir et traiter des fichiers sur disquettes, avec seulement quelques éléments supplémentaires. Ainsi le mode transmis lors de l'ouverture joue un grand rôle. Si l'on a transmis le mode 'ancien', c'est un fichier existant sur la disquette qui est recherché, et il n'est possible que d'en effectuer une lecture. Avec le mode 'nouveau', le fichier est mis en place, et un fichier éventuel existant sous le même nom serait supprimé. Dans un fichier ouvert de cette manière, on peut seulement écrire. Enfin avec le mode 'readwrite', on peut écrire et lire dans un fichier existant, c'est-à-dire le modifier.

Pour les fonctions DOS Read(), Write() et Close(), rien ne se modifie en ce qui concerne l'entrée/sortie à l'écran. Il y a cependant un certain nombre de fonctions en plus, qui sont très utiles lors de l'utilisation de fichiers sur disquettes.

Etant donné que l'on peut lire des données au fur et à mesure sur une disquette, le système doit avoir un moyen pour se souvenir de l'endroit où l'on a accédé pour la dernière fois dans le fichier. Ceci est réalisé à l'aide d'un pointeur, que l'on peut aussi déplacer directement. On dispose à cet effet de la fonction Seek(), grâce à laquelle on peut déplacer le pointeur à volonté vers l'avant ou vers l'arrière. Une position absolue est indiquée dans ce cas, comptée soit relativement à la position actuelle, soit encore par rapport au début ou à la fin du fichier.

Une autre fonction du DOS permet de supprimer un fichier sur une disquette : il s'agit de la fonction Delete(). Elle permet aussi de supprimer des sous-répertoires, à condition évidemment qu'ils soient vides.

Les noms des fichiers sont eux aussi modifiables, à l'aide de la fonction Rename(). On lui transmet simplement l'ancien et le nouveau nom. L'intéressant ici est que l'on peut non seulement changer le nom d'un fichier, mais aussi sa position sur la disquette. Si, en effet, on indique un autre chemin de recherche dans le nom, le fichier est "déplacé" (et non copié) dans ce nouveau sous-répertoire. Ceci ne fonctionne malgré tout que sur une disquette. Si vous cherchez à effectuer le déplacement vers une autre disquette, vous obtenez un message d'erreur.

Un fichier sur disquette peut de plus être protégé contre diverses fonctions. On détermine le type de protection à l'aide d'un masque, transmis à la fonction SetProtection(). Les 4 premiers bits du masque (0-3) indiquent si le fichier est protégé contre les actions suivantes :

Bit	Signification si le bit est positionné
0	On ne peut pas supprimer le fichier
1	On ne peut pas exécuter le fichier
2	On ne peut pas écrire dans le fichier
3	On ne peut pas lire dans le fichier

3.3.3. Interface sérielle

L'interface sérielle peut être traitée à peu près de la même façon que l'entrée/sortie écran. Un canal est ouvert, du nom de SER:, et on peut écrire ou lire dans ce canal. Mais trois problèmes peuvent surgir au cours de cette opération :

- ① Lors de l'appel de la fonction Read(), l'Amiga attend la réception d'un ou plusieurs caractères sur l'interface sérielle. Si ceux-ci ne viennent pas, il continue à attendre en vain, jusqu'à ce que l'écran devienne sombre. C'est pourquoi, dans un programme qui attend des données de cette interface, sans être absolument sûr qu'il en viendra, il vaut mieux faire précédé Read() de la fonction WaitForChar(). Grâce à cette fonction, l'attente est limitée à un temps déterminé (indiqué en microsecondes). Si rien ne vient durant ce laps de temps, on obtient un 0 en retour, et le programme peut afficher le cas échéant un message d'erreur avant de passer la main. Au contraire, lorsque les données attendues arrivent, on obtient en retour la valeur -1, et la lecture de ces données peut commencer.
- ② Les données sont réceptionnées, sans que l'on ait moyen de savoir quelle était leur quantité. Il peut alors se produire le problème décrit en 1). C'est la raison pour laquelle on ne peut pas prendre en considération à partir du CLI des données venant de l'interface sérielle par exemple avec COPY SER: TO *. Le CLI ne sait pas, en effet, à quel moment commence le flux de données et à quel moment il cesse. En conséquence de quoi, il se met en grève. Si l'on a lancé une telle commande, on n'a malheureusement qu'un seul moyen d'y mettre fin : le Reset.
- ③ Un programme peut vouloir envoyer ou recevoir par cette interface des données dont les réglages ne lui correspondent pas. On peut évidemment prévoir ces réglages avec le programme Preferences, et effectuer un nouveau démarrage. Ce qui est malgré tout assez pénible. Tout autant que Preferences, votre propre programme peut évidemment effectuer lui-même les réglages nécessaires. Cela n'est cependant pas possible à l'aide d'une simple commande DOS : il faut en passer par le Serial-Device à l'aide des fonctions I/O. Vous trouverez des commentaires là-dessus dans le chapitre correspondant.

3.3.4. Interface parallèle

La programmation de l'interface parallèle (PAR:) n'est pas nécessaire, car la plupart du temps, c'est l'imprimante qui y est connectée. Cette interface est cependant très intéressante, car on peut non seulement sortir par là des données, mais aussi en lire.

La façon la plus simple de programmer cette interface est le chemin direct par l'intermédiaire des registres du hardware. L'inconvénient de la méthode est qu'il risque alors de surgir des problèmes lors du fonctionnement multitâche, lorsqu'un autre programme cherche lui aussi à avoir accès à cette interface. C'est pourquoi il est plus sûr d'y accéder par l'intermédiaire du DOS. Dans ce cas, le format des données est lui aussi prescrit, et l'on ne dispose plus de la possibilité de programmer des bits un par un à l'entrée, et des bits différents à la sortie.

3.4. Programmes

Un programme créé par un éditeur de liens ou directement par un assembleur peut être lancé par la saisie de son nom dans le CLI. Si on veut le lancer à partir du Workbench, il faut en plus créer un fichier .INFO, contenant l'icône du programme dans la fenêtre Workbench. On peut alors cliquer sur cette icône, et lancer de cette façon le programme.

3.4.1. Lancement d'un programme et paramètres

Comme on le sait déjà pour les commandes CLI, on dispose sur Amiga de la possibilité d'indiquer quelques paramètres dans la ligne qui appelle le programme. Ce dernier reçoit ces paramètres et réagit en conséquence. C'est pourquoi il existe une différence nette en ce qui concerne la transmission des paramètres au programme entre le CLI et le Workbench.

Le programme appelé doit donc savoir à partir de quelle interface il a été lancé, et ensuite aller chercher éventuellement ses paramètres de la manière adéquate. Considérons pour cela tout d'abord le cas le plus simple, qui est le lancement d'un programme à partir du CLI.

3.4.1.1. Appel avec le CLI

Le programme qu'on a fait démarrer à partir du CLI reçoit dans deux registres affectés à cet usage les informations sur les paramètres, introduits éventuellement après le nom du programme. Dans le registre d'adresse A0 est transmise l'adresse de la ligne en mémoire, où se trouve le texte suivant le nom du programme dans la ligne entrée par le CLI. De plus, on obtient dans le registre de données D0 le nombre de caractères qui se trouvent réellement derrière ce nom de programme.

Avec ces deux informations, le programme peut facilement lire les paramètres et agir en conséquence. Pour le montrer sur un exemple, nous allons donner un programme en langage machine, qu'il est possible d'appeler avec et sans paramètre.

Il s'agit ici d'une commande CLI, que vous pouvez également copier dans le répertoire C. Il a la tâche de définir le mode de présentation des textes qui suivent l'appel. Vous pouvez par exemple le placer dans la séquence Startup, si vous désirez souligner ou mettre en italique un message.

Lorsque vous avez entreposé le programme dans le répertoire C sous le nom "Font", vous pouvez l'appeler avec la commande

>Font n

Le paramètre n peut être mis de côté, auquel cas le programme revient à la présentation normale.

Si vous l'indiquez, il faut que ce soit un chiffre entre 0 et 7. Voici les effets de ces chiffres :

0	Présentation normale
1	Gras
3	Italique
4	Souligné
7	inversé

Vous pouvez également choisir une présentation en gras et souligné, si vous sélectionnez l'un après l'autre la fonte 1 et la fonte 4. Voici le programme :

```
;***** Commande FONT *****
;
; Offsets Exec
OpenLib = -30-378
ExecBase = 4
;
; Offsets AmigaDOS
Write = -30-18
Output = -30-30
Exit = -30-114
;
run:
    subq    #1,d0          ;Nombre d'octets -1
    beq    normal           ;pas de paramètre ?
search:
    cmp.b   #$20,(a0)+      ;chercher l'argument
    bne    found             ;trouvé
    dbra   d0,search          ;mise en place du font normal
    bra    normal
;
found:
```

```

move.b -(a0),text+1      ;mise en place du style

normal:
  move.l execbase,a6      ; Pointeur sur la bibliothèque Exec
  lea    dosname,a1
  moveq #0,d0
  jsr    OpenLib(a6)      ;Open DOS-Library
  move.l d0,dosbase
  beq    error             ;pas réussi

  move.l dosbase,a6
  jsr    Output(a6)        ;Chercher Standard-Output-Handle

  move.l d0,d1              ;Output-Handle en D1
  move.l #text,d2            ;adresse du texte en D2
  move.l #tfin-text,d3       ;Longueur en D3
  move.l dosbase,a6
  jsr    Write(a6)           ;affichage du texte
  bra    fin                ;OK: fin

error:
  move.l #1,d0              ;Error-Status

fin:
  move.l d0,d1              ;Paramètre de retour
  move.l dosbase,a6
  jsr    exit(a6)            ;Fin du programme

  rts                         ;ne revient pas

dosbase: dc.l 0
dosname: dc.b 'dos.library',0
text:   dc.b $9b,'0;31;40m'
tfin:

even

```

Si vous voulez écrire un programme en C faisant intervenir ces paramètres, voici comment vous devrez vous y prendre. Il faut mettre en place le fichier 'startup.o' comme premier élément dans l'instruction du linker (dans le compilateur Lattice, en utilisant Aztec, cela se fait automatiquement en liant le fichier C.lib), ce que l'on fait d'ailleurs en règle générale. La ligne des paramètres se trouve alors dans la variable 'argv', et le nombre de caractères dans 'argc'.

La partie du programme correspondant au startup a d'autres capacités encore : elle permet d'ouvrir la bibliothèque DOS, et elle détermine le canal standard d'entrée/sortie, grâce aux fonctions DOS Input() et Output(). Les handles de ces canaux se trouvent alors dans 'stdin' et 'stdout'. La routine met ensuite en route la routine 'main' de votre programme C.

Une autre information transmise par le CLI au programme est la taille du secteur réservé sur la pile. Ce secteur se situe sur la pile derrière l'adresse de retour au CLI et peut être lu et chargé par exemple au moyen de la commande

MOVE.L 4(SP),D0.

De cette manière, le programme peut vérifier s'il y a assez de place ou non sur la pile pour ses besoins particuliers.

En dehors de ceux-ci, il y a encore quelques autres paramètres qui sont transmis à partir du CLI. Ils permettent de simplifier un programme CLI de multiples façons. Vous trouverez là-dessus d'autres détails dans le chapitre se rapportant aux commandes CLI externes. Nous venons donc de voir quelle était la façon de faire pour initialiser un programme mis en route à partir du CLI. Nous allons maintenant passer à l'autre cas : le démarrage à partir du Workbench.

3.4.1.2. Démarrage à partir du Workbench

Si vous cliquez deux fois sur l'icône d'un programme représenté dans la fenêtre Workbench, le programme en question sera mis en route. Il recevra des paramètres de la même façon que précédemment, mais pas sous la forme d'une ligne de texte : il les recevra par l'intermédiaire de messages.

Si le programme est écrit en C, et précédé du programme Startup, lié à lui à l'aide du linker, vous n'avez plus besoin de vous préoccuper de ce message que l'on appelle le message Startup. Le programme Startup se charge lui-même des tâches suivantes, lorsqu'il constate que le démarrage a été effectué à partir du Workbench :

- ① Il commence par ouvrir la bibliothèque DOS.
- ② Il attend le message Startup (WaitPort).
- ③ Il va chercher le message (GetMsg).
- ④ Il teste le nombre des arguments à l'intérieur du message. Si ce nombre est 0, il passe à l'étape suivante.
- ⑤ Les arguments transmis sont interprétés comme des structures Lock, et le répertoire correspondant est transformé en conséquence en répertoire actuel (CurrentDir).
- ⑥ L'argument sm_ToolWindow est vérifié ; s'il est différent de 0, la fenêtre indiquée est ouverte, et son handle est considéré comme entrée standard.

Quel sera l'aspect d'un programme qui ne dispose pas de ce confortable programme Startup ? Comment s'écrira par exemple un programme en langage machine ?

Même si vous n'avez pas besoin du message que le Workbench envoie à votre programme, il vous faut le rechercher malgré tout. Si vous ne le faites pas, le gourou entrera de nouveau dans de profondes méditations, autant dire dans une profonde perplexité, étant donné que lors de l'appel suivant d'une fonction I/O, par exemple l'ouverture d'une fenêtre, il verra arriver un message dans le Message-Port, et ce message n'aura aucun rapport avec la fonction en question.

Vous êtes donc obligé de faire exécuter par votre programme les mêmes fonctions que dans le programme Startup. Appelez d'abord la fonction FindTask() de l'Exec, pour obtenir un pointeur sur la structure du processus, c'est-à-dire de votre programme. Vous placerez par contre un 0 en A1 :

```
execbase = 4
FindTask = -294
WaitPort = -384
GetMsg = -372

move.l execbase,a6          ;Adresse de base Exec en A6
suba.l a1,a1                ;Effacer argument A1
jsr    FindTask(a6)         ;rechercher le pointeur
```

En D0, vous obtiendrez ainsi le pointeur sur votre structure de processus. Dans cette structure, vous trouverez l'information qui vous apprendra si ce processus a été mis en route à partir du CLI ou à partir du Workbench :

```
move.l d0,a4          ;Pointeur sur processus en A4
tst.l  $ac(a4)        ;pr_CLI: CLI ou Workbench ?
bne   fromCLI         ;c'était CLI !
```

Si l'argument testé est 0, le programme a été mis en route à partir du Workbench.

Si c'est bien le cas, la prochaine étape consiste à attendre la réception du message Startup. On y parvient à l'aide de la fonction WaitPort() :

```
lea    $5c(a4),a0      ;pr_MsgPort: MessagePort en A0
jsr    WaitPort(a6)    ;attendre le message
```

Cette fonction attend la réception d'un message en MessagePort. Dans notre cas, ce sera simplement le Startup_Message du Workbench. Ce message doit être relevé, pour qu'il ne figure plus dans la file d'attente des messages. On utilise pour cela la fonction GetMsg() :

```
lea    $5c(a4),a0      ;Adresse RastPort en A0
jsr    GetMsg(a6)       ;Aller chercher le message
```

Vous pouvez faire usage de ce message, si vous en avez besoin. En D0, vous obtenez un pointeur sur la structure de message, par la fonction GetMsg() ; cette structure porte le nom de WBStartup.

Le message contient les éléments suivants :

On trouve au début une structure de message normale. Puis viennent les éléments de la structure Startup proprement dite :

Offset	Nom	Signification
\$14	sm_Process	Descripteur de processus
\$18	sm_Segment	Descripteur de segment programme

Offset	Nom	Signification
\$1C	sm_NumArgs	Nombre d'arguments transmis
\$20	sm_ToolWindow	Description de la fenêtre à ouvrir
\$24	sm_ArgList	Pointeur sur les arguments

sm_Arglist

Pointe sur les éléments des arguments transmis. Ces arguments contiennent les informations sur l'icône du Workbench activée au moment de la mise en route du programme. Il y a des programmes qui utilisent ceci de telle façon qu'un fichier texte activé en supplément par Shift-clic au moment de l'appel du programme soit chargé et affiché par les soins du programme. Les arguments de la liste sur laquelle pointe sm_ArgList sont constitués des pointeurs suivants :

wa_Lock	Lock fichier (description de répertoire)
wa_Name	Pointeurs sur les noms de fichiers

Pour donner un exemple qui montre comment on applique et comment on programme ce traitement des messages, nous allons écrire un programme qui transmet et qui affiche les tool-types. Ceux-ci sont les mentions qui peuvent être portées dans le programme Workbench INFO, à l'intérieur des différents fichiers. Vous sélectionnez pour cela un fichier (cliquez une fois), puis le point Info dans le menu Workbench. Vous voyez s'ouvrir une fenêtre de dialogue, dans laquelle vous pouvez introduire des données à l'intérieur du masque de saisie TOOL TYPES, en cliquant sur Add. Ces données sont utilisées par certains programmes en tant que réglages préalables (par exemple par Notepad).

Les données en question sont mémorisées dans le fichier .info qui appartient au programme. Dans ce fichier se trouvent en outre les données pour l'icône, sa position dans la fenêtre, et bien d'autres choses encore. Pour reporter ces données dans un programme, on dispose d'une autre bibliothèque sur la disquette Workbench, dans le répertoire LIBS : l'Icon-library.

Cette bibliothèque contient des fonctions servant à traiter les fichiers .info. L'une de ces fonctions est GetDiskObject(), qui charge le fichier .info, et retourne un pointeur sur sa structure. Notre programme utilise aussi cette fonction. Avant d'entrer dans les détails de l'Icon-library et de la structure DiskObject, je vais vous présenter ce programme, pour que les choses soient plus claires :

```
;** Message Workbench et traitement de l'info S.D. **

execbase = 4           ;Adresse de base Exec
FindTask = -294        ;chercher Task
WaitPort = -384         ;attendre le message
GetMsg = -372          ;aller prendre le message

OpenLib = -408          ;ouvrir Library
CloseLib = -414         ;fermer Library
Open = -30              ;ouvrir canal
```

```

Close    - -36          ;fermer canal
Read     - -42          ;lire les données
Write    - -48          ;afficher les données
CurrentDir - -126       ;directory actuel
mode_old - 1005         ;mode d'ouverture

GetDiskObject - -78      ;Charger le diskobject

run:
    move.l execbase,a6      ;Adresse de base Exec
    suba.l a1,a1
    jsr    FindTask(a6)      ;chercher le task propre
    move.l d0,a4              ;Pointeur en A4
    tst.l $ac(a4)            ;pr_CLI: CLI ou Workbench ?
    bne    fromCLI           ;CLI ! fin...
    lea    $5c(a4),a0          ;MessageWBench
    jsr    WaitPort(a6)        ;attendre
    jsr    GetMsg(a6)          ;aller chercher le message
    move.l d0,message          ;sauvegarder le pointeur

; ***** Ouvrir bibliothèques et fenêtres *****
    lea    iconname,a1          ;"icon.library"
    clr.l d0
    jsr    OpenLib(a6)          ;ouvrir ICON.library
    move.l d0,iconbase          ;sauvegarder la base
    beq    fin3                ;Une erreur est survenue !
    lea    dosname,a1          ;"dos.library"
    clr.l d0
    jsr    OpenLib(a6)          ;ouvrir DOS
    move.l d0,dosbase          ;sauvegarder la base
    beq    fin2                ;Une erreur est survenue !

    move.l d0,a6
    move.l #connname,d1
    move.l #mode_old,d2
    jsr    Open(a6)             ;Ouvrir la fenêtre CON:
    move.l d0,conbase          ;sauvegarder la base
    beq    fin1                ;Une erreur est survenue !

;***** Mettre en place le répertoire actuel, si nécessaire *****
    move.l message,a0          ;Pointeur sur WBMessag
    move.l $24(a0),a0          ;sm_ArgList: Pointeur sur arguments
    beq    fin                  ;pas d'arguments !
    move.l (a0),d1              ;D1 -> Lock
    move.l dosbase,a6
    jsr    CurrentDir(a6)       ;mettre en place le directory actuel

; ***** Charger le disk-object (fichier .info) *****

```

```

move.l message,a0
move.l $24(a0),a0      ;Pointeur sm_ArgLis
move.l 4(a0),a0         ;wa_Name: Pointeur sur nom
move.l iconbase,a6
jsr    GetDiskObject(a6) ;charger le disk-object

; ***** Afficher dans la fenêtre les entrées tool-type *****

move.l d0,a1            ;Pointeur sur structure DiskObject
move.l $36(a1),a1        ;do_ToolTypes: Pointeur sur
                        ;ToolType-Array
move.l a1,typetext       ;sauvegarder pointeur texte

typesloop:
move.l typetext,a1       ;charger pointeur texte
move.l (a1)+,a0           ;Pointeur sur Text en A0
cmp.l #0,a0               ;le texte existe ?
beq    nomore             ;non : fin des sorties
move.l a1,typetext       ;sinon sauvegarder le pointeur

move.l a0,d2               ;- adresse du texte pour affichage
move.l a0,d3               ;obtenir encore la longueur
lenloop:
tst.b (a0)+                ;rechercher la fin
bne    lenloop
sub.l a0,d3                 ;calculer longueur du texte
not.l d3                     ;et corriger

move.l dosbase,a6
move.l conbase,d1
jsr    Write(a6)           ;afficher le texte dans la fenêtre

move.l conbase,d1
move.l #1f,d2               ;Linefeed:
move.l #1,d3
jsr    Write(a6)           ;ligne suivante

bra   typesloop             ;passer à l'entrée suivante !

; ***** C'était tout, maintenant attendre les touches *****

nomore:
move.l conbase,d1
move.l #1,d3                 ;un caractère
move.l #buffer,d2            ;dans le buffer
jsr    Read(a6)              ;lire (attendre Return)

; ***** Fin du programme: tout fermer et retour *****

fin:   move.l conbase,d1
      move.l dosbase,a6
      jsr    Close(a6)        ;fermeture de la fenêtre
fin1:
move.l execbase,a6
move.l dosbase,a1
jsr    Closelib(a6)         ;fermer le DOS
fin2:
move.l iconbase,a1
jsr    Closelib(a6)         ;fermer ICON.library

```

```

fromCLI:
fin3:
    rts           ;fin du programme

; **** Champs de données ****

dosbase: blk.l 1      ;adresse de base DOS
conbase: blk.l 1      ;base de la fenêtre

iconbase: blk.l 1      ;base d'icon.library
message: blk.l 1       ;Pointeur sur WBMessage
typetext: blk.l 1      ;Pointeur texte

dosname: dc.b 'dos.library',0
iconname: dc.b 'icon.library',0
connname: dc.b 'CON:10/20/300/100/** Affichage du message'.0
lf:      dc.b $a
buffer:  blk.b 2

```

Ce programme ne fonctionne que s'il est mis en route à partir du Workbench. Sinon, il est tout simplement interrompu (fromCLI). Pour le mettre en route, il reste à créer une icône pour ce programme. Vous y parviendrez très facilement à l'aide de l'éditeur d'icônes. Il faut que l'icône soit ensuite sauvegardée sous le même nom que le programme ci-dessus, et l'appendice .info est créé automatiquement.

Une fois que vous avez fait cela, vous pouvez cliquer sur l'icône, et sélectionner le point Info dans le menu Workbench. Dans la fenêtre qui s'ouvre alors, vous pouvez saisir une ou plusieurs données destinées à TOOL TYPES, puis sauvegarder avec SAVE.

Lorsque vous activez ensuite l'icône en cliquant dessus deux fois, le programme correspondant est chargé et mis en route. Le programme exécute les pas nécessaires, pour obtenir et traiter comme il se doit aussi bien le message Workbench Startup que la structure DiskObject.

La structure du DiskObject ou encore du fichier .info est constituée de la façon suivante :

Offset	Nom	Contenu
0	do_Magic	"Nombre magique" qui déclare que le fichier est valide (\$E310)
2	do_Version	Numéro de version (1)
4	do_Gadget	Début d'une structure gadget, qui fixe l'aspect et la position de l'icône
\$30	do_Type	Type d'objet (Tool, projet, etc.)
\$32	do_DefaultTool	Programme standard de la disquette
\$36	do_ToolTypes	Pointeur sur le champ texte des types
\$3A	do_CurrentX	
\$3E	do_CurrentY	Position de l'icône dans la fenêtre

Offset	Nom	Contenu
\$42	do_DrawerData	Pointeur sur la structure de la fenêtre du sous-répertoire
\$46	do_ToolWindow	Fenêtre standard pour les tools
\$4A	do_StackSize	Taille de la pile pour les tools

Le pointeur do_ToolTypes pointe sur une liste de pointeurs, qui pointent à leur tour sur les textes des Tool_Types inscrits dans la fenêtre Info et qui se terminent par un 0. Les pointeurs de cette liste sont utilisés dans le programme pour afficher les textes.

Le programme donné en exemple montre donc comment l'on accède aux Tool_Types du programme. On peut y porter des données fondamentales servant à contrôler la fonction du programme. C'est ce qui se passe pour le programme Notepad du Workbench, où les Tool_Types permettent de fixer des paramètres comme la taille de la fenêtre de saisie ou le type de caractères adoptés. Ces données figurent habituellement sous la forme suivante:

NOM=<Paramètre>[| <Paramètre>]

C'est ce que l'on doit faire également pour Notepad. L'avantage de cette forme réside simplement dans le fait qu'il y a deux fonctions prévues dans la bibliothèque d'icônes et pouvant tester ces lignes.

La première de ces fonctions est FindToolType(), offset -96 ; elle parcourt les données des Tool_Types à la recherche d'un nom déterminé. Dans l'exemple de Notepad, c'est une ligne avec le nom WINDOW qui est recherchée. On obtient en retour un pointeur sur les paramètres correspondant au signe d'égalité, ou alors un 0 lorsqu'aucune ligne n'apparaît avec ce nom.

Ce pointeur est alors transmis à une autre fonction, MatchToolType(), offset -102, en même temps qu'un nouveau pointeur sur un paramètre de comparaison. La valeur qui en résulte permet de savoir si le paramètre de comparaison intervient ou non dans la ligne.

On utilise ce procédé par exemple lorsqu'un programme est en mesure de lire des fichiers, mais n'a le droit de lire que des fichiers d'un type déterminé. On reporte les critères correspondants dans Tool_Types, et ils seront ensuite comparés par le programme avec le type du fichier à charger, pour que le programme sache s'il a le droit de les charger ou non.

3.4.2. Structures d'un fichier programme

Dans l'Amiga, un programme comprend trois segments logiques : code, données et Bss. Nous allons ici envisager la structure interne des fichiers programme dans Amiga.

3.4.2.1. Segments de programme

Le segment du code contient les commandes du programme, c'est-à-dire le programme proprement dit. Dans le segment des données, on trouve les données dont le programme a besoin et qui doivent avoir un contenu déterminé. Ces données sont contenues directement dans le segment du code, de telle sorte que le segment des données manque. Cela ne pose pas de problème particulier. La division en code et données a toutefois l'avantage de permettre au chargeur (Loader) d'emmageriser ces deux segments en des points différents de la mémoire, là où il y a de la place. Lorsqu'on a affaire à un segment de code tout d'une pièce, ce n'est pas si facile, si la mémoire ne dispose pas d'un secteur libre assez grand en un seul morceau.

Le troisième segment est le Bss. Il est simplement destiné à recevoir des secteurs de données, dont l'état initial ou le contenu sont indifférents. Le loader réserve alors pour le programme un secteur de mémoire à un emplacement quelconque, tout en conservant la longueur exigée dans le programme sous forme de secteur Bss (hunk_bss, cf. ci-dessous).

3.4.2.2. Structure d'un programme (hunks)

Un programme n'est en réalité rien d'autre qu'une série de mots-données en binaire, qui constituent dans leur ensemble un programme en langage machine. Sur les bons vieux ordinateurs à 8 bits, la mémorisation de ces programmes ne posait aucun problème : il suffisait de copier en mémoire centrale le programme se trouvant sur la disquette et le tour était joué.

Sur une machine comme l'Amiga, cette méthode est impraticable. Le premier problème qui surgirait concermerait en effet la division de la mémoire. Si la partie de la mémoire où le programme a fonctionné la première fois est occupée lors de l'appel suivant, un simple chargement du programme sans autre forme de procès n'est plus possible. Le programme doit être chargé en un autre point de la mémoire, ce qui signifie que rien ne va plus, puisqu'un programme en langage machine qui se respecte n'utilise que des adresses absolues.

Le problème est résolu par Amiga de la façon suivante : un programme est emmagasiné sur la disquette de telle façon que toutes les adresses absolues qui y sont contenues soient décalées (elles sont en fait interprétées comme adresses relatives par rapport à une adresse de base \$00000). Ainsi, si le programme est chargé à partir de \$20000, il faut que toutes les adresses qu'il contient soient corrigées avant le démarrage, c'est-à-dire augmentées de \$20000. Pour que le DOS, qui se charge de cette opération, puisse retrouver dans le programme toutes les adresses à modifier, on mémorise une table en même temps que le programme proprement dit. Cette table contient les offsets pointant sur les mots longs à modifier.

Il est donc clair qu'une seule section ne saurait suffire à emmagasinier un programme sur une disquette. Jusqu'à présent, nous sommes en présence de deux sections : le programme lui-même et la table de redisposition (relocation table). Mais il y a toute une

série d'autres sections utilisées par l'Amiga. Une partie de programme composée de ces différentes sections est appelée "hunk". Un ou plusieurs "hunks" constituent ensemble une unité de programme (program unit), et plusieurs de ces unités forment un "object file", un fichier objet. En dernier lieu, un ou plusieurs fichiers objets constituent un "load file", un fichier de chargement, qui représente un programme en état de marche.

La différence entre ces deux types de fichiers ("object file" et "load file") est la suivante : un fichier objet contient un programme qui n'est pas encore en état de fonctionner, tel qu'il a été créé par exemple par un compilateur ou un assembleur. Si l'on veut transformer un ou plusieurs de ces fichiers objets en un programme qui soit en mesure de fonctionner, il faut appeler l'éditeur de liens (linker). C'est un programme qui rassemble les fichiers objets pour en faire un programme unique, qui est lui-même ensuite emmagasiné en tant que fichier de chargement, ou load file. Le résultat peut alors être utilisé, si on le fait démarrer par exemple par la saisie de son nom sous le CLI.

L'avantage de ce détour est que des parties de programmes qui risquent de s'appeler réciproquement sont créées et traduites indépendamment l'une de l'autre. Le programme principal, écrit par exemple en C et contenant la routine "main", peut simplement appeler les fonctions ou les sous-programmes contenus dans les autres fichiers.

Le fait que les fonctions soient entreposées hors du programme principal améliore la vue d'ensemble que l'on peut avoir du programme, puisque celui-ci est nettement plus court qu'un programme contenant toutes les composantes développées l'une à la suite de l'autre.

On comprend maintenant pourquoi les programmes pris un à un ne peuvent pas fonctionner. On doit appeler des parties de programme qui ne sont pas explicitement contenues dans ces programmes. Ce n'est qu'à l'aide de l'éditeur de liens que toutes les fonctions et tous les sous-programmes sont réunis dans un fichier programme unique.

Nous allons cependant commencer par les plus petites sections qui constituent les hunks. Quelques-unes de ces parties de fichiers programmes apparaissent seulement dans les fichiers objets, quelques autres seulement dans les fichiers de chargement. Elles commencent toutes par un mot long déterminé, indiqué entre parenthèses dans le tableau qui suit en hexadécimal.

Voici cette liste de toutes les parties (hunks) possibles :

hunk_unit (\$3E7)

Début d'une unité de programme dans les fichiers objets. La caractérisation \$3E7 est suivie de la longueur du nom de cette unité, puis du nom lui-même, qui doit se terminer sur une fin de mot long.

hunk_name (\$3E8)

Ici se trouve le nom du hunk. La caractérisation \$3E8 est suivie de la longueur du nom, puis du nom lui-même, qui doit se terminer sur une fin de mot long.

hunk_code (\$3E9)

Contient une partie de programme pouvant fonctionner après la correction des adresses absolues. Ici encore, la caractérisation \$3E9 est suivie du nombre de mots longs dans le programme, puis de ces mots longs eux-mêmes.

hunk_data (\$3EA)

Ici aussi commence une partie de programme, ne contenant cependant que des données avec un contenu déterminé (data). Quelques-unes d'entre elles peuvent exiger une correction d'adresse. La caractérisation est suivie du nombre de données puis des données elles-mêmes.

hunk_bss (\$3EB)

Les données de cette partie appartiennent certes elles aussi au programme lui-même, mais elles n'ont pas cette fois de contenu déterminé. C'est pourquoi la caractérisation n'est suivie que du nombre des mots longs nécessaires, mais sans que les données soient répertoriées (bss = block storage segment).

hunk_reloc32 (\$3EC)

Ce bloc contient les offsets qui pointent sur les mots longs contenant les adresses à corriger à l'intérieur du programme. Ces offsets sont valables pour l'ensemble du programme. La distribution du bloc est la suivante :

La caractérisation \$3EC est suivie du nombre d'offsets contenus dans le premier tableau. Le mot long suivant désigne le numéro du hunk auquel ces offsets se réfèrent ; puis viennent les offsets eux-mêmes. Le mot long suivant est de nouveau un nombre ; puis vient le numéro du hunk de ce tableau, et ainsi de suite, jusqu'à ce qu'apparaisse le nombre 0, qui conclut cette partie de hunk.

Voici le diagramme d'ensemble :

- * \$3EC (hunk_reloc32)
- * Nombre d'Offsets
- * Numéro de Hunk
- * Offsets...
- * Nombre d'Offsets (ou 0= fin)
- * Numéro de Hunk
- * Offsets...
- .

* 0: Fin de hunk_reloc32

De cette manière, ce tableau recouvre tous les hunks qui constituent le programme contenu finalement en mémoire et prêt à fonctionner.

hunk_reloc16 (\$3ED)

Ce tableau est constitué comme hunk_reloc32, à ceci près que ces offsets se réfèrent à des adresses 16 bits. Ce type d'adresses apparaît dans les adressages relatifs aux PC.

hunk_reloc8 (\$3EE)

A également le même format que hunk_reloc32. Les offsets contenus dans celui-ci sont utilisés pour des adresses 8 bits, qui apparaissent également dans les adressages relatifs aux PC.

hunk_ext (\$3EF)

Dans ce bloc sont reportés les noms des références externes. De telles références n'apparaissent que dans les fichiers objets. Il s'agit des adresses de fonctions ou de sous-routines, qui ne sont pas connues du programme et qui doivent être introduites par le linker.

La caractérisation est suivie de plusieurs "symbol data units", conclue par un mot contenant la valeur 0. Ces définitions de symboles ont la structure suivante :

1 octet : type du symbole. Les différentes possibilités sont les suivantes :

Valeur	Nom	Type de symbole
0	ext_symb	Table de symboles pour recherche d'erreurs
1	ext_def	Définition à corriger
2	ext_abs	Définition absolue
3	ext_res	Relation à la bibliothèque résidente
129	ext_ref32	Correction 32 bits
130	ext_common	Correction générale 32 bits
131	ext_ref16	Correction 16 bits
132	ext_ref8	Correction 8 bits

- ✓ Une valeur en 3 octets pour la longueur du nom (en mots longs).
- ✓ Nom du symbole.

✓ Valeur du symbole et éventuellement d'autres données.

Les données qui suivent le nom ont trois types différents de structure dans ces hunks, en fonction du type de symbole. Pour les types _def, _abs et _res, le dernier mot long est simplement suivi de la valeur absolue du symbole comme mot long.

Le nom des trois types _ref est suivi en revanche d'un mot long, qui contient le nombre des valeurs de référence qui viennent ensuite.

La même chose vaut pour ext_common, à ceci près qu'ici se trouve encore la taille du bloc Common entre le nom et le compteur.

hunk_symbol (\$3F0)

Dans ce bloc, on trouve encore des symboles avec leur nom et leur valeur. Ceux-ci n'intéressent pas le linker, mais un débogueur, c'est-à-dire un programme servant à rechercher les erreurs dans les programmes. On peut ainsi tester une routine dont l'adresse n'est pas disponible en tant que nombre mais en tant que nom, et aussi les emplacements en mémoire des variables. La caractérisation \$3F0 est suivie à nouveau des symbol-data-units, et est conclue par un 0.

hunk_debug (\$3F1)

La structure de ce bloc n'est pas prescrite entièrement. On peut inscrire ici des informations sur le programme, et le programme les aura à sa disposition lorsqu'il s'agira de rechercher des erreurs. La seule prescription est que le bloc doit commencer par la caractérisation \$3F1, à la suite de quoi doit venir le nombre de mots longs constituant le bloc.

hunk_end (\$3F2)

Celui-ci est le seul bloc, parmi tous les hunks, qui soit absolument nécessaire. Il est constitué de sa seule caractérisation, qui est également le dernier mot long d'un programme qui se trouve sur la disquette.

hunk_header (\$3F3)

Ce bloc est le début d'un load file. On indique ici le nombre de hunks qui composent le programme à charger, et la dimension de chacun d'eux. Le bloc contient en outre les noms des bibliothèques résidentes, qui doivent être chargées en même temps que ce programme. Le bloc est constitué de la façon suivante :

- hunk_header (\$3F3)
- longueur (nombre de mots longs) du nom du premier hunk
- nom du hunk
- longueur du nom du deuxième hunk (ou 0 - fin)

- nom du hunk
-
-
-
- 0: fin de la liste des noms
- numéro de hunk le plus élevé+1: longueur de la table
- numéro du hunk à charger en premier
- numéro du hunk à charger en dernier
- ici commencent les hunks du programme

hunk_overlay (\$3F5)

Ce bloc est nécessaire lorsque l'on veut travailler en overlay. Cela signifie que, dans un domaine de la mémoire déjà occupé par un programme, on veut charger un autre segment de programme ou de données. Après la caractérisation, la table contient les indications sur sa longueur, sur le niveau le plus élevé des recouvrements (le nombre de processus de recouvrement), et sur les données à charger.

hunk_break (\$3F6)

Ce numéro d'identification indique à lui seul la fin d'une partie de programme en overlay.

Pour que nous apprenions à nous y retrouver dans ce dédale de la distribution des programmes, voici maintenant un petit exemple. Dans le chapitre sur le CLI, je vous ai présenté un petit programme en langage machine, qui correspondait à la commande FONT en CLI. Vous vous rendrez facilement compte de la façon dont ce programme (à partir de l'assembleur SEKA) est amené sur la disquette, en demandant l'affichage du contenu du fichier 'Font' à l'aide de la commande TYPE en CLI. L'instruction est la suivante :

```
type Font opt h
```

pour obtenir le résultat à l'écran,

```
type from font to prt : opt h
```

pour l'obtenir sur l'imprimante. A la sortie écran ou imprimante, voici ce que vous aurez :

0000:	000003F3	00000000	00000002	00000000
0010:	00000001	00000024	00000001	/ 000003E9
0020:	00000024	/ 53406700	00180C18	00206600
0030:	000A51C8	FFF66000	000813E0	00000083
0040:	2C790000	000443F9	00000076	70004EAE
0050:	FE6823C0	00000072	67000028	2C790000
0060:	00724EAE	FFC42200	243C0000	0082263C
0070:	00000009	2C790000	00724EAE	FFD06000
0080:	0008203C	00000001	22002C79	00000072
0090:	4EAEFF70	4E750000	0000646F	732E6C69
00A0:	62726172	79009B30	3B33313B	34306D00
00B0:	A379A379	/ 000003EC	00000007	00000000

00C0:	00000018	00000024	00000030	0000003A	
00D0:	00000046	00000052	00000068	00000000	/
00E0:	000003F2	/ 000003EB	00000001	000003F2	

Les barres obliques ne sont pas affichées ou imprimées ; elles ne servent ici qu'à séparer les différentes sections, c'est-à-dire les parties de hunks. Considérons-les plus en détail :

Au début, on trouve le numéro caractéristique \$3F3, hunk_header. Le \$0 qui suit signifie qu'il n'y a pas de nom de hunk. Puis le \$2 indique que ce fichier programme ne se compose que de 2 hunks (comme nous l'avons dit, ce n'était qu'un petit exemple). Le premier hunk à charger est le numéro \$0, le dernier est le numéro \$1. Les tailles respectives de ces hunks sont \$24 et \$1.

Avec le numéro caractéristique \$3E9 (hunk_code) commence ensuite la partie dans laquelle se trouvent les données des programmes. La longueur est indiquée par \$24. Puis viennent \$24 (36) mots-longs, qui constituent le code programme.

En \$3EC (hunk_reloc32) commence le domaine contenant les offsets destinés à la correction des adresses. Le nombre d'adresses à corriger est indiqué par \$7, et ces offsets se rapportent au hunk numéro \$0. Puis viennent les 7 offsets. Le premier d'entre eux, \$18, porte la valeur \$0000007F=024, le 24ème mot du code. C'est donc à ce mot long que sera ajoutée, après le chargement du programme, son adresse de début, de telle sorte qu'on y trouve l'adresse effective en mémoire de l'octet adressé. Le procédé est le même pour les autres offsets de la liste reloc32.

La liste est suivie d'un \$0, qui indique la fin de cette liste. La caractérisation suivante, \$3F2 (hunk_end) note la fin du premier hunk. Puis vient le second, dont la longueur était d'une unité.

Vient ensuite le numéro \$3EB (hunk_bss), suivi par le nombre \$1. Ceci signifie qu'il faut réserver un mot long supplémentaire, dans lequel le programme placera l'adresse de base du DOS.

L'ensemble est conclu par le numéro \$3F2 (hunk_end), qui représente la fin du second hunk et donc également du programme tout entier.

Pour pouvoir analyser cette distribution d'un programme ou d'un fichier objet, il faut donc toujours réaliser l'expression hexadécimale proposée, et rechercher les différents hunks ou leur header. Pour automatiser ce travail, mais également pour donner un exemple de la façon dont sont traités les hunks, voici maintenant un petit programme qui recherche un fichier et présente les hunks qui y sont contenus à l'intérieur d'une fenêtre. Ce programme est appelé à partir du CLI avec indication d'un nom de fichier. Si vous nommez donc le programme HUNKS, vous pouvez vérifier les indications données plus haut sur le programme Font, en entrant :

```
Hunks c:Font
```

Le programme Hunks ouvrira par la suite une fenêtre, affichera le nom de fichier transmis et ouvrira le fichier indiqué. Puis sont affichés l'un en dessous de l'autre les noms des hunks trouvés, et le programme se termine par les mots "Veuillez presser Retum". Vous pouvez évidemment obtenir l'affichage sur le périphérique de sortie

standard à l'aide de la fonction OUTPUT de DOS, le handle du périphérique étant donné par la fonction elle-même. Dans ce cas, vous pouvez également dériver les sorties de ce programme par exemple sur une imprimante.

Si les hunks hunk_ext surgissent dans le fichier, il y en a souvent toute une série. Pour empêcher dans ce cas que le reste de se perdre à cause de l'affichage de -zig "hunk_ext", ou de défiler hors de l'écran, on n'obtiendra lors des répétitions que des points à côté de l'indication. Voici le programme, dans lequel vous pourrez reconnaître clairement la distribution des divers hunks :

```
;***** Analyse des hunks dans un fichier; 5/88 S.D. *****

OpenLib    --408
closelib   --414
ExecBase   -4

Open      --30
Close     --36
Read      --42
Write     --48
Seek      --66
mode_old  -1005

lf  -$a

run:
clr.b  -1(a0,d0)
subq #1,d0
beq   ret
move  d0,length
search:
cmp.b  #$20,(a0)+
bne   found
dbra  d0,search
bra   ret

found:
subq.l #1,a0
move.l a0, fname
move.l sp,spsave
clr   ext_nr

move.l execbase,a6
lea    dosname(pc),a1
moveq #0,d0
jsr   openlib(a6)
move.l d0,dosbase
beq   error

move.l #consolename,d1
move.l #mode_old,d2
move.l dosbase,a6
jsr   open(a6)
beq   error
move.l d0,conhandle
```

```

move.l fname,d1           ;File-Name
move.l #mode_old,d2
jsr    open(a6)           ;ouvre un fichier
beq    error
move.l d0,fhandle

move.l fname,d2
move   length,d3
move.l d2,a0
move.b #1f,0(a0,d3)
move.b #1f,1(a0,d3)
addq   #2,d3
move.l conhandle,d1
jsr    Write(a6)          ;affiche un nom de fichier

loop:
bsr    readlw
move.l (a1),d0
du hunk
sub.l #$3e7,d0
move.l d0,hunknr          ;à partir de 0
                           ;sauve le numéro

mulu  #13,d0
move.l #names,d2
add.l d0,d2
move.l #13,d3 move.l conhandle,d1
jsr    Write(a6)          ;affiche le nom du hunk

move.l hunknr,d0
bmi   error               ;ERROR: Nr. trop petit ? ? ?
cmp.l #14,d0
bgt   error               ;ERROR: Nr. trop grand ? ? ?

bsr    next                ;vers le mot long suivant
move.l hunknr,d0
lsl.l #2,d0
lea    jumps,a0
lea    0(a0,d0),a0
move.l (a0),a0
jsr    (a0)                ;pour la routine: sauter le Hunk

bra   loop

error:
move.l #failed,d2
move.l #enter-failed,d3
move.l conhandle,d1
jsr    Write(a6)          ;"Une erreur est survenue !"

end:  move.l spsave,sp      ;cherche l'ancien SP

move.l #enter,d2
move.l #dot-enter,d3
move.l conhandle,d1
jsr    Write(a6)          ;"Presser Return"

move.l conhandle,d1
move.l #puffer,d2         ;adresse du tampon

```

```

move.l #1,d3                      ;1er caractère
move.l dosbase,a6
jsr    read(a6)                   ;Lecture

move.l fhandle,d1
move.l dosbase,a6
jsr    close(a6)                 ;ferme le fichier

move.l conhandle,d1
jsr    close(a6)                 ;ferme la fenêtre

move.l dosbase,a1
move.l execbase,a6
jsr    closelib(a6)              ;ferme DOS.Lib
rts

seek_it:
bsr    seek_it2                  ;modifie le pointeur readlw:
move.l fhandle,d1
move.l #puffer,d2
move.l #4,d3                      ;lit un mot long
move.l dosbase,a6
jsr    Read(a6)
tst.l d0
beq    ende                       ;EOF
move.l #-4,d2                     ;et rétablit le pointeur

seek_it2:
fhandle,d1                        ;modifie le pointeur move.l
move.l #0,d3                      ;offset_current
move.l dosbase,a6
jsr    Seek(a6)
lea    puffer,a1

ret:
rts

; ***** Evaluation des hunks *****

name:                                ;hunk-noms
lws:                                  ;hunk-données
  move.l (a1),d2                    ;nombre dans D2
  lsl.l #2,d2                      ;fois 4
  bsr    seek_it                  ;définit le pointeur

next:                                 ;Pointeur sur prochain LW
  move.l #4,d2
  bra    seek_it

reloc:                                ;hunk_reloc
  move.l (a1),d2
  beq    next                      ;pas d'autres Offsets
  lsl.l #2,d2                      ;sinon nombre fois 4
  addq.l #8,d2                      ;plus 2 LW
  bsr    seek_it                  ;chercher
  bra    reloc                     ;etc...

sdu:                                  ;Symbol Data Unit (seulement
Object-Files)
  move.b (a1),d1                   ;ext Type

```

```

move.l (a1),d0
beq    next           ;Fin de Ext
and.l #$fffff,d0      ;supprime le masque pour la longueur du nom
bsr    lws             ;saute le nom

cmp.b #130,d1          ;ext_common ?
beq    common          ;oui !
cmp.b #3,d1            ;ext_def/abs/res ?
bgt    ref             ;Non: ext_ref

move.l #ext_types,d2
moveq #1,d4             ;Type 1
bsr    pr_ext
bra    sdu             ;prochain SDU...

ref:
move.l #ext_types+19,d2
moveq #2,d4             ;Type 2
bsr    pr_ext
refl:
move.l (a1),d0
bsr    lws             ;sauter les références
bra    sdu             ;prochain SDU...

common: move.l #ext_types+38,d2
moveq #3,d4             ;Typ 3
bsr    pr_ext
bsr    next             ;sauter la taille du Common-Block
bra    refl

pr_ext:
move.l #19,d3
cmp.b ext_nr,d4         ;même type ?
bne    nodot           ;non
move.l #dot,d2          ;prépare l'affichage de "."
move.l #1,d3

nodot:
move.b d4,ext_nr
move.l conhandle,d1
jsr    Write(a6)         ;"- ext_xxx" ou "."
move.l #puffer,a1
rts

header:
move.l (a1),d2           ;hunk_header
beq    tabl             ;pas d'autre nom
ls1.l #2,d2
add.l #4,d2
bsr    seek_it
bra    header           ;sinon sauter les noms
;etc...

tabl:
moveq #12,d2
bsr    seek_it
move.l (a1),d2           ;Pointeur sur 'Last Hunk'
ls1.l #2,d2
bsr    seek_it
bsr    next             ;sauter les tailles des hunks
bra    next

```

```

consolname: dc.b 'RAW:140/0/500/200/** Tableau des hunks **',0
dosname:   dc.b 'dos.library',0
failed:    dc.b lf.lf.'!! Une erreur est survenue!!'
enter:     dc.b lf.lf.'** Veuillez presser Return! **'
dot:       dc.b "."

names:  dc.b lf."hunk_unit      "
dc.b lf."hunk_name      "
dc.b lf."hunk_code      "
dc.b lf."hunk_data      "
dc.b lf."hunk_bss       "
dc.b lf."hunk_reloc32"
dc.b lf."hunk_reloc16"
dc.b lf."hunk_reloc8 "
dc.b lf."hunk_ext       "
dc.b lf."hunk_symbol    "
dc.b lf."hunk_debug     "
dc.b lf."hunk_end       "
dc.b lf."hunk_header    "
dc.b lf."hunk_overlay   "
dc.b lf."hunk_break     "

ext_types: dc.b lf."- ext_def/abs/res "
dc.b lf."- ext_ref32/16/8 "
dc.b lf."- ext_common    "
even

jumps:   dc.l name.name,lws,lws,next
dc.l reloc,reloc,reloc,sdu
dc.l sdu,lws,ret,header,lws,ret

data                                ;début du secteur bss

ext_nr:    dc.w 0
spsave:    dc.l 0
dosbase:   dc.l 0
conhandle: dc.l 0
fhandle:   dc.l 0
hunknr:    dc.l 0
fname:     dc.l 0
length:   dc.w 0
puffer:   dc.l 0

```

3.4.3. Un exemple de handler DOS

```

#include "exec/types.h"
#include "libraries/dos.h"
#include "libraries/dosextens.h"
#include "libraries/filehandler.h"

#define CR          printf ("\n")           /* Carriage Return*/
#define EnvecSize   (DosEnvec->de_TableSize)

extern struct DosLibrary *DOSBase;

```

```

struct DeviceList *DeviceList;
struct DeviceNode *DeviceNode;

struct RootNode *RootNode;
struct DosInfo *DosInfo;

struct FileSysStartupMsg *FSSM;

struct DosEnvec
{
    ULONG de_TableSize;      /* Taille du vecteur d'environnement */
    ULONG de_SizeBlock;      /* Taille du bloc (mots longs) (128) */
    ULONG de_SecOrg;         /* non utilisé: — 0 */
    ULONG de_Surfaces;       /* Nombre de têtes lecture/écrit. (2) */
    ULONG de_SectorPerBlock; /* Non utilisé: — 1 */
    ULONG de_BlocksPerTrack; /* Blocs par Track */
    ULONG de_Reserved;       /* Blocs réservés au début */
                            /* d'une partition (BootBlocks) */
    ULONG de_PreAlloc;       /* Blocs réservés à la fin */
                            /* d'une partition */
    ULONG de_Interleave;     /* Habituellement 0 */
    ULONG de_LowCyl;         /* Cylindre initial (0) */
    ULONG de_HighCyl;        /* Cylindre final (79) */
    ULONG de_NumBuffers;     /* Nombre de buffers (SizeBlock*4 Bytes)*/
    ULONG de_BufMemType;     /* Type de mémoire pour les Buffers */
    ULONG de_MaxTransfer;    /* Nombre max. de blocs, pouvant */
                            /* être lus par unité de temps */
    ULONG de_Mask;           /* Masque d'adresse pour */
                            /* délimiter un emplacement en mémoire */
    LONG de_BootPri;         /* Priorité de bootage pour Auto-Boot */
    ULONG de_DosType;        /* ASCII(HEX)String f. */
                            /* Type de File-System */
                            /* 0X444F5300 — Ancien File-System */
                            /* 0X444F5301 — Fast-Filing-System */
};

struct DosEnvec *DosEnvec;

/*****************************************/
/* printbstr()                         */
/* Fonction: Affichage du string BCPL */
/*-----*/
/* Paramètres d'entrée:                */
/* String:   String BCPL à afficher   */
/*****************************************/

printbstr (String)
BSTR      String;
{
    WORD i;
    BYTE Buffer[80];
    UBYTE *Help;

    Help = (UBYTE *) BADDR (String);

```

```

for (i=0;i<(*Help);i++)
    Buffer[i] = *(Help+i+1);

Buffer[i] = 0;

printf (Buffer);
}

/****************************************/
/*                                main()      */
/*                                */
/****************************************/

main()
{
    RootNode = (struct RootNode *) DOSBase->dl_Root;
    DosInfo = (struct DosInfo *) BADDR(RootNode->rn_Info);
    DeviceNode = (struct DeviceNode *) BADDR(DosInfo->di_DevInfo);

    do
    {
        if (DeviceNode->dn_Type == (ULONG)DLT_DEVICE)
        {
            printf ("Device: ");
            printbstr (DeviceNode->dn_Name);
            printf (":");
            CR;

            printf ("-----");
            CR;

            printf ("Task: 0x%08lx  Stack: %08ld",
                    DeviceNode->dn_Task,
                    DeviceNode->dn_StackSize);
            CR;
            printf ("Pri: 0x%08lx  GlobVec: 0x%08lx",
                    DeviceNode->dn_Priority,
                    BADDR(DeviceNode->dn_GlobalVec));
            CR;

            printf ("Handler: ");
            printbstr (DeviceNode->dn_Handler);
            CR;

            if (DeviceNode->dn_Startup > 21)
            {

                FSSM = (struct FileSysStartupMsg *)
                    BADDR (DeviceNode->dn_Startup);
                printf ("Unit: 0x%08lx ",FSSM->fssm_Unit);
                printf ("Device: ");
                printbstr (FSSM->fssm_Device);
                CR;

                printf ("Flags: 0x%08lx",FSSM->fssm_Flags);
                CR;

                if (FSSM->fssm_Environ != 01)

```

```

{
    CR;
    DosEnvec = (struct DosEnvec *) BADDR (FSSM->fssm_Environ);
    printf ("EnvecSize: %08ld\n", EnvecSize);
    if (EnvecSize > 0)
        printf ("SizeBlock: %08ld ", DosEnvec->de_SizeBlock);
    if (EnvecSize > 1)
        printf ("SecOrg: %08ld\n", DosEnvec->de_SecOrg);
    if (EnvecSize > 2)
        printf ("Surfaces: %08ld ", DosEnvec->de_Surfaces);
    if (EnvecSize > 3)
        printf ("SectorPerBlock: %08ld\n", DosEnvec
                ->de_SectorPerBlock);
    if (EnvecSize > 4)
        printf ("BlocksPerTrack: %08ld ",
                DosEnvec->de_BlocksPerTrack);
    if (EnvecSize > 5)
        printf ("Reserved: %08ld\n", DosEnvec->de_Reserved);
    if (EnvecSize > 6)
        printf ("PreAlloc: %08ld ", DosEnvec->de_PreAlloc);
    if (EnvecSize > 7)
        printf ("Interleave: %08ld\n", DosEnvec->de_Interleave);
    if (EnvecSize > 8)
        printf ("LowCyl: %08ld ", DosEnvec->de_LowCyl);
    if (EnvecSize > 9)
        printf ("HighCyl: %08ld\n", DosEnvec->de_HighCyl);
    if (EnvecSize > 10)
        printf ("NumBuffers: %08ld ", DosEnvec->de_NumBuffers);
    if (EnvecSize > 11)
        printf ("BufMemType: %08ld\n", DosEnvec->de_BufMemType);
    if (EnvecSize > 12)
        printf ("MaxTransfer: 0x%08lx ",
                DosEnvec->de_MaxTransfer);
    if (EnvecSize > 13)
        printf ("Mask: 0x%08lx\n", DosEnvec->de_Mask);
    if (EnvecSize > 14)
        printf ("BootPri: %08ld ", DosEnvec->de_BootPri);
    if (EnvecSize > 15)

        printf ("DosType: 0x%08lx\n", DosEnvec->de_DosType);
    }
    else printf ("No Environment Vector\n");
}
CR;
}
DeviceNode = (struct DeviceNode *) BADDR (DeviceNode->dn_Next);
}
while (DeviceNode != 01);
}
Ce programme fournit comme résultat un affichage de la forme suivante :
Device: DF1:

Task:      0x0000bccc      Stack:      00000000
Pri:       0x00000000      GlobVec:   0x00000000
Handler:
Unit:      0x00000001      Device:   trackdisk.device
Flags:     0x00000000

```

```
EnvecSize:      00000012
SizeBlock:      00000128
Surfaces:       00000002
BlocksPerTrack: 00000011
PreAlloc:        00000011
LowCyl:         00000000
NumBuffers:     00000005      SecOrg:      00000000
                           SectorPerBlock: 00000001
                           Reserved:    00000002
                           Interleave: 00000000
                           HighCyl:    00000079
                           BufMemType: 00000003

Device: PRT:
-----
Task:   0x00000000      Stack:   00001000
Pri:    0x00000000      GlobVec: 0x0000a620
Handler: L:PORT-HANDLER

Device: PAR:
-----
Task:   0x00000000      Stack:   00000800
Pri:    0x00000000      GlobVec: 0x0000a620
Handler: L:PORT-HANDLER

Device: SER:
-----
Task:   0x00000000      Stack:   00000800
Pri:    0x00000000      GlobVec: 0x0000a620
Handler: L:PORT-HANDLER

Device: RAW:
-----
Task:   0x00000000      Stack:   00000700
Pri:    0x00000005      GlobVec: 0x0000a620
Handler:

Device: CON:
-----
Task:   0x00000000      Stack:   00000700
Pri:    0x00000005      GlobVec: 0x0000a620
Handler:

Device: RAM:
-----
Task:   0x0001dcb4      Stack:   00000000
Pri:    0x00000000      GlobVec: 0x00000000
Handler:

Device: DFO:
-----
Task:   0x00000edc      Stack:   00000600
Pri:    0x0000000a      GlobVec: 0x00000000
Handler:
Unit:   0x00000000      Device: trackdisk.device
Flags:  0x00000000

EnvecSize:      00000012
SizeBlock:      00000128
Surfaces:       00000002
BlocksPerTrack: 00000011
PreAlloc:        00000000
LowCyl:         00000000
NumBuffers:     00000005      SecOrg:      00000000
                           SectorPerBlock: 00000001
                           Reserved:    00000002
                           Interleave: 00000000
                           HighCyl:    00000079
                           BufMemType: 00000002
```

3.5. Les disquettes

Le fonctionnement de l'Amiga est fortement orienté vers les disquettes, ce qui veut dire qu'il lui arrive souvent de charger tel ou tel élément dont il a besoin. C'est pourquoi il importe que les informations soient placées en sécurité sur une disquette, et qu'elles puissent être retrouvées assez rapidement. Nous allons ici nous occuper de la division de la disquette, et de l'interprétation des données qu'elle porte.

La division fondamentale d'une disquette est la suivante :

- ✓ Face ou numéro de tête (0 ou 1)
- ✓ Piste ou cylindre (0 à 79)
- ✓ Secteur (0-10)

Les disquettes Amiga sont toujours traitées sur les deux faces.

Chaque face de la disquette est à son tour divisée en 80 pistes, ordonnées en anneaux concentriques. La piste extérieure porte le numéro 0, la piste intérieure le numéro 79. Les pistes sont aussi appelées tracks. Si l'on peut accéder aux deux faces d'une disquette, on dit que les deux tracks disposées au revers l'une de l'autre, et accessibles simultanément aux deux têtes de lecture/écriture du lecteur forment ensemble un cylindre.

Chaque piste, enfin, est elle-même partagée en 11 secteurs, numérotés eux aussi à partir de 0. Ces secteurs sont parfois désignés comme des blocs. Au total les secteurs sont numérotés de 0 à 10, mais les blocs le sont de 0 à 1759, cette dernière numérotation correspondant aux numéros logiques des secteurs de la disquette.

Chacun des secteurs contient 512 octets d'informations disponibles, ce qui fait que chaque disquette peut porter $512 \times 11 \times 80 \times 2$ octets. Tous ne sont cependant pas à la disposition de vos données, étant donné que la gestion de ces données exige un peu de place pour elle-même. Avec le système Fast-Filing (FFS), il faut cependant savoir que l'ensemble des 512 octets est utilisé pour les données.

Le premier secteur logique, donc le premier bloc sur la disquette, se trouve à la face 0, piste 0, secteur 0. Le bloc suivant est le secteur suivant de cette piste, et ainsi de suite. Le bloc 11, par contre, n'est pas le premier secteur de deuxième piste, mais le premier secteur de la première piste sur l'autre face (face 1) de la disquette. De cette manière, la disquette est lue ou écrite alternativement sur l'une et l'autre face.

3.5.1. Le bootage

Le premier contact avec la disquette a lieu dès que l'on allume l'Amiga. Un certain nombre d'opérations d'initialisation de l'ordinateur sont d'abord effectuées sans passer par le logiciel, et aussitôt après le lecteur 0 se met en marche. Que se passe-t-il là ?

Que le Kickstart soit intégré ou non dans l'Amiga, quelque chose est chargé à partir de la disquette qui se trouve dans ce lecteur. Si le Kickstart n'est pas intégré, l'ordinateur le charge à partir de la disquette. Autrement, il recherche la disquette Workbench. Ce chargement au moment de la mise en marche de l'ordinateur est appelé 'bootage' (prononcer 'boutage').

Ce qui est chargé en premier lieu de la disquette insérée, ce sont les blocs de boot (boot-bloc), qui occupent les deux premiers secteurs (0 et 1) de la disquette. C'est là que se trouve l'information disant de quel type de disquette il s'agit. On peut avoir les types suivants :

- ✓ Kickstart
- ✓ DOS, éventuellement une disquette DOS chargeable (Workbench disk)
- ✓ Disquette non formatée, ou formatée dans un format étranger

Les quatre premiers octets du premier bloc de la disquette indiquent donc de quel type il s'agit. C'est là que se trouvent en effet les lettres DOS conclues par un 0 si l'on a affaire à une disquette DOS, ou les lettres KICK pour une disquette Kickstart. S'il n'y a ni l'un ni l'autre, il s'agit d'une disquette étrangère (BAD).

Les quatre octets suivants constituent un mot long, et représentent la somme de vérification du bloc de boot. Si la somme est correcte, l'Amiga conclut qu'il s'agit bien d'une disquette Workbench. Le mot long suivant contient le numéro du bloc de la disquette, nommé bloc racine, et placé normalement dans le bloc \$370. Nous allons parler tout de suite après de la signification de ce bloc. Restons-en pour l'instant au boot-bloc.

A partir du 7ème mot, on trouve un programme. Si la somme servant de contrôle est correcte, ce programme démarre. Il reçoit en A6 un pointeur sur l'adresse de base Exec, et il peut donc exécuter correctement une commande Exec.

Bloc de bootage (secteur 0)

Mot Long	Nom	Contenu	Signification
0	Disk type	DOS, KICK	Type de la disquette sur 4 caractères
1	Contrôle	???	Somme de contrôle du bloc
2	Bloc racine	\$370	Numéro de bloc du bloc racine
3-127	Données		Programme de bootage

Le programme qui se trouve habituellement ici teste au moyen de la commande FindResident() d'Exec si la bibliothèque DOS est résidente. Si ce n'est pas le cas, la

valeur -1 est retournée dans le registre D0. Si c'est le cas, un 0 est retourné en D0 et en A0 un pointeur sur la routine d'initialisation du DOS.

Avec un programme approprié, on peut configurer ici tout le processus de boot et donc l'initialisation de l'Amiga. Vous avez ainsi la possibilité de configurer une 'disquette Workbench' concoctée par vos propres soins, étant donné qu'il y a des programmes moniteurs de disquettes qui peuvent se charger de créer la somme de contrôle. Cette somme de tous les mots du bloc est essentielle, car c'est grâce à elle que l'Amiga reconnaît ce bloc comme bloc racine.

3.5.2. Distribution des données sur la disquette

La distribution des différents blocs de données sur la disquette dépend naturellement de son contenu, et de l'ordre dans lequelles les données ont été enregistrées. Cependant, la distribution des différents secteurs est prescrite à l'avance.

A partir de Workbench 1.3, on trouve sur la disquette le système Fast-Filing (FFS) en plus d'un Filing normal, nous allons envisager dans ce qui se suit les formats de fichiers en deux paragraphes distincts.

3.5.2.1. Filing-system normal

Pour entreposer des données sur une disquette dont la capacité est d'environ 800 Koctets, de telle façon que l'on puisse ensuite retrouver ces données, il existe quelques règles concernant leur distribution. Ces règles sont évidemment connues de l'Amiga DOS, si bien que l'on n'a pas vraiment besoin d'en prendre connaissance. Mais s'il se produit une erreur sur une disquette, encore faut-il pouvoir sauver les données restantes.

Le DISKDOCTOR est justement prévu à cet effet, et l'Amiga le recommande même dans un Requester lorsqu'une erreur se produit. Pour comprendre en cas de besoin comment fonctionne ce sauveur, il faut en passer par quelques remarques approfondies sur les formats de disquettes.

L'un des points importants est la distribution des fichiers sur la disquette, ainsi que la technique du catalogue. Au contraire de ce qui passe pour la plupart des formats de disquette, le catalogue sur les disquettes Amiga ne se trouve pas tout entier dans des secteurs qui se font suite. C'est pourquoi l'affichage du répertoire (par exemple avec la commande DIR du CLI) dure aussi longtemps.

Cette méthode a ses avantages et ses inconvénients. L'inconvénient principal est que l'accès au catalogue peut durer assez longtemps, ce qui peut à la longue porter sur les nerfs de l'utilisateur. Cet inconvénient est cependant contrebalancé par un grand avantage : la possibilité de 'réparer' une disquette endommagée.

Si, pour une disquette provenant d'un autre système, par exemple ATARI ST, il se produit une erreur en particulier dans le secteur 0, là où se trouve l'ensemble du catalogue de la disquette, on a à faire face à de gros problèmes. En effet, dans ce cas, la position

des données et la connexion des secteurs ne peuvent pas être retrouvées, et il faut beaucoup de travail pour sauver les fichiers.

Ce n'est pas le cas avec Amiga. Comme le laisse entendre l'existence de DISKDOCTOR, les fichiers sont assez faciles à retrouver, sans que l'on ait besoin d'un emplacement central contrôlant leur distribution. On y arrive grâce à des redondances importantes, qui prennent certes de la place sur les disquettes, mais qui procurent en revanche une grande sécurité d'emploi.

Comment est-ce que cela fonctionne ? Pour avoir une idée générale de la distribution des données, il nous faut prendre en compte la structure des différents secteurs de la disquette.

Bloc racine (root block)

Indépendamment des secteurs de boot, il existe sur la disquette, à un endroit déterminé, le bloc racine. Celui-ci se trouve habituellement face 0, piste 40, secteur 0, et il porte donc le numéro 880 (\$370). Le troisième mot long du secteur de boot contient ce même numéro.

C'est dans ce bloc que se trouve la racine de la disquette en son entier. On trouve ici le répertoire principal et sa date de création. Ce bloc est constitué de la façon suivante (toutes les valeurs correspondent à des mots longs, donc à des ensembles de 4 octets) :

Bloc racine

Mot	Nom	Contenu	Signification
0	Type	2	Type 2 (T.SHORT) signifie que ce bloc est le bloc initial d'une structure.
1	Header Key	0	N'a pas de signification.
2	High Seq	0	N'a pas de signification.
3	HT-Size	\$48	Taille du tableau (hashtable) portant mention des blocs initiaux des fichiers ou des sous-répertoires reliés par une chaîne.
4	réservé	0	N'a pas de signification.
5	somme contr.	???	Contient une valeur qui annule la somme de tous les mots du bloc.
6	hashtable		Ici commence la table où se trouvent les blocs initiaux des fichiers ou des sous-répertoires.
78	BM-Flag	-1	Ce flag contient -1 (TRUE) si le bitmap de la disquette est valide.

Mot	Nom	Contenu	Signification
79	BM-Pages		La table suivante contient des pointeurs sur les blocs contenant le bitmap. Il s'agit le plus souvent d'un bloc unique, ce qui fait que les autres pointeurs sont nuls.
105	days		Contient la date à laquelle la disquette a été modifiée pour la dernière fois.
106	mins		Heure de la modification.
107	ticks		Secondes de la modification.
108	disk name		Contient le nom de la disquette comme chaîne BCPL: le premier octet représente le nombre de caractères du nom (max. 30).
121	create days		Date de la création de la disquette.
122	create mins		Heure de création.
123	create ticks		Seconde de création.
124	next hash	0	Toujours nul.
125	parent dir	0	Pointeur sur le répertoire immédiatement supérieur : toujours nul.
126	extension	0	Toujours nul.
127	sec.type	1	Ce mot représente le type secondaire du bloc. 1 pour le bloc racine.

Les valeurs portées dans la hashtable indiquent les blocs où commencent les chaînes de fichiers ou de sous-répertoires à l'intérieur du répertoire racine. Etant donné qu'il n'y a pas assez de place dans la table pour que toutes les valeurs puissent y entrer, les fichiers et les sous-répertoires sont constitués en chaînes, leurs noms étant ordonnés d'une certaine façon. A partir de ces noms, on calcule une valeur entre 6 et 77. Cette valeur permet alors d'avoir accès au mot long du tableau hash, là où la chaîne commence.

La fonction qui permet de calculer cette valeur est la suivante :

```

hash = longueur du nom pour chaque caractère du nom:
Hash=Hash *13
Hash=Hash +valeur ASCII du caractère (toujours majuscule)
Hash=Hash & $7FF (ET logique)
Hash=Hash modulo 72
Hash=Hash +6

```

Vous verrez dans le chapitre sur le 'trackdisk-device' comment on programme ceci. Vous y trouverez un programme en langage machine pour déterminer les valeurs de la table hash. Le début de la chaîne se trouve donc là où pointe le pointeur que l'on vient de déterminer ainsi dans la table hash. On a de cette façon le bloc, dans lequel se trouve, au 124ème mot long, le numéro de la prochaine mention de la chaîne ; et l'on continue ainsi jusqu'à ce que l'on ait atteint la fin de la chaîne, signalée par un pointeur de valeur 0. Ces blocs, qui constituent le début d'une structure de fichier ou de répertoire, ont eux

aussi une structure particulière. Commençons par le premier bloc d'un fichier, le file-header-block.

File-header-block

Ce bloc contient les informations sur le fichier correspondant. Ces informations sont : le nom du fichier, la date de création, un commentaire ainsi que la taille et la distribution du fichier sur la disquette. Le bloc est constitué de la façon suivante :

File-header-block

Mot	Nom	Contenu	Signification
0	Type	2	Type 2 (T.SHORT) signifie que ce bloc est le bloc initial d'une structure.
1	header Key		Numéro du bloc.
2	high sequ		Contient le nombre total de blocs de ce fichier.
3	data size	0	
4	first data	0	Contient le numéro du premier bloc de données du fichier. Cette valeur se trouve aussi dans le mot n° 77.
5	somme contr.	???	Contient une valeur qui annule la somme de tous les mots du bloc.
6	data blocks		Ici commence la table où sont reportés les blocs de données du fichier. La table commence au mot 77 et la numérotation se fait ensuite vers l'arrière.
78	réservé	0	
79	réservé	0	
80	protect		Ce mot, dans ses 4 bits inférieurs, contient les informations sur l'état du fichier : Bit Protégé contre
			0 l'effacement 1 la modification 2 l'écriture 3 la lecture
81	byte size		Longueur du fichier en octets.
82	comment		Début du commentaire sur le fichier, en string BCPL (max. 22 caractères).

Mot	Nom	Contenu	Signification
105	days		Contient la date de création du fichier.
106	mins		Heure de la création.
107	tricks		Seconde de la création.
108	nom fichier		Nom du fichier, en string BCPL : le premier octet représente le nombre de caractères du nom (max. 30).
124	hash chain		Nº de bloc du fichier suivant de la chaîne, ou 0.
125	parent		Pointeur sur le répertoire, dans lequel ce fichier apparaît.
126	extension	0	Pointeur sur le bloc d'extension, ou 0. Cette entrée est toujours différente de 0 lorsque la table du bloc des données n'est pas assez longue pour faire figurer tous les blocs de ce fichier. Si c'est le cas, on trouve en cet endroit un pointeur sur un bloc où la liste se poursuit.
127	sec.type	-3	Représente le type secondaire du bloc. -3 (\$FFFD) pour le File-header-block.

File-List-Block

Ce bloc, où la liste se poursuit, est appelé bloc d'extension (file list block), et il est constitué de la façon suivante :

File list block

Mot	Nom	Contenu	Signification
0	Type	\$10	(T.LIST) signifie que ce bloc est le bloc d'extension d'une structure de fichier.
1	header key		Nº de bloc.
2	high seq		Contient le nombre total des entrées dans la table des blocs de données.
3	data size	0	
4	first data	0	Nº du premier bloc de données du fichier. Se trouve aussi dans le mot nº 77 du File header block.

Mot	Nom	Contenu	Signification
5	somme contr.	???	Contient une valeur qui annule la somme de tous les mots du bloc.
6	data blocks		Ici commence la table où sont reportés les blocs de données du fichier. La table commence au mot 77 et la numérotation se fait ensuite vers l'arrière.
78	info	0	Réserve.
124	hash chain	0	Nº de bloc du prochain fichier de la chaîne (toujours 0).
125	parent		Pointeur sur le file header block.
126	extension	0	Pointeur sur le bloc d'extension ou 0.
127	sec.type	-3	Représente le type secondaire du bloc.

L'autre possibilité d'un start-block est le bloc répertoire de l'utilisateur (user directory block), qui se trouve au début d'une structure de répertoire.

User directory block

Chaque sous-répertoire commence par un bloc de ce type, constitué de manière semblable au bloc racine :

Mot	Nom	Contenu	Signification
0	type	2	Type 2 (T.SHORT) signifie que ce bloc est le bloc initial d'une structure.
1	header key		Nº de bloc.
2	high seq	0	N'a pas de signification.
3	HT size	0	N'a pas de signification.
4	réserve	0	N'a pas de signification.
5	somme contr.	???	Contient une valeur qui annule la somme de tous les mots du bloc.
6	hashtable		Ici commence la table où se trouvent les blocs initiaux des fichiers ou des sous-répertoires.
78	réserve	0	N'a pas de signification.
80	protect		Ce mot, dans ses 4 bits inférieurs, contient les informations sur l'état du fichier : Bit Protégé contre

Mot	Nom	Contenu	Signification
			0 L'effacement 1 La modification 2 L'écriture 3 La lecture
81	réservé	0	Sans signification.
82	comment		Ici commence le commentaire sur ce sous-répertoire, comme string BCPL (max. 22 caractères).
105	days		Contient la date à laquelle la disquette a été créée.
106	mins		Heure de la création.
107	ticks		Seconde de la création.
108	dir.name		Contient le nom de la disquette comme string BCPL: le premier octet représente le nombre de caractères du nom (max. 30).
124	next hash		Entrée suivante de la même chaîne.
125	parent dir		Pointeur sur le répertoire immédiatement supérieur.
126	extension	0	Toujours nul.
127	sec.type	2	Ce mot représente le type secondaire du bloc. 1 pour le bloc racine.

En dehors de ces blocs de structure, il y a évidemment aussi des blocs de données sur la disquette. Ces blocs de données ont une forme extrêmement simple :

Data-block

Mot	Nom	Contenu	Signification
0	type	8	Type 8 (T.DATA) signifie qu'il s'agit d'un bloc de données.
1	header key		Nº de bloc du file-header
2	seq num		Nº du bloc de données dans ce fichier.
3	data size	\$1E8	Mots valides de ce bloc de données (\$1E8 ou moins).
4	next data		Nº du bloc de données suivant de ce fichier.
5	somme contr.	???	Contient une valeur qui annule la somme de tous les mots de ce bloc.

Mot	Nom	Contenu	Signification
6	data		Les données proprement dites commencent ici.

Il ne manque plus que le bloc contenant le bitmap mentionné dans le bloc racine. Il y a ici pour chacun des blocs de la disquette un bit indiquant si le bloc correspondant est libre ou s'il est occupé. Il est constitué de façon très simple :

Bitmap block

Mot	Nom	Contenu	Signification
0	somme contrôle	???	Somme de contrôle du bloc.
1-55	motif de bits		Tableau des bits occupés pour tous les blocs. Le bit 0 du premier mot long est mis pour le bloc, et ainsi de suite. Un bit posé signifie que le bloc est libre.

3.5.2.2. Fast-Filing-System (FFS)

La différence entre les deux systèmes de gestion est infime, comparée à l'accroissement de la vitesse qu'elle permet d'obtenir. L'une des raisons de cette accélération est que le nouveau handler FFS a été programmé de manière optimale (en langage machine !), et qu'il tourne donc plus rapidement. Mais c'est l'autre raison qui est décisive : les blocs de données ne contiennent plus que des données !

Pour marquer une disquette FFS ou une partition de disque dur, le type de la disquette dans le bloc de boot, "DOS", est suivi d'un 1 binaire posé. Après formatage avec la nouvelle commande FORMAT et l'option FFS, cet unique bit est donc posé différemment, tandis que le reste du formatage est strictement identique. Seul le contenu initial des secteurs, dans lesquels est écrit DOS1DOS2DOS3 etc, est décalé d'une unité, car on ne commence plus à 0, mais bien à 1.

Les avantages obtenus de cette manière sont évidents : par bloc de données chargé, il y a 512 octets qui sont effectivement chargés, soit 24 octets de plus que précédemment. En outre, on économise le temps dont le handler avait besoin pour former la somme de contrôle, la vérifier, et enfin pour copier les octets de données proprement dits à l'endroit voulu en mémoire. Avec FFS, l'ensemble du contenu du secteur est placé à l'emplacement indiqué. Puisqu'en outre, pour les gros fichiers, les secteurs de données sont placés l'un à la suite de l'autre, le chargement direct d'un grand nombre de secteurs en mémoire permet d'utiliser pleinement le hardware, alors que l'ancien système devait effectuer le chargement secteur par secteur.

Pour obtenir le FFS sur un disque dur, il faut effectuer quelques préparatifs. La première partition du disque doit être préparée comme avant pour l'ancien système, puisqu'il faut commencer par charger le FFS. Il est recommandé de choisir une partition très petite dans ce cas.

Toutes les autres partitions doivent être formatées à nouveau pour l'utilisation du FFS. Si vous avez donc déjà des données sur le disque, commencez par les sauvegarder sur des disquettes.

Copiez ensuite les programmes l:FastFileSystem, c:Mount et c:Format de la disquette Workbench 1.3 sur le disque de boot. Ensuite, pour chaque partition devant fonctionner sous FFS, vous devez ajouter dans la MountList :

```
GlobVec = -1  
FileSystem = l:FastFileSystem  
DosType = 0x4444F5301
```

La ligne "GlobVec" est placée sur -1 car on n'utilise pas de table de vecteurs globaux. L'indication du FileSystem est claire, mais il faut remarquer que le FastFileSystem doit lui aussi être copié dans le répertoire l. Avec la ligne "DosType" enfin, on définit la caractérisation de la partition comme étant "DOS\1". Le premier chiffre 4 de 0x4444F5301 est nécessaire, puisqu'il s'agit d'un string BCPL.

Si vous exécutez maintenant un reset, vous voyez apparaître pour chacune des partitions FFS un requester "Not a DOS-Disk". Cliquez sans crainte sur Cancel, c'est lui qui a raison. En effet, chaque fois que l'on fait démarrer l'ordinateur, les partitions FFS doivent d'abord être formatées selon le bon format. Ceci est réalisé à l'aide de la nouvelle commande de formatage, où l'on ajoute encore FFS à la fin de la ligne de commande. Vous pouvez ensuite utiliser les partitions avec FFS. L'accroissement de vitesse est énorme !

Si vous voulez maintenant créer aussi des disquettes à la norme FFS, le procédé est légèrement différent. Il faut d'abord remarquer qu'une disquette, au contraire du disque dur, peut être échangée. Mais comme FFS ne le sait pas, il faut entrer à nouveau la commande DiskChange (dans le répertoire C du disque WB1.3) après chaque changement de disquette !

Dans la MountList pour la disquette, il faut maintenant taper une entrée supplémentaire, à l'intention des disquettes FFS. Cette entrée se présentera par exemple ainsi :

```
FAST:  
  Device      = trackdisk.device  
  FileSystem  = l:FastFileSystem  
  GlobVec    = -1  
  DosType    = 0x4444F5301  
  StackSize   = 5000  
  Unit        = 1  
  Flags       = 0  
  Surfaces    = 2  
  BlocksPerTrack = 11  
  Reserved   = 2  
  Interleave  = 1  
  LowCyl     = 0
```

```
HighCyl      - 79
Buffers       - 5
BufMemType   - 1
Mount         - 1
#
```

Après la commande Mount FAST:, cette permet d'accéder à une disquette avec FastFileSystem, la disquette devant évidemment être formatée à chaque fois l'option FFS.

Veuillez consulter les chapitres précédents ou encore votre manuel DOS, pour retrouver les significations des différentes entrées dans MountList.

4. Entrée et sortie par les devices

Vous avez sûrement entendu parler des devices, vous avez peut-être eu entre les mains des codes source de programmes utilisant des devices. Mais vous n'avez pas encore pénétré dans les arcanes de ces codes. Nous allons y remédier dans ce chapitre. Nous répondrons d'abord à la question suivante :

4.1. Qu'est-ce qu'un device ?

Les librairies et les devices ont la même structure de base. Les devices s'ouvrent comme les librairies. La fonction Exec responsable de cette ouverture ne s'appelle cependant pas OpenLibrary(), mais OpenDevice().

Il existe malgré tout certaines différences de taille entre les devices et les librairies. Par exemple au moment de l'ouverture, OpenLibrary() fournit l'adresse d'une librairie entièrement initialisée et aussitôt utilisable. En revanche, OpenDevice() rend compte au programmeur d'éventuelles erreurs. C'est pourquoi OpenDevice() a besoin de structures pré-initialisées de la part de l'utilisateur, ces structures étant ensuite liées l'une à l'autre et initialisées entièrement.

La communication avec les devices s'effectue par l'intermédiaire d'une série de fonctions Exec, que nous n'allons pas toutes mentionner dans ce qui suit. Pour acquérir à ce sujet des connaissances exhaustives, vous devez avoir recours à la documentation des fonctions Device dans la librairie Exec. Examinons pour cela d'abord un appel OpenDevice() typique :

```
Error = OpenDevice("trackdisk.device", 0L, DiskRequest, 0L);
if (Error != 0) CloseIt ("OpenDevice() - Error");
```

La structure par laquelle le périphérique (ici le lecteur de disquettes interne) est dirigé, est dans ce cas une structure IORequest :

```
Offsets    struct IORequest
-----  { /* définie dans "exec/io.h" */
0  0x00      struct Message  io_Message;
20 0x14      struct Device   *io_Device;
24 0x18      struct Unit     *io_Unit;
28 0x1c      ULONG        io_Command;
30 0x1e      UBYTE       io_Flags;
```

```

31 0x1f      BYTE          io_Error;
32 0x20 } /* 32 — NbOctets de cette structure */

```

Bien entendu, vous n'avez pas besoin de redéfinir à chaque fois la structure IORequest dans vos programmes, si vous désirez les utiliser. Dans le fichier Include "exec/io.h", cette structure est définie, et elle est disponible pour vos déclarations de structure (par exemple struct IORequest *DiskRequest; cf. plus haut). Nous disions cependant que les structures Device, contrairement aux librairies, doivent être initialisées avant l'utilisation. Cela se fait avec la fonction de support d'Exec CreateExtIO(). "Fonction de support d'Exec" signifie que cette fonction n'est pas déposée dans une librairie du système, mais dans la librairie c.lib (Aztec) de l'éditeur de liens, ou dans amiga.lib (Lattice). Voyons d'abord simplement ce que fait CreateExtIO() :

```

***** CreateExtIO() (Exec-Support) ****
/*
 * Fonction:   Créer un bloc de device
 */
/* IOResponsePort: MsgPort pour WaitIO() etc.
 * Size:        Taille du bloc de device
 */
***** struct IORequest *CreateExtIO(IOResponsePort, Size)
struct MsgPort *IOResponsePort;
LONG           Size;
{
    struct IORequest *Request;
/* pas de IOResponsePort ? alors quitter CreateExtIO() */
    if (IOResponsePort == NULL) return(NULL);
/* Réserver de la mémoire pour Request */
    Request = AllocMem(Size, MEMF_PUBLIC | MEMF_CLEAR);
/* pas de mémoire ? alors quitter CreateExtIO() */
    if (Request == NULL) return(NULL);
/* messages du type MESSAGE */
    Request->io_message.mn_node.ln_Type = NT_MESSAGE;
/* les messages sont longs de Size octets */
/* cf. aussi DeleteExtIO(). */
    Request->io_Message.mn_Length = Size;
/* indiquer le port de messages */
    Request->io_Message.mn_ReplyPort = IOResponsePort;
    return(Request);
}

```

CreateExtIO() réserve de la mémoire pour une structure Device-Request. Parmi ces structures, on trouve non seulement les blocs Device standard comme IOStdReq et IORRequest, mais aussi les blocs Device comme IOAudio (Audio-Device), etc.

Avec CreateExtIO(), vous pouvez créer et initialiser ces différents blocs Device à l'aide desquels s'effectue la communication entre l'ordinateur et le device.

Toutes les structures Device-Request et les blocs Device ont ceci de commun que leur premier élément est une structure IORequest. De ce fait, tous les blocs Device sont traités de la même façon par CreateExtIO(). Les différences n'interviennent qu'au-delà des premiers octets, représentant la structure IORequest. Examinons ainsi la structure IOAudio, le bloc Device pour le device Audio :

```

struct IOAudio
{
    struct IORequest  ioa_Request;      /* IORequest au début */
    WORD              ioa_AllocKey;
    UBYTE             *ioa_Data;
    ULONG             ioa_Length;
    UWORLD            ioa_Period;
    UWORLD            ioa_Volume;
    UWORLD            ioa_Cycles;
    Struct Message   ioa_WriteMessage;
}

```

Le premier élément de la structure IOAudio est, comme on le voit facilement, une structure IORequest. Si l'on transmet alors à CreateExtIO() l'adresse initiale de la structure IOAudio définie par soi-même, cette adresse est identique à celle d'une structure IORequest.

Puisque tous les accès aux éléments de la structure se font par l'intermédiaire des offsets, la structure IORequest du bloc IOAudio est initialisée correctement de manière automatique. Nous initialisons donc notre bloc I/O pour l'ouverture du device Audio au moyen de

```

struct IOAudio *OwnIOAudio;
...
OwnIOAudio = (struct IOAudio *) CreateExtIO(AudioPort, sizeof(struct
IOAudio))
if (OwnIOAudio == 0) CloseIt("CreateExtIO() - Error");
...

```

et l'ouverture est ensuite exécutée à l'aide de OpenDevice() :

```
OpenDevice("audio", OL, OwnIOAudio, OL));
```

Sur cet exemple d'appel, vous pouvez comprendre ce que signifie le paramètre de taille pour CreateExtIO(). Il fournit en effet la taille du bloc I/O à allouer. Le premier paramètre pour CreateExtIO(), le IOReplyPort, a cependant besoin d'être examiné de plus près.

Le IOReplyPort n'est d'abord rien d'autre qu'un port de message habituel. Les ports de message sont nécessaires pour la transmission des messages (message report). Les devices les utilisent pour ainsi dire comme ancrés, qui les relient à l'ensemble du système. Avant d'utiliser un tel port de message (MsgPort), il faut évidemment commencer par l'initialiser :

```

struct MsgPort *MsgPort;
...
MsgPort = (struct MsgPort *)
CreatePort("MsgPort", 0);
if (MsgPort == 0) CloseIt("CreatePort() - Error");
...

```

La routine nécessaire à cet effet s'appelle CreatePort(), et c'est aussi une fonction de support d'Exec :

```
***** CreatePort() (Exec-Support) ****
/*
 * Fonction: Créer un port de messages
 */
/* Nom: Nom du MsgPorts */
/* Priorit : Priorité du Port */
struct MsgPort *CreatePort(Nom, Priorit)
char          *Nom;
BYTE          Priorit;
{
    struct MsgPort *Port;
    BYTE            SignalBit;
    if ((SignalBit = AllocSignal(-1)) == -1) return(NULL);
    /* N'a pu réserver aucun SignalBit */
    Port = (struct MsgPort *)
        AllocMem(sizeof(struct MsgPort), MEMF_PUBLIC|MEMF_CLEAR);
    /* résérer de la mémoire */
    if (Port == Null)
    {
        Freesignal(SignalBit);
        return(Null);
    }
    /* pas de mémoire */
    Port->mp_Node.ln_Nom = Nom;
    Port->mp_Node.ln_Pri = Priorité;
    Port->mp_Node.ln_Type = NT_MSGPORT;
    Port->mp_Flags      = PA_SIGNAL;
    Port->mp_SigBit     = SignalBit;
    Port->mp_SigTask    = FindTask(01);
    /* initialisation */
    if (Nom != OL) AddPort(Port);
    else           NewList(&(Port->mp_MsgList));
    /* Port dans la nouvelle liste */
    return(Port);
}
```

Lorsqu'on utilise des devices, les ports de message ne reçoivent en général pas de nom, si bien que l'initialisation d'un port de message peut être entreprise au moyen de la séquence suivante :

```
struct MsgPort *Port;
Port = (struct MsgPort *)CreatePort(OL, OL);
CreatePort();
```

jette donc l'"ancre", et le device s'y "accroche" par l'intermédiaire de CreateExtIO() et de OpenDevice()

```
struct MsgPort *Port;
struct IORequest *Request;
Port = (struct MsgPort *) CreatePort(OL, OL);
Request = (struct IORequest *) CreateExtIO(Port, sizeof(struct
    IORequest));
```

```
OpenDevice(DEVICENAME, 0L, Request, 0);
...
```

A l'aide ces trois routines, on peut ouvrir n'importe quel device. Puisque les étapes par lesquelles il faut passer pour ouvrir un device sont (presque) toujours les mêmes, nous allons donner quelques routines universelles pour l'allocation et la désallocation des blocs Device, ainsi que pour l'ouverture et la fermeture des devices :

```
*****
/*          Device-Support Functions      */
/*          (c) Bruno Jennrich            */
/*
   8. Juni 1988
*****
/* Compile Info:                      */
/*-----*/
/* cc Devs_Support                   */
*****  

#include "exec/types.h"
#include "exec/io.h"
#include "exec/devices.h"
VOID CloseIt();                         /* CloseIt() existe */
                                         /* dans le programme personnel*/
VOID *CreatePort();                     /* Exec-Support */
VOID *CreateExtIO();
VOID DeletePort();
VOID DeleteExtIO();
*****  

/*          GetDeviceBlock()           */
/*-----*/
/* Fonction :    Créer et initialiser un bloc de device */
/*-----*/
/* Paramètres d'entrée:                */
/*-----*/
/* Size:        Taille du bloc de device en octets */
/*-----*/
/* Valeur retournée:                  */
/*-----*/
/*          Bloc de device initialisé */
*****  

APTR GetDeviceBlock(Size)
ULONG             Size;
{
    struct MsgPort *Device_Port = 0L;
    APTR     Device_Request = 0L;
    /* puisque cette routine doit être mise en oeuvre de */
    /* manière universelle on ne crée pas de structure */
    /* IOrequest, mais une structure quelconque, qui peut */
    /* être transformée en particulier en IORequest par (CASTS) */
    /* Essaie d'allouer le port de device. Si ce n'est pas */
    /* possible, quitte le programme. (CloseIt()). */
    /*-----*/
    Device_Port = (struct MsgPort *) CreatePort(0L, 0L);
    if (Device_Port == 0L) CloseIt("Couldn't get DEVICE-PORT !");

    /* Essaie d'allouer le bloc de device. Si ce n'est pas */
}
```

```

/* possible, rend le Device-Port et quitte le programme */

Device_Request = (APTR) CreateExtIO(Device_Port, Size);
if (Device_Request == 0L)
{
    DeletePort(Device_Port);
    CloseIt("Couldn't get DEVICE-BLOCK !");
}
/* Rend un bloc de device pré-initialisé */
return(Device_Request);
}
/*********************************************
/*                         FreeDeviceBlock()          */
/*                         */
/* Fonction:   Libérer le bloc de device          */
/*-----*/
/* Paramètres d'entrée:                         */
/*-----*/
/* IOResult:   Bloc de device à libérer          */
/*****************************************/
VOID FreeDeviceBlock(IOResult)
struct IOResult *IOResult;
{
    /* Si IOResult a pu être créé, libérer           */
    /* le Device-Port. Libérer ensuite IOResult.      */
    if (IOResult != 0L)
    {
        if (IOResult->io_Message.mn_ReplyPort != 0L)
            DeletePort(IOResult->io_Message.mn_ReplyPort);
        DeleteExtIO(IOResult);
    }
}
/*********************************************
/*                         Open_A_Device()           */
/*                         */
/* Fonction:   Ouvrir un device quelconque       */
/*-----*/
/* Paramètres d'entrée:                         */
/*-----*/
/* Nom:         Nom du device (p.ex. "audio.device") */
/* Unit:        Device-Unit                      */
/* Device_Request: pointeur sur bloc de device à initialiser*/
/*                (ou déjà initialisé)           */
/* Flags:       Device-Flags                     */
/* Size:        Taille du bloc de device          */
/*****************************************/
VOID Open_A_Device(Nom, Unit, Device_Request, Flags, Size)
char          *Nom;
ULONG          Unit;
APTR          *Device_Request;
ULONG          Flags, Size;
{
    UWORLD Error; /* Erreur de OpenDevice() */
    /* si Size > 0, allouer Bloc de device. */
    /* si Size == 0, utiliser le bloc de device*/
    /* initialisé par l'utilisateur.        */
    if (Size != 0L) *Device_Request = GetDeviceBlock(Size);
    /* ouvrir device */
    Error = OpenDevice(Nom, Unit, *Device_Request, Flags);
}

```

```

if (Error != 0)
{
    printf("Open-Device Error #%4lx\n", Error);
    CloseIt("Couldn't get DEVICE !");
}
/* Attention!!! Device_Request est un pointeur */
/*           sur un pointeur! (**DevReq)  */
}
/*****************************************/
/*
*          Close_A_Device
*/
/* Fonction: Libérer le bloc de device et fermer le device */
/*-----*/
/* Paramètres d'entrée: */
/* IORequest: Bloc de device à libérer */
/*****************************************/

VOID Close_A_Device(IORequest)
struct IORequest *IORequest;
{
    /* si IORequest a pu être créé, libérer le. */
    /* Device-Port. Fermer le device. Libérer alors */
    /* IORequest. */
    if (IORequest != OL)
    {
        if (IORequest->io_Message.mn_ReplyPort != OL)
            DeletePort(IORequest->io_Message.mn_ReplyPort);
        if (IORequest->io_Device != OL)
            CloseDevice(IORequest);
        DeleteExtIO(IORequest);
    }
}
/*****************************************/
/*
*          Do_Command()
*/
/*-----*/
/* Fonction: Exécuter une commande */
/*-----*/
/* Paramètres d'entrée: */
/* DeviceBlock: Bloc de device */
/* Command: Commande */
/*****************************************/
VOID Do_Command (DeviceBlock, Command)
struct IORequest *DeviceBlock;
DWORD             Command;
{
    DeviceBlock->io_Command = Command;
    DoIO(DeviceBlock);
}

```

A l'aide de ces fonctions, vous pouvez ouvrir et refermer des devices. Exactement comme vous fermez des librairies, vous devez absolument refermer un device que vous avez ouvert. Alors qu'un programme et un utilisateur peuvent utiliser une librairie en même temps, ce n'est pas possible avec les devices. L'accès à un device n'est permis qu'à un seul utilisateur dans le même temps. Pour assurer un nouvel accès aux autres utilisateurs, il faut donc fermer les devices lorsqu'on quitte le programme. Cela se fait à l'aide des

routines de support d'Exec DeleteExtIO() et DeletePort(), ainsi que de la commande CloseDevice() :

```
***** DeletePort()          (Exec-Support) ****
/*
 * Fonction:    Libérer le port
 */
/* Paramètres d'entrée:
 */
/* IOReplyPort: Port à libérer
 */
VOID DeletePort(IOReplyPort)
struct MsgPort *IOReplyPort;
{
/* si le port n'a pas de nom, alors le supprimer aussitôt */
    if ((IOReplyPort->mp_Node.ln_Nom) != OL)
        RemPort(IOReplyPort);
/* Supprimer type du Port */
    IOReplyPort->mp_Node.ln_Type = 0xff;
/* Enlever le port de la liste */
    IOReplyPort->mp_MsgList.lh_Head = (struct Node *)-1;
/* Libérer la mémoire du port */
    FreeMem(IOReplyPort, sizeof(struct MsgPort));
}

Et voici à quoi ressemble DeleteExtIO() :
***** DeleteExtIO()          (Exec-Support) ****
/*
 * Fonction:    Libérer Bloc de device
 */
/* Paramètres d'entrée:
 */
/* IORequest: Bloc de device à libérer
 */
VOID DeleteExtIO (IORequest)
struct IORequest *IORequest;
{
/* si IORequest n'existe pas, quitter la routine */
/* Sinon Freed-Twice-Alert surgirait */

    if (IORequest == 0) return(OL);
/* IORequest embrouillé, ce qui rend impossible toute */
/* utilisation ultérieure */
    IORequest->io_Message.mn_Node.ln_Type = 0xff;
    IORequest->io_Device = (struct Device *)-1;
    IORequest->io_Unit = (struct Unit *)-1;
/* Libérer la mémoire . (la mémoire n'est pas libérée par FreeMem() */
/* d'où l'embrouillamini indiqué ci-dessus.) */
    FreeMem(IORequest, IORequest->io_Message.mn_Length);
}
```

Outre ces deux fonctions de support d'Exec, qui empêchent toute autre utilisation du IOReplyPort et des structures IORequest, nous avons utilisé dans nos fonctions de support des devices la commande CloseDevice(). Celle-ci permet à d'autres programmes d'utiliser à nouveau le device.

Le device à fermer est communiqué à CloseDevice() par l'intermédiaire du bloc IORequest, qui contient en effet un pointeur sur le device ouvert (IORequest-io_Device). CloseDevice extrait lui-même ce pointeur et il l'utilise pour la fermeture.

Si vous désirez utiliser des routines prises dans Devs_Support, vous devez disposer d'une routine du nom de CloseIt(). On appelle celle-ci à chaque fois qu'une erreur survient par exemple à l'ouverture d'un device.

Dans ce livre, vous rencontrerez encore d'autres routines CloseIt(), mais nous voulons dès maintenant vous familiariser avec les exigences de cette routine :

```
/*********************************************
/*          CloseIt()                      (User)*/
/* Fonction: Afficher l'erreur surgie      */
/*          Tout fermer                   */
/*-----*/
/* Paramètres d'entrée:                  */
/* String:     Error-String             */
/*********************************************
VOID CloseIt(String)
char *String;
{
    WORD Error = 0;
    WORD i;
    WORD *dff180 = (WORD *)0xdff180;

    if (strlen(String) > 0)
    {
        for(i=0;i<0xffff;i++) *dff180 = i;

        puts(String);
        puts("\n");
        Error = 100;
    }
    /* Routine de libération */
    exit(Error);
    /* Quitter le programme */
}
```

Cette routine CloseIt() sert à faire scintiller l'écran en couleurs (grâce à la description du registre affecté à la couleur du fond). Puis vient l'affichage de la chaîne Error, qui vous permet de savoir en quel point vous devez rechercher l'erreur.

Ce n'est qu'ensuite que toutes les librairies Device, etc., sont libérées. Evitez de tout libérer avant l'affichage de la chaîne Error. Si en effet une nouvelle erreur surgit au moment de la fermeture d'un device, vous ne retrouverez plus l'erreur précédente.

N'utilisez enfin CloseIt() que comme une "porte de secours". Il ne faut pas utiliser CloseIt() par commodité pour quitter le programme, qui libère en outre toutes les structures. Nous avons respecté le principe : la routine qui alloue est aussi celle qui libère !

4.2. Communiquer avec les devices

Vous savez maintenant comment on ouvre et comment on referme un device. Mais il faut maintenant examiner comment on utilise un device, et comment s'effectue l'échange de données entre le programme et le device. Comme vous le savez déjà, on peut comparer le device dans une certaine mesure à une librairie. Lorsqu'on a ouvert le device, on a quelques routines disponibles pour assurer la communication entre le device et le programme grâce à IORequestIo_Device. Cette librairie Device contient (le plus souvent) les commandes suivantes :

Commande : Offset :

-----	-----
Open	-0x06
Close	-0x0c
Expunge	-0x12
Extfunc	-0x18
BeginIO	-0x1e
AbortIO	-0x24

Open est une routine appelée par OpenDevice() pour entreprendre des installations spécifiques au device. La fonction Exec OpenDevice() sert avant tout à vous permettre d'accéder à cette librairie Device. La commande Open propre aux devices sert à l'initialisation nécessaire du périphérique adressé.

Close est appelé avant la fermeture du device par CloseDevice(), pour annuler toutes les opérations entreprises par Open.

Grâce à Expunge, la mémoire occupée par les structures Device (par exemple lorsqu'un device a dû être chargé à partir d'un disque) est à nouveau libérée. Avec Open et Close, on a toujours un compteur d'accès (Open-Count) incrémenté ou décrémenté. La mémoire occupée n'est pas libérée en général. C'est Expunge qui en est chargé. Expunge est appelé par la fonction Exec RemDevice().

La routine Extfunc est réservée aux tâches spéciales (par exemple le device d'imprimante DO_SPECIAL). Les routines BeginIO et AbortIO sont en fait les routines les plus intéressantes :

BeginIO() est la routine qui déclenche l'échange de données entre le programme et le device. Une fois que le bloc Device a été entièrement initialisé (pointeur de données correct, commande indiquée, etc.), cette commande est appelée aussi bien par SendIO() que par DoIO(). Cet appel se présente ainsi :

```
BeginIO:  
move.l 20(a1), a6          A1 contient *IORequest *  
jmp    -$1e(a6)           Device-Library vers A6 *  
                           Sauter à BeginIO *
```

La vraie routine BeginIO, enclenchée ici uniquement par l'intermédiaire de votre offset, se présente bien sûr différemment pour chaque device. C'est d'ailleurs logique, puisque les opérations sont entièrement différentes par exemple pour utiliser le device Timer et le device Trackdisk. BeginIO est donc appelé par SendIO() et DoIO() (on trouve

également la commande BeginIO() dans le c.lib du compilateur). Quelle différence y a-t-il entre DoIO() et SendIO() ?

La seule différence est que SendIO() est une commande asynchrone. Cela veut dire que le programme appelant peut être aussitôt traité, après qu'une commande Device a été envoyée. La commande Device est exécutée parallèlement au programme appelant. Au contraire, lorsqu'on utilise DoIO(), le programme appelant doit attendre que la commande ait été entièrement traitée par le device avant de poursuivre sa route. C'est aussi pourquoi on dit que DoIO() est une commande synchrone. Le déroulement du programme et la commande Device sont donc synchronisés.

En ce qui concerne l'exécution d'une commande Device, DoIO() et SendIO() sont rigoureusement identiques. Avec DoIO(), on attend simplement la fin de la commande Device à l'aide de WaitIO() (c'est en effet possible grâce au MsgPort). On peut imiter l'action de DoIO() avec les deux fonctions SendIO() et WaitIO():

```
Second_DoIO (IORRequest)
struct IORRequest *IORRequest;
{
    SendIO(IORRequest);
    WaitIO(IORRequest);
}
```

WaitIO() peut à son tour être imité au moyen de CheckIO():

```
Second_WaitIO (IORRequest)
struct IORRequest *IORRequest;
{
    while (CheckIO (IORRequest)==0);
    /* Commande de device pas encore terminée */
}
```

Si la commande de device est en effet en cours d'exécution, CheckIO() renvoie la valeur 0 (dans D0). Si la commande a été traitée correctement, on trouve après CheckIO() en D0 l'adresse du bloc de device (ici IORRequest) qui a été testée. De cette façon, vous êtes déjà en possession de toutes les possibilités techniques pour mettre en forme utilement la communication entre le device et le programme. Il reste à voir comment on envoie une commande au device : quelles sont les variables de la structure IORRequest (ou IOStdReq) qui jouent ici un rôle ?

Nous répondrons à ces questions lors de la description des différents devices. Nous voulons cependant indiquer ici un certain nombre de points communs entre eux : la commande de device est toujours déposée dans IORRequest-io_Command; d'autre part, la commande de device est constituée en fait par un seul numéro.

Mais le plus souvent, la simple définition de la commande ne sert encore à rien. En effet, lorsqu'on doit transmettre des données, on le fait à l'aide d'un pointeur sur les données et d'une variable, qui indique la longueur ou le nombre des octets de données à transmettre. La transmission de données n'est cependant pas possible avec une simple structure IORRequest. Il y faut la structure IOStdReq, ou une structure spécifique au device (cf. IOAudio) :

```

Offsets      struct IOStdReq
-----      {
                           /* défini dans "exec/io.h" */
                           /* - IORequest - */
0  0x00      struct Message io_Message;
20 0x14      struct Device *io_Device;
24 0x18      struct Unit   *io_Unit;
28 0x1c      ULONG        io_Command;
30 0x1e      UBYTE       io_Flags;
31 0x1f      BYTE         io_Error;
                           /* - IOStdReq - */
32 0x20      ULONG        io_Actual;
36 0x24      ULONG        io_Length;
40 0x28      APTR         io_Data;
44 0x2c      ULONG        io_Offset;
48 0x30      };

```

Le pointeur des données pour la structure IOStdReq s'appelle IOStdReq.io_Data, alors que pour la structure IOAudio il s'appelle IOAudio.ioa_Data. Le nombre d'octets de données est indiqué dans IOStdReq.io_Length. IOStdReq.io_Actual indique souvent le nombre d'octets de données effectivement transmis (écrits ou lus).

Outre ces variables qui n'interviennent qu'avec les blocs de device développés, comme le bloc IORequest, il existe également des variables Flag et des variables Error. A l'aide des variables Flag, on peut diriger le déroulement d'une commande de device. Puisque chaque device, ou presque, possède ses propres flags, nous les décrirons à chaque fois que ce sera nécessaire. Cependant, il existe un flag commun à tous les devices, et c'est IOF_QUICK. Celui-ci est posé lorsqu'une commande peut être traitée sur le coup. Si l'on prend par exemple le device Trackdisk, les commandes de lecture et d'écriture sont adressées à une tâche Trackdisk. Le programme appelant doit attendre que ces commandes soient traitées (cela se fait à l'aide du port de message). Lorsqu'il s'agit de rassembler les éléments de l'état d'une disquette (Write-Protected, etc.), l'attente n'est pas obligatoire, car cette commande est exécutée sans la tâche Trackdisk.

Il faut aussi parler de la variable Error, qui est particulièrement importante. Si elle est égale à 0 après une commande de device, cela veut dire que tout s'est bien passé. Pour une valeur différente de 0, le programmeur doit s'inquiéter. La réaction doit être en rapport avec la gravité de l'erreur (par exemple avertissement à l'utilisateur, sortie du programme, etc.). Mais la même règle vaut aussi pour les erreurs : chaque device possède ses propres erreurs. Il existe toutefois des erreurs communes à tous les devices :

IOERR_OPENFAIL	(-1)	Le device n'a pas pu être ouvert
IOERR_ABORTED	(-2)	La commande a été interrompue par AbortIO()
IOERR_NOCMD	(-3)	Commande invalide
IOERR_BADLENGTH	(-4)	Le champ io_Length contient une valeur invalide

Ce n'est que pour les commandes de device que l'on peut faire ressortir certains points communs. Ainsi presque chaque device utilise la commande de lecture (CMD_READ) ou la commande d'écriture -CMD_WRITE). La commande de reset (CMD_RESET), qui ramène le device dans son état initial, est également utilisée par presque tous les devices. Mais en marge de ces commandes standardisées, il existe des commandes étendues (extended commands) qui dépendent du device envisagé. Ces commandes étendues sont utilisées pour prendre en compte les particularités dont dispose le device.

Nous ajouterons quelques mots sur ce qui va venir : nous avons essayé de vous faciliter autant que possible la tâche pour la programmation des devices. C'est pourquoi, outre les routines de support des devices, qui simplifient de façon générale le maniement des devices, nous présentons également des routines spécifiques aux devices. Il suffit de fournir à ces routines le bloc de device qui lui correspond, et éventuellement certains paramètres. De cette façon, vous êtes dispensé du travail pénible qui consiste à affecter des valeurs aux éléments de la structure.

4.3. Le device parallèle

A l'aide du device parallèle, vous pouvez programmer simplement l'interface parallèle de l'Amiga. Cette interface parallèle permet aussi bien de sortir que de lire des données. Voici les commandes soutenues par le device parallèle :

CMD_RESET	(1)	Ramène le device dans l'état qui est le sien tout de suite après OpenDevice() (y compris TermArray)
CMD_READ	(2)	Lire des données
CMD_WRITE	(3)	Ecrire des données
CMD_STOP	(6)	Arrêter la lecture/écriture (attente du handshake)
CMD_START	(7)	Poursuivre la lecture/écriture
CMD_FLUSH	(8)	Ignorer les commandes Read et Write restantes

Il existe en outre deux commandes spécifiques au device :

PDCMD_SETPARAMS	(9)	Définir des paramètres
PDCMD_QUERY	(10)	Obtenir l'état du port

4.3.1. L'ouverture du device

L'ouverture du device parallèle se fait aussi simplement que possible :

```
struct IOExtPar *ParReq = 0L;
#define PAR_LEN (ULONG) sizeof(struct IOExtPar)
...
Open_A_Device("parallel.device", 0L, ParReq, 0L, PAR_LEN);
...
```

L'utilisateur, s'il est seul, peut toujours avoir accès au device parallèle. Si un autre utilisateur a déjà ouvert le device parallèle, il n'est plus possible d'y accéder, à moins de poser le flag PARF_SHARED (32) avant l'ouverture du device :

```
struct IOExtPar *ParReq = 0L;
#define PAR_LEN (ULONG) sizeof(struct IOExtPar)
VOID *GetDeviceBlock():
...
ParReq = (struct IOExtPar*)GetDeviceBlock(PAR_LEN);
ParReq->io_ParFlags = (UBYTE) PARF_SHARED;
Open_A_Device("parallel.device", 0L, ParReq, 0L, 0L);
...
```

De cette manière, vous pouvez avoir accès à votre tour au device parallèle. Il faut toutefois se montrer très prudent dans ce domaine. Si vous voulez par exemple transférer des données à partir de plusieurs programmes par le device parallèle, vous ne ferez rien d'autre que provoquer un chaos. Cela peut même aller jusqu'au mélange de deux textes envoyés par deux traitements de texte en fonctionnement. Mais revenons à la commande Open_A_Device().

Comme d'habitude, vous devez lui transmettre également l'adresse par un pointeur sur le bloc de device (&ParReq, où ParReq est précisément un pointeur). Le bloc de device pour le device parallèle présente l'aspect suivant :

Offset	Structure
-----	-----
0 0x00	struct IoExtPar
48 0x30	{ struct IOStdReq IOPar;
5 0x32	ULONG io_PExtFlags; /* non utilisé */
0x33	UBYTE io_Status; /* Port-Status */
54 0x34	UBYTE io_ParFlags; /* SHARED+EOFMODE */
62 0x3c }	struct IOPArray io_PTermArray; /* Terminateurs */
	/* défini dans "devices/parallel.h" */

4.3.2. Ecrire des données

L'écriture des données sur le device parallèle est également très simple à mettre en forme. Il vous suffit de définir les données à sortir et le nombre d'octets à écrire, et d'appeler CMD_WRITE :

```
*****
*           Parallel_Write()          (Par_Support)*
*
* Fonction: Sortir des données par l'interface parallèle
*-----*
* Paramètres d'entrée:
*
* ParReq:   Bloc de device
* Data:     Données à sortir
* Len:      Nombre des octets à sortir
*****
VOID Parallel_Write(ParReq, Data, Len)
struct IOExtPar *ParReq:
APTR             Data;
ULONG            Len;
{
    ParReq->IOPar.io_Data = Data;
    ParReq->IOPar.io_Length = Len;
    Do_Command(ParReq, (UWORD) CMD_WRITE);
}
```

Si vous indiquez la valeur -1 pour le nombre de données à écrire, on peut écrire des données jusqu'à ce que l'on tombe sur un octet nul à écrire. Cet octet nul est écrit à son tour, et l'opération s'arrête !

4.3.3. Lire des données

La lecture des données est aussi simple à réaliser que l'écriture :

```
/*
 *          Parallel_Read()          (Par_Support)*
 *
 * Fonction: Lire des données sur l'interface parallèle
 *-----*
 * Paramètres d'entrée:
 *-----*
 * ParReq:   Bloc de device
 * Data:     Tampon des données
 * Len:      Nombre d'octets à lire
 */
VOID Parallel_Read(ParReq, Data, Len)
struct IOExtPar *ParReq;
APTR             Data;
ULONG            Len;
{
    ParReq->IOPar.io_Data = Data;
    ParReq->IOPar.io_Length = Len;
    Do_Command(ParReq, (UWORD) CMD_READ);
}
```

Vous avez ainsi la possibilité d'arrêter avec le device parallèle la lecture des caractères, lorsqu'un caractère déterminé a été lu. On a créé pour cela le IOPArray, qui peut contenir 8 terminateurs :

Offset	Structure
0x00	ULONG PTermArray0;
0x04	ULONG PTermArray1;
0x08	}; /* défini dans "devices/parallel.h" */

Vous indiquerez les 8 "terminateurs" dans deux mots longs :

```
Parallel_SetParams(ParReq, PARF_EOFMODE, 0x00010101, 0x1010101);
/* PARF_EOFMODE = 2 */
```

Si vous définissez les terminateurs avec la fonction Parallel_SetParams() de la manière décrite plus haut, la lecture sera interrompue lorsque 0x00 ou 0x01 (codes ASCII) sont rencontrés. Si vous définissez moins de 8 terminateurs, comme nous l'avons fait ci-dessus, il faut remplir les terminateurs restants avec la valeur du dernier terminateur.

La fonction Parallel_SetParams() se présente ainsi :

```
/*
 *          Parallel_SetParams()          (Par_Support)*
 *
 * Fonction: Modifier les paramètres d'interface
 *-----*
 * Paramètres d'entrée:
 *-----*
```

```

* ParReq:      Bloc de device          *
* Flags:       Nouveaux Flags         *
* TermArray0/1: Nouveaux terminateurs   *
***** */
VOID Parallel_SetParams(ParReq, Flags, TermArray0, TermArray1)
struct IOExtPar    *ParReq;
BYTE                Flags;
ULONG               TermArray0;
ULONG               TermArray1;
{
    ParReq->io_ParFlags = Flags;
    ParReq->io_PTermArray.PTermArray0 = TermArray0;
    ParReq->io_PTermArray.PTermArray1 = TermArray1;
    Do_Command(ParReq, (UWORD)PDCMD_SETPARAMS);
}

```

Après Open_A_Device(), le PTermArray est ignoré, et la seule réaction est celle qui suit 0x00 comme terminateur. Logiquement, vous ne pouvez modifier les paramètres pour le device parallèle que s'il n'y a pas d'accès en écriture ou en lecture en cours. Modifier les paramètres pendant une telle opération signifierait détruire la base de la communication entre l'émetteur et le récepteur, ce qui veut dire qu'il n'y a plus de communication possible. Pour le moment, on peut modifier avec Parallel_SetParams() uniquement les terminateurs (PARF_EOFMODE doit pour cela être posé).

4.3.4. Obtenir l'état de l'interface

Une fonction très intéressante du device parallèle est l'interrogation de l'état du port. A l'aide de la commande PDCMD_QUERY, vous obtenez dans io_Status l'état de l'interface :

Bit 0	IOPTF_PSEL = 1 (Printer Selected)
	- 1: OFFLINE
	- 0: ONLINE
Bit 1	IOPTF_PAPEROUT = 2
	- 1: OK
	- 0: PAPER_OUT
Bit 2	IOPTF_PBUSY = 4 (Printer busy)
	- 1: L'imprimante n'a rien à faire
	- 0: L'imprimante est en fonctionnement
Bit 3	IOPTF_RWDIR = 8 (Direction (Read, Write))
	- 1: Ecriture en cours
	- 0: Lecture en cours
Bit 4-7	réservé

Notez bien que les bits 0, 1 et 2 sont actifs sur Low. S'il y a donc un 0 dans ces bits, cela veut dire que l'état désigné par le flag est actif. Vous reconnaîtrez par exemple PaperOut au fait que le flag IOPTF_PAPER-OUT n'est pas posé ! (les flags sont en outre définis également dans device/parallel.h"). Vous voyez déjà que le device parallèle est réglé sur l'imprimante connectée. Mais vous pouvez aussi utiliser les bits supérieurs pour d'autres périphériques, pour faire savoir à l'ordinateur qu'il n'est pas encore prêt à traiter les données reçues. La séquence suivante permet d'obtenir l'état :

```
DO_Command(parReq, (UWORD)PDCMD_QUERY);  
Status = parReq->io_status;
```

Pour finir, voici encore un programme qui sort une petite chaîne de caractères sur le device parallèle. C'est souvent une imprimante qui est connectée à ce port, si bien que vous pouvez tout de suite reconnaître l'effet de ce programme lorsque l'imprimante est connectée et lorsqu'elle ne l'est pas (message "Imprimante OFFLINE ou non connectée !").

```
*****  
* Par.c (User)*  
* (c) Bruno Jennrich  
* Août 1988  
*  
*****  
/* Compile-Info:  
*  
* cc Par.c  
* ln Par.o Par_Support.o Devs_Support.o -lc  
*****  
#include "exec/types.h"  
#include "exec/memory.h"  
#include "exec/io.h"  
#include "devices/parallel.h"  
struct IOExtPar *ParReq = 0L;  
#define PAR_LEN (ULONG) sizeof(struct IOExtPar)  
VOID *GetDeviceBlock();  
*****  
* CloseIt() (User)*  
*  
* Fonction: Tout fermer en cas d'erreur  
*-----  
* Paramètres d'entrée:  
*  
* String: Error-Message  
*****  
VOID CloseIt(String)  
char *String;  
{  
    WORD i;  
    WORD *dff180 = (WORD *)0xdff180;  
    WORD Error = 0;  
    if (strlen(String) > 01)  
    {  
        for(i=0;i<0xffff;i++) *dff180 = i;  
        puts(String);  
        Error = 10;  
    }  
    if (ParReq != 0L) Close_A_Device(ParReq);  
    exit (Error);  
}  
*****  
* main() (User)*  
*  
*****  
main ()  
{
```

```

BYTE *String = "I write this to the Parallel-Port\015";
ParReq = (struct IOExtPar *)GetDeviceBlock(PAR_LEN);
ParReq->io_ParFlags = (UBYTE) PARF_SHARED;
Open_A_Device("parallel.device", 0L, &ParReq, 0L, 0L);
Do_Command(ParReq, (UWORD) PDCMD_QUERY);
if (((UBYTE)ParReq->io_Status & (UBYTE) IOPTF_PSEL) == (UBYTE)
    IOPTF_PSEL)
    Parallel_Write(ParReq, String, (ULONG) strlen(String));
Close_A_Device(ParReq);
}

```

On peut également vérifier grâce au device de l'imprimante si une imprimante est connectée, par l'intermédiaire du bit Selected de l'état. Si en effet ce bit est toujours égal à 0 au bout de 30 secondes (cas général), le device d'imprimante interrompt son travail.

4.3.5. Messages d'erreur du device parallèle

Il peut évidemment se produire des erreurs lorsqu'on utilise le device parallèle. Voici les erreurs possibles :

ParErr_DevBusy	(1)	Le device parallèle est occupé PDCMD_SETPARAMS ne fonctionne pas !
ParErr_BufToBig	(2)	Le tampon de lecture/écriture est trop gros
ParErr_InvParam	(3)	Cette modification de paramètre n'est pas autorisée (pour cette version du device parallèle). Pour l'instant, seul PARF_EOFMODE est autorisé à modifier les terminateurs
ParErr_LineErr	(4)	Erreur dans le transfert
ParErr_NotOpen	(5)	Erreur au moment de l'ouverture du device (peut-être parallel.device n'existe-t-il pas dans le répertoire devs de la disquette SYS ?)
ParErr_PortReset	(6)	Reset du port Centronics (interface parallèle)
ParErr_InitErr	(7)	Une erreur est survenue lors de l'initialisation du device parallèle (OpenDevice())

4.3.6. Les affectations du port Centronics

Pin	A500	A1000	A2000
1	STROBE	DRDY	STROBE
2	Data0	Data0	Data0
3	Data1	Data1	Data1
4	Data2	Data2	Data2
5	Data3	Data3	Data3
6	Data4	Data4	Data4

7	Data5	Data5	Data5
8	Data6	Data6	Data6
9	Data7	Data7	Data7
10	ACK	ACK	ACK
11	BUSY	BUSY	BUSY
12	POUT	POUT	POUT
13	SEL	SEL	SEL
14	+5v	GND	+5v
15	NC	GND	NC
16	RESET	GND	RESET
17	GND	GND	GND
18	GND	GND	GND
19	GND	GND	GND
20	GND	GND	GND
21	GND	GND	GND
22	GND	GND	GND
23	GND	+5v	GND
24	GND	NC	GND
25	GND	RESET	GND

Avec A1000, vous devez utiliser des prises femelles DSUB à 25 broches, si vous désirez connecter une imprimante, ou un périphérique analogue. Avec A500 et A2000, vous devez utiliser au contraire une prise DSUB mâle.

Attention : N'utilisez jamais de câble d'imprimante IBM pour votre A1000. Cela conduit à un court-circuit qui détruit votre imprimante. Les câbles convenables peuvent être achetées chez un marchand spécialisé.

4.4. Le device série

Le device série permet d'accéder à l'interface série de l'Amiga. Ici aussi, il existe un bloc de device :

```

Offset      Structure
-----
          struct IOExtSer
{
 0 0x00      struct IOStdReq IOSer;
48 0x30      ULONG        io_CtlChar;    /* Protocole de transfert */
52 0x34      ULONG        io_RBufLen;   /* Taille du tampon de
lecture */
56 0x38      ULONG        io_ExtFlags;  /* non utilisé */
60 0x3c      ULONG        io_Baud;      /* Baudrate */
64 0x40      ULONG        io_BrkTime;   /* Durée de pause */

```

```

68 0x44      struct IOTArray io_TermArray; /* Terminateurs */
76 0x4c      UBYTE          io_ReadLen;   /* 7 ou 8 Bits */
77 0x4d      UBYTE          io_WriteLen;  /* 7 ou 8 Bits */
78 0x4e      BYTE           io_StopBits;  /* 0, 1, 2 */
79 0x4f      BYTE           io_SerFlags; /* cf. SetParams */
80 0x50      WORD           io_Status;    /* cf. Query */
82 0x52  };      /* défini dans
"devices/serial.h" */

```

Vous voyez qu'on a créé ici encore un Terminator-Array (cf. device parallèle). Cet array est composé lui aussi de 8 octets ou de 2 mots longs :

Offset	Structure
-----	-----
	struct IOTArray
	{
0 0x00	ULONG TermArray0;
4 0x04	ULONG TermArray1;
8 0x08 }:	/* défini dans "devices/serial.h" */

Voyons maintenant quelles sont les commandes comprises par le device série :

MD_RESET	(1)	Rétablissement le device dans l'état qui suit directement OpenDevice() (y compris TermArray)
CMD_READ	(2)	Lit des données
CMD_WRITE	(3)	Ecrit des données
CMD_STOP	(6)	Arrête lecture/écriture
CMD_START	(7)	Poursuit lecture/écriture
CMD_FLUSH	(8)	Ignorer les commandes Read et Write encore existantes

Il existe en outre trois commandes spécifiques au device :

SDCMD_QUERY	(9)	Obtenir l'état
SDCMD_BREAK	(10)	Arrêter le transfert
SDCMD_SETPARAMS	(11)	Modifier les paramètres

4.4.1. L'ouverture du device

L'ouverture du device série est aussi simple à réaliser que celle du device parallèle :

```

struct IOExtSer *SerReq = 0L;
#define SER_LEN (ULONG) sizeof(struct IOExtSer)
...
    Open_A_Device("serial.device", 0L, &SerReq, 0L, SER_LEN);
...

```

Vous êtes à nouveau le seul à avoir l'autorisation d'accéder à ce device. Si un autre programme a déjà annoncé son accès, votre demande est rejetée. C'est pourquoi vous avez ici encore la possibilité de partager le device série avec d'autres utilisateurs :

```

struct IOExtSer *SerReq = 0L;
#define SER_LEN (ULONG) sizeof(struct IOExtSer)
VOID *GetDeviceBlock();
...
    SerReq = (struct IOExtSer *) GetDeviceBlock(SER_LEN);
    SerReq->io_SerFlags = (UBYTE) SERF_SHARED;

```

```
Open_A_Device ("serial.device", 0L, &SerReq, 0L, 0L);
...
```

SERF_SHARED a ici la valeur 32 (comme PARF_SHARED pour le device parallèle). Veillez à bien déclarer GetDeviceBlock() comme fonction avec un pointeur comme valeur de retour. Sans quoi vous obtiendrez avec le résultat une extension à un mot long (ext.1 d0). Ceci aurait pour effet de provoquer l'effondrement du device série (cette petite négligence nous a personnellement coûté beaucoup de temps et d'énervernement !). Vous pouvez maintenant passer à la lecture et à l'écriture des données :

4.4.2. Lire et écrire par l'intermédiaire de l'interface série

Pour la lecture et l'écriture, vous disposez évidemment des commandes standard CMD_READ et CMD_WRITE:

```
*****
*                               Serial_Read()          (Ser_Support)*
*
* Fonction: Lire des données
*-----*
* Paramètres d'entrée:
*
* SerReq: Bloc de device
* Data:   Tampon des données
* Len:    Nombre des données à lire
*****
VOID Serial_Read(SerReq, Data, Len)
struct IOExtSer *SerReq;
APTR                         Data;
ULONG                        Len;
{
    SerReq->IOSer.io_Data = Data;
    SerReq->IOSer.io_Length = Len;
    Do_Command(SerReq, (UWORD) CMD_READ);
}
*****
*                               Serial_Write()        (Ser_Support)*
*
* Fonction: Ecrire des données
*-----*
* Paramètres d'entrée:
*
* SerReq: Bloc de device
* Data:   Données à sortir
* Len:    Nombre des données à sortir
*****
VOID Serial_Write(SerReq, Data, Len)
struct IOExtSer *SerReq;
APTR                         Data;
ULONG                        Len;
{
    SerReq->IOSer.io_Data = Data;
    SerReq->IOSer.io_Length = Len;
```

```
    Do_Command(SerReq, (UWORD) CMD_WRITE);
}
```

Vous transmettrez à ces deux commandes uniquement l'adresse des données à sortir, ou l'adresse du secteur de mémoire dans lequel les données lues doivent être inscrites, et le nombre des données à transmettre. Il faut tenir compte pour cela de ce qui suit :

Une valeur de -1 pour Len lors de l'écriture a pour effet de mettre fin à l'opération d'écriture après envoi d'un octet nul. Lors de la lecture, il faut cependant distinguer le cas où on utilise le TermArray. Dans ce cas, la lecture se poursuit jusqu'à ce qu'un caractère soit reçu à partir du TermArray. Sinon, la lecture sera interrompue lorsqu'un octet nul interviendra.

Un petit conseil : les fonctions ci-dessus utilisent la commande DoIO() pour l'exécution des commandes. On peut cependant fort bien travailler avec SendIO(), CheckIO() et WaitIO(), pour utiliser de manière cohérente un temps de transfert long lorsqu'on a beaucoup de données.

4.4.3. Les différents paramètres du device série

Vous avez sûrement entendu parler des interfaces série, ou lu quelque chose sur le sujet. Vous avez dans ce cas rencontré des notions comme : protocole de transfert, vitesse en bauds, BreakTime, bits de stop, parité, etc. Ces paramètres peuvent tous être définis et interrogés à l'aide du device série. Nous allons commencer pour cela par dire quelques mots sur l'interface série :

Tout d'abord, il faut savoir qu'un octet est transféré ici bit par bit. Il est clair que des erreurs peuvent surgir durant ce processus. Pour transférer un octet, il peut en effet se produire huit fois plus d'erreurs que dans le cas de l'interface parallèle. Pour éviter ces erreurs, on a mis en place différents protocoles de transfert, à l'aide desquels on peut définir une erreur de transfert et amener l'émetteur à envoyer à nouveau le dernier octet.

Le device série soutient en ce moment le protocole de transfert xON et xOFF. Ce protocole n'est pas utilisé avec un bit SERF_XDISABLED posé (bit 7 = 128). Pour le reste, il est standard (après OpenDevice()). Il est désactivé avec SerReq-io_SerFlags = SERF_XDISABLED. Si vous voulez l'utiliser, vous pouvez utiliser des caractères avec Ctrl. Ceux-ci sont définis dans SerReq-io_CtrlChar. Comme le TermArray, cet élément est constitué d'un ULONG qui peut recevoir les codes ASCII des caractères : les bits 31-24 définissent le caractère xON, et les bits 23-16 définissent le caractère xOFF. Les bits 15-8 doivent recevoir le caractère INQ, alors que les bits 7-0 doivent être utilisés pour le caractère ACK. Les deux derniers protocoles (handshaking) ne sont pas encore soutenus.

En relation avec le protocole de transfert, on trouve aussi le nombre de bits à envoyer pour un octet. Vous avez la possibilité d'envoyer 7 ou 8 bits d'un octet (SerReq-ioReadLen = (BYTE) 7 ou 8 et SerReq-io_WriteLen = (BYTE) 7 ou 8). Après les 7 ou 8 bits envoyés, on a un bit de stop, qui marque la fin de la valeur transférée. On utilise souvent 7 bits pour envoyer un code ASCII au sens strict. Les codes ASCII

prennent ici uniquement les valeurs qui vont de 0 à 0x7f. Pour augmenter la sécurité du transfert, on peut augmenter à 2 unités le nombre de bits de stop lorsque l'on effectue le transfert avec 7 bits. (SerReq-io_StopBit = (BYTE) 2);

Outre le protocole de transfert, il faut évidemment définir la vitesse de transfert des données. Il faut pour cela fournir la vitesse en bauds. Elle indique le nombre de bits transférés par seconde. Avec l'Amiga, vous avez la possibilité de transférer les bits à une vitesse de 112 à 29200 bits par seconde (bauds). Indiquez pour cela simplement la vitesse en bauds dans SerReq-io_Baud. Normalement, on peut aussi effectuer le transfert avec 110 bauds. Mais l'Amiga ne peut traiter, pour des raisons liées au hardware, qu'une vitesse de 112 bauds.

Si vous voulez interrompre le transfert des données, vous devez envoyer un signal Break. Ce signal est généré en mettant tous les conduits sur 0 pendant un laps de temps déterminé. Le temps pendant lequel les conduits doivent rester dans l'état Low est indiqué en micro-secondes dans SerReq-io_BrkTime. Un break est envoyé avec la commande SDCMD_BREAK. Il faut noter que l'on doit spécifier les mêmes paramètres sur l'émetteur et le récepteur, sans quoi le transfert de données n'a pas lieu.

Revenons maintenant au device série. Outre les paramètres ayant trait à la technique de transfert, l'Amiga gère également quelques paramètres qui sont plutôt en rapport avec le niveau logiciel. Le device série gère tout d'abord un tampon de lecture qui lui est propre. Normalement, ce tampon a une taille de 512 octets. Si cette taille vous paraît insuffisante, vous pouvez l'augmenter. Il faut pour cela indiquer dans SerReq-io_RBufLen la nouvelle longueur, et exécuter SDCMD_SETPARAMS. Le nouveau tampon est alors créé. Les données contenues auparavant dans l'ancien tampon sont perdues. En outre, tous les paramètres modifiés sont communiqués au device série une fois seulement que SETPARAMS a été exécuté. C'est aussi le cas pour une modification des terminateurs.

Il suffit pour cela d'indiquer les 8 (ou moins) nouveaux terminateurs qui mettent fin à une commande de lecture (Len = -1), en posant le flag EOFMODE dans SerReq-io_SerFlag. Après SETPARAMS, les terminateurs seront utilisés. Il faut noter que la seule modification de paramètre possible pendant une opération de lecture ou d'écriture est celle du paramètre SERF_XDISABLED. Toutes les autres modifications se concluent par une erreur. Outre les paramètres du bloc de device, il existe encore d'autres flags, qui peuvent influencer le transfert de données. Voici une liste de tous les flags et de leurs actions :

SERF_PARITY_ON (1)

Verifie la parité du bit reçu.

SERF_PARITY_ODD (2)

Vérifie si la parité est impaire (somme des bits = 1). Si ce bit n'est pas posé, c'est la parité paire (Even parity) qui est utilisée.

SERF_7WIRE (4)

Normalement, on n'utilise que trois conduits pour le transfert des données. Ce sont TXD (Transmit Data), RXD (Receive Data) et GND (Ground). Avec SERF_7WIRE, qui ne peut toutefois être posé que devant OpenDevice() (cf. SERF_SHARED), on dispose également des conduits suivants : RTS (REQUEST TO SEND), CTS (CLEAR TO SEND), DSR (DATA SET READY) et DCD (DATA CARRIER DETECT).

SERF_QUEUEDBRK (8)

La commande SDCMD_BREAK n'est envoyée qu'après le processus de lecture/écriture en cours, sinon il est envoyé tout de suite. Ce flag peut être posé sans SETPARAMS.

SERF_RADBOOGIE (16)

Pour augmenter la vitesse de lecture, on se sert de ce flag pour désactiver l'overhead du software au moment de la lecture. La parité n'est plus vérifiée, le protocole xON et XOFF est contourné, et le transfert se fait toujours avec 8 bits par caractère. Cette augmentation de la vitesse est utile surtout pour les freaks MIDI.

SERF_SHARED (32)

Vous partagez l'interface série avec d'autres utilisateurs. Ce flag ne peut être posé que devant OpenDevice() ou Open_A_Device().

SERF_EOFMODE (64)

A l'aide de ce flag, vous pouvez activer le TermArray. Il peut être posé sans SETPARAMS, pour activer et désactiver les terminateurs définis.

SERF_XDISABLED (128)

Le protocole xON-xOFF est désactivé. Il est actif après OpenDevice(). Ce flag n'a d'effet qu'après SETPARAMS. SETPARAMS est utilisé par le device série pour ainsi dire comme indicateur d'une modification de paramètre.

4.4.4. L'état de l'interface

Comme pour l'interface parallèle, vous pouvez obtenir l'état de l'interface série. Appelez pour cela Do_Command (SerReq, (UWORD) SDCMD_QUERY), et vous obtiendrez en retour dans SerReq-ioStatus l'état de l'interface :

- | | |
|-------|--------------------------------------|
| Bit 0 | - 0 (BUSY)
- 1 (pas de transfert) |
| Bit 1 | - 0 Paper Out |

Bit 2	- 1 il y a encore du papier - 0 ONLINE
Bit 3	- 0 Data Set Ready - 1 No Data
Bit 4	- 0 Clear To Send - 1 Not Clear
Bit 5	- 0 Carrier Detect (il y a un signal porteur) - 1 No Carrier
Bit 6	- 0 Ready To Send - 1 Not Ready
Bit 7	- 0 Data Terminal Ready - 1 Not Ready
Bit 8	- 1 Read Buffer Overrun (Readbuffer plein) - 0 no Overrun
Bit 9	- 1 Break Sent (Envoi d'un Break) - 0 No Break
Bit 10	- 1 Break received (Break reçu) - 0 No Break
Bit 11	- 1 transmit XOFFed (x0FF envoyé) - 0 No XOFF
Bit 12	- 1 received XOFFed (x0FF reçu) - 0 No XOFF
Bits 13-15	non utilisé

Outre l'interrogation du mot définissant l'état, on peut aussi interroger la variable `SerReq-IOSer.io_Flags`. Voici les états les plus importants qui sont déposés ici :

IOSERF_OVERRUN	(1) Read Buffer Overrun (dépassement du tampon interne de lecture)
IOSERF_WROTEBREAK	(2) Break envoyé
IOSERF_READBREAK	(4) Break reçu
IOSERF_XOFFWRITE	(8) xOFF écrit
IOSERF_XOFFREAD	(16) xOFF reçu
IOSERF_ACTIVE	(32) Accès lecture ou écriture en cours ou annoncé
IOSERF_ABORT	(32) AbortIO() a été exécuté
IOSERF_QUEUED	(64) Accès lecture/écriture annoncé mais pas encore exécuté. car un autre accès est déjà actif
IOSERF_BUFREREAD	(128) Données lues à partir du tampon interne

Comme vous le constatez, le bit 4 a deux affectations. A cause de cette erreur dans les Includes, et aussi dans les manuels, on ne peut malheureusement pas dire quels sont les bits qui correspondent à tel ou tel état. C'est pourquoi il vaut mieux éviter d'interroger la variable `io_Flags`, et procéder directement par l'intermédiaire de `SDCMD_QUERY` et `io_Status`.

4.4.5. Messages d'erreur

Malheureusement, il faut ici encore fournir un effort d'apprentissage avant d'en récolter les fruits. Cela veut dire que l'on commet très facilement des erreurs au début. Voici la liste des erreurs possibles reconnues par le device série et transmises à l'utilisateur :

SerErr_DevBusy	(1) Lecture ou écriture en cours SETPARAMS ne peut pas être exécuté
----------------	--

SerErr_BaudMismatch	(2)	Les vitesses de transfert ne coïncident pas
SerErr_InvBaud	(3)	la vitesse de transfert ne se trouve pas entre 112 et 29200 bauds
SerErr_BuffErr	(4)	La taille du tampon interne est inférieure à 512 octets, ou est trop grande (plus assez de place en mémoire)
SerErr_InvParam	(5)	Modification de paramètre non autorisée
SerErr_LineErr	(6)	Erreur de transfert (conduit défectueux ?)
SerErr_NotOpenu	(7)	serial.device dans le répertoire devs de la disquette SYS?
SerErr_PortReset	(8)	Reset de l'interface
SerErr_ParityErru	(9)	Erreur de parité lors du transfert
SerErr_InitErr	(10)	Erreur dans l'initialisation du device
SerErr_TimeErru	(11)	Erreur dans io_BrkTime
SerErr_BuffOverflow	(12)	Read Buffer Overflow
SerErr_NoDSRu	(13)	Pas de Data Set Ready Signal
SerErr_NoCTS	(14)	Pas de Clear To Send Signal
SerErr_DetectedBreak	(15)	Break lu

Interface série

Pour conclure, voici encore un tableau qui indique la définition des différentes interfaces série des A1000, A500 et A2000, une liaison courante pour un modem zéro, servant à relier deux ordinateurs par l'intermédiaire de l'interface série, et un petit programme pour utiliser ce modem zéro :

Pin	A500	A1000	A2000	
1	GND	GND	GND	(Ground)
2	TXD	TXD	TXD	(Transmit Data)
3	RXD	RXD	RXD	(Received Data)
4	RTS	RTS	RTS	(Request to send (demande d'écriture))
5	CTS	CTS	CTS	(Clear to send)
6	DSR	DSR	DSR	(Data set ready)
7	GND	GND	GND	(Signal Ground)
8	DCD	DCD	DCD	(Data Carrier Detect (signal porteur reçu))
9	+12v		+12v	
10	-12v		-12v	
11	AUDIO		AUDIO	(Audio Output)
12				
13				
14		-5v		
15		AUDIO		(Audio Output)
16		AUDI		(Audio Input)

17		EB		(716 KHZ)
18	AUDI	INT2*	AUDI	(Interrupt externe IRQ)
19				
20	DTR	DTR	DTR	(Data terminal ready)
21		+5v		
22	RI		RI	(Ring indicator)
23		+12v		
24		C2*		(3.58 MHZ)
25		RESB*		(reset avec tampon)

Veillez à ce que les prises que vous devez introduire derrière l'Amiga soient bien des prises DSUB à 25 broches (mâles pour A1000, femelles pour A500 et A2000). Pour le modem zéro, on croise simplement les conduits RS232 :

Pin	Computer A	Computer B	Pin
1	GND	GND	1
2	TXD	RXD	3
3	RXD	TXD	2
4	RTS	DCD	8
5	CTS	DCD	8
6	DSR	DTR	20
20	DTR	DSR	6
8	DCD	RTS	4
7	GND	GND	7
8	DCD	CTS	4

Vous reliez donc la broche 2 d'un côté avec la broche 3 de l'autre côté, etc. Le programme suivant, qui fonctionne aussi bien comme émetteur que comme récepteur, est très utile pour tester le câble de liaison :

```
*****
*                               Ser.c
*                               Août 1988
*                               (c) Bruno Jennrich
*
* Fonction: Adresser l'interface série
*****
*/
* Compile-Info:
*
* cc Ser
* ln Ser.o Ser_Support.o -Devs_Support.o -lc
*****
#include "exec/types.h"
```

```

#include "exec/io.h"
#include "devices/serial.h"
struct IOExtSer *SerReq;
#define SER_LEN (ULONG) sizeof(struct IOExtSer)
/*********************************************
*           CloseIt()                  (User)*
*
* Fonction: Tout fermer en cas d'erreur
*-----
* Paramètres d'entrée:
* String: Error-Message
*****************************************/
VOID CloseIt(String)
char      *String;
{
    WORD i;
    WORD *dff180 = (WORD *)0xdff180;
    WORD Error = 0;
    if (strlen(String) > 0)
    {
        for(i=0;i<0xffff;i++) *dff180 = i;
        puts(String);
        Error = 10;
    }
    if (SerReq != 0L) Close_A_Device(SerReq);
    exit(Error);
}
/*********************************************
*           main()                   *
*
* Paramètres d'entrée:
* argc > 1 => Lire des données
* argc == 0 > Ecrire des données
*****************************************/
main (argc, argv)
WORD argc;
BYTE    *argv[];
{
    BYTE Buffer[256];
    Open_A_Device("serial.device", 0L, &SerReq, 0L, SER_LEN);
    if (argc > 1)
    {
        Serial_Read(SerReq, Buffer, -1);
        printf ("%s \n", Buffer);
    }
    else
        Serial_Write(SerReq, "HALLO", -1);
    Close_A_Device(SerReq);
}

```

Sur l'ordinateur qui reçoit, appelez le programme avec un paramètre de commande quelconque (par exemple "Ser x"). Cet ordinateur attend alors que des données lui soient envoyées. Sortez la disquette, lancez le programme cette fois sans paramètre de commande ("Ser"), et considérez ce qui se passe : le texte envoyé s'affiche sur l'ordinateur récepteur.

4.5. Le device d'imprimante

Venons-en maintenant au device d'imprimante. A l'aide de ce device, on peut adresser l'imprimante de manière simple. Il faut distinguer ici trois types d'accès :

① Impression de textes

CMD_WRITE(3) et PRD_RAWWRITE(9)

② Sortie de commandes

PRD_PRTCOMMAND(10)

③ Copies d'écran

PRD_DUMPRPORT(11)

Pour ces trois types de tâches du device d'imprimante, on utilise trois blocs de device différents. Pour l'impression des textes, il s'agit d'un bloc IOStdReq simple. Pour les commandes à l'imprimante et les copies d'écran, il existe deux blocs de device spéciaux :

Offset	Structure

	struct IOPrtCmdReq /* Command Request */
	{
0 0x00	struct Message io_Message;
20 0x14	struct Device *io_Device;
24 0x18	struct Unit *io_Unit;
28 0x1c	WORD io_Command; /* PRD_PRTCOMMAND */
30 0x1e	BYTE io_Flags;
31 0x1f	BYTE io_Error;
32 0x20	WORD io_PrtCommand; /* Commande impression*/
34 0x22	BYTE io_Parm0; /* paramètre */
35 0x23	BYTE io_Parm1;
36 0x24	BYTE io_Parm2;
37 0x25	BYTE io_Parm3;
38 0x26	} /* défini dans "devices/printer.h" */
Offset	Structure

	struct IODRPReq /* DumpRastPort Request */
	{
0 0x00	struct Message io_Message;
20 0x14	struct Device *io_Device;
24 0x18	struct Unit *io_Unit;
28 0x1c	WORD io_Command; /* PRD_PRTCOMMAND */
30 0x1e	BYTE io_Flags;
31 0x1f	BYTE io_Error;
32 0x20	struct RastPort *io_RastPort; /* RastPort graphique */
36 0x24	struct ColorMap *io_ColorMap; /* Table couleurs */
40 0x28	ULONG io_Modes; /* ViewPort-Modes */
44 0x2c	WORD io_SrcX; /* point de départ */
46 0x2e	WORD io_SrcY;
48 0x30	WORD io_SrcWidth; /* Largeur */
50 0x32	WORD io_SrcHeight; /* Hauteur */
52 0x34	LONG io_DestCols; /* Largeur impression */
56 0x38	LONG io_DestRows; /* Hauteur impression */

```
60 0x3c      UWORD          io_Special; /* Special-Flags */
62 0x3e    }             /* défini dans "devices/printer.h" */
```

Pour avoir accès à tous ces blocs de device d'un seul coup, il existe le moyen suivant d'ouvrir le device d'imprimante

```
struct IODRPReq *PrtPtr = 0L; /* pointeur dummy*/
struct IOStdReq *Normal;
struct IODRPReq *DumpRastPort;
struct IOPrtCmdReq *Command;
#define PRT_LEN (ULONG) sizeof (struct IODRPReq) /* Bloc de plus grande
taille */
Open_A_Device ("printer.device", 0L, &PrtPtr, 0L, PRT_LEN);
Normal = (struct IOStdReq *)PrtPtr;
DumpRastPort = (struct IODRPReq *)PrtPtr;
Command = (struct IOPrtCmdReq *)PrtPtr;
```

Par l'intermédiaire des pointeurs Normal, DumpRastPort et Command, vous avez ainsi accès au bloc de device, avec une gestion des différentes variables contenues dans ces blocs. Considérons d'abord l'impression des textes :

4.5.1. Imprimer des textes non traités

Vous pouvez envoyer vos textes à l'imprimante avec la commande PRD_RAWWRITE. Les séquences Escape qui se trouvent éventuellement dans le texte ne sont pas remplacées (cf. CMD_WRITE). Ce qui est spécifié comme chaîne de sortie est imprimé tel quel :

```
*****
*           Printer_RawWrite()      (Printer_Support)*
*
* Fonction: Sortir les données
*-----*
* Paramètres d'entrée:
* PrtReq: Bloc de device (Normal)
* Data:   String à sortir
* Len:    Nombre de caractères à sortir
*****
Printer_RawWrite(PrtReq, Data, Len)
struct IOStdReq *PrtReq;
APTR               Data;
ULONG              Len;
{
    PrtReq->io_Data = Data;
    PrtReq->io_Length = Len;
    Do_Command(PrtReq, (UWORD) PRD_RAWWRITE);
}
```

4.5.2. La substitution des séquences Escape

L'Amiga est en mesure de traduire des séquences Escape déterminées en séquences Escape correspondant à l'imprimante utilisée grâce aux pilotes d'imprimante (qui se trouvent d'ailleurs dans le répertoire devs/printers, et qui peuvent être sélectionnés avec le Tool de Preferences). Le tableau suivant indique les séquences Escape soutenues par l'Amiga et traduites pour l'imprimante :

Commande :			
Numéro	Séquence Escape	Signification	
aRIS	0L	ESCC	Reset
aRIN	1L	ESC#1	Initialiser
aIND	2L	ESCD	LineFeed
aNEL	3L	ESCE	LF = CR+LF
aRI	4L	ESCM	LineFeed vers l'arrière
aSGRO	5L	ESC[0m	Caractères imprimante normaux
aSGR3	6L	ESC[3m	italique activé
aSGR23	7L	ESC[23m	italique désactivé
aSGR4	8L	ESC[4m	Underline activé
aSGR24	9L	ESC[24m	Underline désactivé
aSGR1	10L	ESC[1m	gras activé
aSÜR22	11L	ESC[22m	gras désactivé
aSFC	12L	ESC[30m-	couleur de premier plan
		ESC[39m	
aSBC	13L	ESC[40m-	Couleur du fond
		ESC49m	
aSHORPO	14L	ESC[0w	Police normale
aSHORP2	15L	ESC[2w	Elite activé
aSHORP1	16L	ESC[1w	Elite désactivé
aSHORP3	18L	ESC[3w	Condensed désactivé
aSHORP6	19L	ESC[6w	caractères larges activé
aSHORP5	20L	ESC[5w	caractères larges désactivé
aDEN6	21L	ESC[6"z	ombré activé
aDEN5	22L	ESC[5"z	ombré désactivé
aDEN4	23L	ESC[4"z	Doublestrike activé
aDEN3	24L	ESC[3"z	Doublestrike désactivé
aDEN2	25L	ESC[2"z	NLO activé
aDEN1	26L	ESC[1"z	NLO désactivé
aSUS2	27L	ESC[2v	
aSUS1	28L	ESC[1v	

aPLU	32L	ESC L	ligne de partage vers le haut
aPLD	33L	ESC K	ligne de partage vers le bas
aFNT0	34L	ESC(B	Police US
aFNT1	35L	ESC(R	Police française
aFNT2	36L	ESC(K	Police allemande
aFNT3	37L	ESC(A	Police anglaise
aFNT4	38L	ESC(E	1ère police danoise
aFNT5	39L	ESC(H	Police suédoise
aFNT6	40L	ESC(Y	Police italienne
aFNT7	41L	ESC(Z	Police espagnole
aFNT8	42L	ESC(J	Police japonaise
aFNT9	43L	ESC(6	Police norvégienne
aFNT10	44L	ESC(C	2ème police danoise
aPROP2	45L	ESC[2p	Caractères proportionnels activé
aPROPO	47L	ESC[0p	Proportional clear
aTSS	48L	ESC[n E	Définir Proport.offset (n)
aJFY5	49L	ESC[5 F	Aligné à gauche
aJFY7	50L	ESC[7F	Aligné à droite
aJFY6	51L	ESC[6F	Justifié
aJFY3	53L	ESC[3F	Format "Letter Space"
aJFY1	54L	ESC[1F	Centré
aVERPO	55L	ESC[0z	1/8" Interligne
aVERP1	56L	ESC[1z	1/6" Interligne
aSLPP	57L	ESC[nt	Définir la longueur du papier (n)
aPERF	58L	ESC[nq	Intervalle des performances (n>0)
aPERFO	59L	ESC[0q	Pas d'intervalle de perforations
aLMS	60L	ESC#9	Définir la marge gauche
aRMS	61L	ESC#0	Définir la marge droite
aTMS	62L	ESC#8	Définir la marge supérieure
aBMS	63L	ESC#2	Définir la marge inférieure
aSTBM	64L	ESC[Pn1:]	Définir marges supérieure (n1) et inférieure (n2)
			Pn2r
aSLRM	65L	ESC[Pn1:]	Définir marges gauche (n1) et droite (n2)
			Pn2s

aCAM	66L	ESC#3	Supprimer toutes les marges
aHTS	67L	ESCH	Tabulations horizontales
aVTS	68L	ESCJ	Tabulations verticales
aTBC0	69L	ESC[0g	Supprimer tab horizontale
aTBC3	70L	ESC[3g	Supprimer toutes les tabs horizontales
aTBC1	71L	ESC[1g	Supprimer tab verticale
aTBC4	72L	ESC[4g	Supprimer toutes les tabs verticales
aTBCALL	73L	ESC#4	Supprimer toutes les tabs
aTBSALL	74L	ESC#5	Définir Default-Tabs
aEXTEND	75L	ESC[Pn"x	Extended-Font

L'avantage de ce tableau est évident. Si l'on veut par exemple écrire un texte contenant des titres soulignés, le traitement de texte se contente d'envoyer à l'impression la commande "ESC[4m". Il importe peu de savoir quelle est ici l'imprimante connectée. En effet, chaque pilote d'imprimante contient une table semblable, avec les séquences Escape spécifiques à l'imprimante dirigée. Dans cette table, on trouve par exemple comme huitième entrée la séquence "ESC-1" (Star NL-10). Le device d'imprimante remplace "ESC[4m" par "ESC-1", et le texte qui suit est souligné. Cela fonctionne uniquement lorsque l'on utilise la commande CMD_WRITE au lieu de PRD_RAWWRITE (dans ce dernier cas, il n'y a pas substitution effectuée).

```
*****
*           Printer_Write()      (Printer_Support)   *
*
* Fonction: Imprimer les données (Conversion des séquences Escape) *
*-----*
* Paramètres d'entrée:                                              *
*-----*
* PrtReq: Bloc de device (Normal)                                     *
* Data: String à imprimer                                         *
* Len: Nombre de caractères à imprimer                                *
*****
Printer_Write (PrtReq, Data, Len)
struct IOStdReq *PrtReq;
APTR             Data;
ULONG            Len;
{
    PrtReq->io_Data = Data;
    PrtReq->io_Length = Len;
    Do_Command(PrtReq, (UWORD) CMD_WRITE);
}
```

4.5.3. Les commandes d'imprimante

Vous avez en outre la possibilité de formuler ces séquences Escape sous forme de commandes d'imprimante. Si une telle séquence Escape contient en effet des paramètres (par exemple pour la définition des marges de droite et de gauche), on ne peut plus se contenter d'une simple substitution. Ici encore, il existe dans le pilote d'imprimante une routine qui traite les séquences Escape que l'on ne peut pas substituer. Cette routine peut être adressée directement avec la commande PRD_PRTCOMMAND :

```
*****
*                               Printer_Command()      (Printer_Support)*
*
* Fonction: Exécuter une commande imprimante
*-----*
* Paramètres d'entrée:
*
* PrtReq: Bloc de device
* Command: Commande
* P1-P4: Paramètres
*****
Printer_Command      (PrtReq, Command, P0, P1, P2, P3)
struct IOPrtCmdReq *PrtReq;
DWORD                Command;
UBYTE                P0, P1, P2, P3;
{
    PrtReq->io_PrtCommand = Command;
    PrtReq->io_Parm0 = P0;
    PrtReq->io_Parm1 = P1;
    PrtReq->io_Parm2 = P2;
    PrtReq->io_Parm3 = P3;
    Do_Command(PrtReq, (DWORD) PRD_PRTCOMMAND);
}
```

Pour définir alors les marges de droite et de gauche, il suffit d'appeler

```
Printer_Command (PrtReq,651, (BYTE)2, (BYTE)78, (BYTE)0, (BYTE)0);
```

Au lieu du 651, vous pouvez aussi mettre en œuvre le nom de la commande aSLRM. Les variables Parm0-Parm4 sont transmises à la routine du pilote d'imprimante.

4.5.4. Copies d'écran

Voici maintenant l'une des applications les plus intéressantes du device d'imprimante : les copies d'écran. Avec la commande PRD_DUMPRTPORT, vous pouvez imprimer facilement des graphiques. Il suffit de faire attention à quelques détails. Vous devez d'abord indiquer le rastport dans lequel se trouve le graphique à imprimer. Il faut également indiquer le ColorMap dans lequel sont contenues les couleurs du graphique. Ce dernier point est particulièrement important pour les imprimantes couleur. En outre, vous devez indiquer la variable de mode pour le ViewPort, afin que le device d'imprimante sache quel est le mode de présentation du graphique (HIRES, LACE,

HAM, etc.). Il ne reste plus alors qu'à indiquer la position du coin supérieur gauche (SrcX, SrcY) du graphique à imprimer, ainsi que la largeur et la hauteur (SrcWidth, SrcHeight). Vous pouvez de cette façon faire imprimer n'importe quelle section d'écran rectangulaire (Src = Source <M=> rastport).

En dernier, vous devrez encore spécifier la taille de la copie d'écran sur l'imprimante (DestRows, DestCols) (Dest = Destination <M=> Imprimante). Il faut toutefois faire attention à certaines choses : supposons que le flag io_Special soit égal à 0 ; dans ce cas, voici les effets des valeurs positives et négatives pour DestRows et DestCols :

```
DestCols>0
DestRows>0
```

Dans ce cas, le graphique reportera sur le papier des lignes dans DestRows et des colonnes dans DestCols (par exemple 320x200).

```
DestCols=0
DestRows>0
```

Ici, toute la largeur du papier sera utilisée, avec le nombre de lignes indiqué (par exemple 960x200).

```
DestCols=0
DestRows=0
```

Cette fois, le papier sera utilisé sur toute sa hauteur et toute sa largeur (par exemple 960x256).

```
DestCols>0
DestRows=0
```

Dans ce cas, la copie d'écran sera imprimée sur la largeur indiquée, et la hauteur sera adaptée à la largeur.

```
DestCols<0
DestRows>0
```

Vous pouvez ici augmenter ou diminuer les dimensions de la copie. Le facteur d'augmentation (resp. de réduction) se calcule comme suit :

$$\frac{|\text{DestCols}|}{\text{DestRows}} - \text{facteur d'agrandissement}$$

Si DestCols a par exemple la valeur -1, et DestRows la valeur 4, la valeur du facteur d'agrandissement (ici en fait un facteur de réduction) est égale à 1/4. Ces valeurs pour DestCols et DestRows ne sont valides que si io_Special est égal à 0. Dans ce qui suit, vous trouverez les Special_Flags et leurs effets sur l'impression :

SPECIAL_MILCOLS	0x001L	/*DestCols indiqué en 1/1000 de pouce*/
SPECIAL_MILROWS	0x002L	/*DestRows indiqué en 1/1000 de pouce*/
SPECIAL_FULLCOLS	0x004L	/*DestCols défini sur Maximum */
SPECIAL_FULLROWS	0x008L	/*DestRows défini sur Maximum */
SPECIAL_FRACCOLS	0x010L	/*Largeur = Maximum / DestCols*/
SPECIAL_FRACROWS	0x020L	/*Hauteur = Maximum / DestRows*/
SPECIAL_ASPECT	0x080L	/*Si ce flag est posé, Largeur ou Hauteur sont

```

    modifiés, pour conserver les relations entre
    les côtés*/
/*épaisseur d'impression (1=inférieure;
4=supérieure)*/

SPECIAL_DENSITY1 0x100L
SPECIAL_DENSITY2 0x200L
SPECIAL_DENSITY3 0x300L
SPECIAL_DENSITY4 0x400L
SPECIAL_CENTER     0x040L      /*Centrer le graphique.*/

```

Quelques mots encore sur les "maxima" : le pilote d'imprimante contient la largeur et la hauteur de la surface imprimable maximale. Ces deux maxima permettent de calculer le rapport entre les pages (MaximaX/MaximaY), conservé par Special_Aspect.

```

*****
*                               Printer_Dump()      (Printer_Support)*
*
* Fonction: Hardcopy
*-----
* Paramètres d'entrée:
*
* PrtPtr:   Bloc de device
* RastPort: RastPort du graphique à imprimer
* ColorMap: ColorMap contient les couleurs actuelles
* Modes:    Mode d'affichage
* SrcX,
* SrcY:    Coin supérieur gauche du graphique à imprimer
* SrcWidth,
* SrcHeight: Largeur et Hauteur du graphique à imprimer
* DestCols: Nombre de colonnes (Imprimante)
* DestRows: Nombre de lignes (Imprimante)
* Special: Special-Flags
*****
VOID Printer_Dump(PrtPtr, RastPort, ColorMap, Modes, SrcX, SrcY,
SrcWidth, SrcHeight, DestCols, DestRows, Special)
struct IODRPReq *PrtPtr;
struct RastPort      *RastPort;
struct ColorMap      *ColorMap;
ULONG                Modes;
WORD                 SrcX, SrcY;
WORD                 SrcWidth, SrcHeight;
LONG                DestCols, DestRows;
WORD                 Special;
{
    PrtPtr->io_RastPort  = RastPort;
    PrtPtr->io_ColorMap = ColorMap;
    PrtPtr->io_Modes    = Modes;           /* Viewmodes */
    PrtPtr->io_SrcX     = SrcX;           /* Point de départ*/
    PrtPtr->io_SrcY     = SrcY;
    PrtPtr->io_SrcWidth = SrcWidth;       /* Largeur */
    PrtPtr->io_SrcHeight= SrcHeight;      /* Hauteur */
    PrtPtr->io_DestCols = DestCols;       /* Largeur impression*/
    PrtPtr->io_DestRows = DestRows;       /* Hauteur impression*/
    PrtPtr->io_Special  = Special;        /* Special-Flags */
    Do_Command(PrtPtr, (WORD) PRD_DUMPRPORT);
}

```

Le programme qui suit utilise la routine Dump, pour imprimer une section d'écran à partir de la fenêtre actuelle. Après le lancement, vous devez définir la section avec la souris, et c'est parti :

```
*****
*                               Prt.c
*                               Août 1988
*                               (c) Bruno Jennrich
*
* Fonction: Hardcopy de la fenêtre actuelle
*****
/**/
* Compile-Info:
*
* cc Prt
* ln Prt.o Printer_Support.o Devs_Support.o -lc
*****
```

```
#include "exec/types.h"
#include "exec/io.h"
#include "devices/printer.h"
#include "intuition/intuitionbase.h"
#include "intuition/intuition.h"
#include "graphics/gfxbase.h"
#include "graphics/view.h"
union PrinterIO
{
    struct IOStdReq    Normal;
    struct IODRPReq   DumpRastPort;
    struct IOPrtCmdReq Command;
};

struct IODRPReq *PrtPtr = 0L;
struct IOStdReq *Normal;
struct IODRPReq *DumpRastPort;
struct IOPrtCmdReq *Command;
#define PRT_LEN (ULONG) sizeof(struct IODRPReq)
struct IntuitionBase *IntuitionBase = 0L;
struct Window        *Window = 0L;
struct GfxBase       *GfxBase = 0L;
VOID *OpenLibrary();
*****
```

```
*                               CloseIt()          (User)*
*
* Fonction: Tout fermer en cas d'erreur
*-----
* Paramètres d'entrée:
*
* String: Error-Message
*****
```

```
VOID CloseIt(String)
char      *String;
{
    UWORLD i;
    UWORLD *dff180 = (UWORD *)0xdff180;
    UWORLD Error = 0;
    if (strlen(String) > 0L)
    {
        for(i=0;i<0xffff;i++) *dff180 = i;
        puts(String);
    }
}
```

```

        Error = 10;
    }
    if (PrtPtr != OL) Close_A_Device(PrtPtr);
    if (IntuitionBase == OL) CloseLibrary(IntuitionBase);
    if (GfxBase == OL) CloseLibrary(GfxBase);
    exit(Error);
}

/*********************************************
*                               Mark_PrintArea()      (User)*
*
* Fonction: Sélection secteur pour Hardcopy
*-----
* Paramètres d'entrée:
*
* Window: Adresse de la fenêtre à imprimer
* x1, y1, x2, y2: contiennent ultérieurement le coin supérieur
*                  et inférieur droit du secteur à imprimer
*****************************************/
Mark_PrintArea(Window, x1, y1, x2, y2)
struct Window *Window;
ULONG           *x1, *y1, *x2, *y2;
{
    UBYTE *LeftMouse = (UBYTE *)0xbfe001;
    ULONG xold, yold;
    *x1 = (ULONG) 0;
    *y1 = (ULONG) 0;
    *x2 = (ULONG) 0;
    *y2 = (ULONG) 0;
    SetDrMd(Window->RPort, COMPLEMENT);
    while((*LeftMouse & (UBYTE)0x40) == (UBYTE)0x40);
    *x1 = (ULONG)Window->MouseX;
    *y1 = (ULONG)Window->MouseY;
    xold = *x1;
    yold = *y1;
    Move(Window->RPort, *x1, *y1);          /* Premier rectangle */
    Draw(Window->RPort, xold, *y1);
    Draw(Window->RPort, xold, yold);
    Draw(Window->RPort, *x1, yold);
    Draw(Window->RPort, *x1, *y1);
    while((*LeftMouse & (UBYTE)0x40) == (UBYTE)0x0)
    {
        *x2 = (ULONG)Window->MouseX;
        *y2 = (ULONG)Window->MouseY;
        if ((*x2 != xold) || (*y2 != yold))
        {
            Move(Window->RPort, *x1, *y1);    /* Rectangle élastique */
            Draw(Window->RPort, xold, *y1);
            Draw(Window->RPort, xold, yold);
            Draw(Window->RPort, *x1, yold);
            Draw(Window->RPort, *x1, *y1);
            Move(Window->RPort, *x1, *y1);
            Draw(Window->RPort, *x2, *y1);
            Draw(Window->RPort, *x2, *y2);
            Draw(Window->RPort, *x1, *y2);
            Draw(Window->RPort, *x1, *y1);
            xold = *x2;
            yold = *y2;
        }
    }
}

```

```

        }

    }

Move(Window->RPort, *x1, *y1);      /* Supprimer le rectangle */
Draw(Window->RPort, xold, *y1);
Draw(Window->RPort, xold, yold);
Draw(Window->RPort, *x1, yold);
Draw(Window->RPort, *x1, *y1);
SetDrMd(Window->RPort, JAM2);
if (*x1 > *x2)
{
    xold = *x1;
    *x1 = *x2;
    *x2 = xold;
}
if (*y1 > *y2)
{
    yold = *y1;
    *y1 = *y2;
    *y2 = yold;
}
}

/*****
*          main()           (User) *
*****
main()
{
    ULONG x1, y1, x2, y2;
    if ((IntuitionBase = (struct IntuitionBase *) OpenLibrary("intuition.
library", OL)) == (struct IntuitionBase *) OL)
        CloseIt("No Intuition !!!");
    if ((GfxBase = (struct GfxBase *) OpenLibrary("graphics.library", OL)) == (struct GfxBase *) OL)
        CloseIt("No Graphics !!!");
    Window = IntuitionBase->ActiveWindow;
    Mark_PrintArea(Window, &x1, &y1, &x2, &y2);
    Open_A_Device("printer.device", OL, &PrtPtr, OL, PRT_LEN);
    Normal      = (struct IOStdReq *)PrtPtr;
    DumpRastPort = (struct IODRPReq *)PrtPtr;
    Command     = (struct IOPrtCmdReq *)PrtPtr;
    Printer_Dump(DumpRastPort,
                  Window->RPort,
                  GfxBase->ActiView->ViewPort->ColorMap,
                  (ULONG)GfxBase->ActiView->ViewPort->Modes,
                  (UWORD)x1,
                  (UWORD)y1,
                  (UWORD)(x2-x1),
                  (UWORD)(y2-y1),
                  (ULONG)(x2-x1),
                  (ULONG)(y2-y1),
                  (UWORD)SPECIAL_DENSITY4);

    CloseLibrary(GfxBase);
    CloseLibrary(IntuitionBase);
    Close_A_Device(PrtPtr);
}
}
```

4.5.5. Messages d'erreur du device d'imprimante

Malheureusement, des erreurs peuvent se produire pendant l'impression (les erreurs sont retournées dans les variables io_Error du bloc de device) :

`#define PDERR_NOERR 0`

Pas d'erreur - tout va bien.

`#define PDERR_`

L'impression a été interrompue (AbortIO()).

`#define PDERR_NOTGRAPHICS 2`

L'imprimante connectée ne soutient pas le graphisme.

`#define PDERR_INVERTHAM 3`

On ne peut pas sortir d'images HAM inversées ! (valide seulement pour Kick1.1 et Kick1.2).

`#define PDERR_BADDIMENSION 4`

Taille erronée à l'impression.

`#define PDERR_DIMENSIONOVFLOW 5`

La taille sélectionnée à l'impression est trop grande (seulement Kick1.1 et Kick1.2).

`#define PDERR_INTERNALMEMORY 6`

Pas assez de mémoire pour les variables internes.

`#define PDERR_BUFFERMEMORY 7`

Il n'a pas pu être créé une mémoire suffisante par le pilote d'imprimante pour le tampon d'impression.

4.5.6. Le device d'imprimante sous Kickstart V1.3

Il faut commencer par dire que les pilotes d'imprimante fonctionnent environ 20 fois plus vite. En outre, le device d'imprimante possède une nouvelle commande : la commande PRD_QUERY (`#define PRD_QUERY 12`). Vous l'avez déjà rencontrée avec le device parallèle et le device série. Elle permet d'obtenir l'état actuel du port. Le port choisi est celui auquel l'imprimante est connectée (selon Preferences). Pour connaître l'état, il faut créer deux UBYTE ou un UWORLD, et transmettre l'adresse de ce word au pointeur io_Data :

```
UWRD Status;  
Prt-io_Data = (APTR) &Status
```

Après l'appel de PRD_QUERY (Do_Command (PrtReq, (UWORD) PRD_QUERY), on trouve dans cet UWORD l'état demandé. On a besoin pour cela de l'UWORD entier dans le cas d'une imprimante série, alors que l'imprimante parallèle ne nécessite que l'octet inférieur (bits 0-7) de l'UWORD pour emmagasiner l'état. Pour savoir comment interpréter l'état, vous pouvez aller voir dans io_Actual s'il s'agit d'une imprimante série (io_Actual == 2) ou parallèle (io_Actual == 1). Pour connaître la signification des bits, reportez-vous aux sections sur les devices série et parallèle. Outre cette nouvelle commande, quelques nouveaux flags ont été également introduits :

```
#define SPECIAL_DENSITY5 0x0500
#define SPECIAL_DENSITY6 0x0600
#define SPECIAL_DENSITY5 0x0700
```

Vous pouvez maintenant sélectionner 7 épaisseurs d'impression au lieu de 4 auparavant (à condition que l'imprimante les soutienne).

```
#define SPECIAL_NORMFEED 0x0800
```

Après une copie d'écran, les imprimantes fonctionnant par page (par exemple les imprimantes laser ou les imprimantes avec alimentation feuille à feuille) exécutent normalement un formfeed. Cela veut dire que vous ne pouvez jamais imprimer plus d'un graphique sur une feuille de papier. Si vous posez toutefois ce flag, il n'y aura pas de formfeed. Pour poursuivre alors l'impression sur la même page, vous devez empêcher l'exécution d'un reset de l'imprimante;

```
#define SPECIAL_TRUSTME 0x1000
```

En effet, l'imprimante effectue un reset avant chaque copie d'écran. Pour l'empêcher, vous pouvez poser précisément ce flag.

```
#define SPECIAL_NOPRINT 0x2000
```

A l'aide de ce flag, vous empêchez l'imprimante d'imprimer une copie d'écran. Ce flag semble à première vue absurde : pourquoi aurait-on ouvert le device d'imprimante, pour décider ensuite de ne pas imprimer ? Mais sa signification devient claire si l'on se rappelle que les nouveaux pilotes d'imprimante calculent la taille de la copie d'écran sur la base des données définies par vos soins dans io_DestCols, io_YDotsInch, etc. La taille de la copie d'écran sur l'imprimante est alors retournée dans les variables io_DestCols et io_DestRows, qui contiennent le nombre de colonnes et de lignes dont la copie d'écran aura besoin. Le programme peut alors tester par exemple si la taille voulue peut être atteinte, et modifier ensuite au besoin certains paramètres.

4.6. Le device de clavier (Keyboard-device)

A l'aide du device de clavier, il est possible d'interroger directement le clavier. Nous disposons pour cela des commandes suivantes :

KBD_READEVENT	(9)	Préparer saisies clavier comme structures Input-Event
KBD_READMATRIX	(10)	Lire l'état de toutes les touches (pressées ou non)
CMD_CLEAR	(5)	Effacer le tampon de clavier

De plus, le device de clavier propose des routines servant à enclencher des routines de reset propres :

KBD_ADDRESETHANDLER	(11) Insérer Reset-Handler
KBD_REMOVESETHANDLER	(12) Supprimer Reset-Handler
KBD_RESETHANDLER-DONE	(13) Faire savoir qu'une routine de reset propre est traitée.

Malheureusement, nous sommes obligés de vous signaler sans tarder une défectuosité de ce device. La seule commande qui fonctionne est KBD_READEVENT. Toutes les autres commandes retournent une erreur, indiquant que la commande n'est pas implémentée (-0xfc). On doit en conclure que la tâche d'input (Input-Task) vérifie ce device et n'autorise aucun autre utilisateur. Nous décrirons cependant toutes les commandes.

4.6.1. L'ouverture du device de clavier

Le device de clavier utilise un bloc Standard-Request normal. Vous avez ensuite accès au device.

```
struct IOStdReq *KeyRequest = 0L;
#define KEY_LEN (ULONG) sizeof(struct IOStdReq)
...
    Open_A_Device("keyboard.device", 0L, &KeyRequest, 0L, KEY_LEN);
...
```

4.6.2. La lecture du device

Comme nous l'avons mentionné plus haut, vous pouvez lire le statut du clavier. Il existe pour cela deux moyens. Le premier consiste à faire remplir une structure Input-Event par le device d'input, et à interpréter ensuite la structure :

```
*****
*                               KeyBoard_ReadEvent()      (Key_Support)*
*
* Fonction: Lire un KeyBoard-Event
*-----
* Paramètres d'entrée:
*
* KeyRequest:   Bloc de device
* InputEvent:   Adresse du Input-Event à remplir
*****
VOID KeyBoard_ReadEvent(KeyRequest, InputEvent)
struct IOStdReq           *KeyRequest;
struct InputEvent          *InputEvent;
{
    KeyRequest->io_Data  = (APTR) InputEvent;
    KeyRequest->io_Length = (ULONG) sizeof(struct InputEvent);
    Do_Command(KeyRequest, (UWORD)KBD_READEVENT);
}
```

Le second moyen consiste à lire l'état de toutes les touches. Malheureusement, cela ne fonctionne pas toujours. Mais peut-être que ceci tient simplement à une erreur du système d'exploitation. Lorsque celle-ci aura été supprimée, vous pourrez vous servir de cette seconde façon d'interroger l'état du clavier. On transmet pour cela au pointeur io_Data du bloc de device l'adresse d'une byte-array, qui contiendra ultérieurement l'état de chaque touche. Chaque bit de cet array représente en effet une touche (bit = 0 = touche non pressée).

Le premier octet contient l'état pour les touches 0-7, le second pour les touches 8-15, etc. (cf. device d'input). Dans io_Length, il vous suffit d'indiquer le nombre d'octets de l'array, pour exécuter la commande KBD_READMATRIX. L'array devrait alors être rempli avec l'état de chacune des touches.

```
*****
*                               KeyBoard_ReadMatrix()      (Key_Support)*
*
* Fonction: Lire l'état des touches
*-----*
* Paramètres d'entrée:
*
* KeyRequest: Bloc de device
* Array:     Tableau d'octets (byte-array) pour l'état
* Len:       Taille du tableau (array)
*****
VOID KeyBoard_ReadMatrix(KeyRequest, Array, Len)
struct IOStdReq          *KeyRequest;
APTR                      Array;
ULONG                     Len;
{
    KeyRequest->io_Data   = (APTR) Array;
    KeyRequest->io_Length = Len;
    Do_Command(KeyRequest, (UWORD)KBD_READMATRIX);
}
```

4.6.3. Resets par l'intermédiaire du device de clavier

A l'aide du device de clavier, il est (sera ?) possible d'enclencher quelques routines de reset dans le système. Ces routines sont appelées lorsque l'on actionne la combinaison de touches Ctrl-Amiga-Amiga. Lors d'un reset, vous devez par exemple fermer les fichiers ouverts. On installe cette routine par l'intermédiaire d'une structure Interrupt, dont les éléments is_Code (code de programme) et is_Data (données) ont été initialisés. Cette structure Interrupt est transmise au pointeur io_Data du bloc de device, où la variable io_Length est également fixée à la valeur sizeof(struct Interrupt). On appelle alors la commande ADDRESETHANDLER :

```
struct Interrupt OwnReset;
...
OwnReset.is_Code = RoutinePropre;
OwnReset.is_Data = DonnéesPropres;
KeyReq->io_Data = (APTR) OwnReset;
KeyReq->io_Length = (ULONG) sizeof(struct Interrupt);
Do_Command(KeyReq, (UWORD) KBD_ADDRESETHANDLER);
```

Malheureusement, cette commande ne fonctionne pas non plus. La seule chose que l'on puisse faire avec un peu de "chance", c'est d'aboutir à un effondrement total, qui oblige à éteindre l'appareil, si c'est un A1000. La commande REMRESETHANDLER ne fonctionne pas plus. C'est la commande qui devrait permettre de supprimer le reset-handler personnel peu avant la fin du programme. Les variables qui doivent être initialisées sont les mêmes que pour ADDRESETHANDLER.

La dernière commande soutenue par le device de clavier est KBD_RESETHANDLERDONE. Cette commande doit être appelée après exécution de la routine de reset personnelle, pour dire au système que le reset-handler a été entièrement traité, et que le reset peut se poursuivre. Malheureusement, nous ne pouvons pas savoir si cette commande fonctionne ou pas : nous n'avons jamais pu installer de reset-handler.

4.6.4. Un exemple de programme

Le programme suivant lit à partir du device de clavier les événements Input, et sort le code de la touche pressée (ie_Code). Veillez à ne pas taper trop vite, car cela risque de déranger la sortie.

```
*****
*                               Key.c
*                               (c) Bruno Jennrich
*                               Août 1988
*****
/* Compile-Info:
 */
/* cc Key.c
 * ln Key.o Key_Support.o Devs_Support.o -lc
*/
#include "exec/types.h"
#include "exec/io.h"
#include "exec/devices.h"
#include "devices/inputhead.h"
#include "devices/keyboard.h"
struct InputEvent Event;
struct IOStdReq *KeyRequest=0L;
#define KEY_LEN (ULONG) sizeof(struct IOStdReq)
*****
*           CloseIt()                                (User)*
* Fonction: Tout fermer en cas d'erreur
*-----
* Paramètres d'entrée:
* String: Error-Message
*****
VOID CloseIt(String)
char      *String:
{
    WORD i;
    WORD *dff180 = (WORD *)0xdff180;
    WORD Error = 0;
    if (strlen(String) > OL)
```

```

    {
        for(i=0;i<0xffff;i++) *dff180 = i;
        puts(String);
        Error = 10;
    }
    if (KeyRequest != 0L) Close_A_Device(KeyRequest);
    exit(Error);
}
main()
{
    Open_A_Device("keyboard.device", 0L, &KeyRequest, 0L, KEY_LEN);
    do
    {
        KeyBoard_ReadEvent(KeyRequest, &Event);
        printf("ie_Code: %d\n", Event.ie_Code);
    } while(Event.ie_Code != (UWORD) 0x45); /* Escape */
    Close_A_Device(KeyRequest);
}

```

4.7. Le device du port de jeu (gameport)

Avec le device du port de jeu, il est possible d'interroger la souris et la manette connectées au port de jeu. Voici les commandes disponibles à cet effet :

GPD_READEVENT	(9) Lire état du contrôleur
GPD_ASKCTYPE	(10) Lire type du contrôleur
GPD_SETCTYPE	(11) Définir type du contrôleur
GPD_ASKTRIGGER	(12) Lire les conditions de l'annonce
GPD_SETTRIGGER	(13) Définir les conditions de l'annonce
CMD_CLEAR	(5) Effacer le tampon du gameport

Dès l'ouverture du device, vous devez définir le port de jeu que vous désirez interroger. Le paramètre Unit avec OpenDevice() contient 0 pour le port de jeu 1, ou 1 pour le port de jeu 2 :

```

#define GP_LEN (ULONG) (sizeof(struct IOStdReq))
struct IOStdReq *GamePortRequest = 0L;
...
Open_A_Device("gameport.device", 1L, &GamePortRequest, 0L, GP_LEN);
...
Close_A_Device(GamePortRequest);

```

Cette séquence met à votre disposition le port de jeu 2. Si vous voulez utiliser les deux ports de jeu, par exemple pour un jeu, vous devez ouvrir deux fois le device du port de jeu. Une fois avec Unit = 0L, et la seconde avec Unit = 1L. Si OpenDevice() a fonctionné correctement, vous devez définir le type de contrôleur (joystick ou souris) qui doit être interrogé au port de jeu. Voici les types de contrôleur :

GPCT_MOUSE	(1) Souris
GPCT_RELJOYSTICK	(2) Joystick relatif
GPCT_ABSJOYSTICK	(3) Joystick absolu

Si vous sélectionnez GPCT_MOUSE, ce sera la souris de l'Amiga qui sera interrogée au port de jeu. Notez que vous ne pouvez pas interroger la souris au gameport 1 par l'intermédiaire du device du port de jeu, car le device Input est impliqué, et que la souris considère le gameport 1 comme son "domaine privé". Vous pouvez cependant faire savoir au device Input qu'une manette de jeu doit être interrogée à la place d'une souris (cf. device Input).

Le port de jeu 1 se trouve dans les mains du device Input jusqu'à ce que vous ayez pris en charge l'ensemble du système, ou désactivé la tâche Input. Cette dernière opération n'est toutefois pas recommandée, car elle provoque facilement des difficultés dans l'ensemble du système. Si vous voulez utiliser malgré tout le gameport 1 sans renoncer à la tâche Input, vous pouvez interroger le port de jeu directement par l'intermédiaire du registre de hardware joy0dat (\$dff00a). Mais revenons au device du port de jeu.

A l'aide de la commande GP_SETCTYPE, vous définissez le contrôleur qui doit être interrogé au port de jeu indiqué avec OpenDevice(). Il suffit de transmettre à la routine suivante les flags GPCT_MOUSE, GPCT_RELJOYSTICK et GPCT_ABJOYSTICK :

```
/*********************************************
*                               GamePort_Set CType()      (Game_Support)*
*
* Fonction: Définir le type du contrôleur
*-----*
* Paramètres d'entrée:
*
* GamePortRequest:   GamePort-Bloc de device
* Type:             Controller-Type
*****************************************/
VOID GamePort_Set CType(GamePortRequest, Type)
struct IOStdReq      *GamePortRequest;
UBYTE                  Type;
{
    UBYTE GP_Type;
    GP_Type           - Type;
    GamePortRequest->io_Data   - (APTR) &GP_Type;
    GamePortRequest->io_Length - 1L;
    Do_Command (GamePortRequest, (UWORD)GPD_SETCTYPE);
}
```

On transmet au bloc du device de gameport l'adresse de la variable contenant le type à définir. Puisque ce type doit être défini comme variable UBYTE, il est logique que io_Length contienne la valeur 1L.

Etant donné que l'histoire du système d'exploitation de l'Amiga n'est pas encore terminée, ce que l'on peut discerner en considérant les différences versions du Kickstart, il pourrait arriver que vous essayiez d'utiliser un contrôleur nouveau venu (Kick 1.4 pourrait également soutenir un lightpen). Puisque ce nouveau contrôleur n'est pas soutenu par les versions de Kickstart, le programme s'effondrerait alors sans espoir, s'il on le lançait avec une version inférieure du Kickstart.

Pour empêcher cela, on sort dans io_Error du bloc de device du gameport, pour un contrôleur non soutenu, la valeur -1 pour GPDERR_SETCTYPE. Votre programme doit en conclure qu'il devra se retirer discrètement. Les contrôleurs Mouse, Reljoystick

et Absjoystick sont cependant soutenus par toutes les versions existantes du Kickstart, si bien qu'une interrogation Error n'est pas nécessaire lorsqu'on définit ces contrôleurs. Si vous désirez savoir quel contrôleur est soutenu actuellement au port de jeu, vous pouvez l'apprendre avec la routine suivante :

```
*****
*          GamePort_AskCType()      (Game_Support)*
*
* Fonction: Quels sont les contrôleurs soutenus?      *
*-----*
* Paramètres d'entrée:                                *
*
* GamePortRequest: GamePort-Bloc de device           *
* Type:          Adresse de l'octet, qui doit recevoir   *
*                 le type de contrôleur                  *
*****
VOID GamePort_AskCType (GamePortRequest, Type)
struct IOStdReq      *GamePortRequest;
UBYTE                *Type;
{
    GamePortRequest->io_Data  = (APTR) Type;
    GamePortRequest->io_Length = 1L;
    Do_Command(GamePortRequest, (UWORD)GPD_ASKCTYPE);
}
```

Il vous suffit d'indiquer ici, outre le bloc de device du gameport, l'adresse de l'octet dans lequel est reporté le type actuel du contrôleur. Si vous obtenez en retour la valeur 0 dans Type (GPCT_NOCONTROLLER), vous pouvez utiliser le port de jeu pour votre propre contrôleur (en ce moment, vous n'utilisez pas de contrôleur). Avec Type == -1 (GPCT_ALLOCATED), le port de jeu est occupé par un autre programme. Si vous obtenez en retour la valeur 1, 2 ou 3 dans Type, cela veut dire que votre programme utilise en ce moment le port de jeu, et que le contrôleur en question est soutenu par le device du port de jeu.

```
*****
*          GamePort_ReadEvent()     (Game_Support)*
*
* Fonction: Lire l'état du contrôleur               *
*-----*
* Paramètres d'entrée:                                *
*
* GamePortRequest: Bloc de device du gameport        *
* ReadEvent:      Adresse d'une structure Input-Event dans laquelle *
*                 est écrit l'état du contrôleur       *
*****
VOID GamePort_ReadEvent(GamePortRequest, ReadEvent)
struct IOStdReq      *GamePortRequest;
struct InputEvent      *ReadEvent;
{
    GamePortRequest->io_Data  = (APTR) ReadEvent;
    GamePortRequest->io_Length = (ULONG) (sizeof(struct InputEvent));
    GamePortRequest->io_Command = (UWORD) GPD_READEVENT;
    DoIO(GamePortRequest);
}
```

Le statut du contrôleur (avec la manette de jeu, par exemple le mouvement de la manette) est écrit dans une structure d'événement Input indiquée par vous-même. Dans ReadEvent.ie_position.ie_x ou dans ReadEvent.ie_position.ie_y, vous obtenez la position du contrôleur. Il faut cependant faire ici la distinction entre les types de contrôleurs :

GPCT_MOUSE

Si vous interrogez la souris ou le trackball, vous obtenez dans les deux variables citées le nombre des pas parcourus par la souris. Plus la souris se déplace rapidement, plus les valeurs sont grandes. Si vous déplacez la souris vers le haut, les valeurs de Y sont négatives. Pour un mouvement vers le bas, cette coordonnée est au contraire positive. Les mouvements vers la gauche sont annoncés par un signe négatif, et les mouvements vers la droite par un signe positif.

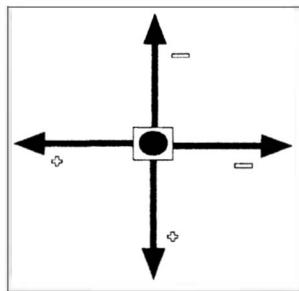


Figure 4 - 49 : Mouvement du joystick

Notez que les valeurs transmises n'indiquent pas la position du pointeur de la souris. Seul le mouvement est indiqué. Si vous voulez donc gérer un pointeur de souris qui vous est propre, il vous suffit d'ajouter les valeurs qui se trouvent dans ReadEvent.ie_X et ReadEvent.ie_Y pour obtenir la position actuelle du pointeur de la souris.

GPCT_RELJOYSTICK

Si vous avez connecté un joystick relatif au port de jeu, la position de la manette est restituée par l'intermédiaire des valeurs -1, 1 et 0. Ici, les valeurs positives et négatives correspondent aux mouvements indiqués dans la figure. La position moyenne du joystick dans la direction horizontale ou verticale est indiquée par l'intermédiaire d'un 0 dans chacune des variables.

GPCT_ABSJOYSTICK

La représentation des mouvements est la même pour le joystick relatif et pour le joystick absolu. La différence entre les deux tient seulement au fait que pour GPCT_RELJOYSTICK la position est fournie sans interruption, alors que ABSJOYSTICK ne fournit une annonce que lorsque le joystick se déplace dans une nouvelle position. Vous vous demandez sans doute ce que deviennent alors les boutons de la souris et du joystick, ou encore comment on peut les interroger.

Ici encore, on a besoin de la structure ReadEvent. Le champ ie_Code est rempli toutefois avec le statut des boutons de la souris et du bouton feu. Si cet élément a la valeur IEPCODE_LBUTTON (0x68), c'est qu'on a pressé le bouton feu, ou le bouton gauche s'il s'agit de la souris. Le bouton droit de la souris est interrogé par l'intermédiaire de IEPCODE_RBUTTON (0x69). Mais vous avez également la possibilité de demander l'annonce du fait que ces boutons ont été relâchés. Si vous désirez obtenir cette annonce, vous devez le faire savoir au device du port de jeu. On le fait par l'intermédiaire de SETTRIGGER :

```
*****
*          GamePort_SetTrigger()      (Game_Support)   *
*
* Fonction: Définir le seuil de mouvement           *
*-----*
* Paramètres d'entrée:                            *
*
* GamePortRequest: Bloc de device du gameport      *
* GPT:          Structure GamePortTrigger, pour indiquer   *
*               quand doit être envoyée l'annonce du mouvement,   *
*               et si la pression ou le relâchement d'une touche*   *
*               doivent être annoncés.                      *
*****
VOID iGamePort_SetTrigger;(GamePortRequest, GPT)
struct IOStdReq      *GamePortRequest;
struct GamePortTrigger      *GPT;
{
    GamePortRequest->io_Data  = (APTR) GPT;
    GamePortRequest->io_Length = (ULONG) (sizeof(struct
GamePortTrigger));
    Do_Command(GamePortRequest, (UWORD) GPD_SETTRIGGER);
}
```

Ne vous laissez pas induire en erreur par le commentaire sur la fonction de cette routine. En effet, elle a été conçue principalement pour régler la vitesse du pointeur de la souris. Nous aurons à y revenir. Mais nous allons nous intéresser ici au fait qu'un bouton est pressé ou non. Dans la structure GamePort à indiquer, vous pouvez en effet définir par l'intermédiaire de l'élément Keys si vous voulez être informé seulement sur le fait qu'un bouton est pressé (GPTF_DOWNKEYS = 0x01), seulement sur le fait qu'un bouton est relâché (GPTF_UPKEYS = 0x02), ou sur les deux (GPTF_UPKEYS+GPTF_DOWNKEYS = 0x03). Mais l'élément Keys n'est le seul de la structure GamePortTrigger :

Offset	Structure
-----	-----
0	struct GamePortTrigger
	{ /* défini dans "devices/gameport.h" */
2	UWORD gpt_Keys;
4	UWORD gpt_TimeOut;
6	UWORD gpt_XDelta;
8	UWORD gpt_YDelta;

Avec Timeout, vous pouvez indiquer au bout de combien de balayages verticaux (cinquantièmes de seconde) un ReadEvent doit être envoyé à partir du device de gameport, de façon à pouvoir être lu. XDelta et YDelta indiquent le nombre d'impulsions

du contrôleur au bout duquel un ReadEvent doit être déclenché. Ceci n'a toutefois de sens que lorsqu'on utilise une souris. Puisqu'il y a deux plaques trouées à l'intérieur de la souris, qui fonctionnent avec deux sources de lumière, on obtient une impulsion à chaque changement de lumière et d'ombre, et c'est celle-ci qui est interrogée par le device de gameport.

Ces impulsions sont additionnées. Lorsque la somme dépasse les valeurs indiquées dans XDelta et YDelta, la souris se déplace du nombre de points indiqués dans XDelta et YDelta. Si vous indiquez à chaque fois la valeur 1, vous obtenez le déplacement le plus rapide possible de la souris.

Pour le dire autrement : vous avez besoin de moins de place sur votre bureau pour déplacer le pointeur de la souris du coin supérieur gauche jusqu'au coin inférieur droit. Vous pouvez également régler la vitesse de déplacement avec le programme Preferences. La fonction qui se charge de conserver la vitesse indiquée à chaque processus de bootage est certes une routine de device Input, mais cette routine a recours à son tour à la commande du port de jeu.

Revenons un instant sur le programme Preferences : comment le programme connaît-il les valeurs qui ont été définies pour la vitesse de la souris ? Il existe évidemment ici encore une fonction qui permet de lire le statut actuel du port de jeu adressé pour le GamePortTrigger :

```
*****
*          GamePort_AskTrigger()      (Game_Support)*
*
* Fonction: A quelles conditions obéissent les contrôleurs
*-----*
* Paramètres d'entrée:
*
* GamePortRequest: GamePort-Bloc de device
* GPT:           Adresse de la structure GamePortTrigger
*             à remplir. (cf. SetTrigger)
*****
VOID iGamePort_AskTrigger (GamePortRequest, GPT)
struct IOStdReq      *GamePortRequest;
struct GamePortTrigger      *GPT;
{
    GamePortRequest->io_Data  = (APTR) GPT;
    GamePortRequest->io_Length = (ULONG) (sizeof(struct
GamePortTrigger));
    Do_Command(GamePortRequest, (UWORD) GPD_ASKTRIGGER);
}
```

Cette routine remplit la structure GameportPortTrigger indiquée par vos soins avec les valeurs Trigger du port de jeu adressé, que vous pouvez alors lire. Mais nous allons maintenant passer de la théorie, qui risque de paraître un peu abstraite, à la pratique. Voici un programme qui interroge la souris par l'intermédiaire du gameport 2. Les clics sur le bouton gauche de la souris sont annoncés; une pression sur le bouton droit de la souris met fin au programme.

Lorsque vous bougez la souris, les modifications de sa position sont affichées. Plus la souris se déplace vite, plus les valeurs affichées sont grandes :

```

/*****
*          GamePort.c
*          (c) Bruno Jennrich
*          Août 1988
*****/
/* Compile-Info:
*/
/* cc GamePort
* ln GamePort.o Game_Support.o Devs_Support.o -lc
*****/
#include "exec/Types.h"
#include "exec/memory.h"
#include "exec/io.h"
#include "exec/devices.h"
#include "devices/inputevent.h"
#include "devices/gameport.h"
#define GP_LEN (ULONG) (sizeof(struct IOStdReq))
struct IOStdReq      *GamePortRequest=OL;
struct GamePortTrigger GPT;
struct InputEvent     ReadEvent;
/****************************************
*          CloseIt()           (User)*
*
* Fonction: Libérer structures et mémoire en cas d'erreur
*-----
* Paramètres d'entrée:
*
* String: Error-Message
*****/
VOID CloseIt(String)
UBYTE    *String;
{
    UWORLD *dff180 = (UWORD *)0xdff180;
    UWORLD i;
    UWORLD Error;
    Error = 0;
    if (strlen(String) > 0)
    {
        for(i=0;i<0xffff;i++) *dff180 = i;
        Error = 10;
    }
    puts(String);
    if (GamePortRequest != OL) Close_A_Device(GamePortRequest);
    exit(Error);
}
/****************************************
*          The_GamePort_Device()       (User)*
*
* Fonction: utiliser le device Gameport
*****/
The_GamePort_Device()
{
    Open_A_Device("gameport.device", 11, &GamePortRequest, OL, GP_LEN);
    printf(" Connecter la souris au second gameport !\n");
    printf(" Bouton droit de la souris — Fin du programme \n");
    GamePort_SetCType(GamePortRequest, (UBYTE)GPCT_MOUSE);
    GPT.gpt_Keys = (UWORD)(GPTF_UPKEYS+GPTF_DOWNKEYS);
/* annoncer les deux */
}

```

```

GPT.gpt_Timeout = (UWORD) 0;
GPT.gpt_XDelta = (UWORD) 1;
GPT.gpt_YDelta = (UWORD) 1;
GamePort_SetTrigger(GamePortRequest, &GPT);
ReadEvent.ie_Code = 0;
while(ReadEvent.ie_Code != IE_CODE_RBUTTON)
{
    if (ReadEvent.ie_Code == IE_CODE_LBUTTON) printf("Left Button\n");
    if (ReadEvent.ie_Code == IE_CODE_LBUTTON+IE_CODE_UP_PREFIX)
        printf("Left Button released\n");
    GamePort_ReadEvent(GamePortRequest, &ReadEvent);
    printf("x: %d\n y: %d\n", ReadEvent.ie_X, ReadEvent.ie_Y);
    Do_Command(GamePortRequest, (UWORD)CMD_CLEAR);
}
GamePort_SetCType(GamePortRequest, (UBYTE)GPCT_NOCONTROLLER);
Close_A_Device(GamePortRequest);
}
main()
{
    The_GamePort_Device();
}

```

4.8. Le device Input

Nous venons de décrire le device de clavier et le device de port de jeu, et nous allons nous tourner maintenant vers le device d'input. Ce device est basé sur les deux précédents.

4.8.1. Les fonctionnalités du device d'input

Voici les commandes existantes pour le device d'input

IND_ADDHANDLER	(9)	Insérer Input-Handler propre
IND_REMHANDLER	(10)	Supprimer le handler propre
IND_WRITEEVENT	(11)	Envoyer Input-Event à tous les autres utilisateurs du device Input.
IND_SETTHRESH	(12)	Définir le seuil pour la fonction Repeat
IND_SETPERIOD	(13)	Définir la vitesse de répétition
IND_SETMPORT	(14)	Définir Mouse-Port
IND_SETMTYPE	(15)	Définir Mouse-Port-Controller
IND_SETMTRIG	(16)	Définir le seuil pour Mouse-Port-Trigger.

Ces commandes sont définies dans "devices/input.h". Outre les commandes propres au device, le device d'input soutient également les commandes standard suivantes :

CMD_RESET	(1)	Effectuer le reset du device Input sans désactiver le handler.
CMD_CLEAR	(5)	Effacer le tampon du device Input. Tous les Input-Events envoyés jusqu'ici et non encore traités par le handler sont avalés.
CMD_STOP	(6)	Arrêter le device Input.
CMD_START	(7)	Relancer le device Input.

4.8.2. Le premier pas : l'ouverture

Le device d'input doit - comme tout autre device - être ouvert avant de pouvoir être utilisé. Ici encore, vous utiliserez la fonction `Open_A_Device()` :

```
#define INPUT_LEN (ULONG) (sizeof(struct IOStdReq))
struct IOStdReq *InputRequest;
...
    Open_A_Device("input.device", 0L, &InputRequest, 0L, INPUT_LEN);
...
```

A la fin de son utilisation, le device doit évidemment être fermé. On a besoin pour cela comme d'habitude de la fonction `Close_A_Device()` :

```
Close_A_Device (InputRequest);
```

4.8.3. Utilisation du device d'input

Venons-en maintenant à l'utilisation du device d'input. Vous avez vu qu'il n'existe pas de commande READ pour le device d'input. Comment pouvons-nous dans ce cas obtenir par exemple les pressions sur les boutons à partir du device d'input, et les traiter en conséquence ? Le mot magique qui intervient ici est "Input-Handler". Un handler d'input est appelé par la tâche d'input, qui contrôle toutes les pressions sur les boutons et tous les mouvements de la souris. On transmet à ce handler une structure Input-Event qui a été générée par la tâche d'input. Une telle structure Input-Event a l'aspect suivant :

Offset:	Structure:
<hr/>	

	struct InputEvent
	{/* défini dans "devices/inputevent.h" */
0 0x00	struct InputEvent *ie_NextEvent;
4 0x04	UBYTE ie_Class;
5 0x05	UBYTE ie_SubClass;
6 0x06	UWORD ie_Code;
8 0x08	UWORD ie_Qualifier;
	union
	{
	struct
	{
10 0x0A	WORD ie_x;
12 0x0C	WORD ie_y;
	} ie_xy;
10 0x0A	APTR ie_addr;
	} ie_position;
14 0x0E	struct timeval ie_TimeStamp;
22 0x16	};

Examinons cette structure de plus près. La variable `ie_Class` contient le type d'événement contenu dans la structure Input-Event. Voici les classes Input-Event existantes :

IECLASS_Null	(0x00)	NOP Input-Event
IECLASS_RAWKEY	(0x01)	Code clavier
IECLASS_RAWMOUSE	(0x02)	Mouvement de la souris
IECLASS_EVENT	(0x03)	Événement interne
IECLASS_POINTERPOS	(0x04)	Position de la souris
!!!IECLASS	(0x05)	n'existe pas
IECLASS_TIMER	(0x06)	Timer-Event
IECLASS_GADGETDOWN	(0x07)	Un gadget a été cliqué
IECLASS_GADGETUP	(0x08)	Le gadget a été relâché
IECLASS_REQUEST	(0x09)	Requérir actuellement à l'écran
IECLASS_MENU_LIST	(0x0A)	Clic sur le menu
IECLASS_CLOSEWINDOW	(0x0B)	Clic sur le gadget Window Close
IECLASS_SIZEWINDOW	(0x0C)	Taille de la fenêtre modifiée
IECLASS_REFRESHWINDOW	(0x0D)	"rafraîchit" la fenêtre
IECLASS_NEWPREFS	(0x0E)	Nouvelles Préférences
IECLASS_DISKREMOVED	(0x0F)	Disque enlevé
IECLASS_DISKINSERTED	(0x10)	Disque introduit
IECLASS_ACTIVEWINDOW	(0x11)	Window activée
IECLASS_INACTIVEWINDOW	(0x12)	Window désactivée

Nous allons maintenant examiner une à une ces variables, pour voir comment il faut les interpréter selon ie_Class :

IECLASS_NULL

Cet événement Input est un événement Input NOP. Les données de l'événement n'ont aucune signification.

IECLASS_RAWKEY

La structure Input contient dans ie_Code le code de clavier (pas le code ASCII!) de la touche qui a été pressée. Dans ie_Qualifier, on trouve le statut des touches Ctrl, Shift et Alt, ainsi que d'autres touches spéciales :

IEQUALIFIER_LSHIFT	(0x0001)	Linke Shift-Taste
IEQUALIFIER_RSHIFT	(0x0002)	Touche Shift droite
IEQUALIFIER_CAPSLOCK	(0x0004)	Capslock allumé
IEQUALIFIER_CONTROL	(0x0008)	Touche Ctrl
IEQUALIFIER_LALT	(0x0010)	Touche Alt gauche
IEQUALIFIER_RALT	(0x0020)	Touche Alt droite
IEQUALIFIER_LCOMMAND	(0x0040)	Touche Amiga gauche
IEQUALIFIER_RCOMMAND	(0x0080)	Touche Amiga droite
IEQUALIFIER_NUMERICPAD	(0x0100)	Touche du bloc numérique
IEQUALIFIER_REPEAT	(0x0200)	Touche répétée
IEQUALIFIER_INTERRUPT	(0x0400)	???
IEQUALIFIER_MULTIBROADCAST	(0x0800)	???
IEQUALIFIER_LBUTTON	(0x1000)	Bouton gauche
IEQUALIFIER_RBUTTON	(0x2000)	Bouton droit
IEQUALIFIER_MBUTTON	(0x4000)	Bouton du milieu (n'existe pas!)
IEQUALIFIER_RELATIVEMOUSE	(0x8000)	Annonce d'une position relative de la souris

Chaque fois que le bit est posé, cela veut dire la touche Shift, Alt ou Ctrl a été pressée en même temps que la touche dans ie_Code. Pour que vous sachiez quelle est la

correspondance entre les touches et les codes de clavier, nous avons reproduit une image du clavier sur la page suivante.

Les numéros qui se trouvent sur les touches indiquent la valeur retournée dans ie_Code. Il faut noter qu'il y a deux événements Input envoyés par le device d'input chaque fois que l'on presse une touche. C'est d'abord le fait que la touche est pressée qui est annoncé, puis le fait qu'elle est relâchée. On peut faire la différence entre les deux événements Input grâce au fait que le bit supérieur (bit 7 = 0x80) au moment où l'on relâche la touche est posé (flag IDCMP: RAWKEY) dans le code de clavier proprement dit (ie_Code).

IECLASS RAWMOUSE

On obtient ici un mouvement de la souris dans `ie_x` et `ie_y`. Comme vous le savez déjà, ces deux variables font partie d'une union. Pour ne pas être obligé d'écrire à chaque Input-Event `ie_position.ie_x` pour obtenir le mouvement de la souris dans la direction X (ou Y), on a défini deux macros :

```
#define ie_X ie_position.ie_x;  
#define ie_Y ie_position.ie_y;
```

Il est maintenant possible d'accéder par exemple à la coordonnée en X de la modification du mouvement par l'intermédiaire de Input-Event.ie_X. Le flag IDCMP_DELTAMOVE doit être posé dans la fenêtre actuelle, pour que la tâche Input puisse envoyer cet Input_Event.

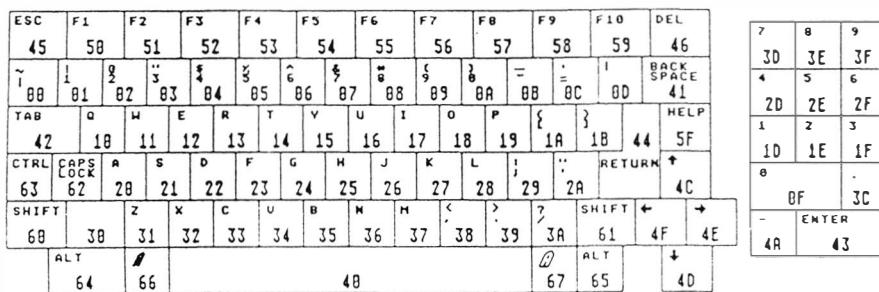


Figure 4 - 50 : Clavier ASCII Américain

Figure 4 - 51 : Clavier français AZERTY

IECLASS EVENT

Cet événement est un événement interne du device d'input. On en a besoin pour faire savoir que la fenêtre de saisie actuelle a été modifiée. ie_Code a la valeur IEPCODE NEWACTIVE (0x01).

IECLASS POINTERPOS

On trouve la position absolue de la souris dans ie_X et ie_Y. Le flag IDCMP_MOVEMOVE doit être posé dans la fenêtre actuelle.

IECALSS TIMER

C'est ici qu'est transmise dans timeval la nouvelle heure système. Cet événement est envoyé par la tâche Input toutes les cinquantièmes de seconde (flag IDCMP: INTUITICKS).

IECLASS GADGETDOWN

Lorsqu'il existe des gadgets dans la fenêtre de saisie actuelle, l'adresse du gadget sur lequel s'est opéré le clic est indiquée dans `ie_position.ie_addr`. Cet Input-Event n'est évidemment envoyé que si le flag `IDCMP_GADGETDOWN` a été posé dans la fenêtre. Les gadgets système comme `WINDOW_TO_FRONT` ou `WINDOW_TO_BACK` ne peuvent pas être interrogés de cette façon.

IECLASS_GADGETUP

Lorsqu'il existe des gadgets dans la fenêtre de saisie actuelle, l'adresse du gadget relâché est indiquée dans ie_position.ie_addr. Cet Input-Event n'est évidemment envoyé que si le flag IDCMP GADGETUP a été posé

IECLASS_REQUESTER

Lorsqu'un requester est affiché dans la fenêtre actuelle, c'est cet input-Event qui est envoyé. Dans ie_Code, on trouve IEPCODE_REQSET (0x01) pour le premier requester. Si d'autres requesters sont affichés sans que le premier disparaisse, cet événement n'est plus envoyé. Lorsque tous les requesters ont disparu, on obtient un autre Input-Event. Cette fois, c'est la valeur IEPCODE_REQCLEAR (0x00) qui est transmise dans ie_Code. Pour recevoir cet Input-Event, il faut que les flags IDCMP REQCLEAR et/ou REQSET aient été posés.

IECLASS_MENU LIST

Si un menu a été sélectionné dans la fenêtre actuelle, on peut savoir grâce à ie_Code quel est le menu sélectionné. Le tout ne fonctionne évidemment que si le flag IDCMP MENUPICK a été posé.

La sélection du menu fonctionne comme pour Intuition !

IECLASS_CLOSEWINDOW

Si on a cliqué sur le gadget Close de la fenêtre actuelle, on reçoit cet Input-Event. Cela ne fonctionne que si le flag IDCMP CLOSEWINDOW a été posé dans la fenêtre actuelle.

IECLASS_SIZEWINDOW

Si la taille de la fenêtre actuelle a été modifiée, c'est cet Input-Event qui est envoyé par la tâche d'input. Le flag IDCMP correspondant est NEWSIZE.

IECLASS_REFRESHWINDOW

Lorsque la fenêtre actuelle doit être effacée, c'est cet Input-Event qui est envoyé (flag IDCMP : REFRESHWINDOW).

IECLASS_DISKREMOVED

Si une disquette a été enlevée de l'un des lecteurs, c'est cet Input-Event qui est envoyé. La fenêtre actuelle doit avoir le flag IDCMP DISKREMOVED posé.

IECLASS_DISKINSERTED

Si une disquette est introduite dans l'un des lecteurs connectés, c'est l'Input-Event envoyé (flag IDCMP : DISKINSERTD).

IECLASS_ACTIVEWINDOW

On a cliqué sur une nouvelle fenêtre, qui devient la fenêtre de saisie actuelle.

IECLASS_INACTIVEWINDOW

La fenêtre de saisie actuelle a été désactivée.

Si Intuition vous est quelque peu familier, vous constaterez que beaucoup de fonctions d'entrée/sortie se déroulent par l'intermédiaire du device d'input, et n'ont plus ensuite qu'à être prolongées jusqu'à votre fenêtre Intuition. Mais nous allons maintenant voir comment sont reçus ces événements envoyés par la tâche d'input. Il faut pour cela, comme nous l'avons mentionné plus haut, qu'un handler d'input ait été installé. On y parvient avec la commande de device IND_ADDHANDLER :

```

ULONG User_Routine;
VOID Input_Code();
struct Interrupt Input_Handler;

/*********************************************
*                         Input_AddHandler()      (Input_Support) *
*
* Fonction : Intégrer une routine C propre dans le handler d'input*
*-----*
* Paramètres d'entrée:
*
* InputRequest: Bloc du device Input
* Handler:     Adresse de la routine Handler propre (C)
* Data:        Adresse sur le secteur des données personnelles
*             pour la routine handler
*****************************************/
VOID Input_AddHandler(InputRequest, Handler, Data)
struct IOStdReq    *InputRequest;
VOID                *Handler;
APTR                Data;
{
    User_Routine           = (ULONG) Handler;
    Input_Handler.is_Data = Data; /* cf. a1 */
    Input_Handler.is_Code = (VOID (*)())Input_Code;
    Input_Handler.is_Node.In_Pri = 51;
    InputRequest->io_Data = (APTR)&Input_Handler;
    Do_Command(InputRequest, (UWORD) IND_ADDHANDLER);
}

```

Comme à toutes les routines soutenues par le device, on transmet à celle-ci aussi le bloc de device avec lequel le device a été ouvert (ou une copie du bloc de device original). On transmet ensuite l'adresse de la routine de handler, écrite en C. Vous avez également la possibilité d'indiquer un domaine de données, qui se tient à la disposition de votre routine de handler (par exemple pour les variables, etc.).

Malheureusement, la tâche d'input n'est pas en mesure d'appeler des routines écrites en C, car dans celles-ci les paramètres sont transmis par l'intermédiaire de la pile. La tâche d'input transmet les paramètres (Input-Event et domaine de données) dans les registres de hardware. C'est pourquoi nous devons mettre en place une interface qui écrit les paramètres sur la pile en les prenant dans le registre du hardware, et appelle ensuite le programme en C. Nous avons développé dans ce but une petite routine en langage machine :

```

*****
*           _Input_Code.asm      (Input_Support) *
*
* Fonction: Appeler la routine handler d'Input (Aztec C)
*
* -----
* Paramètres d'entrée:
*
* A0: pointeur sur l'événement Input
* A1: pointeur sur les données personnelles
*
* -----
* Valeur retournée:
*
* D0: contient InputEvent à traiter ultérieurement
* Celui-ci est rendu disponible par la routine en C
*****
#asm
    public _geta4
    public _Input_Code
_Input_Code:
    move.l  a4, -(sp)
    jsr     _geta4
    movem.l a0/a1, -(sp)      ; paramètre sur la pile
    move.l  _User_Routine, a0  ; appeler la routine en C
    jsr     (a0)
    movem.l (sp)+, a0/a1
    move.l  (sp)+, a4
    rts
#endifasm

```

Puisque le compilateur Aztec s'assure de l'accès aux variables du programme par l'intermédiaire du registre de hardware A4, ce registre doit être réinitialisé avant chaque appel de la routine de handler en C. On le fait avec la routine geta4, qui transporte l'adresse de base des variables dans A4. Cette initialisation du registre A4 est nécessaire car le registre A4 contient une autre valeur pendant le traitement du handler d'input. Il faut pour cette raison rétablir l'ancienne valeur de ce registre lorsqu'on quitte le handler. Après initialisation du registre A4, on peut alors porter les paramètres sur la pile, et appeler la routine en C.

Dans `Input_AddHandler()`, cette routine en langage machine est fournie comme `InputHandler` véritable. On crée pour cela une structure d'interrupt, dont le champ `is_Code` est chargé avec l'adresse de la routine en langage machine, et le champ `is_Data` est chargé avec l'adresse du domaine des données. Nous définissons une priorité plus haute pour notre handler que pour le handler d'input système, pour recevoir en premier tous les événements d'input générés par la tâche d'input. La priorité du handler système a la valeur 50, et celle de notre handler la valeur 51.

La structure `Interrupt` initialisée est alors transmise à notre bloc de device d'input (`io_Data`), et la commande `IND_ADDHANDLER` est envoyée. Notez que vous n'avez pas le droit d'utiliser des variables de noms `Input_Handler` et `User_Routine`, si vous voulez utiliser les routines ci-dessus. `Input-Handler` est la structure `Interrupt` par l'intermédiaire de laquelle le segment en langage machine `Input_Code` est appelé. Quant à `User_routine`, il contient l'adresse de notre routine en C à appeler, destinée à traiter les événements. Si vous voulez quitter le programme qui a installé un handler d'input,

vous devez évidemment supprimer à nouveau cet handler d'input. Voici comment on y parvient :

```
*****
*           Input_RemHandler()          (Input_Support)*
*
* Fonction: Désactiver le handler d'Input
*-----
* Paramètres d'entrée:
*
* InputRequest: Bloc du device d'Input
*****/
```

```
VOID Input_RemHandler(InputRequest)
struct IOStdReq      *InputRequest;
{
    InputRequest->io_Data     = (APTR) &Input_Handler;
    Do_Command(InputRequest, (UWORD)IND_REMHANDLER);
}
```

A l'aide de ces deux routines, il est possible par exemple d'écrire un enregistreur de macros, destiné à enregistrer tous les événements d'input qui se présentent :

```
*****
*           Recorder.c               *
*           Août 1988                *
*           (c) Bruno Jennrich       *
*****/
```

```
*****
*   Compile-Info:                  *
*                               *
*   cc Recorder.c                 *
*   ln Recorder.o Input_Support.o Devs_Support.o -lc
*****/
```

```
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/nodes.h"
#include "devices/input.h"
#include "devices/inputevent.h"
VOID *AllocMem();
VOID *Open();
ULONG User_Routine;
#define MODE_NEWFILE    1006L
#define INPUT_LEN        (ULONG) (sizeof(struct IOStdReq))
#define RECORD_SIZE     5000L
#define INEV_LEN         ((ULONG) (sizeof(struct InputEvent)))
#define MEMTYPE          (MEMF_CHIP | MEMF_CLEAR)
ULONG HowMuchEvents;
ULONG ActualEvents;
BOOL End;
UWORD *FileHandle = 0L;
struct InputEvent *Recorder = 0L;
struct IOStdReq  *InputRequest = 0L;
struct InputEvent *Pointeur;
```

```
*****
```

```

*
*                               CloseIt()          (User)      *
*                               *                      *
* Fonction : Libérer toutes les structures réservées, la mémoire, etc.*-----*
*-----*                               *
* Paramètres d'entrée:          *                      *
*                               *                      *
* String: Error-Message        *                      *
******/
```

CloseIt(String)
 BYTE *String;
{
 WORD *dff180 = (WORD *)0xdff180;
 WORD i;
 WORD Error;
 Error = 0;
 if (strlen(String) > 0)
 {
 for(i=0;i<0xffff;i++) *dff180 = i;
 Error = 10;
 }
 puts(String);
 if (FileHandle != 0L) Close(FileHandle);
 if (InputRequest != 0L) Close_A_Device(InputRequest);
 if (Recorder != 0L) FreeMem(Recorder, HowMuchEvents*INEV_LEN);
 exit(Error);
}

******/

```

*                               C_Handler()          (User)*
*                               *                      *
* Fonction: Traiter InputEvents de la tâche Input           *
*-----*                               *
* Paramètres d'entrée:          *                      *
*                               *                      *
* Input: InputEvent            *                      *
* Data: pointeur sur les données personnelles             *-----*
*-----*                               *
* Valeur renvoyée:          *                      *
*                               *                      *
* InputEvent devant être traitée ultérieurement.           *-----*
******/
```

struct InputEvent *C_Handler(Input, Data)
 struct InputEvent *Input;
 ULONG , *Data;
{
 if (!End)
 if (Input->ie_Class == IECLASS_RAWKEY)
 {
 if (Input->ie_Code == 0x45) End = TRUE; /* Escape */
 else
 if (*Data < HowMuchEvents)
 {
 Pointeur = &Recorder[*Data]; /* InputEvent */
 Pointeur->ie_NextEvent = 0;
 Pointeur->ie_Class = Input->ie_Class;
 Pointeur->ie_SubClass = Input->ie_SubClass;

```

        Pointeur->ie_Code      = Input->ie_Code;
        Pointeur->ie_Qualifier = Input->ie_Qualifier;
        Pointeur->ie_TimeStamp.tv_secs = Input->ie_TimeStamp.tv_secs;
        Pointeur->ie_TimeStamp.tv_micro = Input->ie_TimeStamp.tv_micro;
        *Data+=1;
    }
    else End = TRUE;
}
return(Input);
}

/*********************************************
*                         (User)*
*
* Fonction: InputDevice verwenden
*****************************************/
Input_Device()
{
    Open_A_Device("input.device", 0L, &InputRequest, 0L, INPUT_LEN);
    End = FALSE;
    ActualEvents = 0;
    Input_AddHandler(InputRequest, C_Handler, &ActualEvents);
    while(!End); /* Attendre la fin de l'enregistrement*/
    Input_RemHandler(InputRequest);
    Close_A_Device(InputRequest);
}

/*********************************************
*                         (User)*
*
* Paramètres d'entrée:
* argv[1]: Nom du fichier des macros
* argv[2]: Nombre d'Events à enregistrer
*****************************************/
main (argc, argv)
UWORD argc;
BYTE    **argv;
{
    if (argc != 3)
    {
        printf("USAGE: %s Fichier de macros HowMuchEvents\n", argv[0]);
        CloseIt("");
    }
    if ((HowMuchEvents = atoi(argv[2])) < 0)
        /* Combien d'Events à enregistrer?*/
        CloseIt("HowMuchEvents < 0 !!!");
    Recorder = (struct InputEvent *) AllocMem(HowMuchEvents*INEV_LEN,
MEMTYPE); /* prendre mémoire pour Events */
    if (Recorder == 0L)
        CloseIt("No Memory for InputEvents !!!");
    Input_Device();
    FileHandle = Open(argv[1], MODE_NEWFILE);
    if (FileHandle == 0L)
        CloseIt("Cannot Open 'Fichier macros' !!!");
    Write(FileHandle, &ActualEvents, 4L);
    Write(FileHandle, Recorder, ActualEvents*INEV_LEN);
    Close(FileHandle);
}

```

```
    FreeMem(Recorder, HowMuchEvents*INEV_LEN);  
}
```

Ce programme ouvre d'abord le device d'input. Puis il installe le handler d'input qui appelle notre routine de handler en C. Dans cette routine de handler en C sont alors enregistrés les événements RAWKEYS. L'événement d'input reçu, qu'il ait été enregistré ou non, est envoyé aux autres handlers d'input (tous les handlers d'input sont organisés dans une liste, comme les serveurs d'interrupt). On obtient ce résultat en retournant simplement l'événement d'input reçu. L'instruction return() écrit pour cela l'adresse de l'événement d'input reçu dans D0. Si vous voulez transmettre la valeur 0 dans D0, vous devez veillez à ne pas sauter aux handler qui viennent ensuite !

D'ailleurs, vous devez aussi noter que les événements Input peuvent être liés entre eux. On emploie pour cela le champ ie_NextEvent de chaque structure Input-Event (ce pointeur pointe sur le successeur de l'événement d'input). Il peut parfaitement se produire que vous receviez uniquement le premier Input-Event d'une longue liste dans votre handler d'input. Vous avez alors le loisir de modifier cette liste, par exemple en ajoutant d'autres événements. Mais dans ce cas, vous devez gérer vous-même la mémoire pour les événements d'input que vous avez ajoutés. L'émetteur des Input-Events est toujours responsable de ces derniers !

Revenons maintenant au programme ci-dessus. Nous y avons enregistré uniquement des événements RAWKEYS. Vous pouvez ainsi enregistrer toutes les touches actionnées pendant que le programme est en cours (dans n'importe quelle fenêtre, en tout cas tant qu'il n'existe pas de handler d'input dont la priorité est plus élevée). Pour cela, vous devez indiquer le nombre maximal d'événements à enregistrer. Un appel du programme aura par exemple l'aspect suivant :

"Recorder Macro 100".

Cet appel du programme vous permet d'enregistrer jusqu'à 50 pressions sur des touches, et de les conserver dans la macro File. Vous ne pouvez pas en enregistrer plus, car une pression sur une touche se décompose en deux : le moment où vous appuyez, et le moment où vous la relâchez, il faut envoyer un Input-Event pour chacun. Mais vous pouvez par exemple tester dans ie_Code le bit7, et ne pas enregistrer l'événement correspondant lorsqu'il est posé.

Si jamais vous voulez mettre fin à l'enregistrement plus tôt que prévu, il vous suffit de presser sur Escape. Tous les événements enregistrés jusque-là sont conservés, et on quitte le programme. Evidemment, un enregistreur de macros ne sert à rien, si l'on ne peut faire fonctionner la macro enregistrée. On a besoin pour cela d'une fonction qui envoie les Input-Events enregistrés. A l'aide de cette fonction, nous faisons donc "jouer" la tâche d'input. Nous envoyons des événements d'input à tous les handlers d'input, dont il existe plusieurs exemplaires dans le système, par exemple au handler système, et à nos handlers d'input propres, installés avec Input_AddHandler. Cette fonction s'appelle WRITEEVENT :

```
*****
*           Input_WriteEvent()          (Input_Support)*
*
* Fonction: Transmettre InputEvent à d'autres handlers d'Input *
*-----*
* Paramètres d'entrée:                                     *
*   *
* InputRequest: Bloc de device d'Input                      *
* Event:          InputEvent à reconduire                   *
*****
```

VOID Input_WriteEvent(InputRequest, Event)
 struct IOStdReq *InputRequest;
 struct InputEvent *Event;
{
 InputRequest->io_Data = (APTR) Event;
 InputRequest->io_Length = INEV_LEN;
 InputRequest->io_Flags = (UBYTE)0;
 Do_Command(InputRequest, (UWORD)IND_WRITEEVENT);
}

Grâce à cette routine, vous êtes en mesure d'envoyer un Input-Event à tous les handlers d'input existants. Si vous avez installé un handler personnel, celui-ci reçoit aussi votre Input-Event, à condition qu'un handler de priorité plus élevée ne modifie par les événements d'input, ou ne les "avale" pas.

Avec cette routine, il est très facile de programmer l'inverse de ce que fait l'enregistreur ci-dessus. Notre programme doit seulement ouvrir le fichier généré par l'enregistreur, lire le nombre d'Input-Events enregistrés, réservé de la mémoire en conséquence, et charger les événements d'input dans cette mémoire réservée. Il ne reste plus alors qu'à lancer ces événements d'input avec Input_WriteEvent() :

```
*****
*           Play.c                         *
*           Août 1988                      *
*           (c) Bruno Jennrich             *
*****
```

* Compile-Info:
* *
* cc Play
* ln Play.o Input_Support.o Devs_Support.o -lc

```
*****
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/interrupts.h"
#include "exec/nodes.h"
#include "devices/input.h"
#include "devices/inputevent.h"
VOID *AllocMem();
VOID *Open();
#define MODE_OLDFILE    1005L
#define INPUT_LEN        (ULONG) (sizeof(struct IOStdReq))
#define RECORD_SIZE      5000L
#define INEV_LEN         ((ULONG) (sizeof(struct InputEvent)))
#define MEMTYPE          (MEMF_CHIP | MEMF_CLEAR)
DWORD      *FileHandle = 0L;
struct InputEvent *Player = 0L; /* mémoire pour Input-Events */
```

```

ULONG Length = 0L; /* Nombre d'Input-Events */
struct IOStdReq *InputRequest = 0L;
/*********************************************
* CloseIt() (User)*****
*
* Fonction: libérer mémoire et structures en cas d'erreur
* -----
* Paramètres d'entrée:
*
* String: Error-String
*****************************************/
VOID CloseIt(String)
BYTE *String;
{
    UWORLD *dff180 = (UWORD *)0xdff180;
    UWORLD i;
    UWORLD Error;
    Error = 0;
    if (strlen(String) > 0)
    {
        for(i=0;i<0xffff;i++) *dff180 = i;
        Error = 10;
    }
    puts(String);
    if (FileHandle != 0L) Close(FileHandle);
    if (Player != 0L) FreeMem(Player, Length*INEV_LEN);
    if (InputRequest != 0L) Close_A_Device(InputRequest);
    exit(Error);
}
/*********************************************
* main (User)*****
*
* Paramètres d'entrée:
* argv[1]: Fichier macros
*****************************************/
main (argc, argv)
UWORD argc;
BYTE **argv;
{
    UWORLD i;
    if (argc != 2)
    {
        printf("USAGE: %s 'Fichier macro'\n", argv[0]);
        CloseIt("");
    }
    FileHandle = Open(argv[1], MODE_OLDFILE);
    if (FileHandle == 0L) CloseIt("File not exists !!!");
    if (Read(FileHandle, &Length, 4L) != 4L) /* Le nombre des événements*/
        CloseIt("Read Error #1 !!!"); /* enregistrés n'a pas pu
être lu*/
    Player = AllocMem(Length*INEV_LEN, MEMTYPE);
    if (Player == 0L) CloseIt("No Memory !!!");
        /* réservrer mémoire pour Length-Events
*/
    if (Read(FileHandle, Player, Length*INEV_LEN) != Length*INEV_LEN)
        CloseIt("Read-Error #2 !!!");
    Close(FileHandle);
    Open_A_Device("input.device", 0L, &InputRequest, 0L, INPUT_LEN);
    for(i=0;i<Length;i++) /* Vous ne vous êtes pas beaucoup

```

```
fatigué! */
    Input_WriteEvent(InputRequest, &Player[i]);
    Close_A_Device(InputRequest);
    FreeMem(Player, Length*INEV_LEN);
}
```

Dans ce programme, on voit bien combien il est facile d'ouvrir un device et de l'utiliser. La partie du programme qui envoie les événements d'input ne contient que trois lignes et une instruction de contrôle (for(;;);).

Après ce sujet réellement complexe des handlers d'input, nous allons envisager les autres commandes du device d'input. Celles-ci s'occupent uniquement de la souris et du clavier. Pour bien comprendre, vous devriez avoir lu la section précédente sur le device de gameport, car nous allons recourir ici aux connaissances acquises dans cette section.

4.8.4. Le device d'input, souris et clavier

Si vous êtes déjà occupé du réglage dans Preferences et chaque possesseur d'un Amiga l'a fait une fois dans sa vie, vous trouverez dans la présente section quelques analogies entre les possibilités du programme Preferences et celles du device d'input.

Régler la vitesse de répétition

Le programme Preferences offre d'abord la possibilité de modifier à l'aide de barres de défilement la vitesse de répétition, c'est-à-dire la vitesse à laquelle un caractère est répété lorsqu'on maintient une touche pressée. Pour cela, la valeur Repeat définie est lue dans le fichier devs/system-configuration au moment où le système est booté, et déterminé au moyen de IND_SETTHRESH. Si vous désirez modifier la vitesse de répétition après que le système a été booté, vous devez d'abord charger le programme Preferences, effectuer la modification, et booter à nouveau la disquette. La routine suivante (avec le device d'input ouvert) permet cependant de modifier directement la vitesse de répétition :

```
/*
 *          Input_SetPeriod()      (Input_Support)*
 *
 * Fonction: Définir la vitesse de répétition
 *-----
 * Paramètres d'entrée:
 *
 * InputRequest: Input-Device-Bloc
 * Secs, Micro: Durée entre deux répétitions
 */
VOID Input_SetPeriod(InputRequest, Secs, Micro)
struct timerequest *InputRequest;
ULONG                         Secs, Micro;
{
```

```

InputRequest->tr_time.tv_secs = Secs;
InputRequest->tr_time.tv_micro = Micro;
Do_Command(InputRequest, (UWORD)IND_SETPERIOD);
}

```

Il suffit que vous indiquez à cette routine le laps de temps qui doit s'écouler entre deux répétitions d'une touche pressée.

Modifier le seuil de répétition

Vous avez sans doute déjà constaté qu'il se passe d'abord un certain temps, avant qu'une touche pressée ne soit répétée. Si une touche était répétée dès qu'on appuie dessus, il serait impossible de saisir les caractères un par un. Le temps qui s'écoule entre la pression sur la touche et le début de la répétition s'appelle le seuil de répétition ou "threshold" en anglais. Avec la routine suivante, vous pouvez modifier ce seuil à votre guise :

```

*****
*           Input_SetThresh()      (Input_Support)*
*
* Fonction: Définir le seuil de répétition
* -----
* Paramètres d'entrée:
*
* InputRequest: Input-Device-Block
* Secs, Micro: Après combien de secondes et de micro-secondes
*               la touche doit-elle être répétée?
*****
VOID Input_SetThresh(InputRequest, Secs, Micro)
struct timerequest *InputRequest; /* !!! */
ULONG                         Secs, Micro;
{
    InputRequest->tr_time.tv_secs = Secs;
    InputRequest->tr_time.tv_micro = Micro;
    Do_Command(InputRequest, (UWORD)IND_SETTHRESH);
}

```

Essayez donc de travailler une fois avec un seuil égal à 0 seconde et 0 micro-seconde. Vous verrez que le simple fait d'effleurer une touche conduit à une série de répétitions sans fin.

Définir la vitesse de la souris

Passons maintenant à la souris. On ne peut atteindre ici qu'une seule des fonctions du device d'input par l'intermédiaire de Preferences :

```

*****
*           Input_SetMTrig()      (Input_Support)*
*
* Fonction: Définir le seuil pour l'annonce
*           des mouvements de la souris
* -----
* Paramètres d'entrée:
*
* InputRequest: Input-Device-Block

```

```

* Keys:           Annoncer pression ou relâchement du bouton?   *
* Timeout:       Envoyer MousReport après combien de           *
*                retours verticaux?                                *
* XDelta, YDelta: Annoncer le mouvement de la souris            *
*                à quelle valeur du compteur?                      *
******/  

VOID Input_SetMTrig(InputRequest, Keys, Timeout, XDelta, YDelta)
struct IOStdReq    *InputRequest;
ULONG                 Keys, Timeout, XDelta, YDelta;
{
    struct GamePortTrigger GPT;
    GPT.gpt_Keys      - Keys;
    GPT.gpt_Timeout   - Timeout;
    GPT.gpt_XDelta    - XDelta;
    GPT.gpt_YDelta    - YDelta;
    InputRequest->io_Data      - (APTR) &GPT;
    InputRequest->io_Length     - (ULONG) (sizeof(struct
GamePortTrigger));
    Do_Command(InputRequest, (UWORD)IND_SETMTRIG);
}

```

Avec cette routine, il est possible de faire savoir au device d'input le nombre d'impulsions reçues par les plaques trouées de la souris à partir duquel un mouvement doit être annoncé dans la direction X ou Y. Comme vous le savez, il existe dans une souris deux plaques perpendiculaires l'une par rapport à l'autre, qui passent à travers deux faisceaux de lumière. Lorsque la souris se déplace, les plaques tournent, et elles interrompent les faisceaux de lumière. Ces impulsions sont envoyées au device d'input, ou encore le device d'input les lit dans les registres du hardware.

Lorsque le nombre d'impulsions atteint la valeur indiquée, le device d'input se charge de déplacer le pointeur de la souris d'un point à l'écran. XDelta et YDelta indiquent le nombre d'impulsions dans les directions X et Y, nécessaires pour déplacer le pointeur d'un point à l'écran. Plus ces valeurs sont grandes, plus la souris est lente.

Avec le paramètre Timeout, vous pouvez définir le nombre de balayages verticaux au bout duquel un événement de la souris ou un MouseEvent doit être envoyé, dans le cas où la souris n'a pas été suffisamment déplacée. Pour Intuition, ce paramètre a la valeur 1. Dans la fenêtre Intuition (Intuition Screen), la position de la souris est modifiée après chaque balayage vertical. Avec le paramètre Keys, vous avez enfin la possibilité de demander l'annonce du fait que les boutons sont relâchés (GPTF_UPKEYS), ou pressés (GPTF_DOWNKEYS).

Indiquer le port de la souris

Vous aurez de la peine à le croire, mais cette commande permet de connecter la souris au second port de jeu, après quoi il est possible de l'utiliser comme à l'accoutumée :

```

*****  

*                               Input_SetMPort()          (Input_Support)*  

*  

* Fonction: Définir le port de la souris  

*-----*  

* Paramètres d'entrée:  

*
```

```

*
* InputRequest: Input-Device-Block
* Port:          0: GamePort 1
*                 1: GamePort 2
*****
VOID Input_SetMPort:(InputRequest, Port)
struct IOStdReq    *InputRequest;
UBYTE                  Port;
{
    UBYTE PointerToPort;
    PointerToPort = Port;
    InputRequest->io_Data      = (APTR) &PointerToPort;
    InputRequest->io_Length    = (ULONG) 1L;
    Do_Command(InputRequest, (UWORD)IND_SETMPORT);
}

```

Il suffit pour cela de transmettre à cette routine, comme d'ailleurs aux précédentes, le bloc du device d'input, et un 0 ou un 1. D'après la valeur indiquée, la souris sera interrogée au port de jeu 1 ou 2. Notez bien qu'avec cette commande de device l'adresse doit être transmise à la valeur 1 ou 0.

Joystick comme souris

Si vous en avez envie, vous pouvez non seulement modifier le port de la souris, mais aussi le contrôleur de celle-ci. Cela veut dire qu'un joystick pourra par exemple se charger des fonctions de la souris (sauf celles qui sont assumées par le bouton droit de la souris). Vous avez uniquement besoin pour cela d'appeler la routine qui suit :

```

*****
*                         Input_SetMType()      (Input_Support)*
*
* Fonction: Définir le type du contrôleur de souris
*-----
* Paramètres d'entrée:
*
* InputRequest: Input-Device-Block
* Type:        Nouveau type du contrôleur
*****
VOID Input_SetMType:(InputRequest, Type)
struct IOStdReq    *InputRequest;
UBYTE                  Type;
{
    UBYTE MouseType;
    MouseType = Type;
    InputRequest->io_Data      = (APTR) &MouseType;
    InputRequest->io_Length    = 1L;
    Do_Command(InputRequest, (UWORD)IND_SETMTYPE);
}

```

Dans Type sera alors indiqué le nouveau type du contrôleur :

GPCT_MOUSE	Souris connectée au port
GPCT_RELJOYSTICK	Joystick relatif connecté au port
GPCT_ABSJOYSTICK	Joystick absolu connecté au port
GPCT_NOCONTROLLER	Plus rien de connecté au port

Notez bien qu'il faut ici encore transmettre l'adresse sur le type. C'est pourquoi le paramètre transmis sera écrit dans une variable créée spécialement à cet effet, l'adresse de cette variable étant transmise elle-même au bloc de device.

4.9. Le device de console

On utilise le device de console pour pouvoir sortir des textes à l'écran sans difficulté. Il faut pour cela qu'une fenêtre soit ouverte, à partir de laquelle le device de console pourra lire vos saisies, et dans laquelle le même device de console effectuera les sorties. La fenêtre ouverte est transmise avant ouverture du device de console au bloc de device correspondant :

```
struct IOstReq *ConsoleRead = 0L;
#define CON_LEN ( ULONG ) ( sizeof( struct IOStdReq ) )
...
Window = OpenWindow( &NewWindow );
ConsoleRead = ( struct IOstdReq * ) GetDeviceBlock( CON_LEN );
ConsoleRead->io_Data = ( APTR ) Window;
ConsoleRead->io_Length = ( ULONG ) ( sizeof( struct Window ) );
Open_A_Device( "console.device", 0L, &ConsoleRead, 0L, 0L );
```

Vous pouvez alors travailler avec le device de console et les commandes suivantes :

CMD_READ	(2)	Lire les pressions des touches
CMD_WRITE	(3)	Afficher les textes
CMD_CLEAR	(5)	Supprimer le Console-Buffer
CD_ASKKEYMAP	(9)	Obtenir la définition actuelle du clavier
CD_SETKEYMAP	(10)	Redéfinir les touches du clavier
CD_ASKDEFAULTKEYMAP	(11)	Obtenir définition clavier par défaut
CD_SETDEFAULTKEYMAP	(12)	Définir clavier par défaut

Comme vous pouvez le constater, le device de console met très peu de commandes à votre disposition; il n'en reste pas moins que vous avez entre les mains de cette manière un outil très puissant, permettant de développer des applications complexes comme les éditeurs (cf. DiskEd), des traitements de texte, etc. Cela provient du fait que le device de console est contrôlé par l'intermédiaire de chaînes de contrôle. Ces chaînes sont envoyées à l'aide de la commande CMD_WRITE au device de console. Vous devez créer un nouveau bloc de device pour la commande CMD_WRITE, afin que les commandes READ et WRITE n'entrent pas en conflit :

```
struct IOStdReq *ConsoleWrite = 0L;
...
ConsoleWrite = ( struct IOStdReq* ) GetDeviceBlock( CON_LEN );
Console_Copy( ConsoleRead, ConsoleCopy );
...
```

Pour pouvoir utiliser également le second bloc de device, il faut copier quelques éléments de structure :

```
*****
*           Console_Copy()          (Con_Support)*
*
* Fonction: Copier le bloc de device
*-----
* Paramètres d'entrée:
*
* OldStdReq: Original
* NewStdReq: Copie
*****
VOID Console_Copy(OldStdReq, NewStdReq)
struct IOStdReq *OldStdReq,*NewStdReq;
{
    NewStdReq->io_Device =
        OldStdReq->io_Device;
    NewStdReq->io_Unit =
        OldStdReq->io_Unit;
}
```

Vous pouvez maintenant lire les pressions sur les touches et les afficher à l'écran à l'aide de *Console_Read()* et *Console_Write()*

```
*****
*           Console_Write()         (Con_Support)*
*
* Fonction: Sortir la chaîne sur le device de console
*-----
* Paramètres d'entrée:
*
* ConWrite: Device-Block
* String: Chaîne à afficher
* Len: Longueur de la chaîne
*****
VOID Console_Write;(ConWrite, String, Len)
struct IOStdReq *ConWrite;
BYTE             *String;
ULONG            Len;
{
    ConWrite->io_Data = (APTR) String;
    ConWrite->io_Length = Len;
    Do_Command(ConWrite, (UWORD)CMD_WRITE);
}
*****
*           Console_Read()         (Con_Support)*
*
* Fonction: Lire la chaîne à partir du device de console
*-----
* Paramètres d'entrée:
*
* ConWrite: Device-Block
* String: Adresse du tampon
* Len: Nombre de caractères à lire
*****
VOID Console_Read(ConRead, String, Len)
struct IOStdReq *ConRead;
```

```
    BYTE          *String;
    ULONG         Len;
{
    ConRead->io_Data     = (APTR) String;
    ConRead->io_Length   = Len;
    ConRead->io_Command  = (UWORD) CMD_READ;
    DoIO(ConRead);
}
```

Vous transmettez à ces routines le bloc de device et l'adresse de la chaîne à afficher, ou l'adresse du domaine de données, dans lequel les pressions de touches lues devront être écrites.

Indiquez également le nombre des caractères à afficher ou à lire. Si vous indiquez -11 comme nombre de caractères à afficher, tous les caractères de la chaîne devant se terminer par un octet zéro seront affichés.

Si vous voulez lire au moyen de `Console_Read()` des pressions de touches au clavier, vous obtenez le code ASCII représenté par la touche pressée. Notez que ce code n'est pas affiché. Vous devez vous occuper vous-même de l'affichage au moyen de `Console_Write()` ou `CMD_WRITE`. Comme nous l'avons déjà mentionné, il existe une foule de chaînes de contrôle, avec lesquelles vous pouvez déterminer la forme de l'entrée et de la sortie :

BELL (0x7)

Clignotement de l'écran.

BACKSPACE (0x08)

Le curseur est déplacé d'une position vers la gauche. Le caractère qui se trouve à gauche du curseur n'est pas effacé! Vous pouvez l'effacer cependant en affichant un espace, et en déplaçant à nouveau le curseur d'une position.

LINEFEED (0x0a)

Le curseur est amené une position vers le bas. S'il se trouve dans la ligne la plus basse, l'écran défile d'une ligne vers le haut (cf. SET MODE).

VERTCAL TAB (0x0b)

Le curseur est déplacé d'une ligne vers le haut. S'il se trouve dans la ligne la plus haute, le reste de l'écran défile d'une ligne le bas.

FORM FEED (0x0c)

Efface la fenêtre console.

CR (0x0d)

Le curseur est déplacé dans la première colonne, mais pas dans la ligne suivante.

SHIFT IN (0x0e)

Active la touche Shift.

SHIFT OUT (0x0f)

Désactive la touche Shift. A la sortie, on n'obtient que des majuscules.

Touche CAPSLOCK

Cf. La définition des touches du clavier

ESC (0x1b)

Touche Escape

CSI (0x9b)

Control Sequence Introducer. Toutes les chaînes de commandes commencent par ce code ASCII.

RESET ("<CSI>c")

Reset du device de console.

INSERT [N] SPACES ("<CSI>[N]@")

Insère N espaces à partir de la position actuelle du curseur. Si on ne fait pas figurer N, un seul espace sera inséré. N est indiqué sous forme de chaîne décimale. "<CSI>12@" insère 12 espaces. @ est le caractère de code ASCII 64.

CURSOR UP [N] ("<CSI>[N]A")

Le curseur est déplacé de N positions vers le haut (avec "<CSI>2A" par exemple, le curseur est déplacé de 2 positions vers le haut). Lorsque N ne figure pas, le curseur est déplacé d'une position.

CURSOR DOWN [N] ("<CSI>[N]B")

Le curseur est déplacé de N positions vers le haut (avec "<CSI>2A" par exemple, le curseur est déplacé de 2 positions vers le haut). Lorsque N ne figure pas, le curseur est déplacé d'une position.

CURSOR FORWARD [N] ("<CSI>[N]C")

Le curseur est déplacé de N positions vers la droite (avec "<CSI>20C" par exemple, le curseur est déplacé de 20 positions vers la droite). Lorsque N ne figure pas, le curseur est déplacé d'une position.

CURSOR BACKWARD [N] ("<CSI>[N]D")

Le curseur est déplacé de N positions vers la gauche (avec "<CSI>20C" par exemple, le curseur est déplacé de 20 positions vers la gauche). Lorsque N ne figure pas, le curseur est déplacé d'une position.

CURSOR NEXT LINE [N] ("<CSI>[N]E")

Le curseur est déplacé de N lignes vers le bas et ramené à la première colonne. "<CSI>E" a le même effet que <Retum>

CURSOR PRECEDING LINE [N] ("<CSI>[N]F")

Le curseur est déplacé de N lignes vers le haut et ramené à la première colonne. "<CSI>F" a le même effet que <Retum>

MOVE CURSOR ("<CSI>[N][;M]H")

Le curseur est placé dans la ligne N et, si M est indiqué, dans la colonne M. Si M n'est pas indiqué, il ne faut pas entrer le point-virgule. "<CSI>1H", "<CSI>;H" ou "<CSI>1H" ou "<CSI>H" placent le curseur dans le coin supérieur gauche (Curseur Home).

ERASE TO END OF DISPLAY ("<CSI>J")

Efface l'écran jusqu'à la fin, à partir de la position actuelle du curseur. Pour effacer l'écran en entier, vous disposez de la séquence suivante: "<CSI>1;1H" (Curseur Home) et "<CSI>J" (Effacer).

ERASE TO END OF LINE ("<CSI>K")

La ligne dans laquelle se trouve le curseur est effacée à partir de la position actuelle du curseur (Fonction Ctrl-Y de ED).

INSERT LINE ("<CSI>L")

Insère une ligne au-dessus de celle dans laquelle se trouve le curseur.

DELETE LINE ("<CSI>M")

Efface la ligne dans laquelle se trouve le curseur, déplace d'une ligne vers le haut toutes les lignes qui suivent, et efface la dernière ligne (Fonction Ctrl-B de ED, où la dernière ligne est réécrite).

DELETE CHARACTER ("CSI>[N]P")

Efface le caractère sur lequel se trouve le curseur, et N autres caractères sur la droite, si N est défini.

SCROLL UP [N] LINES ("<CSI>[N]S")

Décale l'ensemble de l'écran de N lignes (ou de 1 ligne) vers le haut. Les lignes libérées dans le bas sont remplacées par des lignes vides.

SCROLL DOWN [N] LINES ("<CSI>[N]T")

Décale l'ensemble de l'écran de N lignes (ou de 1 ligne) vers le bas. Les lignes libérées dans le haut sont remplacées par des lignes vides.

SETMODE ("<CSI>20h")

Linefeed est considéré comme Return+Linefeed (0x0c + 0x0a). Le curseur se retrouve avec Linefeed (0x0a) dans la ligne suivante de la première colonne.

RESET MODE ("<CSI>20L")

Cette fois, Linefeed est considéré à lui tout seul. Si l'on envoie un Linefeed (0x0a), le curseur se trouve dans la ligne suivante, mais il reste dans la même colonne.

DEVICE STATUS REPORT ("<CSI>6n")

Cette chaîne de contrôle demande au device de console d'envoyer un rapport sur l'état, de la forme suivante: "<CSI>Ligne;ColonneR". Dans Ligne et Colonne, on trouve les chaînes décimales indiquant précisément la ligne et la colonne dans lesquelles se trouve le curseur.

Ce rapport peut être reçu au moyen de CMD_READ. Notez que vous devez convertir les chaînes décimales en valeurs décimales, si vous désirez calculer avec elles, par exemple pour définir de nouvelles positions.

SELECT GRAPHIC RENDITION "<CSI><Style>; <Foreground>; <Background>m")

A l'aide de cette commande, vous pouvez déterminer l'aspect du caractère à sortir. "Style" peut prendre les valeurs suivantes en tant que code ASCII

- 0 Texte normal
- 1 Gras
- 3 Italique
- 4 Souligné
- 7 Inversé

Dans "Foreground" et "Background", vous indiquez la couleur de premier plan et la couleur de fond des caractères à afficher :

Foreground		Background	
30	Couleur 0	40	Couleur 0
31	Couleur 1	41	Couleur 1
...		...	
37	Couleur 7	47	Couleur 7

Si vous voulez par exemple que le texte soit souligné et en gras dans les couleurs 0 et 1, vous devez envoyer les séquences suivantes :

"<CSI>4;30;40m" (souligné, sans couleur)
"<CSI>1;20;41m" (gras, en couleur)

Notez qu'il faut indiquer les valeurs pour Style, Foreground et Background.

SET PAGE LENGTH "<CSI><LEN>t")

Cette commande définit le nombre de lignes que le device de console devra gérer dans la fenêtre. Normalement, on utilise toujours la fenêtre en entier pour l'affichage du texte, mais vous pouvez limiter le secteur de texte avec la commande présente et celles qui suivent. Lorsqu'on modifie la taille de la fenêtre, le device de console calcule en outre les nouvelles valeurs à partir du jeu de caractères actuel, et définit en conséquence la taille du secteur de texte. Mais cela n'est réalisé que si vous n'avez pas défini de valeurs personnelles. Si vous désirez que le device de console gère lui-même la taille de la fenêtre ou du secteur de texte, vous devez appeler uniquement les commandes, sans indication de valeurs, par exemple "<CSI>t".

SET LINE WIDTH ("<CSI><width>u")

Cette commande définit le nombre de caractères par ligne. La place restant éventuellement libre à droite peut être utilisée par exemple pour intégrer des graphiques.

SET LEFT OFFSET ("<CSI><offset>x")

Cette commande indique à partir de quelle ligne verticale (et non colonne), le secteur de texte doit commencer. "<CSI>8x" permet d'avoir par exemple 8 lignes sur la marge de gauche de la fenêtre, pour les petits graphiques, la bande de défilement, etc.

SET TOP OFFSET ("<CSI><offset>y")

Cette commande indique à partir de quelle ligne graphique (non pas ligne de texte) le secteur de texte doit commencer. Le secteur laissé libre peut être utilisé avec un traitement de texte pour l'affichage des positions de Tab (cf. BeckerText).

CURSOR ON ("<CSI>0 p")

Activer le curseur.

CURSOR OFF ("<CSI>p")

Désactiver le curseur (invisible).

WINDOW STATUS ("<CSI>0 q")

Envoie le Window-Status-Request. Vous obtenez un message sur l'état, de la forme suivante au moyen de CMD_READ :

```
"<CSI>1;1;<limite inférieure>;<limite à droite> r"
```

Vous obtenez donc la position du coin supérieur gauche (1;1) et la position du coin inférieur droit.

RAW EVENTS

Si les informations envoyées par le device de console par l'intermédiaire des touches ne suffisent pas, vous pouvez recevoir ce que l'on appelle des RAW EVENTS. Il faut pour cela faire savoir au device de console quels sont les RAW EVENTS devant être envoyés :

- | | |
|----|---|
| 0 | Pas d'opération |
| 1 | Pression sur les touches et relâchement des touches |
| 2 | Pressions sur les boutons de la souris |
| 3 | Une fenêtre a été activée |
| 4 | Position du pointeur de la souris |
| 5 | non utilisé |
| 6 | Timer Events |
| 7 | Clic sur un gadget |
| 8 | Gadget relâché |
| 9 | Présentation du requester |
| 10 | Sélection du menu |

- | | |
|----|---|
| 11 | Sélection du gadget de fermeture |
| 12 | Modification de la taille de la fenêtre |
| 13 | Nouveau tracé de la fenêtre |
| 14 | Les Preferences ont été modifiées |
| 15 | Disquette enlevée |
| 16 | Disquette introduite |

Si vous avez déjà eu affaire au device d'input, vous constaterez que les événements que vous pouvez interroger à l'aide du device de console sont précisément les mêmes que ceux du device d'input. Ici encore, les flags IDCMP correspondants pour l'interrogation des événements doivent être posés. Vous provoquez avec <CSI>Numéro_Evénement l'envoi par le device de console des événements voulus. Vous recevez alors un message de la forme suivante, lorsque surgit l'événement attendu :

```
"<CSI><Class>;<Subclass>;<KeyCode>;<Qualifiers>;<X>;<Y>;<Seconds>;<MicroSeconds>|"
```

Si vous envisagez la structure Input-Event, vous remarquerez de grandes similitudes entre cette structure et la chaîne de contrôle renvoyée par le device de console. Mais que signifient les différents chaînes décimales ?

<Class> Indique le numéro de l'événement (chaîne décimale), constaté par le device de console. Vous avez la possibilité de faire interroger par le device de console en même temps plusieurs événements, ou même tous les événements. Si vous désirez que les pressions sur les touches et les clics avec la souris soient signalés, vous pouvez l'obtenir à l'aide de "<CSI>1" suivi de "<CSI>2", ou simplement de "<CSI>1;2".

<Subclass> Égal à 0 le plus souvent. Prend la valeur 1 uniquement dans le cas où la souris a été connectée au port de droite (Gameport 2) (SetMPort()).

<KeyCode> Reçoit la chaîne décimale qui reproduit le code clavier de la touche pressée ou relâchée (cf. device d'input).

<Qualifiers> Indique quelle est la touche pressée : Ctrl, Alt ou Shift :

- | | |
|-------|---|
| 1 | Shift de gauche |
| 2 | Shift de droite |
| 4 | Caps Lock |
| 8 | Ctrl |
| 16 | Alt de gauche |
| 32 | Alt de droite |
| 64 | Amiga de gauche |
| 128 | Amiga de droite |
| 256 | Touche sur le bloc numérique |
| 512 | Touche répétée (fonction Repeat) |
| 1024 | Interrupt (non utilisé) |
| 2048 | Multi-Broadcast (événement pour fenêtre actuelle) |
| 4096 | Bouton de gauche sur la souris |
| 8192 | Bouton de droite sur la souris (non utilisé) |
| 16384 | Bouton de droite sur la souris |
| 32768 | Mouvement relatif de la souris |

Si l'on presse plusieurs de ces "qualificateurs" en même temps, les valeurs correspondantes sont additionnées et transmises dans <Qualifier>.

<X><Y> Ces chaînes décimales contiennent le mouvement relatif de la souris ou l'adresse du gadget sélectionné (x<<16+Y)

<Seconds> Contient l'heure système

<MicroSeconds> Contient l'heure système

C'est à l'utilisateur qu'il revient alors de décoder la chaîne fournie par le device de console, et d'interpréter les valeurs obtenues - à moins d'avoir recours à Intuition. L'utilisateur n'a pas besoin de faire de longues conversions, il reçoit en effet directement le flag IDCMP de l'événement survenu, et par exemple le code clavier. Nous avons ainsi décrit toutes les chaînes de contrôle qui peuvent être envoyées et reçues. Pour que vous puissiez vous familiariser un peu avec elles, nous avons développé un éditeur du device de console.

Vous pouvez entrer par l'intermédiaire du clavier les différentes chaînes de contrôle, et envisager directement le résultat. Du fait que le Control-Sequence-Introducer n'est pas accessible directement par le clavier, nous interrogeons la touche Escape, et interprétons Escape comme <CSI>.

Pour sortir de l'éditeur, il faut actionner q <Return>.

```
*****
*                                         Connici.c
*                                         (c) Bruno Jennrich
*                                         Août 1988
*
*****/
*****/
* Compile-Info:
* *
* cc Connici.c
* ln Connici.o Con_Support.o Devs_Support.o -lc
*****/
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "devices/console.h"
#include "devices/keymap.h"
#include "intuition/intuitionbase.h"
#include "intuition/intuition.h"
#define MEMTYPE (MEMF_CHIP | MEMF_CLEAR)
#define CON_LEN (ULONG) (sizeof(struct IOStdReq))
VOID *Open();
VOID *AllocMem();
VOID *GetDeviceBlock();
VOID *OpenLibrary();
VOID *OpenScreen();
VOID *OpenWindow();
struct Screen      *Screen      = 0L;
struct Window       *Window       = 0L;
struct IntuitionBase *IntuitionBase = 0L;
```

```

struct NewScreen      NewScreen = {
    0, 0, 640, 255, 4,
    0, 1,
    HIRES,
    CUSTOMSCREEN,
    (UBYTE*) "No Name",
    0L,
    0L
};

struct NewWindow      NewWindow = {
    0, 0,
    640, 255,
    0, 1,
    0L,
    (ULONG) ACTIVATE,
    0L,
    0L,
    (UBYTE*) "Console-Device-Editor (c)",
    Bruno Jennrich",
    0L,
    0L,
    0, 0,
    0, 0,
    CUSTOMSCREEN
};

struct IOStdReq *ConsoleRead = 0L,
                 *ConsoleWrite = 0L;
/*************************************************************************/
*                           CloseIt()                               (User)*
*
* Fonction: Tout fermer en cas d'erreur
* -----
* Paramètres d'entrée:
*   *
* String: Error-Message
/*************************************************************************/
VOID CloseIt(String)
char      *String;
{
    WORD i;
    WORD *dff180 = (WORD *)0xdff180;
    WORD Error = 0;
    if (strlen(String) > 0L)
    {
        for (i=0;i<0xffff;i++) *dff180 = i;
        puts(String);
        Error = 10;
    }
    if (Window != 0L) CloseWindow(Window);
    if (Screen != 0L) CloseScreen(Screen);
    if (IntuitionBase != 0L) CloseLibrary(IntuitionBase);
    if (ConsoleRead != 0L) Close_A_Device(ConsoleRead);
    if (ConsoleWrite != 0L) FreeDeviceBlock(ConsoleWrite);
    exit (Error);
}

/*************************************************************************/
*                           Open_Screen_and_Window()           (User)*

```

```

*
* Fonction: Ouvrir Editor Screen et Window
*****
VOID Open_Screen_and_Window()
{
    IntuitionBase = (struct IntuitionBase*)
        OpenLibrary("intuition.library", OL);
    if (IntuitionBase == OL) CloseIt("No IntuitionBase !");
    Screen = (struct Screen *) OpenScreen(&NewScreen);
    if (Screen == OL) CloseIt("No Screen !");
    NewWindow.Screen = Screen;
    Window = (struct Window *) OpenWindow(&NewWindow);
    if (Window == OL) CloseIt("No Window !");
}
 ****
*             Open_Screen_and_Window()           (User)*
*
* Fonction: Fermer Editor Screen et Window
*****
VOID Close_Screen_and_Window()
{
    CloseWindow(Window);
    CloseScreen(Screen);
    CloseLibrary(IntuitionBase);
}
 ****
*             main()                     (User)*
*
****

main ()
{
    UWORLD i;
    BYTE InputString[256];
    BYTE *BufPointer; /* Position actuelle de saisie à l'intérieur
*/
                           /* de InputString */
    UWORLD Pos; /* Nombre des caractères dans InputString */
    BOOL Quit = FALSE; /* Quitter le programme? */
    BOOL Return = FALSE; /* Return pressé? */
    Open_Screen_and_Window();
    ConsoleRead = (struct IOStdReq *)GetDeviceBlock(CON_LEN);
    ConsoleWrite = (struct IOStdReq *)GetDeviceBlock(CON_LEN);
    ConsoleRead->io_Data = (APTR) Window;
    ConsoleRead->io_Length = (ULONG) (sizeof(struct Window));
    Open_A_Device("console.device", OL, &ConsoleRead, OL, OL);
    Console_Copy(ConsoleRead, ConsoleWrite);
    BufPointer = InputString;
    Pos = 0;
    while(!Quit)
    while(!Return)
    {
        *BufPointer = (BYTE)0;
        Console_Read(ConsoleRead, BufPointer, 1L);
        if (*BufPointer == (BYTE)0x08) /* Backspace */
        {
            if (Pos>0)
            {
                *BufPointer = (BYTE)0;
                BufPointer--;
            }
        }
    }
}

```

```

        if (*BufPointer == (BYTE) 0x1b)
        {
            /* < C S I > */
            Console_Write(ConsoleWrite, "\010\010\010\010\010", -1L);
            Console_Write(ConsoleWrite, "\n", -1L);
            Console_Write(ConsoleWrite, "\010\010\010\010\010", -1L);
        }
        else
        {
            Console_Write(ConsoleWrite, "\010", 1L);
            Console_Write(ConsoleWrite, "\n", 1L);
            Console_Write(ConsoleWrite, "\010", 1L);
        }
        *BufPointer = (BYTE)0;
        Pos--;
    }
}
else
{
    if (Pos < 256)
    {
        if (*BufPointer == (BYTE) 0x9b)
        {
            /* Remplacer CSI reçu par 0x07 */
            *BufPointer = 0x7;
        }
        if (*BufPointer == 0x0d) /* Return */
        {
            Return = TRUE;
            Console_Write(ConsoleWrite, "\012", 1L);
            if (*InputString == 'q') Quit = TRUE; /* q pressé? */
        }
        else
        if (*BufPointer == 0x1b) /* Escape */
        {
            Console_Write(ConsoleWrite, "<CSI>", 5L);
        }
        else Console_Write(ConsoleWrite, BufPointer, 1L);
        BufPointer++;
        Pos++;
    }
}
Return = FALSE;
*BufPointer = (BYTE)0;
if (*InputString == (BYTE)0x1b)
{
    /* après Return afficher chaîne de contrôle*/
    *InputString = (BYTE) 0x9b;
    Console_Write(ConsoleWrite, InputString, -1L);
}
BufPointer = InputString;
Pos = 0;
}
Close_A_Device(ConsoleRead);
FreeDeviceBlock(ConsoleWrite);
Close_Screen_and_Window();
}

```

Remarque : Après "<CSI>1{", il n'y a pas de retour.

Lors de la sortie des chaînes de contrôle reçues, les codes <CSI> sont remplacés par BELL (0x07). Certaines pressions sur les touches (par exemple les touches de direction du curseur) envoient également un <CSI>. Il arrive ainsi que les touches du curseur et les touches de fonction ne fonctionnent pas, ou sortent A, 10, etc. Un petit conseil: les chaînes de contrôle affichées ici fonctionnent également dans une fenêtre "CON:!"

4.9.1. Keymapping

Voici une fonctionnalité très intéressante du device de console: la possibilité de redéfinir les touches. Ainsi à la place d'un "a", on peut sortir la chaîne "Hello, comment allez-vous?". Le device de console reçoit en effet du device d'input le code de la touche pressée (que le device d'input connaît par l'intermédiaire du device de clavier). Le device de console extrait alors à l'aide de tables le code ASCII correspondant à la touche pressée, et le transmet à l'utilisateur. Vous avez maintenant la possibilité de modifier ces tables. Il vous faut pour cela structure Keymap, que voici :

Offset	Structure
0x00	struct KeyMap
0x04	{
0x00	UBYTE *km_LoKeyMapTypes;
0x04	ULONG *km_LoKeyMap;
0x08	UBYTE *km_LoCapsable;
0x0c	UBYTE *km_LoRepeatable;
0x10	UBYTE *km_HiKeyMapTypes;
0x14	ULONG *km_HiKeyMap;
0x18	UBYTE *km_HiCapsable;
0x1c	UBYTE *km_HiRepeatable;
0x20	}; /* défini dans "devices/keymaps.h" */

Cette structure contient des pointeurs sur les secteurs de données, qui indiquent ce qui sera envoyé au device de console lors de la pression sur une touche déterminée. On distingue entre le Hi-Map et la Lo-Map. Le Lo-KeyMap est destiné aux codes allant de 0x0 à 0x3f. Le Hi-KeyMap contient les données des touches 0x40 à 0x67.

Quelle est la structure d'une table de clavier? Pour répondre à cette question, nous devons d'abord nous intéresser aux KeyMapTypes et au KeyMap. Les pointeurs km_LoKeyMapTypes et km_HiKeyMapTypes sont dirigés sur un Byte-Array qui définit les "qualifiers" (Shift, Alt, Ctrl) soutenus pour chacune des touches, ou qui indique si une chaîne (KCF_STRING) doit être envoyée à la place d'un simple code ASCII. Voici les valeurs autorisées :

```
#define KC_NOQUAL    0x00
#define KCF_SHIFT    0x01
#define KCF_ALT     0x02
#define KCF_CONTROL  0x04
#define KC_VANILLA   0x07 /* Shift+Alt+Ctrl */
#define KCF_STRING   0x40 /* String */
/* défini dans "devices/keymaps.h" */
```

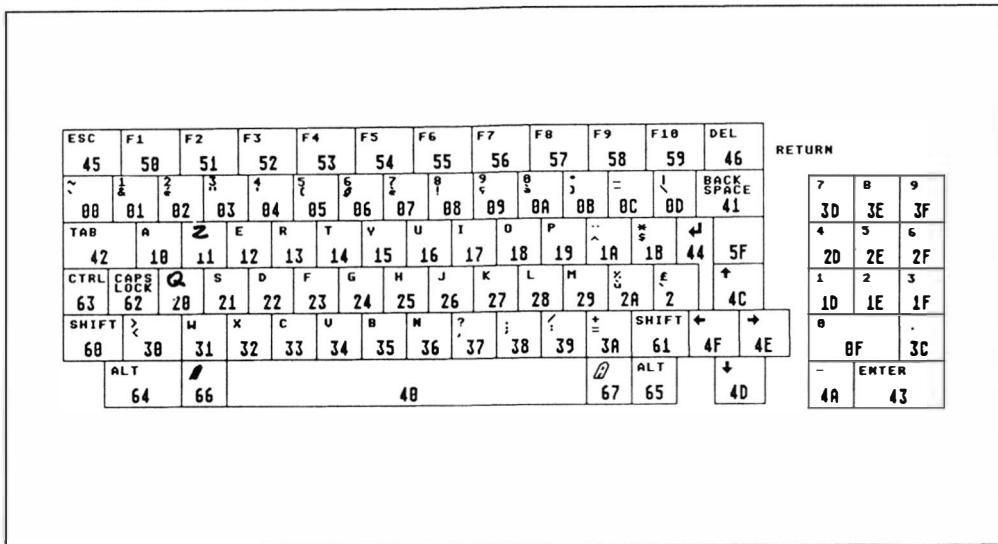


Figure 4 - 52 : Clavier

Dans km_LoKeyMap et km_HiKeyMap, on trouve alors les codes ASCII devant être envoyés lorsqu'une touche est pressée seule ou avec un qualifieur. Ce secteur de mémoire est à chaque fois un tableau de mots longs (4 octets = 4 codes ASCII).

Il en résulte qu'une touche ne peut (réellement) soutenir que deux qualifieurs (par exemple Shift et Alt ou Ctrl+Alt). On obtient des codes ASCII correspondant aux trois cas suivants: pression d'une touche seule, pression d'une touche en relation avec l'un des deux qualifieurs (par exemple Shift ou Alt), et pression d'une touche avec deux qualifieurs (par exemple Shift et Alt). Lorsqu'on utilise des chaînes à la place de codes ASCII simples, on peut envoyer jusqu'à 8 chaînes différentes avec une pression de touche (et les qualifieurs correspondants). Examinons un Keymap, qui met à la disposition de l'utilisateur aussi bien des codes ASCII simples que des chaînes :

```
#asm
;ownkeymap.asm (c) Bruno Jennrich
KC_NOQUAL equ 0
KC_VANILLA equ 7
KCF_SHIFT equ 1
KCF_ALT equ 2
KCF_CONTROL equ 4
KCF_STRING equ 64
CSI equ $9b
public _LoKeyMapTypes
even
_LoKeyMapTypes:
dc.b KC_VANILLA ;$00 Tilde
dc.b KC_VANILLA ;$01 1
dc.b KC_VANILLA ;$02 2
dc.b KC_VANILLA ;$03 3
dc.b KC_VANILLA ;$04 4
dc.b KC_VANILLA ;$05 5
```

dc.b KC_VANILLA	:\$06	6
dc.b KC_VANILLA	:\$07	7
dc.b KC_VANILLA	:\$08	8
dc.b KC_VANILLA	:\$09	9
dc.b KC_VANILLA	:\$0a	0
dc.b KC_VANILLA	:\$0b	8
dc.b KC_VANILLA	:\$0c	
dc.b KC_VANILLA	:\$0d	\
dc.b 0 :\$0e	non défini !	
dc.b 0 :\$0e	non défini	
dc.b KC_NOQUAL	:\$0f	0 (pavé numérique)
dc.b KC_VANILLA	:\$10	q
dc.b KC_VANILLA	:\$11	w
dc.b KC_VANILLA	:\$12	e
dc.b KC_VANILLA	:\$13	r
dc.b KC_VANILLA	:\$14	t
dc.b KC_VANILLA	:\$15	z
dc.b KC_VANILLA	:\$16	u
dc.b KC_VANILLA	:\$17	i
dc.b KC_VANILLA	:\$18	o
dc.b KC_VANILLA	:\$19	p
dc.b KC_VANILLA	:\$1a	ü
dc.b KC_VANILLA	:\$1b	+
dc.b 0 :\$1c	non défini	
dc.b KC_NOQUAL	:\$1d	1 (pavé numérique)
dc.b KC_NOQUAL	:\$1e	2 (pavé numérique)
dc.b KC_NOQUAL	:\$1f	3 (pavé numérique)
dc.b KC_VANILLA	:\$20	a
dc.b KC_VANILLA	:\$21	s
dc.b KC_VANILLA	:\$22	d
dc.b KC_VANILLA	:\$23	f
dc.b KC_VANILLA	:\$24	g
dc.b KC_VANILLA	:\$25	h
dc.b KC_VANILLA	:\$26	j
dc.b KC_VANILLA	:\$27	k
dc.b KC_VANILLA	:\$28	l
dc.b KC_VANILLA	:\$29	ö
dc.b KC_VANILLA	:\$2a	ä
dc.b 0 :\$2b	réserve	
dc.b 0 :\$2c	non défini	
dc.b KC_NOQUAL	:\$2d	4 (pavé numérique)
dc.b KC_NOQUAL	:\$2e	5 (pavé numérique)
dc.b KC_NOQUAL	:\$2f	6 (pavé numérique)
dc.b 0 :\$30	réserve	
dc.b KC_VANILLA	:\$31	y
dc.b KC_VANILLA	:\$32	x
dc.b KC_VANILLA	:\$33	c
dc.b KC_VANILLA	:\$34	v
dc.b KC_VANILLA	:\$35	b
dc.b KC_VANILLA	:\$36	n
dc.b KC_VANILLA	:\$37	m
dc.b KC_VANILLA	:\$38	.
dc.b KC_VANILLA	:\$39	.
dc.b KC_VANILLA	:\$3a	-
dc.b 0 :\$3b	non défini	
dc.b KC_NOQUAL	:\$3c	, (pavé numérique)
dc.b KC_NOQUAL	:\$3d	7 (pavé numérique)
dc.b KC_NOQUAL	:\$3e	8 (pavé numérique)
dc.b KC_NOQUAL	:\$3f	9 (pavé numérique)

```

public _LoKeyMap
even
_LoKeyMap:
dc.b "", "", "[" , "[" :$00 [ 
dc.b $21+$80, $31+$80, "!", "1" :$01 1 
dc.b $22+$80, $32+$80, "$2", "2" :$02 2 
dc.b $a7+$80, $33+$80, "", "3" :$03 3 
dc.b $24+$80, $34+$80, "$", "4" :$04 4 
dc.b $25+$80, $35+$80, "%", "5" :$05 5 
dc.b $26+$80, $36+$80, "&", "6" :$06 6 
dc.b $2f+$80, $37+$80, "/", "7" :$07 7 
dc.b $28+$80, $38+$80, "(", "8" :$08 8 

dc.b $29+$80, $39+$80, ")" , "9" :$09 9 
dc.b $3d+$80, $30+$80, "-", "0" :$0a 0 
dc.b $3f+$80, $df+$80, "?", "8" :$0b 8 
dc.b $27+$80, $60+$80, "", "" :$0c 
dc.b $7c+$80, $5c+$80, "|", "\\" :$0d \ 
dc.l $0 :$0e non défini 
dc.l $0 :$0e non défini 
dc.b $00, $00, $00, "0" :$0f 0 (pavé numérique) 
dc.b $51+$80, $71+$80, "q" :$10 q 
dc.b $57+$80, $77+$80, "W" :$11 w 
dc.b $45+$80, $65+$80, "E" :$12 e 
dc.b $52+$80, $72+$80, "R" :$13 r 
dc.b $54+$80, $74+$80, "T" :$14 t 
dc.b $5a+$80, $7a+$80, "Z" :$15 z 
dc.b $55+$80, $75+$80, "U" :$16 u 
dc.b $49+$80, $69+$80, "I" :$17 i 
dc.b $4f+$80, $6f+$80, "O" :$18 o 
dc.b $50+$80, $70+$80, "P" :$19 p 
dc.b $dc+$80, $fc+$80, "Ü" :$1a ü 
dc.b $2a+$80, $2b+$80, "+" :$1b + 
dc.b $00, $00, $00, $00 :$1c non défini 
dc.b $00, $00, $00, "1" :$1d 1 (pavé numérique) 
dc.b $00, $00, $00, "2" :$1e 2 (pavé numérique) 
dc.b $00, $00, $00, "3" :$1f 3 (pavé numérique) 
dc.b $41+$80, $61+$80, "A" :$20 a 
dc.b $53+$80, $73+$80, "S" :$21 s 
dc.b $44+$80, $64+$80, "D" :$22 d 
dc.b $46+$80, $66+$80, "F" :$23 f 
dc.b $47+$80, $67+$80, "G" :$24 g 
dc.b $48+$80, $68+$80, "H" :$25 h 
dc.b $4a+$80, $6a+$80, "J" :$26 j 
dc.b $4b+$80, $6b+$80, "K" :$27 k 
dc.b $4c+$80, $6c+$80, "L" :$28 l 
dc.b $d6+$80, $f6+$80, "Ö" :$29 ö 
dc.b $c4+$80, $e4+$80, "Ä" :$2a ä 
dc.b $00, $00, $00, $00 :$2b réservé 
dc.b $00, $00, $00, $00 :$2c non défini 
dc.b $00, $00, $00, "4" :$2d 4 (pavé numérique) 
dc.b $00, $00, $00, "5" :$2e 5 (pavé numérique) 
dc.b $00, $00, $00, "6" :$2f 6 (pavé numérique) 
dc.b $00, $00, $00, $00 :$30 réservé 
dc.b $59+$80, $79+$80, "Y" :$31 y 
dc.b $58+$80, $78+$80, "X" :$32 x 
dc.b $43+$80, $63+$80, "C" :$33 c 
dc.b $56+$80, $76+$80, "V" :$34 v

```

```

dc.b $42+$80, $62+$80, "B", "b"      ;$35    b
dc.b $4e+$80, $6e+$80, "N", "n"       ;$36    n
dc.b $4f+$80, $6f+$80, "M", "m"       ;$37    m
dc.b $3b+$80, $2c+$80, ":", ".", ":" ;$38    .
dc.b $3a+$80, $2e+$80, ":", ".", ":" ;$39    .
dc.b $5f+$80, $2d+$80, "-", "-", "-" ;$3a    -
dc.b $00, $00, $00, $00                ;$3b    non défini
dc.b $00, $00, $00, ".", " "          ;$3c    , (pavé numérique)
dc.b $00, $00, $00, "1"              ;$3d    7 (pavé numérique)
dc.b $00, $00, $00, "2"              ;$3e    8 (pavé numérique)
dc.b $00, $00, $00, "3"              ;$3f    9 (pavé numérique)
public _HiKeyMapTypes
even
-HiKeyMapTypes:
dc.b KC_NOQUAL                      ;$40    Space
dc.b KC_NOQUAL                      ;$41    BackSpace
dc.b KC_NOQUAL                      ;$42    Tab
dc.b KC_NOQUAL                      ;$43    Enter
dc.b KC_NOQUAL                      ;$44    Return
dc.b KC_NOQUAL                      ;$45    Escape
dc.b KC_NOQUAL                      ;$46    Delete
dc.b 0     ;$47    undefined
dc.b 0     ;$48    undefined
dc.b 0     ;$49    undefined
dc.b KC_NOQUAL                      ;$4a    Numernfeld
dc.b 0     ;$4b    undefined
dc.b KCF_STRING+KCF_SHIFT           ;$4c    Up Arrow
dc.b KCF_STRING+KCF_SHIFT           ;$4d    Down Arrow
dc.b KCF_STRING+KCF_SHIFT           ;$4e    Forward Arrow
dc.b KCF_STRING+KCF_SHIFT           ;$4f    Backward Arrow
dc.b KCF_STRING+KCF_SHIFT           ;$50    F1
dc.b KCF_STRING+KCF_SHIFT           ;$51    F2
dc.b KCF_STRING+KCF_SHIFT           ;$52    F3
dc.b KCF_STRING+KCF_SHIFT           ;$53    F4
dc.b KCF_STRING+KCF_SHIFT           ;$54    F5
dc.b KCF_STRING+KCF_SHIFT           ;$55    F6
dc.b KCF_STRING+KCF_SHIFT           ;$56    F7
dc.b KCF_STRING+KCF_SHIFT           ;$57    F8
dc.b KCF_STRING+KCF_SHIFT           ;$58    F9
dc.b KCF_STRING+KCF_SHIFT           ;$59    F10
dc.b 0     ;$5a    undefined
dc.b 0     ;$5b    undefined
dc.b 0     ;$5c    undefined
dc.b 0     ;$5d    undefined
dc.b 0     ;$5e    undefined
dc.b KCF_STRING                      ;$5f    Help
dc.b 0     ;$60    Shift left
dc.b 0     ;$61    Shift right
dc.b 0     ;$62    Caps Lock
dc.b 0     ;$63    Ctrl
dc.b 0     ;$64    Alt left
dc.b 0     ;$65    Alt right
dc.b 0     ;$66    Amiga left
dc.b 0     ;$67    Amiga right
public _HiKeyMap
even
-HiKeyMap:
dc.b $00, $00, $00, $20              ;$40    Space
dc.b $00, $00, $00, $08              ;$41    BackSpace

```

```

dc.b $00, $00, $00, $09      ;$42 Tab
dc.b $00, $00, $00, $0d      ;$43 Enter
dc.b $00, $00, $00, $0d      ;$44 Return
dc.b $00, $00, $9b, $1b      ;$45 Escape
dc.b $00, $00, $00, $7f      ;$46 Delete
dc.b $00, $00, $00, $00      ;$47 undefined
dc.b $00, $00, $00, $00      ;$48 undefined
dc.b $00, $00, $00, $00      ;$49 undefined
dc.b $00, $00, $00, " - "   ;$4a Numeric Pad
dc.b $00, $00, $00, $00      ;$4b undefined
dc.l Up_Arrow                ;$4c Up Arrow
dc.l Down_Arrow               ;$4d Down Arrow
dc.l Forward_Arrow            ;$4e Forward Arrow
dc.l Backward_Arrow           ;$4f Backward Arrow
dc.l F1                      ;$50 F1
dc.l F2                      ;$51 F2
dc.l F3                      ;$52 F3
dc.l F4                      ;$53 F4
dc.l F5                      ;$54 F5
dc.l F6                      ;$55 F6
dc.l F7                      ;$56 F7
dc.l F8                      ;$57 F8
dc.l F9                      ;$58 F9
dc.l F10                     ;$59 F10
dc.b $00, $00, $00, $00      ;$5a undefined
dc.b $00, $00, $00, $00      ;$5b undefined
dc.b $00, $00, $00, $00      ;$5c undefined
dc.b $00, $00, $00, $00      ;$5d undefined
dc.b $00, $00, $00, $00      ;$5e undefined
dc.l Help                     ;$5f Help
dc.b $00, $00, $00, $00      ;$60 Shift left
dc.b $00, $00, $00, $00      ;$61 Shift right
dc.b $00, $00, $00, $00      ;$62 Caps Lock
dc.b $00, $00, $00, $00      ;$63 Ctrl
dc.b $00, $00, $00, $00      ;$64 Alt left
dc.b $00, $00, $00, $00      ;$65 Alt right
dc.b $00, $00, $00, $00      ;$66 Amiga left
dc.b $00, $00, $00, $00      ;$67 Amiga right

Up_Arrow:
dc.b 2                         ; Longueur du string (sans shift)
dc.b Up_Arrow_UnShift-Up_Arrow ; Offset
dc.b 2                         ; Longueur du string (avec shift)
dc.b Up_Arrow_Shift-Up_Arrow   ; Offset

Up_Arrow_UnShift:
dc.b CSI, "A"

Up_Arrow_Shift:
dc.b CSI, "T"

Down_Arrow:
dc.b 2                         ; Longueur du string (sans shift)
dc.b Down_Arrow_UnShift-Down_Arrow ; Offset
dc.b 2                         ; Longueur du string (avec shift)
dc.b Down_Arrow_Shift-Down_Arrow ; Offset

Down_Arrow_UnShift:
dc.b CSI, "B"

Down_Arrow_Shift:
dc.b CSI, "S"

Forward_Arrow:
dc.b 2                         ; Longueur du string (sans shift)
dc.b Forward_Arrow_UnShift-Forward_Arrow ; Offset

```

```

        dc.b 3                      ; Longueur du string (avec shift)
        dc.b Forward_Arrow_Shift-Forward_Arrow  ; Offset
Forward_Arrow_UnShift:
        dc.b CSI. "C"
Forward_Arrow_Shift:
        dc.b CSI. "A"
Backward_Arrow:
        dc.b 2                      ; Longueur du string (sans shift)
        dc.b Backward_Arrow_UnShift-Backward_Arrow  ; Offset
        dc.b 3                      ; Longueur du string (avec shift)
        dc.b Backward_Arrow_Shift-Backward_Arrow  ; Offset
Backward_Arrow_UnShift:
        dc.b CSI. "D"
Backward_Arrow_Shift:
        dc.b CSI. "@"
F1:
        dc.b 3                      ; Longueur du string (sans shift)
        dc.b F1_UnShift-F1          ; Offset
        dc.b 4                      ; Longueur du string (avec shift)
        dc.b F1_Shift-F1          ; Offset
F1_UnShift:
        dc.b CSI. "0"
F1_Shift:
        dc.b CSI. "10"
F2:
        dc.b 3                      ; Longueur du string (sans shift)
        dc.b F2_UnShift-F2          ; Offset
        dc.b 4                      ; Longueur du string (avec shift)
        dc.b F2_Shift-F2          ; Offset
F2_UnShift:
        dc.b CSI. "1"
F2_Shift:
        dc.b CSI. "11"
F3:
        dc.b 3                      ; Longueur du string (sans shift)
        dc.b F3_UnShift-F3          ; Offset
        dc.b 4                      ; Longueur du string (avec shift)
        dc.b F3_Shift-F3          ; Offset
F3_UnShift:
        dc.b CSI. "2"
F3_Shift:
        dc.b CSI. "12"
F4:
        dc.b 3                      ; Longueur du string (sans shift)
        dc.b F4_UnShift-F4          ; Offset
        dc.b 4                      ; Longueur du string (avec shift)
        dc.b F4_Shift-F4          ; Offset
F4_UnShift:
        dc.b CSI. "3"
F4_Shift:
        dc.b CSI. "13"
F5:
        dc.b 3                      ; Longueur du string (sans shift)
        dc.b F5_UnShift-F5          ; Offset
        dc.b 4                      ; Longueur du string (avec shift)
        dc.b F5_Shift-F5          ; Offset
F5_UnShift:
        dc.b CSI. "4"
F5_Shift:

```

```

        dc.b CSI, "14"
F6:
        dc.b 3          ; Longueur du string (sans shift)
        dc.b F6_UnShift-F6 ; Offset
        dc.b 4          ; Longueur du string (avec shift)
        dc.b F6_Shift-F6 ; Offset
F6_UnShift:
        dc.b CSI, "5"
F6_Shift:
        dc.b CSI, "15"
F7:
        dc.b 3          ; Longueur du string (sans shift)
        dc.b F7_UnShift-F7 ; Offset
        dc.b 4          ; Longueur du string (avec shift)
        dc.b F7_Shift-F7 ; Offset
F7_UnShift:
        dc.b CSI, "6"
F7_Shift:
        dc.b CSI, "16"
F8:
        dc.b 3          ; Longueur du string (sans shift)
        dc.b F8_UnShift-F8 ; Offset
        dc.b 4          ; Longueur du string (avec shift)
        dc.b F8_Shift-F8 ; Offset
F8_UnShift:
        dc.b CSI, "7"
F8_Shift:
        dc.b CSI, "17"
F9:
        dc.b 3          ; Longueur du string (sans shift)
        dc.b F9_UnShift-F9 ; Offset
        dc.b 4          ; Longueur du string (avec shift)
        dc.b F9_Shift-F9 ; Offset
F9_UnShift:
        dc.b CSI, "8"
F9_Shift:
        dc.b CSI, "18"
F10:
        dc.b 3          ; Longueur du string (sans shift)
        dc.b F10_UnShift-F10 ; Offset
        dc.b 4          ; Longueur du string (avec shift)
        dc.b F10_Shift-F10 ; Offset
F10_UnShift:
        dc.b CSI, "9"
F10_Shift:
        dc.b CSI, "19"
Help:
        dc.b 3          ; Longueur du string (sans shift)
        dc.b Help_UnShift-Help ; Offset
Help_UnShift:
        dc.b CSI, "?"
        even
#endifasm

```

Remarque : Pour des raisons indiscernables, il faut deux octets au lieu d'un dans la table KeyMapTypes pour le code clavier \$0e, et 8 octets au lieu de 4 dans la table KeyMap. Si vous n'y prêtez pas garde, toutes les pressions de touche

seront pourvues de codes erronées, par exemple "c" au lieu de C ou "m" au lieu de M.

Notez que les tableaux Keymap et KeyMapTypes doivent commencer à des adresses de mot (even). Après avoir défini les tableaux, vous devez également veiller à ce que le reste du programme se poursuive avec une adresse paire, car vous obtiendrez sinon dès le démarrage un gourou, annonçant une erreur d'adresse (n° de gourou: \$00000003).

Considérons maintenant de plus près les entrées dans LoKeyMap: vous remarquerez que le premier octet pour le code de touche \$20 (touche A) contient la valeur "A"+\$80. Le second octet contient "a"+\$80, et les deux derniers octets les valeurs "A" et "a".

Malheureusement l'assembleur de Aztec ne comprend pas les expressions du genre "A"+\$80. C'est pourquoi nous avons traduit le caractère "A" par son code ASCII \$41. Voyons quel est celui des 4 codes ASCII qui est envoyé lorsqu'une touche a été pressée en relation avec un qualifieur :

Vous voyez que c'est le plus souvent l'un des codes ASCII "a", "A", "A"+\$80 ou "a"+\$80, mais seulement lorsque les qualifiants Alt et Shift sont pressés.

Si les trois qualifiants sont autorisés, les bits 5 et 6 (\$30) sont effacés lorsque l'on presse la touche Ctrl. Vous n'avez donc aucun moyen de déterminer le code ASCII à envoyer en relation avec Ctrl, indépendamment des 4 codes définis dans le Keymap.

La situation est tout à fait différente lorsque nous utilisons des chaînes à la place des codes ASCII simples. Il faut noter ici que les 4 octets dans le Keymap pointent pour une entrée donnée sur un ou plusieurs descripteurs de chaîne. Un descripteur de chaîne a le format suivant :

Premier octet: Longueur de la chaîne à afficher

Second octet: Offset de la chaîne jusqu'au début des descripteurs de chaînes.

Voici un exemple :

```
StringDescriptor:
  dc.b 8                      :Longueur
  dc.b StringAAfficher1-StringDescriptor :Offset
  dc.b 14                     :Longueur
  dc.b StringAAfficher2-StringDescriptor :Offset
StringAAfficher1:  dc.b "String 1"
StringAAfficher2:  dc.b "Second string"
```

Du fait que les strings à afficher sont adressés par l'intermédiaire d'offsets, les strings doivent se trouver dans le secteur allant de +127 à -128 octets, comptés à partir du début des descripteurs de string.

Vous pouvez cependant faire en sorte de représenter les trois qualifiants et leurs combinaisons par d'autres chaînes. La figure qui suit montre quelles sont les chaînes qui peuvent être affichées à l'aide des différentes combinaisons de qualifiants autorisés et pressés :

(keyMapTypes)								
	S	A	C	S+A	C+A	S+C	S+A+C	
	a	a	a	a	a	a	a	a
	a	A	a	a	A	a	A	a
	a	a	A	a	a+\$80	A	a	a+\$80
	a	a	a	A	a	a+\$80	a+\$80	a~\$30
	a	A	A	a	A+\$80	A	A	A+\$80
	a	a	A	A	a+\$80	A+\$80	a+\$80	a+\$80~-30
	a	A	a	A	A	a+\$80	A+\$80	A~\$30
	a	A	A	A	A+\$80	A+\$80	A+\$80	A+\$80~-30
Abaisssé								

S = Shift
A = Alt
C = Control

Exemple : Autorisé : S+A Résultat: "a"+\$80

Abaissé : C+A

Entrée KeyMap :
dc.b "A"+\$80,"a"+\$80,"A","a"

Figure 4 - 53 : Qualif1

Notez que pour un qualifieur autorisé, il doit y avoir deux chaînes disponibles, puis quatre chaînes pour deux qualifieurs autorisés, et huit chaînes pour 3 qualifieurs autorisés. Pour chacune des chaînes à afficher, il faut également créer un descripteur de chaîne qui lui soit propre! Il ne reste plus qu'à expliquer ce que sont les pointeurs Lo/HiCapsable et Lo/HiRepeatable.

(KeyMapTypes)								
	S	A	C	S+A	C+A	S+C	S+A+C	
A	A	A	A	A	A	A	A	
S	A	B	A	A	B	A	B	B
A	A	A	B	A	C	B	A	C
C	A	A	A	B	A	C	C	E
S+A	A	B	B	A	D	D	B	D
C+A	A	A	B	B	C	B	C	G
S+C	A	B	A	B	A	C	D	F
S+A+C	A	B	B	B	D	D	D	H

S = Shift
A = Alt
C = Control

Entrée KeyMap

Newa:

DC . b 1
DC . b A -Newa
DC . b 1
DC . b B-Newa
DC . b 1
DC . b C-Newa
DC . b 1
DC . b D-Newa
DC . b 1
DC . b E-Newa
DC . b 1
DC . b F-Newa
DC . b 1
DC . b G-Newa
DC . b 1
DC . b H-Newa

A: dc.b "A"
B: dc.b "B"
C: dc.b "C"
D: dc.b "D"
E: dc.b "E"
F: dc.b "F"
G: dc.b "G"
H: dc.b "H"

Figure 4 - 54 : Qualif2

Comme vous le constaterez vous-même, certaines touches ne sont pas représentées par leur valeur avec Shift, malgré le fait que la touche Caps-Lock ait été pressée. Le

pointeur Capsable est dirigé sur un tableau de 8 octets, dont chaque bit est responsable du fait qu'une touche obéit au Caps-Lock (bit == 1) ou non (bit == 0).

Pour la touche N 0, le bit Capsable 0 du premier octet est posé (LoCapsable). Pour la touche 8, c'est le premier bit du second octet qui est concerné. Pour la touche 0x40, le bit concerné est le premier bit de HiCapsable. Les deux pointeurs sont dirigés sur un tableaux de 64 bits = 8 octets.

La situation est la même avec LoRepeatable et HiRepeatable. Ici aussi, il y a un bit réservé pour chaque touche dans HiKeyMap et LoKeyMap. Mais les bits posés indiquent cette fois si les touches correspondantes seront répétées lorsqu'on garde assez longtemps la pression sur elles (cf. device d'input). Si ce bit est égal à 0 par exemple pour la touche Return, cette touche ne sera pas répétée.

Comment peut-on utiliser un nouveau Keymap par l'intermédiaire du device de console? Il faut d'abord remplir une structure Keymap propre avec les valeurs de la structure Console-Window-Keymap à l'aide de `Console_AskKeyMap()` (cf. 4.12.3, structure ConUnit).

```
struct KeyMap:
{
    ...
    Console_AskKeyMap(ConsoleRead, &KeyMap);
}
```

Les pointeurs correspondants de la structure KeyMap se modifient alors, et envoient la commande `CD_SETKEYMAP` :

```
KeyMap.km_LoKeyMapTypes = (UBYTE*) &LoKeyMapTypes;
KeyMap.km_LoKeyMap     = (ULONG*) &LoKeyMap;
KeyMap.km_HiKeyMapTypes = (UBYTE*) &HiKeyMapTypes;
KeyMap.km_HiKeyMap     = (ULONG*) &HiKeyMap;
Console_SetKeyMap(ConsoleRead, &KeyMap);
```

On a installé de la sorte le nouveau Keymap. Voici les routines Con_Support utilisées plus haut :

```
*****
*                               Console_AskKeyMap()          (Con_Support)*
*
* Fonction: Remplir la structure KeyMap
* -----
* Paramètres d'entrée:
*
* ConReq:   Device-Block
* KeyMap:   Pointeur sur structure KeyMap
* -----
* Valeur en retour:
*
* FALSE: Erreur !!!
*****
BOOL Console_AskKeyMap(ConReq, KeyMap)
struct IOStdReq           *ConReq;
struct KeyMap              *KeyMap;
{
    ConReq->io_Length = (sizeof(struct KeyMap));
    ConReq->io_Data   = (APTR)KeyMap;
```

```

Do_Command(ConReq, (UWORD)CD_ASKKEYMAP);
if (ConReq->io_Error != (BYTE)0) return(FALSE);
return(TRUE);
}
/*********************************************
*                               Console_SetKeyMap()      (Con_Support)*
*
* Fonction: Installer KeyMap de console
*-----
* Paramètres d'entrée:
*
* ConReq: Device-Block
* KeyMap: Pointeur sur structure KeyMap*
*-----
* Valeur en retour:
*
* FALSE: Erreur !!!
*****************************************/
BOOL Console_SetKeyMap(ConReq, KeyMap)
struct IOStdReq           *ConReq;
struct KeyMap              *KeyMap;
{
    ConReq->io_Length = (sizeof(struct KeyMap));
    ConReq->io_Data   = (APTR)KeyMap;
    Do_Command(ConReq, (UWORD)CD_SETKEYMAP);
    if (ConReq->io_Error != (BYTE)0) return(FALSE);
    return(TRUE);
}

```

Vous avez également la possibilité de demander au device de console d'équiper de nouveaux Console-Windows avec une Keymap déterminé. Il existe pour cela les commandes CD_ASKDEFAULTKEYMAP et CD-SETDEFAULTKEYMAP. Si vous installez un nouveau Keymap à l'aide de ces deux commandes, ce sera le nouveau Keymap qui sera utilisé lors de chaque appel de Open_A_Device à venir ("console.device", 0L, &ConsoleRead, 0L, 0L). Ces commandes sont d'ailleurs utilisées par la commande SetMAP de CLI. La commande SetMAP entreprend toutefois d'autres modifications encore dans la structure ConUnit de la fenêtre CLI actuelle.

```

/*********************************************
*                               Console_AskDefaultKeyMap()      (Con_Support)*
*
* Fonction: Remplir structure KeyMap
*-----
* Paramètres d'entrée:
*
* ConReq: Device-Block
* KeyMap: Pointeur sur structure KeyMap
*-----
* Valeur en retour:
*
* FALSE: Erreur !!!
*****************************************/
BOOL Console_AskDefaultKeyMap;(ConReq, KeyMap)
struct IOStdReq           *ConReq;
struct KeyMap              *KeyMap;
{
    ConReq->io_Length = (sizeof(struct KeyMap));

```

```

ConReq->io_Data = (APTR)KeyMap;
Do_Command(ConReq, (UWORD)CD_ASKDEFAULTKEYMAP);
if (ConReq->io_Error != (BYTE)0) return(FALSE);
return(TRUE);
}

/*********************************************
*          Console_SetDefaultKeyMap()      (Con_Support)*
*
* Fonction: Installer Console-Default-KeyMap
*-----
* Paramètres d'entrée:
*
* ConReq: Device-Block
* KeyMap: Pointeur sur la structure KeyMap
*-----
* Valeur en retour:
*
* FALSE: Erreur !!!
*****************************************/
BOOL Console_SetDefaultKeyMap;(ConReq, KeyMap)
struct IOStdReq           *ConReq;
struct KeyMap              *KeyMap;
{
    ConReq->io_Length = (sizeof(struct KeyMap));
    ConReq->io_Data = (APTR)KeyMap;
    Do_Command(ConReq, (UWORD)CD_SETDEFAULTKEYMAP);
    if (ConReq->io_Error != (BYTE)0) return(FALSE);
    return(TRUE);
}

```

4.9.2. Eléments internes à la console

Après Open_A_Device(), ConsoleRead->io_Unit pointe sur une structure ConUnit. Cette structure contient toutes les variables importantes pour l'utilisation de la console :

Offset	Structure	
0	-----	struct ConUnit
	{	
0 0x00	struct MsgPort cu_MP;	/* Message-Port pour envoi */ /* et réception */
34 0x22	struct Window *cu_Window;	/* Console-Window */
38 0x26	WORD cu_XCP;	
40 0x28	WORD cu_YCP;	/* Position des caractères */
42 0x2a	WORD cu_XMax;	
44 0x2c	WORD cu_YMax;	/* Caractères maximaux */ /* Position */
46 0x2e	WORD cu_XRSize;	
48 0x30	WORD cu_YRSize;	/* Taille des caractères*/
50 0x32	WORD cu_XROrigin;	
52 0x34	WORD cu_YROrigin;	/* Début du */ /* texte */
54 0x36	WORD cu_XRExtant;	/* taille maximale */
56 0x38	WORD cu_YRExtant;	/* du texte */
58 0x3a	WORD cu_XMinShrink;	/* secteur minimal */
60 0x3c	WORD cu_YMinShrink;	/* non concerné */

```

62 0x3e      WORD          cu_XCCP;           /* Window-Resize */
64 0x40      WORD          cu_YCCP;           /* Position du curseur*/
66 0x42      struct KeyMap cu_KeyMapStruct; /* KeyMap */
98 0x62      DWORD         cu_TabStops[80]; /* Positions de Tab*/
                                         /* Cf. structure Rastport:
*/
178 0xb2      BYTE          cu_Mask;
179 0xb3      BYTE          cu_FgPen;
180 0xb4      BYTE          cu_BgPen;
181 0xb5      BYTE          cu_AOLPen;
182 0xb6      BYTE          cu_DrawMode;
183 0xb7      BYTE          cu_AreaPtSz;
184 0xb8      APTR          cu_AreaPtnr; /* Modèles de curseur */
188 0xbc      UBYTE         cu_Minterms[8];
196 0xc4      struct TextFont *cu_Font;
200 0xc8      UBYTE         cu_AlgoStyle;
201 0xc9      UBYTE         cu_TxFlags;
202 0xca      WORD          cu_TxHeight;
204 0xcc      WORD          cu_Tx_Width;
206 0xce      WORD          cu_TxBaseline;
208 0xd0      WORD          cu_TxSpacing;
210 0xd2      UBYTE         cu_Modes[3]; /* Mémoire pour Modes */
                                         /* et RAW EVENTS */
                                         /* (chaque fois 1 Bit) */
213 0xd5      UBYTE         cu_RawEvents[3];
216 0xd8 }     /* défini dans
"devices/conunit.h" */

```

4.9.2.1. Les fonctions de console

Outre les commandes comme CMD_WRITE, etc., il existe pour le device de console comme pour le device Timer des fonctions auxquelles on accède par l'intermédiaire d'offsets, comme dans le cas des fonctions habituelles de librairie. L'adresse de base est ici toutefois ConsoleDevice = ConsoleRead->io_Device.

Offset	Commande
-----	-----
0xa	CDInputHandler(&InputEvent)
	AO
-0x30	Actual = RawKeyConvert(&InputEvent, Buffer, Length, KeyMap)
	DO AO , A1 , D1 , A2

CDInputHandler() envoie l'événement défini dans Input-Event au Console-Window actuel. Vous pouvez par exemple initialiser une structure Input-Event et l'envoyer au device de console au moyen de CDInputHandler. L'action définie dans Input-Event est alors visible dans le device de console. CDInputHandler() peut être comparé à CMD_WRITE, à ceci près que ce ne sont pas ici des chaînes que l'on obtient à la sortie, mais une structure Input-Event.

CDInputHandler() appelle RawKeyConvert(). Cette commande traduit le Input-Event en une chaîne qui commence dans "Buffer" et dont la longueur maximale est "Length". Le Keymap indiqué est appelé à la rescousse pour la traduction. "Actual" contient le nombre

de caractères contenus dans la chaîne générée. Si Actual a la valeur -1, cela veut dire que le tampon n'était pas assez grand. L'appel d'une telle fonction se présente ainsi :

```
move.l _ConsoleDevice, a6  
;Initialisation des paramètres  
jsr    -$2a(a6)
```

4.9.2.2. Autres codes pour les touches

Si vous considérez avec attention le Keymap dont le listing se trouve plus haut, vous constaterez qu'outre les quelques codes de clavier non définis, il existe deux codes de clavier réservés (\$2b et \$30). Ces codes de clavier sont prévus pour les claviers étrangers, et pour une touche entre le Ä et la touche Return (code 0x2B), ainsi que pour une touche entre la touche Shift gauche et le Y (code 0x30).

Mais outre les codes de clavier allant de 0x00 à 0x67, il en existe encore quelques autres. Celles-ci ne peuvent cependant pas être influencées par un Keymap :

0x68	Bouton gauche de la souris
0x69	Bouton droit de la souris
0x6a	Bouton du milieu de la souris

Ces trois codes de touche ne sont jamais envoyés par le device de console en fonctionnement normal. Ces codes de touche ne sont accessibles par l'intermédiaire de la chaîne de contrôle reçue que si vous activez le mode RAW pour les clics sur les boutons de la souris à l'aide de "<CSI>2|".

0x80-0xe7

Ces codes de touche indiquent que la touche a été relâchée avec un code situé entre 0x00 et 0x67 (0x80 pour la touche 0x00, etc.). Ce code ne peut être reçu que si le flag KCF_DOWNUP (0x08) a été posé pour la touche en question dans KeyMapTypes.

0xf9

Le dernier code envoyé par le clavier était faux.

0xfa

Le tampon interne de clavier (10 caractères) est plein.

0xfb

Catastrophe au clavier - Erreur fatale !!!

0xfd

Keyboard Power-Up - des touches ont été pressées pendant le bootage. Elles sont envoyées en étant encadrées par 0xfd et 0xfe (par exemple: 0xfd, 0x03, 0x04, 0xfe).

0xfe

Keyboard Power-Up terminé - le clavier est initialisé et toutes les touches pressées entre-temps ont été envoyées.

0xff

La souris a été déplacée (sans pression sur un bouton).

4.10. Le device Clipboard

Vous avez sans doute déjà travaillé avec un traitement de texte, et utilisé les différentes opérations de déplacement de blocs. Si vous êtes occupé du développement d'un tel système de traitement de texte, vous aurez probablement pensé que vous devez créer et gérer des emplacements en mémoire pour ces opérations sur des blocs. C'est une entreprise passablement complexe (elle dépend d'ailleurs de l'organisation des données dans le traitement de texte). Mais si vous utilisez le device de clipboard, tout se simplifie. Vous n'avez plus besoin que d'écrire le bloc dans le device de clipboard, et celui le conserve aussi longtemps qu'il le faut, jusqu'à ce que vous le lisiez à nouveau, ou le déclariez invalide.

Voici comment on ouvre le device de clipboard :

```
struct IOClipReq *ClipReq = 0L;
#define CLIP_LEN (ULONG) sizeof(struct IOClipReq)
...
Open_A_Device("clipboard.device", Unit, &ClipReq, 0L, CLIP_LEN);
...
```

Etant donné que l'on ne peut jamais avoir qu'un seul bloc dans une unité (Unit) du device de clipboard, il faut ouvrir autant de fois le device de clipboard avec des numéros d'unité différents que vous avez de blocs. Pour comprendre le mode de fonctionnement du device de clipboard, il est important de connaître le bloc Device :

Offset	Structure
0	struct IOClipReq
0x00	struct Message io_Message;
0x14	struct Device *io_Device;
0x18	struct Unit *io_Unit; /* Quelle unité? */
0x1c	UWORD io_Command;
0x1e	UBYTE io_Flags;
0x1f	UBYTE io_Error;

```

32 0x20 ULONG          io_Actual: /* Nombre d'octets*/
                           /*effectivement transférés*/
36 0x24 ULONG          io_Length: /* Nombre d'octets */
                           /* à transférer */
40 0x28 SPTR           io_Data: /* Données (Stringpointer) */
44 0x2c ULONG           io_Offset: /* Offset à l'intérieur de l'unité
*/
48 0x30 LONG            io_ClipID: /* Numéro d'identification
                           /* du clip */
52 0x34 }; /* défini dans "devices/clipboard.h" */

```

Les variables io-Message, io_Device, etc. vous sont déjà connues, puisque vous les avez rencontrées dans les chapitres précédents. Les deux dernières variables vont nous intéresser ici tout particulièrement. Pour pouvoir emmagasiner les données en mémoire, il faut évidemment que le device de clipboard réserve de la place dans la mémoire. Pour savoir à partir de quelle position se sont effectuées les dernières opérations de lecture ou d'écriture, io_Offset contient précisément l'offset d'octet à l'intérieur de la mémoire Clipboard qui indique la dernière position de lecture/écriture.

On peut comparer cette position avec la position du fichier mise à votre disposition par DOS, pour savoir à quel endroit à l'intérieur du fichier on se trouve exactement au moment de la lecture ou de l'écriture. io_ClipID contient le nombre de blocs déjà écrits dans le device de clipboard, et à nouveau effacés. Voyons maintenant comment on écrit les données dans le device de clipboard. Nous nous aiderons comme d'habitude de la commande CMD_WRITE :

```

*****
*             Clip_Write()          (Clip_Support)*
*
* Fonction: Ecrire des données dans le device Clipboard *
*-----*
* Paramètres d'entrée: *
* *
* ClipReq: Device-Bloc *
* Data:   Données à écrire *
* Len:    Nombre des octets à écrire *
* FirstTime: TRUE  -> Première commande en écriture *
*             FALSE -> Commande en écriture d'une séquence *
*-----*
* Valeur en retour: *
* *
* Nombre des données effectivement écrites *
*****
ULONG Clip_Write(ClipReq, Data, Len, FirstTime)
struct IOClipReq *ClipReq;
APTR             Data;
LONG              Len;
BOOL             FirstTime;
{
    if (FirstTime==TRUE)
        ClipReq->io_Offset = 0L;
    ClipReq->io_Data    = (STRPTR) Data;
    ClipReq->io_Length  = Len;
    Do_Command(ClipReq, (UWORD) CMD_WRITE);
    return(ClipReq->io_Actual);
}

```

Pour que le device de clipboard ne mélange pas tout, il faut que vous mettiez les variables io_Offset et io_ClipID sur 0 lors du premier accès en écriture. Pour faire savoir au device de clipboard que toutes les données ont été écrites, une commande CMD-UPDATE est envoyée après commande CMD_WRITE. Vous pouvez ainsi écrire avec succès un grand bloc dans le device de clipboard.

Le bloc n'est pas forcément un texte. Il peut contenir des données quelconques (images, samples, etc.). Ce bloc se trouve désormais dans le device de clipboard. S'il ne reste plus assez de mémoire, le device de clipboard écrit votre bloc sur disquette, dans le répertoire "SYS:devs/clipboards". Le nom de fichier est le numéro de l'Unit, sous forme de chaîne décimale (par exemple 0). Pour pouvoir lire à nouveau les données, il vous suffit d'exécuter la commande CMD_READ :

```
*****
* Clip_Read()          (Clip_Support)*
*
* Fonction: Lire des données à partir du device Clipboard *
*-----*
* Paramètres d'entrée: *
* *
* ClipReq: Device-Bloc *
* Data:    Tampon de données *
* Len:     Nombre des octets à lire *
* FirstTime: TRUE  => Première commande de lecture *
*             FALSE => Commande de lecture d'une séquence *
*-----*
* Valeur en retour: *
* *
* Nombre des données effectivement lues (Erroné!) *
*****
ULONG Clip_Read(ClipReq, Data, Len, FirstTime)
struct IOClipReq *ClipReq;
APTR           Data;
LONG            Len;
BOOL           FirstTime;
{
    if (FirstTime==TRUE)
        ClipReq->io_Offset = 0L;
    ClipReq->io_Data   = (STRPTR) Data;
    ClipReq->io_Length = Len;
    Do_Command(ClipReq, (UWORD) CMD_READ);
    return(ClipReq->io_Actual);
}
```

Ici aussi, les variables doivent être remises à 0 lors de la première lecture. Malheureusement, nous devons faire observer une petite erreur dans le device de clipboard. Pour signaler que toutes les données ont été lues, la valeur 0 est normalement renvoyée dans io_Actual. Vous devez conserver vous-même la valeur de io_Length à l'esprit, pour que les données ne soient pas perdues. En outre, vous devez - une fois que toutes les données ont été lues - lire encore une fois ces données, et cette fois avec une valeur égale à 0 dans io_Length. Cela provient du fait qu'une séquence Read n'est envoyée pour le device de clipboard qu'une fois la valeur 0 retournée dans io_Actual. Mais comme c'est toujours la valeur de io_Length qui est transférée dans io_Actual au moment de la lecture, vous devez lire des données de 0 octet.

Si vous n'avez plus besoin du bloc, il vous suffit d'exécuter la commande CMD_CLEAR. Les données actuelles se trouvant dans le device de clipboard sont effacées, et le compteur ClipID est incrémenté d'une unité. Lors de nouveaux accès en écriture et en lecture, il vous suffira ensuite d'effacer le champ io_Offset. Il ne faut pas toucher à io_ClipID, pour des raisons que nous allons indiquer tout de suite.

En effet, si vous avez affaire à de très grands blocs, que vous désirez écrire dans le device de clipboard après les avoir convertis (par exemple en format IFF), cela peut demander beaucoup de temps. Pour aller plus vite, vous avez la possibilité d'annoncer un "clip" (c'est ainsi qu'on appelle le transfert de blocs de données dans le device de clipboard et à partir de ce device). Vous transmettez pour cela au pointeur io_Data de votre bloc de device l'adresse d'un port de message, par l'intermédiaire duquel vous obtenez un message Satisfy lorsque les données rendues ainsi disponibles sont demandées par quelqu'un, et vous envoyez alors la commande CBD_POST. Le message Satisfy présente l'aspect suivant :

Offset	Structure
-----	-----
	struct SatisfyMsg
	{
0 0x00	struct Message sm_Message;
20 0x14	UWORD sm_Unit; /* de quelle unité? */
22 0x16	LONG sm_ClipID;
26 0xa	}

Vous pouvez alors, à l'aide du Message = GetMessage(OwnPort), voir si un message est arrivé (Message !=0). Si ce n'est pas le cas, votre programme peut vaquer à d'autres tâches. Mais si un message est arrivé, vous devez écrire les données dans le device de clipboard, comme nous l'avons décrit plus haut. Il faut noter ici que le clip annoncé avec POST peut être parfois inutile. Entre-temps, on peut écrire d'autres blocs dans le device de clipboard, et les lire ensuite à partir de là. Dans ce cas, les variables io_ClipID ne sont évidemment pas les mêmes.

Si vous voulez maintenant conclure une commande POST avec une commande CMD_WRITE après avoir obtenu le message Satisfy, vous devez prendre connaissance de la variable io_ClipID de la commande de lecture actuelle avec CBD_CURRENTREADID. Si cette valeur est supérieure à celle de la variable io_ClipID du bloc de device avec lequel la commande CBD_POST a été exécutée, vos données ne sont pas nécessaires.

Si vous voulez encore tester, avant de quitter le programme, si vous devez exécuter une commande POST envoyée précédemment, il vous suffit d'interroger la variable io_ClipID de la dernière commande d'écriture avec CBD_CURRENTWRITEID, et de la comparer avec la variable ClipID du bloc de device POST. Si la valeur retournée par CURRENTWRITEID dans io_ClipID est supérieure à celle de la variable ClipID du bloc de device POST, cela veut dire qu'il y a eu entre-temps d'autres commandes CMD_WRITE exécutées, et le transfert de données annoncé n'a pas besoin d'être exécuté.

4.11. L'Amiga comme synthétiseur de sons - le device Audio

Vous avez sans doute entendu parler des capacités, que l'on peut qualifier de fantastiques, de l'Amiga. Mais pour les programmer, vous n'aviez le plus souvent qu'un seul moyen: passer par les registres du hardware. Ici aussi, un device est nécessaire: le device Audio. A l'aide de ce device, il est possible d'envoyer toutes les formes d'ondes (waveforms) imaginables, en différents volumes et fréquences (periods) par l'intermédiaire des canaux (channels) du son. Il faut pour cela ouvrir évidemment le device Audio :

```
#define AUDIO_LEN (ULONG) sizeof(struct IOAudio)
struct IOAudio *Audio_Request=0L;
...
    Open_A_Device("audio.device", 0L, Audio_Request, 0L, AUDIO_LEN);
...
```

Le bloc de device, par l'intermédiaire duquel toutes les commandes sont indiquées et exécutées, s'appelle IOAudio, et il se présente ainsi :

```
Offsets      struct IOAudio
-----  /* défini dans "devices/audio.h" */
0   0x00      struct IORequest  ioa_Request; /* IORequest au début */
32  0x20      WORD            ioa_AllocKey;
34  0x22      UBYTE           *ioa_Data; /* Pointeur des données*/
38  0x26      ULONG           ioa_Length; /* Taille du champ des
données */
42  0x2a      WORD            ioa_Period; /* Fréquence */
44  0x2c      WORD            ioa_Volume; /* Volume */
46  0x2e      WORD            ioa_Cycles; /* Cycles */
48  0x30      struct Message ioa_WriteMessage;
62  0x3d      };
```

Voici le tableau des commandes et des flags :

Commandes :

ADCMD_ALLOCATE	(32) Allocation des canaux Sound
ADCMD_FINISH	(11) Mettre fin à la sortie du son
ADCMD_FREE	(9) Libérer Lock
ADCMD_LOCK	(13) Poser Lock
ADCMD_PERVOL	(12) Définir Fréquence & Volume
ADCMD_SETPREC	(10) Définir Precedence
ADCMD_WAITCYCLE	(14) Attendre la fin du cycle
CMD_FLUSH	(8) Supprimer toutes les commandes WRITE
CMD_READ8	(2) Obtenir le Write-IO-Block actuel
CMD_RESET	(1) Reset de la machine Audio-State
CMD_START	(7) Lancer la sortie
CMD_STOP	(6) Arrêter la sortie
CMD_WRITE	(3) Débuter la sortie du son

Flags:

ADIOF_PERVOL	(16) Définir Period et Volume dans ADCMD_ALLOCATE
ADIOF_SYNCCYCLE	(32) Synchroniser les actions Cycles
ADIOF_NOWAIT	(64) Avec ADCMD_ALLOCATE ne pas attendre

Errors:

AUDIOERR_NOALLOCATION (-10)	Surgit lorsque AllocKey n'est pas exact
AUDIOERR_ALLOCFAILED (-11)	Canal >Allocation erroné
AUDIOERR_CHANNELSTOLEN (-12)	La canal a été "volé"

4.11.1. Définir les canaux Audio

Il existe deux moyens pour allouer les canaux Audio, par lesquels le son doit sortir. Puisque d'autres programmes (Multitasking) peuvent également produire des sons, et donc utiliser les canaux de son que vous désirez utiliser vous-même, ces canaux Audio doivent être réservés. Le premier moyen de s'assurer l'accès à un canal Audio est de passer par OpenDevice(). Ici, dès l'appel de OpenDevice() (fonction Exec), on a une tentative d'allouer les canaux. Il faut pour cela initialiser toutefois à l'avance la structure IOAudio (le bloc I/O du device Audio). L'initialisation par CreateExtIO() ne suffit pas dans le cas présent. Vous devez en effet transmettre à ce bloc de device Audio l'adresse de votre "channel allocation map", c'est-à-dire de votre masque de réservation de canal Audio :

```
struct IOAudio *AudioReq;
char Channel_Map[...] = {...};

...
    AudioReq = (struct IOAudio *) GetDeviceBlock(AUDIO_LEN);
    AudioReq->ioa_Data = Channel_Map;
    AudioReq->ioa_Length = sizeof(Channel_Map);
    Open_A_Device("audio.device", OL, AudioReq, OL, AUDIO_LEN);
```

Un tel masque d'allocation comprend jusqu'à 16 octets, à l'aide desquels vous pouvez définir les canaux concernés. Les quatre bits inférieurs (le "low-nibble") de chaque octet définissent plus précisément les canaux à réserver. Si vous désirez donc envoyer vos formes d'onde sur un canal de son à gauche et à droite, votre masque d'allocation se présentera ainsi :

```
#define Left_Channel_0 1 /* Bit pour le premier canal de gauche */
#define Right_Channel_1 2 /* Bit pour le premier canal de droite */
#define Right_Channel_2 4 /* Bit pour le second canal de droite */
#define Left_Channel_3 8 /* Bit pour le second canal de gauche */
UBYTE Channel_Map[] = {Left_Channel_0 | Right_Channel_1,
                      Left_Channel_0 | Right_Channel_2,
                      Left_Channel_3 | Right_Channel_1,
                      Left_Channel_3 | Right_Channel_2};
```

Le tableau Channel_Map contient alors quatre entrées (sizeof(Channel_Map) == 4), qui déterminent chacun un canal de son à gauche et à droite pour la réservation. Ce tableau sera transmis alors à la structure IOAudio (pointeur ioa_Data) à initialiser. Le nombre d'entrées dans ce tableau est indiqué par l'intermédiaire du pointeur ioa_Length. Dans

le cas présent, la longueur est de 4 octets. Pour l'allocation des canaux par OpenDevice(), on aurait besoin de la séquence suivante :

```
#define Left_Channel_0 1
#define Right_Channel_1 2
#define Right_Channel_2 4
#define Left_Channel_3 8
#define Precedence -40
#define AUDIO_LEN sizeof(struct IOAudio)
UBYTE Channel_Map[] = {Left_Channel_0 | Right_Channel_1,
                      Left_Channel_0 | Right_Channel_2,
                      Left_Channel_3 | Right_Channel_1,
                      Left_Channel_3 | Right_Channel_2};

struct IOAudio *AudioReq;
...
    AudioReq = (struct IOAudio *) GetDeviceBlock(AUDIO_LEN);
    AudioReq->ioa_Data = (UBYTE*)Channel_Map;
    AudioReq->ioa_Length = (ULONG)sizeof(Channel_Map); /* 4 */
    AudioReq->ioa_Request.io_Message.mn_Node.ln_pri = Precedence;
    Open_A_Device("audio.device", 0L, &AudioReq, 0L, 0L);
```

Puisque nous avons créé nous-même le bloc de device Audio au moyen de GetDevice Block, on n'a plus besoin de le faire par Open_A_Device() (devssupport). Pour cette raison, on transmet aussi la valeur 0 à Open_A_Device(), pour la taille du bloc de device Audio. On dispose alors des canaux Audio demandés (à condition que l'on ait bien Our-Sounds->ioa_Request.io_Error == 0). Si vous êtes un lecteur attentif, vous êtes sans doute déjà demandé quelle peut bien être la fonction de la variable :

```
AudioReq->ioaRequest.io_Message.mn_Node.ln_Pri
```

Ce paramètre indique l'urgence de vos sons. Plus le niveau d'urgence (la priorité) est grande, et plus il est improbable que votre canal vous soit "volé". Cela veut dire que si un programme tente de réserver vos canaux Audio, cette tentative sera rejetée. Mais si l'autre programme a une priorité supérieure au vôtre, vous devez libérer vos canaux aussi vite que possible. Commodore a émis des recommandations, définissant les relations entre sons et priorités :

Priorité :

- 128** Ce niveau de priorité est prévu pour les programmeurs paresseux. Si vous lancez en effet vos sons au niveau 127, plus personne ne peut accéder à vos canaux réservés. Vous n'avez plus besoin de cette façon de libérer vos canaux avant la fin du programme. Mais par pitié, faites-en usage le moins possible!
- 90-100** Sounds en urgence. Si quelque chose ne va pas dans votre ordinateur (par exemple si une librairie n'a pas pu être ouverte), vous pouvez demander un signal sonore sur ce niveau.
- 80-90** Cas moins urgents.
- 75** C'est le niveau "Precedence" du device Narrator.

- 50-70** Pour les sons destinés à attirer votre attention sur quelque chose qui est encore invisible (par exemple messages radar dans les jeux).
- 50-50** Programme musical
- 70-0** Effets sonores (explosions, etc.)
- 100- -80** Musique de fond
- 127** Pourquoi réserver des canaux? A ce niveau, n'importe quel autre Request aura la priorité sur vous.

Nous avons dit au début qu'il y avait deux moyens de réserver des canaux Audio. Le second moyen passe par une commande de device Audio. Nous avons déjà revêtu la commande (ADCMD_ALLOCATE) de ses atours, constitués par une fonction support de device Audio

```
*****
*          Audio_Allocate()      (Audio_Support)   *
*
* Fonction: Allouer des canaux son ou préparer le bloc Audio-Device   *
*           pour l'allocation (OpenDevice)                                *
*-----*
* Paramètres d'entrée:                                                 *
*   *
* Audio_Device_Block: Bloc de device pour la définition               *
* Channel_Map:             Masque de définition du canal            *
* Size:                  Taille du masque d'allocation (OCTETS)    *
* Precedence:             Priorité du son (-127 - 128)                *
* Wait:                  Attendre la libération des canaux           *
*                      souhaités?                                     *
*-----*
* Valeur en retour:                                                 *
*   *
* Erreur dans l'exécution de la commande                               *
*****/
BYTE Audio_Allocate;(Audio_Device_Block, Channel_Map, Size, Precedence,
Wait)
struct IOAudio      *Audio_Device_Block;
UBYTE               *Channel_Map;
ULONG               Size;
BYTE                Precedence;
BOOL               Wait;
{
  Audio_Device_Block->ioa_Data           = Channel_Map;
  Audio_Device_Block->ioa_Length         = Size;
  Audio_Device_Block->ioa_Request.io_Message.mn_Node.ln_Pri = -
Precedence;
  if (!Wait) /* Attendre que les canaux soient libérés ? */
    Audio_Device_Block->ioa_Request.io_Flags |= (UBYTE) ADIOF_NOWAIT;
  if (Audio_Device_Block->ioa_Request.io_Device != OL)
  {
    Audio_Device_Block->ioa_Request.io_Command = (UWORD) ADCMD_ALLOCATE;
    if (Wait) DoIO(Audio_Device_Block);
    else
    {
      SendIO(Audio_Device_Block);
    }
  }
}
```

```

        if (CheckIO(Audio_Device_Block) == 0)
            return(ADIOERR_ALLOCFAILED);
    }
    return(Audio_Device_Block->ioa_Request.io_Error);
}
return(0x00);
}

```

Cette routine place également le pointeur ioa_Data (Channel_Map) et la variable ioa_Length (taille de Channel_Map), ainsi que la "sound-precedence" sur les valeurs que vous indiquez. De plus, elle pose le flag NOWAIT, si vous le demandez. Dans le cas où le flag est posé (WAIT = FALSE), si vous essayez de réserver des canaux déjà retenus par un Request de priorité supérieure, vous obtenez en retour pour la fonction OpenDevice() et pour ADCMD_ALLOCATE les valeurs d'erreur ADIOERR_ALLOCFAILED et un flag d'erreur (AudioReq->ioa_Request.io_Error).

Si en revanche le flag NOWAIT n'est pas posé, la routine attend jusqu'à ce que le Request qui occupe les canaux demandés les libère. Cela pose cependant un petit problème, lorsqu'on veut utiliser le flag NOWAIT en relation avec ADCMD_ALLOCATE, et non pas pour l'allocation des canaux avec OpenDevice(). On dit, il est vrai, qu'avec le flag NOWAIT posé, ADCMD_ALLOCATE revient aussitôt au programme appelant si les canaux demandés ne sont pas accessibles, mais ADCMD_ALLOCATE attend malheureusement toujours que les canaux soient libérés.

Nous avons corrigé cette petite erreur à l'aide des commandes SendIO() et CheckIO(). Si on ne veut pas attendre que les canaux soient libérés, il suffit en effet d'envoyer un Request asynchrone (SendIO()), et d'examiner directement si la commande envoyée est encore en cours d'exécution, ou si elle a déjà été traitée. Si CheckIO() renvoie la valeur 0, cela veut dire que les canaux sont occupés par quelqu'un d'autre. Mais comme nous ne voulons pas attendre qu'ils soient libérés, nous quittons aussitôt la routine avec l'erreur ADIOERR_ALLOCFAILED.

Si au contraire les canaux ont pu être alloués, on obtient dans *Audio_Device_Block->ioa_Request.io_Unit la combinaison de bits des canaux effectivement alloués. La connaissance des canaux alloués se révèlera nécessaire ultérieurement. Lorsqu'il y a diverses indications pour les canaux à réserver dans le Channel_Map (jusqu'à 16), ADCMD_ALLOCATE cherche toujours à prendre en compte les combinaisons qui demanderont le moins de temps d'attente.

Si entre-temps un programme quelconque libère les canaux Audio qu'il occupe, ADCM_ALLOCATE ou OpenDevice() vérifient si une réservation peut être réalisée. Cette vérification ne s'effectue évidemment que si l'on attend une libération de canaux. Si la réservation n'a pas réussi, on le reconnaît au fait que la variable Error de Audio Device Block pose le flag ALLOC FAILED :

```
*****  
*                               Audio_NoAlloc()          (Audio_Support)*  
*  
* Fonction: Allocation réussie?  
*-----  
* Paramètres d'entrée:  
*
```

```
* Audio_Device_Bloc : Bloc de device à tester
*-----*
* Valeur en retour:
*
* TRUE: pas d'allocation
* FALSE: allocation réussie
*****
```

```
BOOL Audio_NoAlloc;(Audio_Device_Bloc)
struct IOAudio      *Audio_Device_Bloc;
{
    if ((Audio_Device_Bloc->ioa_Request.io_Error&AUDIOERR_NOALLOCATION) ==
        AUDIOERR_NOALLOCATION)
        return(TRUE);
    else
        return(FALSE);
}
```

Comment le device reconnaît-il cependant plus tard qu'un programme peut utiliser des canaux déterminés? Il existe pour cela AllocKey, une variable contenant un nombre. Ce nombre indique votre numéro d'utilisateur du device Audio depuis que l'ordinateur est allumé (ou a subi un reset). Si cet AllocKey, lors de l'exécution d'une commande Audio-Device, ne coïncide pas avec la valeur conservée de manière interne, toutes les commandes du device Audio annoncent une erreur NOALLOCATION.

Pour que cette erreur ne surgisse pas lorsque vous voulez utiliser une copie du bloc de device original, cet AllocKey est copié également lorsque l'on copie un bloc de device Audio. Avant d'en venir là, nous allons nous occuper quelque peu de l'allocation. En effet, vous pouvez économiser encore du travail dans la définition par OpenDevice(). Lorsque le flag ADIOF_PERVOL a été posé, la fréquence (Period) et le volume (Volume) sont également définis. Evidemment, on peut aussi le faire à tout moment, à l'aide d'une commande Audio.

4.11.2. Définir le volume

Dans la routine suivante, on a intégré la commande de device Audio servant à définir le volume :

```
*****  
*                               Audio_Pervol()          (Audio_Support)*  
*-----*  
* Fonction: Définir Volume et Fréquence  
*-----*  
* Paramètres d'entrée:  
*-----*  
* Audio_Device_Bloc : Bloc de device Audio, pour définir Volume*  
*                   et Fréquence  
* Period:           Sample Fréquence  
* Volume:          Volume  
*-----*  
* Valeur en retour:  
*-----*  
* Erreur lors d'une exécution éventuelle de la commande  
*****
```

```

BYTE Audio_Pervol;(Audio_Device_Bloc, Period, Volume)
struct IOAudio      *Audio_Device_Bloc;
UWORD                           Period;
UWORD                           Volume;
{
    Audio_Device_Bloc->ioa_Request.io_Flags     = (UBYTE) ADIOF_SYNCYCLE;
    Audio_Device_Bloc->ioa_Period               = Period;
    Audio_Device_Bloc->ioa_Volume                = Volume;
    if (Audio_Device_Bloc->ioa_Request.io_Device != 0L)
        /* Définition ultérieure de Fréquence et Volume */
        Audio_Device_Bloc->ioa_Request.io_Command = (UWORD) ADCMD_PERVOL;
        DoIO(Audio_Device_Bloc);
        return(Audio_Device_Bloc->ioa_Request.io_Error);
    }
    Audio_Device_Bloc->ioa_Request.io_Flags |= ADIOF_PERVOL;
    /* ADIOF_PERVOL doit être défini pour OpenDevice() */
    return(0x00);
}

```

Cette routine est en mesure de remarquer si le volume doit être défini avant ou après OpenDevice(). Avant OpenDevice(), le pointeur est en effet sur 0, ainsi que toutes les variables non définies, car la réservation du bloc de device Audio (GetDeviceBloc()) a été exécutée par AllocMem(), avec les exigences MEMF_PUBLIC et MEMF_CLEAR.

Mais il existe encore une troisième méthode pour définir le volume et la fréquence. Le flag ADIOF_PERVOL en relation avec CMD_WRITE (sortie du son) a le même effet qu'en relation avec OpenDevice(). Mais avant d'en venir à la sortie du son proprement dite, nous allons encore donner quelques explications sur la fréquence et le volume.

Nous avons déjà indiqué que Period détermine la fréquence. Cela ne veut pas dire qu'une valeur de Period égale par exemple à 20000 provoque une sortie de 20000 octets par seconde. Il faut malheureusement se livrer ici à quelques calculs, pas très compliqués cependant. A partir du nombre des octets (samples) qui déterminent la forme des ondes, et de la fréquence avec laquelle cette forme d'onde doit être entendue, on peut calculer la période :

$$\text{Période} = \frac{1}{\text{Sampling-Rate} * 28 * 10^{-8}}$$

où la vitesse "sampling-rate" est constituée du nombre d'octets et de la fréquence. Un exemple: vous voulez sortir une forme d'onde composée de 40 octets 440 fois par seconde (toute la forme d'onde avec 440 Hz). La vitesse de sampling sera calculée ainsi :

$$\text{Sampling-rate} = 40 * 440 \text{ (40 octets * 440 Hz)}$$

La période sera donc la suivante :

$$\text{Period} = \frac{1}{40 * 440 * 28 * 10^{-8}} = 202,922 = 203$$

Pour le calcul de la période également, nous allons présenter ici une routine. Puisque nous voulons renoncer à l'arithmétique en virgule flottante, nous avons multiplié la fraction écrite ci-dessus par 10^8 ($28 * 10^8 = 280$ ns, ce qui représente le temps qu'il faut pour sortir un octet), et nous obtenons donc la routine qui suit :

```
*****  
*          Audio_Period()      (Audio_Support)*  
*  
* Fonction: Calculer de Period à partir du nombre des octets      *  
*           à jouer et de la fréquence disponible à cet effet      *  
*-----  
* Paramètres d'entrée:  
*  
* Bytes: Nombre des octets à jour  
* Hz:   Fréquence, avec laquelle les Bytes Samples doivent  
*       être joués  
*-----  
* Valeur en retour:  
*  
* Période calculée  
*****
```

UWORD Period(Bytes, Hz)
UWORD Bytes, Hz;
{
 return((UWORD) (100000000L / (Bytes * Hz * (UWORD)28)));
}

Notez bien que cette routine ne retourne que des variables du type UWORD. Le domaine des valeurs va donc de 0 à 65535. Si la période est supérieure à 65535, les 16 bits supérieurs du calcul seront coupés (ceci est exécuté avec des mots longs), et seuls les 16 bits inférieurs seront utilisés (cf. les registres du hardware), ce qui risque parfois de conduire à des valeurs très faibles, en dehors du domaine autorisé. La période ne doit en effet pas être inférieure à 124. La fréquence de sortie des octets se calcule alors ainsi :

$$\begin{aligned} \text{Period} &= 100000000L / (\text{Bytes} * \text{Hz} * 28) \\ \leftrightarrow \text{Hz} &= 100000000L / (\text{Bytes} * \text{Period} * 28) \end{aligned}$$

En supposant que notre forme d'onde ne comprend qu'un octet, on obtient pour la fréquence la plus élevée :

$$\text{Hz} = 100000000L / (1 * 124 * 28) = 28800 \text{ s}^{-1};$$

Pour définir le volume, nous aurons besoin de très peu de mathématiques. Il suffit de savoir que le volume peut aller de 0 (faible) à 64 (fort). De l'un à l'autre, la croissance du volume est linéaire. Ajoutons un petit conseil en passant: avant de sortir pour la première fois des données avec CMD_WRITE, définissez la période et le volume sur 0.

En effet, il peut arriver que vous entendiez des "blips" - un grincement bref, du fait que les canaux DMA s'en vont un peu au début dans tous les sens, et que tout ce qui est lu pendant ce bref laps de temps est envoyé dans les haut-parleurs. Si le volume est égal à 0, vous ne l'entendrez pas. Mais n'oubliez pas de monter le volume tout de suite après avec la commande CMD_WRITE.

4.11.3. Enfin, on entend quelque chose

Venons-en maintenant à la commande CMD_WRITE, dont nous avons si souvent parlé. Nous l'avons également habillée d'une petite routine :

```

*****
*                               Audio_Write()      (Audio_Support)*
*
* Fonction: Sortir les données sur la canal du son
*-----*
* Paramètres d'entrée:
*
* Audio_Device_Bloc: Bloc de device
* WaveForm:           Adresse du tableau de la forme d'onde
* WaveLength:         Nombre d'octets à jouer
* Cycles:            Nombre des répétitions de formes d'onde
* ComeBack:          Attendre la fin de CMD_WRITE (FALSE)?
*-----*
* Valeur en retour:
*
* Erreur lors de l'exécution de la commande
*****
BYTE Audio_Write(Audio_Device_Bloc, WaveForm, WaveLength, Cycles,
                 ComeBack)
struct IOAudio *Audio_Device_Bloc;
UBYTE             *WaveForm;
ULONG            WaveLength;
UWORD            Cycles;
BOOL             ComeBack;
{
    Audio_Device_Bloc->ioa_Data           = WaveForm;
    Audio_Device_Bloc->ioa_Length        = WaveLength;
    Audio_Device_Bloc->ioa_Cycles        = Cycles;
    Audio_Device_Bloc->ioa_Request.io_Flags = (UBYTE) 0;
    Audio_Device_Bloc->ioa_Request.io_Command = (UWORD) CMD_WRITE;
    if (ComeBack) SendIO(Audio_Device_Bloc);
    else          DoIO(Audio_Device_Bloc);
    return(Audio_Device_Bloc->ioa_Request.io_Error);
}

```

Il est clair qu'ici encore, comme pour presque toutes les routines Audio_Support, nous devons transmettre le bloc de device avec lequel le device a été ouvert. Mais que signifient les autres paramètres?

WaveForme et WaveLength déterminent la forme d'onde à sortir. WaveForm contient l'adresse de début du tableau BYTE, dans lequel la forme d'onde est déposée. WaveLength ne contient pas la longueur d'onde du résultat audible, mais le nombre d'octets nécessaires pour la description de la forme d'onde.

Cycles détermine le nombre de répétitions d'une forme d'onde. Si par exemple vous voulez que la forme d'onde indiquée ne soit jouée qu'une fois, vous devez donner à Cycles la valeur 1. Pour plusieurs répétitions, vous devez évidemment indiquer le nombre correspondant.

Si vous désirez toutefois obtenir une répétition ininterrompue, il faut donner à Cycles la valeur 0. Dans ce cas, à condition que vos canaux ne soient pas "volés" entre-temps, votre forme d'onde sera répétée ad vitam aeternam. En outre, toute modification dans le tableau de la forme d'onde est tout de suite audible, au moment de la sortie de cet emplacement modifié. Notez que les données Sound doivent être déposées dans la mémoire Chip. Comme vous le savez, le bus d'adresses des customs-chips (Blitter, Paula,

Agnus, Denise, Copper) ne contient que 19 canaux d'adresses. Il ne permet donc d'adresser que la partie inférieure de la mémoire (a peu près les 512 Ko inférieurs).

4.11.4. Autres fonctionnalités du device Audio

Vous êtes maintenant capable de produire des sons sur votre Amiga. Vous pouvez réserver les canaux nécessaires, définir la fréquence et le volume, et rendre audibles les données. Mais imaginez que vous travailliez avec votre device Audio, et que soudain vous n'entendiez plus rien, ou du moins pas ce que vous attendiez. Vous vous demanderez ce qui se passe. L'événement est facile à interpréter: on vous a dérobé vos canaux.

Que faut-il faire? Vous pouvez quitter votre programme ou effectuer un reset, s'il n'est pas possible de faire autrement. Il existe cependant un moyen de savoir si les canaux ont été dérobés, à l'aide d'un programme, et non pas seulement à l'oreille. Ce moyen est en relation avec la commande ADCMD_LOCK :

```
*****
*          Audio_Lock()      (Audio_Support)*
*
* Fonction: Protéger le canal contre un nouvel accès
*-----*
* Paramètres d'entrée:
*-----*
* Audio_Device_Bloc: Bloc de device des canaux à protéger
*-----*
* Valeur en retour:
*-----*
* Adresse du Lock
*****
struct IOAudio *Audio_Lock(Audio_Device_Bloc)
struct IOAudio           *Audio_Device_Bloc;
{
    struct IOAudio *Lock;
    Lock = (struct IOAudio *)GetDeviceBloc(AUDIO_LEN);
    Audio_Copy(Audio_Device_Bloc, Lock);
    Lock->ioa_Request.io_Command = (UWORD)ADCMD_LOCK;
    SendIO(Lock);
    return(Lock);
}
```

ADCMD_LOCK est une commande exécutée uniquement lorsque les canaux réservés ont été "volés". Cela veut dire que tout va bien tant que ADCMD_LOCK est exécuté. Si vous voulez utiliser vos canaux Audio, il vaut mieux tester à intervalles réguliers si le droit d'accès est encore à vous. La petite routine qui suit vous aidera dans ce sens :

```
*****
*          Audio_Channel_Stolen()  (Audio_Support)*
*
* Fonction: Le canal a-t-il été "volé?"
*-----*
* Paramètres d'entrée:
*-----*
```

```

* Lock: Bloc de device du Lock, devant être testé      *
*-----*                                           *
* Valeur en retour:                                *
*                                               *
* TRUE: Le canal a été "volé" -> Quitter le programme   *
* FALSE: Canal encore sous votre contrôle           *
*****                                                 */
BOOL Audio_Channel_Stolen(Lock)
struct IOAudio          *Lock;
{
    if (CheckIO(Lock) != 0)
        return(TRUE);
    else
        return(FALSE);
}

```

Cette petite routine teste si la commande LOCK a été envoyée (CheckIO() !=0), ou si le droit d'accès aux canaux réservés est toujours à vous. Vous penserez sans doute: comment puis-je sortir des données si j'ai bloqué les canaux Audio, pour que la commande ADCMD_LOCK prenne fin lorsque les canaux sont volés, et pour empêcher le traitement de deux commandes de device avec un seul bloc de device?

Et vous aurez raison! Il faut un second bloc de device. On le crée comme tous les blocs de device avec GetDeviceBloc(). Il faut ensuite s'assurer que l'on peut utiliser le nouveau bloc de device. Nous avons pour cela développé une fonction de copie, qui copie également le AllocKey, nécessaire pour identifier l'utilisateur dans le device Audio :

```

*****                                                 */
*                               Audio_Copy()          (Audio_Support)*
*-----*                                           *
* Fonction: Copier le bloc de device                *
*-----*                                           *
* Paramètres d'entrée:                            *
*                                               *
* Old_Audio_Bloc: Original                      *
* New_Audio_Bloc: Copie                         *
*****                                                 */
VOID Audio_Copy(Old_Audio_Bloc, New_Audio_Bloc)
struct IOAudio *Old_Audio_Bloc,*New_Audio_Bloc;
{
    New_Audio_Bloc->ioa_Request.io_Device =
        Old_Audio_Bloc->ioa_Request.io_Device;
    New_Audio_Bloc->ioa_Request.io_Unit =
        Old_Audio_Bloc->ioa_Request.io_Unit;
    New_Audio_Bloc->ioa_Allockey =
        Old_Audio_Bloc->ioa_Allockey;
}

```

Que se passe-t-il lorsque vous voulez quitter ce programme? Vous ne pouvez tout de même pas laisser les canaux verrouillés tels quels. Cela empêcherait toute sortie de son ultérieure, car seules les demandes de canal de "precedence" (de priorité) moindre seraient mises en oeuvre. Pour empêcher cela, vous devez libérer les canaux verrouillés. Le mieux est de le faire à l'aide de la routine suivante :

```

*****                                                 */
*                               Audio_Free()          (Audio_Support)*
*
```

```

*
* Fonction: Supprimer la protection pour les canaux
*
* Paramètres d'entrée:
*
* Lock: Bloc de device du Lock à libérer
*****
VOID Audio_Free(Lock);
struct IOAudio *Lock;
{
    Do_Command(Lock, (UWORD)ADCMD_FREE);
    FreeDeviceBloc(Lock);
}

```

Cette routine libère les canaux de votre accès en exclusivité, et libère également le bloc de device précédemment réservé. Outre la méthode utilisant Lock, il existe toutefois une autre solution - pas très élégante - pour se prémunir contre les accès d'autres utilisateurs: on place simplement la priorité sur 127. On peut le faire lors de la réservation, mais aussi en utilisant la commande Audio ADCMD_SETPREC.

```

*****
*           Audio_SetPrecedence()          (Audio_Support)*
*
* Fonction: Modifier ultérieurement la priorité
*
* Paramètres d'entrée:
*
* Audio_Device_Bloc: Bloc de device
* Precedence: Nouvelle priorité
*****
VOID Audio_SetPrecedence(Audio_Device_Bloc, Precedence)
struct IOAudio           *Audio_Device_Bloc;
BYTE                      Precedence;
{
    Audio_Device_Bloc->ioa_Request.io_Message.mn_Node.ln_Pri =
(BYTE)Precedence;
    Do_Command(Audio_Device_Bloc, (UWORD)ADCMD_SETPREC);
}

```

Chaque appel de ADCMD_SETPREC a en outre pour effet de tester les commandes ALLOCATE en attente, pour voir si la priorité nouvellement définie n'est pas inférieure à la leur. Si c'est le cas, ALLOCATE peut demander à occuper les canaux en question.

Si l'on élève la priorité à 127, la priorité la plus élevée, il n'y a plus aucune chance pour que les autres commandes ALLOCATE puissent avoir accès à vos canaux, puisque la priorité des canaux susceptibles d'être "volés" doit être strictement inférieure. Avec ces commandes, nous pouvons écrire des programmes Sound qui nous sont propres. Voici des exemples de programmes :

```

*****
*           SoundEditor.c               *
*           (c) Bruno Jennrich          *
*           Août 1988                  *
*****
/*****
* Compile-Info:                         *

```

```

*
* cc SoundEditor
* ln SoundEditor.o Audio_Support.o Devs_Support.o -lc
*****
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "intuition/intuition.h"
#include "intuition/intuitionbase.h"
#include "graphics/gfxbase.h"
#include "graphics/gfxmacros.h"
#include "devices/audio.h"
#define ScreenHeight 255L           /* Editor-Screen */
#define ScreenWidth 320L
#define ScreenDepth 2L
#define ScreenMode 0L
#define ERROR 100
#define AUDIO_LEN (ULONG)(sizeof (struct IOAudio))
VOID *OpenLibrary();
VOID *OpenScreen();
VOID *OpenWindow();
VOID *AllocMem();
VOID *GetDeviceBloc();
VOID *Audio_Lock();
struct Screen *Screen=0L;
struct Window *Window=0L;
struct NewScreen NewScreen;
struct NewWindow NewWindow;
struct GfxBase      *GfxBase=0L;
struct IntuitionBase *IntuitionBase=0L;
/* Structures concernant le device Audio */
#define Left_Channel_0 1
#define Right_Channel_1 2
#define Right_Channel_2 4
#define Left_Channel_3 8
#define SoundPrecedence (BYTE) -40
struct IOAudio *Left_Side      =0L,
               *Right_Side     =0L,
               *Left_Lock       =0L,
               *Right_Lock      =0L;
BYTE *WaveLeft   = 0L; /* Définition de la forme d'onde (signed) */
BYTE *WaveRight  = 0L;
UBYTE  Left_Channels[] = {Left_Channel_0, Left_Channel_3};
UBYTE  Right_Channels[] = {Right_Channel_1, Right_Channel_2};
               /* Channel_Map */
               /* Allouer un canal quelconque*/
               /* à gauche ou à droite */
#define CHANNELS_LEFT  (ULONG) sizeof( Left_Channels)
#define CHANNELS_RIGHT (ULONG) sizeof(Right_Channels)

*****
*                                     CloseIt()          (User)*
*
* Fonction: Fermer et libérer le tout
*-----
* Paramètres d'entrée:
*
* String: Error-String
*****

```

```

VOID CloseIt(String)
char      *String;
{
    WORD i;
    WORD *dff180 = (WORD *)0xdff180;
    WORD Error;
    if (strlen(String) > 0)
        for(i=0;i<0xffff;i++) *dff180 = i;
    puts(String);
    if (Window != OL)      CloseWindow(Window);
    if (Screen != OL)     CloseScreen(Screen);
    if (GfxBase != OL)    CloseLibrary(GfxBase);
    if (IntuitionBase != OL) CloseLibrary(IntuitionBase);
    if (Left_Lock != OL)   Audio_Free(Left_Lock);
    if (Right_Lock != OL)  Audio_Free(Right_Lock);
    if (Left_Side != OL)   Close_A_Device(Left_Side);

    if (Right_Side != OL)  FreeDeviceBloc(Right_Side);
    if (WaveLeft != OL)   FreeMem(WaveLeft, (ULONG)ScreenWidth);
    if (WaveRight != OL)  FreeMem(WaveRight, (ULONG)ScreenWidth);
    exit(10);
}
/*********************************************
*                         InstallScreenWindow()          (User)*
*
* Fonction: Initialiser Editor Window et Screen
*****************************************/
VOID InstallScreenWindow()
{
    NewScreen.LeftEdge    = 0;
    NewScreen.TopEdge     = 0;
    NewScreen.Width       = ScreenWidth;
    NewScreen.Height      = ScreenHeight;
    NewScreen.Depth       = ScreenDepth;
    NewScreen.DetailPen   = 1;
    NewScreen.BlocPen     = 0;
    NewScreen.ViewModes   = ScreenMode;
    NewScreen.Type        = CUSTOMSCREEN;
    NewScreen.Font         = (struct TextAttr *) OL;
    NewScreen.DefaultTitle = (UBYTE *) " (c) Bruno Jennrich";
    NewScreen.Gadgets      = (struct Gadget *) OL;
    NewScreen.CustomBitMap = (struct BitMap *) OL;
    NewWindow.LeftEdge    = 0;
    NewWindow.TopEdge     = 0;
    NewWindow.Width       = ScreenWidth;
    NewWindow.Height      = ScreenHeight;
    NewWindow.DetailPen   = 1;
    NewWindow.BlocPen     = 0;
    NewWindow.IDCMPFlags  = BORDERLESS | ACTIVATE | RMBTRAP | NOCAREREFLASH;
    NewWindow.FirstGadget = (struct Gadget *) OL;
    NewWindow.CheckMark   = (struct Image *) OL;
    NewWindow.Title        = (UBYTE *) " Waveform-Editor";
    NewWindow.Screen       = (struct Screen *) OL;
    NewWindow.BitMap       = (struct BitMap *) OL;
    NewWindow.MinWidth    = 0;
    NewWindow.MaxWidth    = 0;
}

```



```

{
    WORD i;
    ULONG x, y;
    UBYTE *LeftMouse = (UBYTE *) 0xbfe001;
    UWORLD *RightMouse = (UWORD *) 0xdff016;
    Move(Window->RPort, 0L, ScreenHeight/2L);
    Draw(Window->RPort, ScreenWidth, ScreenHeight/2L);
    for(i=0;i<ScreenWidth; i++)
    {
        Left_Channels,
        CHANNELS_RIGHT,
        SoundPrecedence,
        FALSE) == ADIOERR_ALLOCFAILED)
        CloseIt("Couldnt get Right-Channels !");
    if (Audio_Allocate(Left_Side,
        Left_Channels,
        CHANNELS_LEFT,
        SoundPrecedence,
        FALSE) == ADIOERR_ALLOCFAILED)
        CloseIt("Couldnt get Left-Channels !");
    Left_Lock = Audio_Lock(Left_Side);
    Right_Lock = Audio_Lock(Right_Side);
    if (Right_Lock == 0L) CloseIt("Right_Lock failed !");
    if (Left_Lock == 0L) CloseIt("Left_Lock failed !");
    Audio_Pervol(Left_Side, (UWORD)0, (UWORD)0);
    Audio_Pervol(Right_Side, (UWORD)0, (UWORD)0);
    Audio_Write(Left_Side, WaveLeft, ScreenWidth,
                (UWORD) 0, (BOOL) TRUE);
    Audio_Write(Right_Side, WaveRight, ScreenWidth,
                (UWORD) 0, (BOOL) TRUE);
    Audio_Pervol(Right_Side, (UWORD)1500, (UWORD)64);
    Audio_Pervol(Left_Side, (UWORD)1500, (UWORD)64);
}
/*********************************************
*           Close_Audio_Device()          (User)*
*
* Fonction: Libérer les canaux et fermer le device Audio      *
*****************************************/
Close_Audio_Device()
{
    Audio_Free(Left_Lock);
    Audio_Free(Right_Lock);
    Close_A_Device(Left_Side);
    FreeDeviceBloc(Right_Side);
}
/*********************************************
*           Edit()                      (User)*
*
* Fonction: Editer la forme d'onde      *
*****************************************/
Edit ()
{
    WORD i;
    ULONG x, y;
    UBYTE *LeftMouse = (UBYTE *) 0xbfe001;
    UWORLD *RightMouse = (UWORD *) 0xdff016;
    Move(Window->RPort, 0L, ScreenHeight/2L);
    Draw(Window->RPort, ScreenWidth, ScreenHeight/2L);
    for(i=0;i<ScreenWidth; i++)
    {
        Left_Channels,
        CHANNELS_RIGHT,
        SoundPrecedence,
        FALSE) == ADIOERR_ALLOCFAILED)
        CloseIt("Couldnt get Right-Channels !");
    if (Audio_Allocate(Left_Side,
        Left_Channels,
        CHANNELS_LEFT,
        SoundPrecedence,
        FALSE) == ADIOERR_ALLOCFAILED)
        CloseIt("Couldnt get Left-Channels !");
    Left_Lock = Audio_Lock(Left_Side);
    Right_Lock = Audio_Lock(Right_Side);
    if (Right_Lock == 0L) CloseIt("Right_Lock failed !");
    if (Left_Lock == 0L) CloseIt("Left_Lock failed !");
    Audio_Pervol(Left_Side, (UWORD)0, (UWORD)0);
    Audio_Pervol(Right_Side, (UWORD)0, (UWORD)0);
    Audio_Write(Left_Side, WaveLeft, ScreenWidth,
                (UWORD) 0, (BOOL) TRUE);
    Audio_Write(Right_Side, WaveRight, ScreenWidth,
                (UWORD) 0, (BOOL) TRUE);
    Audio_Pervol(Right_Side, (UWORD)1500, (UWORD)64);
    Audio_Pervol(Left_Side, (UWORD)1500, (UWORD)64);
}
/*********************************************
*           Close_Audio_Device()          (User)*
*
* Fonction: Libérer les canaux et fermer le device Audio      *
*****************************************/
Close_Audio_Device()
{
    Audio_Free(Left_Lock);
    Audio_Free(Right_Lock);
    Close_A_Device(Left_Side);
    FreeDeviceBloc(Right_Side);
}

```

```

        WaveLeft[i] = 0;
        WaveRight[i] = 0;
    }
SetDrMd(Window->RPort, (ULONG)COMPLEMENT);
The_Audio_Device();
while((*RightMouse & 0x0400) == 0x0400)
{
    if (Audio_Channel_Stolen (Left_Lock) ||
        Audio_Channel_Stolen (Right_Lock))
        CloseIt("Channel stolen");
        /* Cas où des canaux ont été volés! */
    if ((*LeftMouse & 0x40) == 0)
    {
        x = Screen->MouseX;
        y = Screen->MouseY;
        if (WaveLeft[x] != (ScreenHeight/2-y))
        {
            Move(Window->RPort, x, ScreenHeight/2);
            Draw(Window->RPort, x, ScreenHeight/2-WaveLeft[x]);
            WaveLeft[x] = (ScreenHeight/2-y);
            WaveRight[x] = WaveLeft[x];
            Move(Window->RPort, x, ScreenHeight/2);
            Draw(Window->RPort, x, ScreenHeight/2-WaveLeft[x]);
        }
    }
}
Close_Audio_Device();
}
/*********************************************
*                                     (User)*
*****************************************/
main()
{
WaveLeft = (BYTE*) AllocMem(ScreenWidth, (ULONG) MEMF_CHIP|MEMF_CLEAR);
WaveRight = (BYTE*) AllocMem(ScreenWidth, (ULONG) MEMF_CHIP|MEMF_CLEAR);
if ((WaveLeft == 0L) || (WaveRight == 0L))
    CloseIt("No Wave Buffer !");
OpenLibs();
OpenScreenWindow();
Edit();
CloseScreenWindow();
CloseLibs();
FreeMem(WaveLeft, ScreenWidth);
FreeMem(WaveRight, ScreenWidth);
}
}

```

Le programme qui va suivre présente un type particulier d'engendrement du son.

Si vous voulez par exemple sortir continuellement du son, vous constaterez que votre mémoire ne suffit pas pour jouer un morceau durant une demi-heure. Grâce à une astuce, on peut parvenir à jouer un morceau de musique aussi long que l'on veut (par exemple une musique de fond dans un jeu). Il suffit pour cela de faire jouer de courts morceaux l'un après l'autre. Mais si vous faites jouer ces morceaux à l'aide de commande CMD_WRITE successives,

```

Audio_Write /*Sound 1 */;
Audio_Write /*Sound 2 */;

```

```
Audio_Write /*Sound 3 */;
...
```

vous constaterez que l'on entend des bruits désagréables (blips) entre la dernière note du morceau actuel et la première note du morceau suivant. Cela tient au fait que les canaux DMA Audio observent un moment de pause entre la fin du dernier CMD_WRITE et le début du suivant: ils "chassent" un moment dans la mémoire, sans coordination, ce qui provoque ces sons désagréables.

Mais si vous appliquez la technique du "double buffering", que certains d'entre vous connaissent sûrement, puisqu'on l'utilise dans la programmation des graphiques, ces sons parasites disparaissent. La sortie s'effectue selon le schéma suivant :

```
Jouer le premier morceau
Boucle
    Calculer le nouveau morceau (ou le charger à partir du disque)
    Envoyer la commande Write pour le nouveau morceau
    Attendre la fin du morceau précédent
        (à partir d'ici, on entend le nouveau morceau)
    Calculer le morceau suivant
    Envoyer la commande Write pour un autre morceau
    Attendre l'ancien morceau
Répéter la boucle
```

Les commandes Write avec copies de blocs de device sont traitées dans l'ordre de manière interne. C'est pourquoi une double commande CMD_WRITE ne crée pas de chaos pour ces canaux. Voici le programme à double tampon :

```
*****
*                               Double.c
*                               (c) Bruno Jennrich
*                               Août 1988
*****
/* Compile-Info:
 *   cc Double.c
 *   ln Double.o Audio_Support.o Devs_Support.o -lc
 */
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "devices/audio.h"
#define AUDIO_LEN      (ULONG) (sizeof(struct IOAudio))
VOID *OpenLibrary();
VOID *AllocMem();
VOID *GetDeviceBloc();
VOID *Audio_Lock();
/* Structures concernant le device Audio */
#define Left_Channel_0 1
#define Right_Channel_1 2
#define Right_Channel_2 4
#define Left_Channel_3 8
#define SoundPrecedence (BYTE) -40
#define WaveLength1 320L
#define WaveLength2 600L
struct IOAudio *FirstPlay           -OL.
```

```

        *SecondPlay      -OL,
        *FirstLock       -OL,
        *SecondLock      -OL;
char *FirstWave = OL; /* Définition de la forme d'onde (signed) */
char *SecondWave = OL;
UBYTE Channels[] = {Left_Channel_0, Left_Channel_3};
                         /* Channel_Map */
                         /* Allouer l'importe quel */
                         /* canal de gauche */
#define CHANNEL_SIZE (ULONG) sizeof(Channels)
                         /* Taille du masque */
/*********************************************
*                                     (User)*
*
* Fonction: Fermer et libérer le tout
*-----
* Paramètres d'entrée:
*
* String: Error-String
*****************************************/
VOID CloseIt (String)
char      *String;
{
    UWORLD i;
    UWORLD *dff180 = (UWORD *)0xdff180;
    UWORLD Error;
    if (strlen(String) > 0)
        for(i=0;i<0xffff;i++) *dff180 = i;
    puts(String);
    if (FirstLock != OL)     Audio_Free(FirstLock);
    if (SecondLock != OL)    Audio_Free(SecondLock);
    if (SecondPlay != OL)   FreeDeviceBloc(SecondPlay);
    if (FirstPlay != OL)    Close_A_Device(FirstPlay);
    if (FirstWave != OL)   FreeMem(FirstWave, WaveLength1);
    if (SecondWave != OL)  FreeMem(SecondWave, WaveLength2);
    exit(10);
}
/*********************************************
*                                     (User)*
*
* Fonction: Utiliser le device Audio
*****************************************/
The_Audio_Device()
{
    UWORLD i, j;
    UBYTE *bfe001 = (UBYTE*) 0xbfe001;
    FirstPlay = (struct IOAudio *) GetDeviceBloc(AUDIO_LEN);
    SecondPlay = (struct IOAudio *) GetDeviceBloc(AUDIO_LEN);
    FirstPlay->ioa_Data = (UBYTE *)Channels;
    FirstPlay->ioa_Length = CHANNEL_SIZE;
    FirstPlay->ioa_Request.io_Message.mn_Node.ln_Pri = SoundPrecedence;
    Open_A_Device("audio.device", OL, &FirstPlay, OL, OL);
    Audio_Copy(FirstPlay, SecondPlay);
    FirstLock = Audio_Lock(FirstPlay);
    SecondLock = Audio_Lock(SecondPlay);
    Audio_Pervol(FirstPlay, (UWORD)0, (UWORD)0);
    for (i=0;i<WaveLength1;i++)
        FirstWave[i] = 0;
    for (i=0;i<WaveLength2;i++)

```

```

SecondWave[i] = 0;
Audio_Pervol(FirstPlay, (UWORD)1500, (UWORD)64);
for (i=0;i<WaveLength1;i+=2) FirstWave[i] = 128;
for (i=0;i<WaveLength2;i+=4) SecondWave[i] = 128;
/* Calculer FirstWave (déjà fait) */
Audio_Write(FirstPlay, FirstWave, (ULONG) 320,
            (UWORD) 1, (BOOL) TRUE);
while ((*bfe001 & 0x40) == 0x40)
{
    /* Calculer SecondWave (déjà fait) */
    Audio_Write(SecondPlay, SecondWave, (ULONG) WaveLength2,
                (UWORD) 1, (BOOL) TRUE);
    WaitIO(FirstPlay);
    /* Calculer à nouveau FirstWave (n'est pas fait ici!) */
    Audio_Write(FirstPlay, FirstWave, (ULONG) WaveLength1,
                (UWORD) 1, (BOOL) TRUE);
    WaitIO(SecondPlay);
}
Audio_Free(FirstLock);
Audio_Free(SecondLock);
FreeDeviceBloc(SecondPlay);
Close_A_Device(FirstPlay);
}
/*********************************************
*                               main()          (User)*
*****************************************/
main()
{
    FirstWave = (BYTE *) AllocMem(WaveLength1,
(ULONG)MEMF_CHIP|MEMF_CLEAR);
    SecondWave = (BYTE *) AllocMem(WaveLength2,
(ULONG)MEMF_CHIP|MEMF_CLEAR);
    if (FirstWave == 0L) CloseIt("No Memory for Wave 1 !");
    if (SecondWave == 0L) CloseIt("No Memory for Wave 2 !");
    The_Audio_Device();

    FreeMem(FirstWave, WaveLength1);
    FreeMem(SecondWave, WaveLength2);
}

```

Il est clair que les commandes de device Audio décrites plus haut ne sont pas les seules possibles. Le device Audio possède beaucoup plus de commandes. Mais celles-ci ne sont pas particulièrement importantes, et elles ne sont pas nécessaires au programmeur normal. Pour être complet, nous en donnerons cependant la liste.

```

/*********************************************
*           Audio_Read()          (Audio_Support)*
*
* Fonction: Obtenir le bloc de device
*-----
* Paramètres d'entrée:
*-
* ReadRequest: Bloc de device
* Channel:   Canal (0, 1, 2, 3) dont on doit obtenir le bloc
*-
* Valeur en retour:
*-
* Adresse du bloc de device responsable de la sortie sur le
*
```

```

* canal indiqué ou -1 s'il n'y a pas de CMD_WRITE en cours      *
***** */
UBYTE *Audio_Read(Read_Request, Channel)
struct IOAudio    *Read_Request;
ULONG             Channel;
{
    Read_Request->ioa_Request.io_Unit = (struct Unit *) Channel;
    Do_Command(Read_Request, (UWORD) CMD_READ);
    if (Read_Request->ioa_Request.io_Error != 0)
        return ((UBYTE*)0xffffffff);
    return(Read_Request->ioa_Data);
}

```

A l'aide de cette fonction, vous pouvez connaître l'adresse du bloc de device Audio responsable de la sortie du son sur un canal déterminé. Le bit correspondant du canal est indiqué dans Channel, et transmis à l'élément Unit du bloc de device Audio. Après CMD_READ, on trouve dans ioa_Data soit l'adresse du bloc de device Audio, qui exécute actuellement une commande CMD_WRITE sur le canal indiqué, soit la valeur 0 lorsque le canal indiqué n'est pas occupé actuellement en écriture.

La fonction CMD_READ peut être utilisée pour définir les priorités des autres utilisateurs du device Audio. On peut, à partir de ces informations, préciser la priorité de réservation, et améliorer ses chances d'accès au canal.

ADCMD_WAITCYCLE

Dans la routine ci-dessus, nous sommes servi du flag SYNCYCLE pour exécuter la commande de manière synchronisée avec la fin de la sortie du son. Cette synchronisation peut être également obtenue par l'intermédiaire d'une commande device Audio :

```
Do_Command(Audio_Device_Bloc, (UWORD) ADCMD_WAITCYCLE);
```

Cette commande ne revient au programme que lorsque le Cycle en cours a pris fin.

ADCMD_RESET

Cette commande ramène le device Audio au stade de départ :

```
Do_Command(Audio_Device_Bloc, (UWORD) CMD_RESET);
```

La commande CMD_FLUSH est exécutée, les vecteurs d'interruption du son sont réinitialisés, et une commande CMD_STOP précédente est supprimée.

CMD_FLUSH

Le Write-Request actuel et tous sont ceux qui sont en attente (double buffering) sont arrêtés ou ne sont plus exécutés.

```
Do_Command(Audio_Device_Bloc, (UWORD) CMD_FLUSH);
```

CMD_FINISH

Cette commande interrompt également la sortie du son :

```
*****
*           Audio_Finish()          (Audio_Support)*
*
* Fonction: Arrêter la sortie du son
*-----*
* Paramètres d'entrée:
*
* Audio_Device_Bloc: Bloc de device, dont le son
*                     doit être arrêté
* Sync:            Synchronise fin du son avec fin de cycle?*
*****
VOID Audio_Finish(Audio_Device_Bloc, Sync)
struct IOAudio *Audio_Device_Bloc;
BOOL             Sync;
{
    Audio_Device_Bloc->ioa_Request.io_Command = (UWORD)ADCMOD_FINISH;
    if (Sync) Audio_Device_Bloc->ioa_Request.io_Flags |= (UBYTE)ADIOF_SYNCCYCLE;
    else     Audio_Device_Bloc->ioa_Request.io_Flags &=
(UBYTE)ADIOF_SYNCCYCLE;
    DoIO(Audio_Device_Bloc);
}
```

Si vous désirez synchroniser l'interruption du son actuel avec la fin de la sortie du son, vous devez transmettre à `Audio_Finish()` la valeur TRUE comme second paramètre (`Sync`). Le flag `SYNCCYCLE`, responsable de cette synchronisation, est alors posé. `ADCMOD_FINISH` s'occupe de l'exécution des Write-Requests en attente. C'est ce qui le distingue de `CMD_FLUSH`.

CMD_START et CMD_STOP

A l'aide de ces deux commandes, vous pouvez interrompre la sortie du son (`Do_Command(Audio_Device_Bloc, (UWORD)CMD_STOP)`), et la relancer à nouveau (`Do_Command(Audio_Device_Bloc, (UWORD)CMD_START)`).

4.11.5. Son dans le code Interrupt

Les commandes Audio-Device qui suivent ne doivent pas être employées au niveau d'interruption 5 et aux niveaux plus élevés :

ADCMOD_FINISH	
ADCMOD_FREE	(ne pas employer dans le code Interrupt!)
ADCMOD_LOCK	(ne pas employer dans le code Interrupt!)
ADCMOD_PERVOL	
ADCMOD_SETPREC	(ne pas employer dans le code Interrupt!)
ADCMOD_WAITCYCLE	
CMD_FLUSH	
CMD_RESET	

```
CMD_START
CMD_STOP
CMD_WRITE
```

Cela paraît d'ailleurs tout à fait logique, puisque les interruptions qui annoncent que le DMA en a terminé avec la sortie du son se trouvent au niveau 4. Les interruptions nanties d'une priorité CPU supérieure doivent être réservées pour le système.

4.12. ...et comme synthétiseur de parole

Venons-en maintenant à un avatar du device Audio: le device Narrator. Celui-ci permet de faire parler l'Amiga. Le device Narrator permet en effet d'accéder à la parole synthétisée, ou plutôt au synthétiseur de parole de l'Amiga. Malheureusement, ce synthétiseur ne parle que l'anglais. Si vous voulez le faire parler français, vous obtiendrez notre langue avec un fort accent américain, ce qui est plutôt gênant.

Mais on peut tout de même parvenir à faire prononcer au synthétiseur des phrases intelligibles en français avec un accent acceptable, à l'aide de quelques astuces. En effet, le synthétiseur est constitué de ce qu'on appelle des codes de phonèmes, c'est-à-dire des codes qui imitent les sons de la langue. Il existe ainsi des phonèmes pour les sons a, t, etc. En mettant ces phonèmes bout à bout, on obtient des mots prononcés par une voix humaine!

Mais étant donné que la conversion des mots en phonèmes est plutôt complexe à réaliser, Commodore a créé la librairie Translator, qui possède une seule commande: la routine Translate(). Cette routine est en mesure de convertir (presque) tous les mots anglais en leurs codes de phonèmes. Elle est en effet capable de traduire la plupart des mots en anglais à l'aide de certaines règles. Mais comme il existe en anglais aussi certaines exceptions aux règles, Translate() utilise également une table contenant les mots différant de la règle.

Translate maîtrise bien par exemple la différence entre les phonèmes dans la prononciation de "the car" et "the others". Dans le premier cas, comme on le sait, "the" se prononce par le son E, tandis que dans le second cas il se prononce avec le son I, car le mot suivant commence par une voyelle. Les phonèmes sont DHAX CAA3R (the car), et DHIY AH2DHERZ (the others).

Comment fait-on pour appeler Translate()? C'est extrêmement simple. Une fois que la librairie Translator a été ouverte (TranslatorBase = OpenLibrary ("translator.library", 0)), Translate() est appelé avec comme paramètres la chaîne à prononcer et sa longueur, l'adresse du secteur de mémoire pour les phonèmes ainsi que sa longueur :

```
char *String; /* Null terminated */
char Phoneme[1000];
#define LEN sizeof(Phoneme)
Error = Translate(String, strlen(String), Phoneme, LEN);
```

Si l'emplacement en mémoire est suffisant pour les phonèmes, Error contient la valeur 0. Si Error est différent de 0, la valeur absolue de Error (car Error a alors une valeur négative) indique l'emplacement dans la chaîne à partir duquel la traduction n'a pas pu se faire, car la mémoire était insuffisante. Dans notre exemple ci-dessus, nous aurions pu marquer cet endroit au moyen de `String(-Error)`, et continuer à traduire :

```
Error = Translate(&String[-Error], strlen(&String[-Error]), NouvMémoire,
Longueur);
```

Vous pouvez alors faire prononcer la chaîne des phonèmes générée par `Translate`, par l'intermédiaire du device `Narrator`. Pour cela, il faut néanmoins commencer par ouvrir le device `narrator` :

```
#define NARRAT_RB_LEN sizeof(struct narrator_rb)
struct narrator_rb *WriteRequest = 0L;
...
Open_A_Device("narrator.device", 0L, &WriteRequest, 0L, NARRAT_RB_LEN);
```

Le bloc de device `Narrator` (`narrator_rb`) se présente comme suit :

Offset	Structure
-----	-----
0 0x00	struct IOStdReq message; /* Comme toujours! */
14 0x0e	UWORD rate; /* Mots par minute*/
16 0x10	UWORD pitch; /* Fréquence de base */
18 0x12	UWORD mode; /* humain//robot */
20 0x14	UWORD sex; /* sexe*/
22 0x16	UBYTE *ch_masks; /* Channel Allocation Map */
26 0x1a	UWORD nm_masks; /* sizeof (ch_masks) */
28 0x1c	UWORD volume; /* Volume */
30 0x1e	UWORD sampfreq; /* Fréquence de Sampling */
32 0x20	UBYTE mouths; /* Flag pour la forme de la bouche */
33 0x21	UBYTE chanmask; /* Quels canaux sont /*/ réellement utilisés */
34 0x22	UBYTE numchan; /* Combien de masques? */
35 0x23	UBYTE pad; /* pour adresse paire */
36 0x24	}

Il ne s'agit cependant pas du seul bloc de device pour le device `narrator`. Celui-ci utilise en effet le bloc de device ci-dessus uniquement pour la sortie au moyen du device `Narrator`. Dans le cas de la lecture, c'est un autre bloc qui est utilisé :

Offset	Structure
-----	-----
0 0x00	struct mouth_rb;
-----	-----
36 0x24	struct narrator_rb voice;
37 0x25	UBYTE width; /* Largeur de la bouche */
38 0x26	UBYTE height; /* hauteur de la bouche */
39 0x27	UBYTE shape; /* Interne ! */
40 0x28	UBYTE pad; /* Adresse paire */

Comme vous pouvez le deviner sans difficulté à partir du nom de cette structure (`mouth` = bouche) et des commentaires, on utilise cette structure pour lire la forme de la bouche au moment de la prononciation du mot. Nous en venons maintenant aux commandes

du device Narrator. La commande la plus importante est CMD_WRITE. C'est elle qui est responsable de la prononciation du code de phonème par l'intermédiaire du device Audio :

```
*****
/*          Narrator_Write()      (Narrat_Support)*/
/*
/* Fonction:    Sortie Narrator
/*
/*-----*/
/* WriteRequest: Bloc IO par lequel se fait la sortie */
/* string:       Chaîne à prononcer (en anglais) */
/* rate:         Combien de mots par minute (40-400) */
/* pitch:        Hauteur de base pour le son (65-320) */
/* mode:         Voix de robot (1) ou naturelle (0) */
/* sex:          Sexe (0 masc.../1 fémin...) */
/* Channels:    Channel-Map */
/* Size:         sizeof (Channel-Map) */
/* vol:          Volume (0-64) */
/* freq:         Fréquence de sortie (5000-28000) */
/* mouths:       Générer la forme de la bouche (1) */
*****
VOID Narrator_Write
    (WriteRequest, string, rate, pitch, mode, sex, Channels, Size, vol,
 freq, mouths)
struct narrator_rb *WriteRequest;
char             *string;
UWORD            rate, pitch, mode, sex;
UBYTE           *Channels;
UWORD            Size, vol, freq, mouths;
{
    struct mouth_rb *ReadRequest; /* Request pour forme de la
bouche */
    UBYTE SpokenString[1000];   /* Mémoire des phonèmes */
    ULONG Mouth_Rout_Count;   /* Combien d'appels */
                                /* de Mouth_Routine() */

    WriteRequest->rate;
    WriteRequest->pitch;
    WriteRequest->mode;
    WriteRequest->sex;
    WriteRequest->ch_masks;
    WriteRequest->nm_masks;
    WriteRequest->volume;
    WriteRequest->sampfreq;
    WriteRequest->mouths;
    if (Translate (string, (ULONG) strlen(string), SpokenString, 1000L)
!= 0)
        CloseIt("TranslateError");
        /* Phonémiser une phrase en anglais */
    WriteRequest->message.io_Data = (APTR) SpokenString;
    WriteRequest->message.io_Length = (ULONG) strlen(SpokenString);
    WriteRequest->message.io_Command = (UWORD) CMD_WRITE;
    if (mouths == 1) /* Génère la forme de la bouche */
    {
        ReadRequest = (struct mouth_rb *) GetDeviceBloc(MOUTH_RB_LEN);
        Narrator_Copy(WriteRequest, ReadRequest);
        /* Prépare ReadRequest */
        Mouth_Init();           /* Initialise Mouth-Routine */
        ReadRequest->width
                                = (UBYTE) 0;
    }
}
```

```

ReadRequest->height           = (UBYTE) 0;
ReadRequest->voice.message.io_Command = (UWORD) CMD_READ;
ReadRequest->voice.message.io_Error   = (UBYTE) 0;
/* Prépare la commande de lecture */
if (SendIO(WriteRequest) != 0) CloseIt ("SpeakError");
/* Envoie une commande Write */
Mouth_Rout_Count = 0;          /* Mouth-Routine appelée 0 fois */
while (ReadRequest->voice.message.io_Error != ND_NoWrite)
{
    DoIO(ReadRequest);
    Mouth_Routine(ReadRequest->width, ReadRequest->height,
    Mouth_Rout_Count);
    /* lecture tant que c'est possible */
    /* et appel de Mouth_Routine() */
    Mouth_Rout_Count++;
}
FreeDeviceBloc(ReadRequest);
Mouth_Expunge();               /* Met fin à Mouth_Routine */
}
else DoIO(WriteRequest);        /* ne pas générer de forme de la
bouche */
}                                /* Sortir uniquement la phrase*/
}

```

Cette routine relie la prononciation (CMD_WRITE) avec l'obtention de la forme de la bouche (CMD_READ). On pose tout d'abord les paramètres nécessaires dans le WriteRequest, donc dans la demande de synthèse de parole. Les paramètres comme Channels, Size et Vol sont clairs par eux-mêmes, ou vous sont connus car vous les avez rencontrés dans le device Audio (section précédente). Nous allons donc nous occuper des autres paramètres.

Le premier paramètre à transmettre dans NarratorWrite() est le bloc de device narrator initialisé par Open_A_Device() (pour l'écriture, c'est-à-dire ici pour la synthétisation de la parole). Vient ensuite comme d'habitude la chaîne à prononcer, indiquée en toutes lettres (et non pas en phonèmes). Puis "rate" détermine le nombre de mots à prononcer à la minute. Plus la valeur de "rate" est grande, et plus les mots sont prononcés rapidement. Il y a toutefois des limites, puisque "rate" doit se trouver entre 40 et 400.

"pitch" détermine la hauteur de la voix. Ce paramètre est soumis lui aussi à des limitations, et doit se situer entre 65 et 320. "pitch" détermine la hauteur de voix avec lequel l'Amiga va vous adresser la parole.

Lorsqu'on parle, l'accent mis naturellement sur les mots provoque cependant des déviations par rapport à cette hauteur de base. La langue naturelle oscille en quelque sorte autour de la hauteur de base de la voix. Si vous voulez interdire cette oscillation de façon à ce que le synthétiseur parle de manière "monotone", il faut mettre le paramètre "mode" sur 1. La valeur 0 a pour effet la reproduction d'une parole naturelle.

"sex" permet de déterminer le sexe du synthétiseur. La valeur 1 donne une voix féminine, alors que la valeur 0 donne une voix masculine. La fréquence du synthétiseur peut également être déterminée. Plus cette fréquence est grande, et plus la prononciation est naturelle. C'est d'ailleurs tout à fait logique. En effet, plus les formes d'onde sont

calculées de manière précise et fine, et meilleur sera le résultat. Avec la valeur 5000 pour "freq", on ne peut pas dire qu'on entend une reproduction de la parole humaine, puisqu'on obtient une série de bips. Avec la valeur 28000 pour "freq", le résultat est correct pour l'Amiga.

Le dernier paramètre pour `Narrator_Write()` indique si la forme de la bouche doit être lue (`mouths =1`). De même que l'on remue la bouche en parlant, on peut amener l'ordinateur à faire de même, par exemple à l'aide d'un petit graphique. Il faut cependant communiquer celui-ci, ce que l'on fait à l'aide de la commande `CMD_READ`. Nous y reviendrons en détail plus loin. Pour le moment, nous allons nous intéresser aux autres éléments qui interviennent dans la reproduction de la parole.

Il faut tout d'abord transmettre les variables indiquées ci-dessus (`vol`, `freq`, `sex`, etc.) au `WriteRequest` (Bloc IO du device `Narrator`). Ensuite, on peut traduire la chaîne à prononcer au moyen de `Translate()`. Il suffit alors de transmettre le code de phonème, qui se termine d'ailleurs par le caractère de fin de chaîne \0, au pointeur `io_Data` du `WriteRequest`, ainsi que la longueur de cette chaîne à la variable `io_Length` du `WriteRequest`.

Une fois la commande définie (`WriteRequest->message-io_Command = (UWORD) CMD_WRITE;`), on peut lancer la sortie à l'aide de `DoIO()`. Outre cette situation simple (celle de la synthétisation de parole), `Narrator_Write()` offre également la possibilité d'obtenir la forme de la bouche. Celle-ci est lue, nous l'avons dit, par le `Narrator_Device()` au moyen de `CMD_READ`.

Si vous avez défini le paramètre sur 1 pour obtenir l'indication de la forme de la bouche, vous obtiendrez d'abord un `ReadRequest`, donc un bloc de device pour la lecture (`GetDeviceBloc()`). Puis les variables `io_Unit` et `io_Device` seront copiées dans le `ReadRequest` à partir du bloc de device original. Nous avons souvent rencontré ce type de routine. Nous nous contenterons donc ici d'en écrire l'appel à l'aide des deux premières lignes de la définition de la fonction :

```
VOID Narrator_Copy (Old_Request, New_Request)
struct narrator_rb *Old_Request;
struct mouth_rb      *New_Request;
{...}
```

Après la mise en place du `ReadRequest`, on appelle la routine `Mouth_Init()`. Cette routine fait partie du trio constitué de `Mouth_Init()`, `Mouth_Routine()` et `Mouth_Expunge()`. Ces trois routines disponibles sont destinées aux préparations indispensables avant l'indication de la forme de la bouche. Dans `Mouth_Init()`, on ouvre par exemple des écrans (screens) et des fenêtres (windows), dans lesquels on peut dessiner ensuite la bouche en mouvement à l'aide de `Mouth_Routine()`. Lorsque la prononciation prend fin, `Mouth_Expunge()` ferme les écrans et fenêtres ouvertes par `Mouth_Init()`.

Mais avant d'appeler `Mouth_Routine()`, nous devons traiter le `ReadRequest`. Il faut d'abord initialiser la forme de la bouche. Cette forme est définie par la hauteur et la largeur de la bouche, et elle doit être placée d'abord sur 0. Nous indiquons ensuite au bloc `ReadRequest` sa tâche, c'est-à-dire la commande à exécuter (`CMD_READ`).

Il faut alors lire la forme de la bouche jusqu'à ce que la commande Write précédente prenne fin, c'est-à-dire jusqu'à ce que le dernier mot soit prononcé. Lorsque le device Narrator n'a plus rien à dire, l'erreur ND_NoWrite surgit quand on tente de lire de nouvelles formes de bouche. CMD_READ est une commande synchrone, et il faut toujours l'appeler par DoIO(). DoIO() ne revient réellement avec CMD_READ que lorsqu'une modification intervient dans la forme de la bouche.

Lorsqu'il y a une telle modification, la routine Mouth_Routine() est appelée dans Narrator_Write(), et celle-ci transmet la largeur et la hauteur enregistrées par elle, ainsi que le nombre des appels. Le programme suivant utilise la routine Narrator-Write que nous venons de présenter :

```
*****  
/*          The-Narrator-Device      */  
/*          */  
/*          Juni 1988                */  
/*          (c) Bruno Jennrich       */  
*****  
/* Compile - Info:                 */  
/*-----*/  
/*          */  
/* cc Say.c                         */  
/* ln Narrat_Support.o Say.o Devs_Support.o -lc */  
*****  
#include "exec/types.h"  
#include "exec/types.h"  
#include "exec/memory.h"  
#include "exec/devices.h"  
#include "intuition/intuition.h"  
#include "intuition/intuitionbase.h"  
#include "graphics/gfxbase.h"  
#include "graphics/gfxmacros.h"  
#include "libraries/translator.h"  
#include "Narrat_Support.h"  
#define WIDTH 150                  /* Taille de la fenêtre Mouth */  
#define HEIGHT 75                 /* Fonctions utilisées */  
VOID *OpenLibrary();  
VOID *OpenWindow();  
struct Library     *TranslatorBase = OL; /* Librarys */  
struct IntuitionBase *IntuitionBase = OL;  
struct GfxBase     *GfxBase = OL;  
struct NewWindow   *NewWindow;        /* Window-Defs */  
struct Window      *Window = OL;  
struct narrator_rb *WriteRequest = OL; /* Blocs de device Narrator */  
UBYTE Channels[4] = {3, 5, 10, 12};    /* Masque d'allocation des  
canaux */  
                                         /* s. Audio-Device */  
#define MOUTH_RB_LEN    (ULONG) sizeof(struct mouth_rb)  
ULONG Mouth_Rout_Count;                 /* Combien d'appels de */  
                                         /* Mouth_Routine() */  
*****  
/*          CloseIt()           (User) */  
/*-----*/  
/* Fonction: Libérer toutes les structures allouées */  
/*-----*/  
/* String: Error-Message */
```

```

/********************************************* */
VOID CloseIt(String)
char      *String;
{
    UWORLD Error = 0, i;
    UWORLD *dff180 = (UWORD *)0xdff180;      /* Registre pour la couleur
du fond*/
    if (strlen(String) > 0)
    {
        for (i=0;i<0xffff;i++) *dff180 = i; /* Clignotement de l'écran*/
        puts(String);                      /* Sortie de la chaîne */
        Error = 100;                      /* Error-Code != 0 (EXIT()) */
    }
    if ((WriteRequest != 0L) && (WriteRequest->message.io_Device != 0L))
        Close_A_Device(WriteRequest);
    if (TranslatorBase != 0L)
        CloseLibrary(TranslatorBase);
    exit(Error);
}
/********************************************* */
/*          Mouth_Init()           (User)*/
/* Fonction: Routine d'initialisation de Mouth */
/********************************************* */
VOID Mouth_Init()
{
    if ((GfxBase = (struct GfxBase *) OpenLibrary("graphics.library",
0L)) == 0)
        CloseIt("No Graphics !!!");
    if ((IntuitionBase = (struct IntuitionBase *) OpenLibrary("intuition.library", 0L)) == 0)
        CloseIt("No Intuition !!!");
    NewWindow.LeftEdge   = 640/2-WIDTH/2;
    NewWindow.TopEdge    = 200/2-HEIGHT/2;
    NewWindow.Width     = WIDTH;

    NewWindow.Height     = HEIGHT;
    NewWindow.DetailPen = 0;
    NewWindow.BlocPen   = 1;
    NewWindow.IDCMPFlags = 0;
    NewWindow.Flags     = ACTIVATE | RMBTRAP | NOCAREREFRESH;
    NewWindow.FirstGadget = (struct Gadget *) 0L;
    NewWindow.CheckMark = (struct Image *) 0L;
    NewWindow.Title     = (UBYTE *) "Mouth-Shape";
    NewWindow.Screen    = (struct Screen *) 0L;
    NewWindow.BitMap    = (struct BitMap *) 0L;
    NewWindow.MinWidth  = 0;
    NewWindow.MaxWidth  = 0;
    NewWindow.MinHeight = 0;
    NewWindow.MaxHeight = 0;
    NewWindow.Type      = WBENCHSCREEN;
    if ((Window = (struct Window *) OpenWindow(&NewWindow)) == 0)
        CloseIt("No Window !!!");
    SetAPen(Window->RPort, 2);
    SetDrMd(Window->RPort, (ULONG)(COMPLEMENT|JAM1));
}
/********************************************* */
/*          Mouth_Expunge()        (User)*/
/* */

```

```

/* Fonction: Mouth-Expunge-Routine */  

/*******************************************/  

VOID Mouth_Expunge()  

{  

    if (Window != OL) CloseWindow(Window);  

    if (GfxBase != OL) CloseLibrary(GfxBase);  

    if (IntuitionBase != OL) CloseLibrary(IntuitionBase);  

}  

/*******************************************/  

/* Mouth_Routine() (User) */  

/* Fonction: Mouth-Routine */  

/*-----*/  

/* width: forme de la bouche-Largeur */  

/* height: forme de la bouche-Hauteur */  

/*******************************************/  

VOID Mouth_Routine(width, height, Count)  

UBYTE width, height;  

ULONG Count;  

{  

    static ULONG old_width, old_height;  

    if (Count != OL)  

    {  

        /* Supprimer l'ancienne forme */  

        Move(Window->RPort, (ULONG)(WIDTH/2), (ULONG)(HEIGHT/2-height));  

        Draw(Window->RPort, (ULONG)(WIDTH/2), (ULONG)(HEIGHT/2+height));  

        Move(Window->RPort, (ULONG)(WIDTH/2-width), (ULONG)(HEIGHT/2));  

        Draw(Window->RPort, (ULONG)(WIDTH/2+width), (ULONG)(HEIGHT/2));  

    }  

    /* Dessiner la nouvelle forme*/  

    width *= 2;  

    WaitTOF();  

    Move(Window->RPort, (ULONG)(WIDTH/2), (ULONG)(HEIGHT/2-height));  

    Draw(Window->RPort, (ULONG)(WIDTH/2), (ULONG)(HEIGHT/2+height));  

    Move(Window->RPort, (ULONG)(WIDTH/2-width), (ULONG)(HEIGHT/2));  

    Draw(Window->RPort, (ULONG)(WIDTH/2+width), (ULONG)(HEIGHT/2));  

    old_width = (ULONG)width;  

    old_height = (ULONG)height;  

}  

/*******************************************/  

/* The_Narrator_Device() (User) */  

/*-----*/  

/* Fonction: Utiliser le device Narrator */  

/*-----*/  

/* Paramètres d'entrée: */  

/*-----*/  

/* string: Chaîne à prononcer */  

/* rate: Mots par minute */  

/* pitch: Fréquence de nase */  

/* mode: Voix de robot ou humaine */  

/* sex: Sexe */  

/* vol: Volume */  

/* freq: Fréquence de sampling */  

/* mouths: Générer forme de la bouche? */  

/*******************************************/  

VOID The_Narrator_Device(string, rate, pitch, mode, sex, vol, freq,  

mouths)  

char *string;  

UWORD rate, pitch, mode, sex, vol, freq, mouths;

```

```

{
    TranslatorBase = OpenLibrary("translator.library", OL);
    if (TranslatorBase == OL) CloseIt("NoTranslatorBase");
    Open_A_Device("narrator.device", OL, &WriteRequest, OL, MOUTH_RB_LEN);
    Narrator_Write
        (WriteRequest, string, rate, pitch, mode, sex, Channels,
        (UWORD)sizeof (Channels), vol, freq, mouths);
    Close_A_Device(WriteRequest);
    CloseLibrary(TranslatorBase);
}
//*********************************************************************
/*          AtoUWORD()           (User)*/
/*
/* Fonction: Convertir la chaîne du nombre (ASCII)   */
/*          en format WORD          */
/*
-----*/
/* Paramètres d'entrée:                         */
/*          */
/* Bufpointer: Chaîne numérique à convertir      */
/*-----*/
/* Valeur en retour: Valeur de la chaîne numérique */
//*********************************************************************
UWORD AtoUWORD(BufPointer)
char *BufPointer;
{
    UWORD Result;
    Result = 0;
    while (*BufPointer == ' ') BufPointer++; /* Sauter les espaces */
    while (*BufPointer != '\000')
    {
        Result *= 10;
        Result += *(BufPointer++) - 0;
    }
    return(Result);
}
//*********************************************************************
/*          main()            */
//*********************************************************************
main (argc, argv)
UWORD argc;
char **argv;
{
    UWORD Param[7];
    ULONG i;
    Param[0] = DEFRATE; /* rate */ /* Default's */
    Param[1] = DEFPITCH; /* pitch */
    Param[2] = DEFMODE; /* mode */
    Param[3] = DEFSEX; /* sex */
    Param[4] = DEFVOL; /* volume */
    Param[5] = DEFFREQ; /* frequency */
    Param[6] = 1; /* mouths */
    if ((argc < 2) || (argc > 9))
        printf("Usage: SAY \"string\" [rate] [pitch] [mode] [sex] [volume]
               [sampfreq] [mouths]\n");
    else
    {
        for (i=0;i<(argc-2);i++)
            Param[i] = AtoUWORD (argv[i+2]);
        The_Narrator_Device

```

```

        (argv[1], Param[0], Param[1], Param[2], Param[3], Param[4],
        Param[5], Param[6]);
    }
}

```

Dans ce qui suit, nous présentons d'autres routines de support de Narrator. Elles vous permettront de voir quelles sont les autres commandes soutenues par le device Narrator :

```

/*********************************************
/*           Narrator_Copy()      (Narrat_Support)*/
/*
/* Fonction: Copie du Narrator-Bloc device      */
/*-----*/
/* Old_Request: IO-Bloc original                */
/* New_Request: Copie du bloc IO                 */
/*********************************************
VOID Narrator_Copy (Old_Request, New_Request)
struct narrator_rb *Old_Request;
struct mouth_rb          *New_Request;
{
    New_Request->voice.message.io_Device = /* On n'a besoin que de */
        Old_Request->message.io_Device;      /* de copier io_Device et
io_Unit */
    New_Request->voice.message.io_Unit = 
        Old_Request->message.io_Unit;
}
/*********************************************
/*           Narrator_Stop()      (Narrat_Support)*/
/*
/* Fonction: Arrêter la sortie de Narrator      */
/*-----*/
/* WriteRequest: Bloc IO dont la sortie doit être arrêtée */
/*********************************************
VOID Narrator_Stop (WriteRequest)
struct narrator_rb *WriteRequest;
{
    WriteRequest->message.io_Command = (UWORD) CMD_STOP;
    DoIO(WriteRequest);
}
/*********************************************
/*           Narrator_Start()      (Narrat_Support)*/
/*
/* Fonction: Lancer la sortie de Narrator      */
/*-----*/
/* WriteRequest: Bloc IO dont la sortie doit être lancée */
/*             (vers Narrator_Stop())            */
/*********************************************
VOID Narrator_Start(WriteRequest)
struct narrator_rb *WriteRequest;
{
    WriteRequest->message.io_Command = (UWORD) CMD_START;
    DoIO(WriteRequest);
}
/*********************************************
/*           Narrator_Flush()      (Narrat_Support)*/
/*
/* Fonction: Mettre fin à la sortie de Narrator */
/*-----*/
/* WriteRequest: Bloc IO dont la sortie doit être arrêtée */

```

```

/*****
VOID Narrator_Flush(WriteRequest)
struct narrator_rb *WriteRequest;
{
    WriteRequest->message.io_Command = (UWORD) CMD_FLUSH;
    DoIO(WriteRequest);
}
/*****
/*           Narrator_Reset()      (Narrat_Support)*/
/*           */
/* Fonction:     Remplacer le device narrator dans l'état */
/* connu (reset) */
/*-----*/
/* WriteRequest: Bloc IO par l'intermédiaire duquel le */
/* device narrator doit subir un reset */
/*****
VOID Narrator_Reset(WriteRequest)
struct narrator_rb *WriteRequest;
{
    WriteRequest->message.io_Command = (UWORD) CMD_RESET;
    DoIO(WriteRequest);
}

```

4.13. Le device Timer

Nous allons maintenant nous occuper du device Timer, dont le programmeur normal n'a sans doute jamais entendu parler. Ce device sert à adresser les timers internes. L'Amiga possède en effet deux types de timers. D'une part, il est possible de mesurer le temps par l'intermédiaire du retour vertical du faisceau d'électrons. D'autre part, on peut se servir du timer CIA. Tous les deux ont leurs avantages et leurs inconvénients: le timer du retour vertical reste constant sur de longues périodes. Il ne possède toutefois qu'une seule résolution: un cinquantième de seconde.

```

UNIT_MICROHZ 0 (CIA-Timer)
UNIT_VBLANK 1 (Vertical-Blank-Timer)

```

Le timer du retour vertical s'ouvre de la façon suivante :

```

struct timerequest *TimeRequest = 0L;
#define TIME_LEN (ULONG) (sizeof (struct timerequest))
...
Open_A_Device("timer.device", (ULONG) UNIT_VBLANK,
              &TimeRequest, TIME_LEN);
...

```

On peut maintenant programmer le device Timer. Voici les commandes disponibles :

TR_ADDREQUEST	Attendre un laps de temps déterminé
TR_GETSYSTEME	Obtenir l'heure système
TR_SETSYSTEME	Définir l'heure système

Il existe en outre trois commandes de device :

Offset	Commande
-0x2a	AddTime()
-0x30	SubTime()
-0x36	CmpTime()

Voyons d'abord les commandes de device. La routine suivante permet de laisser passer un temps déterminé :

```
*****
*           WaitTime()          (Timer_Support)*
*
* Fonction: Attendre un laps de temps déterminé
*
*-----*
* Paramètres d'entrée:
*
* TimeRequest: Timer-Device-Bloc
* Secs, Micro: Attendre combien de secondes et de
*               micro-secondes?
*****
VOID WaitTime(TimeRequest, Secs, Micro)
struct timerequest *TimeRequest;
ULONG             Secs, Micro;
{
    TimeRequest->tr_time.tv_secs   = Secs;
    TimeRequest->tr_time.tv_micro = Micro;
    Do_Command(TimeRequest, (UWORD)TR_ADDREQUEST); /* Commande */
}
```

Ici, on transmet au bloc du device Timer les secondes et les microsecondes indiquées en tant que paramètres, puis on appelle la commande TR_ADDREQUEST. Cette routine ne revient que lorsque le temps indiqué s'est écoulé. Notez qu'avec UNIT_BLANK, l'indication exacte du temps en micro-secondes est inutile. Entrez ici la valeur 0. Voyons maintenant de plus près la structure du TimerRequest :

Offset	Structure
0 0x00	struct IORequest tr_node;
32 0x20	struct timeval tr_time;
40 0x28	; /* défini dans "devices/timer.h" */

La structure timeval contenue dans cette structure se présente ainsi :

Offset	Structure
0 0x00	ULONG tv_secs;
4 0x04	ULONG tv_micro;
8 0x08	; /* défini dans "devices/timer.h" */

Dans cette structure sont définis tous les laps de temps définis ou lus. Si vous voulez par exemple lire l'heure système, vous la transférez dans cette variable :

```
*****
*           GetSysTime()        (Timer_Support)*
*
* Fonction: Obtenir l'heure système
*****
```

```

*-----*
* Paramètres d'entrée: *
*-----*
* TimeRequest: Timer-Device-Bloc *
*-----*
* Valeur en retour: *
*-----*
* Pointeur sur la structure Timeval du bloc de device Timer *
*****/struct timeval *GetSysTime(TimeRequest)
struct timerequest      *TimeRequest:
{
    Do_Command(TimeRequest, TR_GETSYSTIME);
    return(&TimeRequest->tr_time);
}

```

Avec Time1 = (struct timeval *) GetSysTime(TimeRequest), vous obtenez dans Time1 l'adresse de la structure timeval du bloc de device avec l'heure système actuelle. L'heure système indique l'heure et la date actuelles. Dans les ordinateurs possédant une horloge à accu, l'heure système ne pose aucun problème. Mais que fait l'utilisateur qui ne possède pas une telle horloge?

Il doit régler lui-même à chaque fois l'heure et la date. Il peut toutefois se consoler. Lorsqu'on écrit en effet sur une disquette, l'heure de cette modification est enregistrée dans le bloc de bootage, elle est lue ensuite à chaque bootage, et déclarée heure système. Evidemment, ici aussi il peut surgir avec le temps de grandes inexactitudes, et il faut les supprimer à l'aide de la commande CLI Date si on ne veut pas renoncer à avoir l'heure exacte.

Expliquons maintenant comment est décodée l'heure système. On ne doit pas croire que l'horloge interne compte toutes les secondes qui se sont écoulées depuis la naissance du Christ. Les secondes sont comptées, certes, mais à partir du 1.1.1978. Si vous voulez afficher l'heure système, normalement indiquée en secondes et micro-secondes, dans un format compréhensible, vous devez effectuer les conversions suivantes :

```

ULONG Days_of_Months[] = {31L, /* Jan */ /* Combien de jours dans */
                           28L, /* Feb */ /* chaque mois? */ 31L, /* Mar */
                           30L, /* Abr */
                           31L, /* Mai */
                           30L, /* Jun */
                           31L, /* Jul */
                           31L, /* Aug */
                           30L, /* Sep */
                           31L, /* Oct */
                           30L, /* Nov */
                           31L /* Dec */};

char *Days_of_Week[] = {"Sunday", /* Jours de la semaine */
                       "Monday",
                       "Tuesday",
                       "Wednesday",
                       "Thursday",
                       "Friday",
                       "Saturday"};

```

```

*****
*          AnnBiss()          (TimerSupport)  *
*
* Fonction: L'année est-elle bissextile      ?
*-
* Paramètres d'entrée:
*-
* Year: Année à examiner
*-
* Valeur en retour:
*-
* TRUE: Year est bissextile
* FALSE: Year n'est pas bissextile
*****
```

BOOL AnnBiss(Year)

ULONG Year;

{

if (((Year/4L)*4L) == Year) &&
 (((Year/100L)*100L) != Year)) return(TRUE);
 else return(FALSE);
 /* Si Year divisible par 4, mais pas par*/
 /* 100 Year est bissextile */

}

```

*****
*          MakeMonthTable()          (TimerSupport)  *
*
* Fonction: Mise à jour du tableau des mois (29 ou 28 février?)*
*-
* Paramètres d'entrée:
*-
* Year: Année à examiner
* Table: Adresse du tableau des jours de mois (par exemple
*        Days_of_Month)
*****
```

VOID MakeMonthTable(Year, Table)

ULONG Year;

ULONG *Table;

{

if (AnnBiss (Year))
 Table[1] = 29;
 else
 Table[1] = 28;
 /* Si Year est bissextile, définir le nombre de jour en février à */
 /* 29 sinon à 28 */
}

```

*****
*          SysTime_to_TimeDate()          (TimerSupport)  *
*
* Fonction: Transférer l'heure système dans la structure
*           TimeDate
*-
* Paramètres d'entrée:
*-
* TimeRequest: Timer-Device-Bloc
* TimeDate: Structure TimeDate à remplir
*****
```

VOID SysTime_to_TimeDate;(TimeRequest, TimeDate)

```

struct timerequest      *TimeRequest;
struct TimeDate          *TimeDate;
{
    struct timeval *SysTime;      /* pour GetSysTime */
    ULONG SysTimeSecs;
    ULONG all_Days;              /* Combien de jours depuis le 1.1
1978 */
    ULONG year_Days;             /* Combien de jours dans l'année */
    UWORLD leap_year, years;    /* Variables de boucle */
    ULONG month;
    SysTime = GetSysTime(TimeRequest); /* Prendre l'heure système*/
    SysTimeSecs = SysTime->tv_secs; /* Extraire les secondes */
    if (SysTimeSecs>Secs_1980)      /* Au-delà de 1980? */
    {
        SysTimeSecs -= Secs_1980; /* Oui, alors soustraire (1980-1978-2)
*/
        TimeDate->Year = 1980L; /* Années de la date système */
        all_Days = (ULONG)(2*365); /* Définir l'année avec 1980+*/
        /* le nombre de jours depuis */
        /* "Heure zéro" (1.1 1978) */
        /* sur 2*365 */
    }
    else
    {
        all_Days = 0L;           /* Date antérieure à 1980 */
        TimeDate->Year = 1978L; /* Année = 1978, jours passés = 0 */
        if (SysTimeSecs > SecondsPerYear)
        {
            /* 1979? */
            SysTimeSecs -= SecondsPerYear; /* oui */
            all_Days += 365L;
            TimeDate->Year++;
        }
        goto Get_Month;           /* car année déjà définie */
        /* passer au calcul du mois*/
    }
    for (leap_year = 0; leap_year < 34; leap_year++)
    {
        /* Calcul de l'année exacte*/
        if (SysTimeSecs >= SecondsPerLeapYear) /* (postérieur à 1980) */
        {
            SysTimeSecs -= SecondsPerLeapYear; /* Année bissextile */
            TimeDate->Year++;
            all_Days += 366L;
        }
        for (years = 0; years < 3; years++)
        {
            if (SysTimeSecs >= SecondsPerYear) /* Trois années normales */
            {
                SysTimeSecs -= SecondsPerYear;
                TimeDate->Year++;
                all_Days += 365L;
            }
        }
    }
    MakeMonthTable(TimeDate->Year, Days_of_Months); /* Février = 28 ou 29
jours? */
    /* ! 1978 & 1979
n'étaient pas bissextiles! */
Get_Month:
    year_Days = 0L;           /* jours dans l'année = 0 */
    for (month = 0L; month < 11L; month++)
    {

```

```

    if (SysTimeSecs >= (Days_of_Months[month]*SecondsPerDay))
    {
        SysTimeSecs -= (Days_of_Months[month]*SecondsPerDay);
        all_Days += Days_of_Months[month];
        year_Days += Days_of_Months[month];
    }
    else break;
}
TimeDate->Month = month+1; /* car sinon 0 = Janvier, etc. */
TimeDate->Day = (SysTimeSecs/SecondsPerDay)+1L; /* Jour 0 d'un */
                                                 /* est le premier du mois
*/
SysTimeSecs -= (SysTimeSecs/SecondsPerDay)*SecondsPerDay;
year_Days += TimeDate->Day-1L;                      /* Ex: Le 02.02. on a */
all_Days += TimeDate->Day-1L;                        /* 32 jours passés */
TimeDate->Hour = SysTimeSecs/SecondsPerHour;          /* Heure */
SysTimeSecs -= (SysTimeSecs/SecondsPerHour)*SecondsPerHour;
TimeDate->Mins = SysTimeSecs/SecondsPerMinute;         /* Minute */
SysTimeSecs -= (SysTimeSecs/SecondsPerMinute)*SecondsPerMinute;

TimeDate->Secs = SysTimeSecs;                         /* seconde */
TimeDate->Week = year_Days/7;                          /* Semaine dans
l'année */
TimeDate->Week_Day = all_Days % 7;
}                                                       /* Jour de la semaine (1.1 1978 était un
dimanche (0)) */

```

Parmi ces trois routines, la seule qui ait une importance réelle est `SysTime_toTimeDate()`. Cette routine remplit une structure créée par nos soins, et facilement interprétable :

Offset	Structure
-----	-----
	struct TimeDate
	{
0 0x00	ULONG Year; /* Année */
4 0x04	ULONG Month; /* Mois */
8 0x08	ULONG Day; /* Jour du mois */
12 0x0c	ULONG Week; /* Semaine de l'année */
16 0x10	ULONG Week_Day; /* Jour de la semaine (0=dimanche, etc.) */
20 0x14	ULONG Hour; /* Heure */
24 0x18	ULONG Mins; /* Minute */
28 0x2c	ULONG Secs; /* Seconde */
	}

Voici ce qu'il faut savoir sur le fonctionnement de la routine, et il faut bien avouer qu'elle n'est pas particulièrement facile à comprendre. Une fois que vous avez compris le système de décodage, elle ne devrait poser cependant aucun problème. Lorsque l'heure système est chargée, et en supposant que l'on ne s'occupe que des secondes, en laissant de côté les micro-secondes, la routine examine si l'heure système actuelle se trouve ou non avant l'année 1980. En effet, à partir de 1980, on définit les années passées tout à fait simplement. 1980 étant une année bissextile, on peut examiner les années suivantes à l'aide d'une boucle.

Pour savoir combien d'années se sont écoulées, on retire de l'heure système actuelle le nombre des secondes écoulées en une année jusqu'à ce que le résultat de cette soustraction soit inférieur à une année. On doit ici tenir compte des années bissextilles,

mises en évidence par la fonction AnnBiss(). Une fois que l'année a été déterminée, on peut calculer le mois. Puisque les douze mois ont des nombres de jours différents, la soustraction expliquée ci-dessus est un peu plus compliquée, mais elle fonctionne. Si l'année est une année bissextille, la routine MakeMonth() permet de définir à 29 le nombre de jours du mois de février.

Le jour du mois se calcule à l'aide du nombre de secondes restantes, divisé par le nombre de secondes en un jour. On fait de même pour les heures, et les minutes. Lorsqu'on soustrait les secondes, on soustrait en fait le nombre calculé de jours/heures/minutes du nombre de secondes par jours/heures/minutes. Les secondes restantes donnent le nombre de secondes écoulées dans la minute commencée.

Pour calculer le jour de la semaine, on ajoute dans tous les calculs le nombre de jours écoulés. Le reste de la division par 7 de ces jours écoulés depuis le 1.1.1978 jusqu'à aujourd'hui donne le jour de la semaine. Avec `printf("%s\n", Days_of_week [TimeDate->WeekDay]);` on peut afficher le jour correspondant. Nous avons eu besoin des constantes suivantes pour les calculs dans la routine ci-dessus :

```
#define SecondsPerMinute    (ULONG)(60L)
#define SecondsPerHour       (ULONG)(60L*60L)
#define SecondsPerDay        (ULONG)(60L*60L*24L)
#define SecondsPerYear       (ULONG)(60L*60L*24L*365L)
#define SecondsPerLeapYear   (ULONG)(60L*60L*24L*366L) /* AnnBiss */
#define Secs_1980            (ULONG)(2L*SecondsPerYear)
                                         /* Secondes depuis le */
                                         /* 1.1.1978 jusqu'au */
                                         /* 1.1.1980 */
```

Nous avons évidemment prévu une routine pour le calcul inverse. A partir d'une structure TimeDate donnée, on calcule et l'on définit la nouvelle heure système :

```
*****  
*          TimeDate_to_SysTime()      (TimerSupport)*  
*  
* Fonction: TimeDate devient la date système  
*-----  
* Paramètres d'entrée:  
*  
* TimeRequest: Timer-Device-Bloc  
* TimeDate:     Structure TimeDate, destinée à devenir date  
*             système  
*****  
BYTE TimeDate_to_SysTime;(TimeRequest, TimeDate)  
struct timerequest      *TimeRequest;  
struct TimeDate           *TimeDate;  
{  
    ULONG i;  
    ULONG SysTimeSecs=0L;  
    if (TimeDate->Hour > 24) return(TDERR_HOUR_OUT_OF_RANGE);  
    if (TimeDate->Mins > 59) return(TDERR_MINS_OUT_OF_RANGE);  
    if (TimeDate->Secs > 59) return(TDERR_SECS_OUT_OF_RANGE);  
    if ((TimeDate->Year < 19781) && (TimeDate->Year > 21141))  
        return(TDERR_YEAR_OUT_OF_RANGE);           /* test de TimeDate  
*/  
    if (TimeDate->Month > 12)
```

```

        return(TDERR_MONTH_OUT_OF_RANGE);
    if ((TimeDate->Day > Days_of_Months[TimeDate->Month-1]) ||
        (TimeDate->Day == 0))
        return(TDERR_DAY_OUT_OF_RANGE);
    SysTimeSecs = TimeDate->Hour*SecondsPerHour+
                  TimeDate->Mins*SecondsPerMinute+
                  TimeDate->Secs+
                  (TimeDate->Day-1)*SecondsPerDay; /* Heures, minutes,
secondes et nombre */
                                         /* de jours à
convertir en secondes */
    MakeMonthTable(TimeDate->Year, Days_of_Months); /* février 28 ou 29
jours? */
    for (i=0L;i<(TimeDate->Month-1);i++)
        SysTimeSecs += Days_of_Months[i]*SecondsPerDay; /* mois en
secondes */
    for (i=1978L;i<TimeDate->Year;i++)
        if (AnnBiss (i)) SysTimeSecs += SecondsPerLeapYear;
        else             SysTimeSecs += SecondsPerYear; /* années en
secondes */
    SetSysTime(TimeRequest, SysTimeSecs, 0L);
}                                         /* Définir la nouvelle date système
*/
*/

```

Cette routine multiplie simplement les valeurs provenant de la structure TimeDate par le nombre de secondes pour l'année, le jour, le mois, etc (en prenant en compte les années bissextiles), et elle fait la somme de ces résultats. La dernière étape est le définition de l'heure système à partir de ces résultats :

```

/********************* SetSysTime() (TimerSupport) *****/
/*
 * Fonction: Définir l'heure système
 * -----
 * Paramètres d'entrée:
 *   *
 *   * TimeRequest: Timer-Device-Bloc
 *   * Secs, Micro: Nouvelle heure système en secondes et
 *   *               micro-secondes
 */
VOID SetSysTime;(TimeRequest, Secs, Micro)
struct timerequest *TimeRequest;
ULONG                     Secs, Micro;
{
    TimeRequest->tr_time.tv_secs = Secs;
    TimeRequest->tr_time.tv_micro = Micro;
    TimeRequest->tr_node.io_Command = TR_SETSYSTIME;
    DoIO(TimeRequest);
}

```

C'est tout pour les commandes de device. Comment se présentent maintenant les commandes SubTime(), AddTime() et CmpTime()? On transmet toujours à ces commandes deux structures timeval, qui contiennent évidemment deux valeurs pour l'heure. Par exemple: SubTime(timeval1, timeval2). Le résultat de cette commande se déduit de son nom. SubTime() soustrait l'heure de la seconde structure timeval de la première, et conserve le résultat dans la première structure timeval.

AddTime() ajoute les deux structures timeval et conserve le résultat dans la première structure timeval. CmpTime() compare les deux structures timeval. Le résultat de cette fonction est 0 en cas d'égalité, >0 si timeval1 > timeval2, et <0 si timeval1 < timeval2. Pour appeler ces fonctions, il faut d'abord ouvrir la TimerBase. Dans le cas présent, cette ouverture se fait très simplement :

```
struct Device *TimerBase;
...
    TimerBase = TimerRequest->tr_node.io_Device;
```

Vous disposez maintenant des routines AddTime(), SubTime() et CmpTime(). Le programme suivant les utilise pour jouer avec l'heure système :

```
*****
*          Timer-Device
*          (c) Bruno Jennrich
*          Juin 1988
*
*****/
*****/
*  Compile-Info: (TimerComp)
*
*  cc Timer
*  ln Timer.o Timer_Support.o Devs_Support.o -lc
*****/
#include "exec/types.h"
#include "exec/nodes.h"
#include "exec/lists.h"
#include "exec/memory.h"
#include "exec/ports.h"
#include "exec/libraries.h"
#include "exec/io.h"
#include "exec/devices.h"
#include "devices/timer.h"
#include "Timer_Support.h"
VOID *GetSysTime();
extern ULONG Days_of_Month[];
extern char *Days_of_Week[];
struct TimeDate TimeDate;
struct timerequest *TimeRequest=0L;
struct timeval *SystemTime=0L, OneDay = { (ULONG)SecondsPerDay, 0L};
struct Device *TimerBase;
*****
*          CloseIt()          (User)*
*
* Fonction: Afficher l'erreur survenue et tout fermée
*-----
* Paramètres d'entrée:
*
* String: Error-String
*****/
VOID CloseIt(String)
char *String;
{
    WORD i;
    WORD *dff180 = (WORD *)0xdff180;
    WORD Error = 0;
    if (strlen(String) > 0)
```

```

{
    for (i=0;i<0xffff;i++) *dff180 - i;
    puts(String);
    Error = 100;
}
if (TimeRequest != DL) Close_A_Device(TimeRequest);
exit(Error);
}
/*********************************************
*                               (User)*
*
* Fonction: Afficher la structure TimeDate (heure)
* -----
* Paramètres d'entrée:
*
* TimeDate: Structure TimeDate à afficher
*****************************************/
VOID TimePrintout(TimeDate)
struct TimeDate *TimeDate;
{
    printf("%s %02ld %02ld %04ld Time: %02ld:%02ld:%02ld\n",
           Days_of_Week[TimeDate->Week_Day],
           TimeDate->Day,
           TimeDate->Month,
           TimeDate->Year,
           TimeDate->Hour,
           TimeDate->Mins,
           TimeDate->Secs);
}
/*********************************************
*                               (User)*
*
* Fonction: Utiliser Timer-Device et Timer-Support
* -----
*****************************************/
VOID The_Timer_Device()
{
    Open_A_Device("timer.device", (ULONG) UNIT_VBLANK, &TimeRequest, 0L,
    TIME_LEN);
    TimerBase = TimeRequest->tr_node.io_Device;
    printf("System Time is:\n");
    SysTime_to_TimeDate(TimeRequest, &TimeDate);
    TimePrintout(&TimeDate);
    printf("Now add one Day to System Time\n");
    SystemTime = (struct timeval*) GetSysTime(TimeRequest);
    AddTime(SystemTime, &OneDay);
    SetSysTime(TimeRequest, SystemTime->tv_secs, SystemTime->tv_micro);
    printf("System Time now is:\n");
    SysTime_to_TimeDate(TimeRequest, &TimeDate);
    TimePrintout(&TimeDate);
    printf("Subtract the Day from System Time and wait 15 Seconds\n");
    SubTime(SystemTime, &OneDay);
    SetSysTime(TimeRequest, SystemTime->tv_secs, SystemTime->tv_micro);
    WaitTime(TimeRequest, 15L, DL);
    printf("This is Northern-System-Time:\n");
    SysTime_to_TimeDate(TimeRequest, &TimeDate);
    TimePrintout(&TimeDate);
    Close_A_Device(TimeRequest);
}

```

```
*****
*                                main()                               *
*****
```

```
main()
{
    The_Timer_Device();
}
```

4.14. Le device TrackDisk (TrackDisk-device)

Nous allons nous consacrer maintenant au contrôle des lecteurs de l'Amiga. Il existe pour cela un device particulier : le device Trackdisk. Ce device utilise un bloc IOStdReq étendu pour ses opérations :

Offset	Structure
-----	-----
	struct IOExtTD
	{
0 0x00	struct IOStdReq iotd_Req;
48 0x30	ULONG iotd_Count;
52 0x34	ULONG iotd_SecLabel;
56 0x38	} /* défini dans "devices/trackdisk.h" */

Les deux variables supplémentaires vont être expliquées dans ce qui suit. Pour le moment, il importe de savoir quelles sont les fonctions soutenues par le device Trackdisk :

CMD_READ	(2)	lire un secteur
CMD_WRITE	(3)	écrire un secteur (dans TrackBuffer)
CMD_UPDATE	(4)	écrire TrackBuffer sur disquette
CMD_CLEAR	(5)	déclarer TrackBuffer invalide
TD_MOTOR	(9)	activer/désactiver le moteur
TD_SEEK	(10)	positionner la tête de lecture/écriture
TD_FORMAT	(11)	formater un track
TD_REMOVE	(12)	installer un interrupt pour le changement de disquettes
TD_CHAGENUM	(13)	obtenir le nombre de changements de disquettes
TD_CHANGESTATE	(14)	disquette introduite
TD_PROTSTATUS	(15)	tester Write Protect
TD_RAXREAD	(16)	lire un track (raw data)
TD_RAWWRITE	(17)	écrire un track (raw data)
TD_GETDRIVETYPE	(18)	obtenir le type de lecteur (31/2 ou 51/4)
TD_GETNUMTRACKS	(19)	obtenir le nombre de tracks
TD_ADDCHANGEINT	(20)	installer un interrupt pour le changement de disquettes
TD_REMCHANGEINT	(21)	désactiver l'interrupt pour le changement de disquettes

Outre ces commandes normales, il existe également des commandes étendues (extended commands).

Pour ces commandes, le bit 15 est posé dans le numéro de commande. Ces commandes ont les mêmes fonctions que celles qui se trouvent dans la liste ci-dessus, avec cette

différence que cette fois les commandes suivies d'un changement de disquettes ne sont pas mentionnées :

ETD_READ	(32770)	lire un secteur
ETD_WRITE	(32771)	écrire un secteur (dans TrackBuffer)
ETD_UPDATE	(32772)	écrire TrackBuffer sur disquette
ETD_CLEAR	(32773)	déclarer TrackBuffer invalide
ETD_MOTOR	(32777)	allumer/éteindre le moteur
ETD_SEEK	(32778)	positionner la tête de lecture/écriture
ETD_FORMAT	(32779)	formater le track
ETD_RAWREAD	(32784)	lire le track (raw data)
ETD_RAWWRITE	(32785)	écrire le track (raw data)

Pour ces commandes, la variable `iold_Count` est importante (cf. `IOExtTD`). Cette variable contient le nombre de changements de disquettes effectués depuis la dernière opération de bootage. Un "changement de disquettes" ne veut pas dire que vous ayez enlevé la disquette actuelle du lecteur pour la remplacer par une autre disquette. Le simple fait d'enlever une disquette et de la remplacer par une autre augmente le nombre de "changements de disquettes" de 2 unités.

Mais n'allons pas trop vite. Occupons-nous d'abord de la façon d'accéder à un lecteur de disquettes. Il faut pour cela, comme d'habitude, ouvrir un device :

```
struct IOExtTD *DiskExtIO=0L;
#define TD_LEN (ULONG) (sizeof (struct IOExtTD))
...
    OpenDevice("trackdisk.device", Unit, &DiskExtIO, 0L, TD_LEN);
```

Notez que, dès l'ouverture du device, vous devez indiquer le lecteur que vous désirez adresser. Le paramètre `Unit` prend les valeurs 0, 1, 2 ou 3 selon le lecteur de disquettes que vous voulez utiliser (DF0:,... DF3:). Vous ne pouvez ouvrir qu'un seul lecteur avec `Open_A_Device()`. Si vous désirez utiliser plusieurs lecteurs de disquettes, vous devez appeler plusieurs fois `Open_A_Device()` avec des blocs de device différents et des valeurs distinctes pour `Unit`.

4.14.1. Lire et écrire des secteurs

Après `Open_A_Device()`, on peut commencer les opérations, par exemple par la lecture d'un secteur :

```
/*********************************************
*                               TrackDisk_ReadSector()      (Track_Support)*
*
* Fonction: Lecture d'un secteur
*-----
* Paramètres d'entrée:
*
* DiskExtIO:   Device-Bloc
* SectorBuffer: Données du secteur
* LabelBuffer: Données Label-Area
* Offset:      Numéro de secteur
******************************************/
```

```

VOID TrackDisk_ReadSector(DiskExtIO, SectorBuffer, LabelBuffer, Offset)
struct IOExtTD           *DiskExtIO;
APTR                      SectorBuffer;
APTR                      LabelBuffer;
ULONG                     Offset;
{
    DiskExtIO->iotd_Count      = TrackDisk_GetDiskChangeCount(DiskExtIO);
    DiskExtIO->iotd_Req.io_Offset = Offset*512L;
    DiskExtIO->iotd_Req.io_Data   = (APTR) SectorBuffer;
    DiskExtIO->iotd_Req.io_Length = (ULONG) TD_SECTOR;
    DiskExtIO->iotd_SecLabel    = (ULONG) LabelBuffer;
    Do_Command(DiskExtIO, (UWORD) ETD_READ);
}

```

Cette routine utilise la commande ETD_READ, pour lire un secteur de la disquette. On conserve pour cela dans iotd_Count le nombre actuel de changements de disquettes au moyen de TrackDisk_GetDiskChangeCount(). Si la disquette est changée avant la commande ETD_READ, l'exécution de ETD_READ constatera que la valeur dans iotd_Count est inférieure à la valeur actuelle du nombre de changements de disquettes, et ETD_READ ne sera pas exécuté. Outre ETD_READ, toutes les fonctions ETD travaillent selon le même principe, après avoir vérifié iotd_Count.

Vous pouvez évidemment éviter ces opérations en plaçant la valeur 0xffffffff dans iotd_Count. Cette fois, iotd_Count sera toujours supérieur ou égal au nombre actuel de changements de disquettes, mais il vaut mieux utiliser CMD_READ (au lieu de ETD_READ), qui ne se préoccupe pas de iotd_Count.

Revenons cependant à la lecture. Pour faire savoir au device Trackdisk quel secteur il doit lire, on indique dans iotd_Req.io_Offset l'offset du secteur à lire. Il faut noter ici que le device Trackdisk numérote tous les secteurs octet par octet. Pour lire le secteur 0, il faut transmettre au device Trackdisk la valeur 0. Pour le secteur 1, on transmet la valeur 512 (un secteur contient 512 octets), pour le secteur 2, la valeur 1024, etc. Pour ne pas être obligé de multiplier à chaque fois le numéro du secteur à lire par 512, il suffit d'affecter l'offset au bloc de device. On transmet alors uniquement le numéro du secteur à lire (0-1759).

On pourrait avoir l'idée de lire la moitié inférieure du secteur 0 et la moitié supérieure du secteur 1. Il faudrait pour cela indiquer un offset égal à 256. Malheureusement, cette technique n'est pas autorisée. L'offset doit toujours être un multiple de 512. Le tampon dans lequel vous pouvez lire les données du secteur doit lui aussi avoir une taille d'au moins 512 octets (et se trouver dans la mémoire Chip). Si vous désirez lire plus d'un secteur, vous devez avoir à votre disposition plus de mémoire en un seul morceau. La routine ci-dessus ne lit à chaque fois qu'un seul secteur. Dans SectorBuffer est transmise l'adresse de début de la mémoire des données, qui doit avoir une taille d'au moins 512 octets.

Si vous indiquez comme valeur pour la longueur de SectorBuffer, un nombre inférieur à 512 octets, seuls par exemple les 200 premiers octets seront lus. Les autres octets resteront inaccessibles. Outre les données du SectorBuffer, un secteur sur disquette contient encore un domaine avec des données: le LabelBuffer. Celui-ci est placé avant chaque secteur proprement dit, et il a une taille de 16 octets. On trouve ici uniquement

des octets nuls. Mais vous pouvez par exemple utiliser ce LabelBuffer comme mémoire de données supplémentaire, ou y copier les données du copyright.

Pour la lecture, vous devez mettre à la disposition de chaque secteur à lire 16 octets de LabelBuffer. Cette mémoire doit être en un seul morceau lorsqu'il y a plusieurs secteurs à lire. L'écriture se fait de manière analogue :

```
/*
 *          TrackDisk_WriteSector()      (Track_Support)*
 *
 * Fonction: Ecrire dans un track
 *-----
 * Paramètres d'entrée:
 *
 * DiskExtIO:    Device-Bloc
 * SectorBuffer: Données du secteur
 * LabelBuffer:  Données LabelBuffer
 * Offset:       Numéro de secteur
 */
VOID TrackDisk_WriteSector(DiskExtIO, SectorBuffer, LabelBuffer, Offset)
struct IOExtTD           *DiskExtIO;
APTR                      SectorBuffer;
APTR                      LabelBuffer;
ULONG                     Offset;
{
    DiskExtIO->iotd_Count      = TrackDisk_GetDiskChangeCount(DiskExtIO);
    DiskExtIO->iotd_Req.io_Offset = Offset*512L;
    DiskExtIO->iotd_Req.io_Data  = (APTR) SectorBuffer;
    DiskExtIO->iotd_Req.io_Length = (ULONG) TD_SECTOR;
    DiskExtIO->iotd_SecLabel   = (ULONG) LabelBuffer;
    Do_Command(DiskExtIO, (UWORD) ETD_WRITE);
    Do_Command(DiskExtIO, (UWORD) ETD_UPDATE);
}
```

Il faut transmettre ici aussi un SectorBuffer et un LabelBuffer. Cette fois, les tampons contiennent des données devant être écrites sur la disquette. L'offset aussi est indiqué comme avec TrackDisk_ReadSector(). La seule différence avec ReadSector() tient dans le fait qu'une commande comme UPDATE est exécutée après la commande WRITE. Cette commande a pour effet d'écrire le secteur physiquement sur la disquette.

ETD_WRITE et CMD_WRITE sont uniquement responsables de l'opération consistant à écrire les données d'abord dans un tampon du device Trackdisk, ce tampon devant être reporté ensuite sur la disquette au moment de l'accès à un nouveau track. Le tampon interne contient assez de place pour recevoir un track entier (11 secteurs). Avec les commandes WRITE et READ, l'accès se réduit le plus souvent uniquement à ces tampons. Cela permet d'avoir des accès très rapides. Lorsqu'on accède à un nouveau track, et à ce moment-là seulement, l'ancien TrackBuffer est retourné, ou alors un nouveau track est lu dans le tampon interne. Avec AddBuffers, vous pouvez agrandir le tampon interne, ce qui permet encore d'augmenter la vitesse.

Les problèmes commencent lorsque vous écrivez un secteur dans le tampon interne à l'aide de ETD_WRITE ou CMD_WRITE, et que vous essayez de quitter le programme. Il peut arriver alors que les nouvelles données ne soient pas reportées sur la disquette. C'est pourquoi on exécute la commande UPDATE après une commande d'écriture, ce

qui assure l'écriture instantanée du tampon interne sur la disquette. Pour l'utilisation de la commande Extended, il importe de connaître le nombre actuel de changements de disquettes. On s'aide pour cela de la commande TD_CHANGENUM :

```
*****
*           TrackDisk_GetDiskChangeCount()      (Track_Support)*
*
* Fonction: Retrouver le nombre de changements de disquettes *
*-----*
* Paramètres d'entrée:                                         *
*-----*
* DiskExtIO: Device-Bloc                                         *
*-----*
* Valeur en retour:                                              *
*-----*
* Nombre de changements de disquette                           *
*****
ULONG TrackDisk_GetDiskChangeCount(DiskExtIO)
struct IOExtTD          *DiskExtIO;
{
    Do_Command(DiskExtIO, (UWORD)TD_CHANGENUM);
    return(DiskExtIO->iotd_Req.io_Actual);
}
```

Après TD_ChangeNUM, on trouve dans io_Actual le nombre actuel de changements de disquettes. La commande qui active et désactive le moteur du lecteur de disquettes adressé est aussi très importante :

```
*****
*           TrackDisk_Motor()      (Track_Support)*
*
* Fonction: Activer et désactiver le moteur                   *
*-----*
* Paramètres d'entrée:                                         *
*-----*
* DiskExtIO: Device-Bloc                                         *
* Flag: TRUE  -> Moteur activé                               *
*        FALSE -> Moteur désactivé                            *
*****
VOID TrackDisk_Motor(DiskExtIO, Flag)
struct IOExtTD          *DiskExtIO;
BOOL                    Flag;
{
    if (Flag)  DiskExtIO->iotd_Req.io_Length = 1L; /* Motor an */
    else      DiskExtIO->iotd_Req.io_Length = 0L; /* Motor aus */
    Do_Command(DiskExtIO, (UWORD) TD_MOTOR);
}
```

Cette commande est importante car les commandes de lecture et d'écriture activent certes le moteur, mais elles ne l'éteignent pas à la fin de l'opération. La désactivation du moteur revient à l'utilisateur. La routine ci-dessus réalise la désactivation très simplement: avec TrackDisk_Motor (DiskExtIO, FALSE). Cette routine écrit un 0 dans l'élément Length de la structure DiskExtIO. Avec TrackDisk_Motor (DiskExtIO, TRUE), on obtient un 1 écrit dans io_Length, ce qui permet de placer le moteur sur High, c'est-à-dire de l'activer. Dans io_Actual, on obtient l'état précédent du moteur (0=désactivé, 1=activé).

4.14.2. Ecriture et lecture des tracks non traités

Outre l'écriture et la lecture des différents secteurs, le device Trackdisk propose aussi des commandes destinées à lire et à écrire dans des tracks entiers. Il se pose toutefois ici un petit problème. Les données lues et écrites n'ont pas le format d'octet habituel et facilement compréhensible, auquel nous sommes familiarisés pour l'avoir rencontré dans ETD_READ et ETD_WRITE.

Comme vous le savez sans doute, les bits des octets sont codés avant d'être physiquement reportés sur la disquette. Avec l'Amiga, on utilise le codage MFM. Cela nous mènerait trop loin ici de décrire le format MFM. Pour nous, il importe seulement de savoir que les commandes lisant ou écrivant dans le track lisent les données de la disquette comme elles sont physiquement. Cela veut dire que les bits Sync et Valid sont lus également, ce qui présente l'avantage de permettre également la lecture des formats étrangers, comme ceux d'Atari.

Les disquettes qui utilisent le format d'enregistrement GCR peuvent être lues également. Pendant l'opération d'écriture, vous devez vous rappeler que les données à écrire sont déjà codées dans le format adéquat. Voici la routine de lecture et d'écriture :

```
#define RAW_TRACK_LEN 0x397cL
/*********************************************
*           TrackDisk_RawReadSector()  (Track_Support)*
*
* Fonction: Lecteur des tracks (non traité!)
*-----*
* Paramètres d'entrée:
*
* DiskExtIO: Device-Bloc
* TrackBuffer: Données du track (0x397c octets)
* Offset: Numéro de track
*****************************************/
VOID TrackDisk_RawReadSector(DiskExtIO, TrackBuffer, Offset)
struct IOExtTD          *DiskExtIO;
APTR                      TrackBuffer;
ULONG                     Offset;
{
    if (Offset > 159) DiskExtIO->iotd_Req.io_Error = 0xfc;
    else
    {
        DiskExtIO->iotd_Count = TrackDisk_GetDiskChangeCount(DiskExtIO);
        DiskExtIO->iotd_Req.io_Offset = Offset;
        DiskExtIO->iotd_Req.io_Data = (APTR) TrackBuffer;
        DiskExtIO->iotd_Req.io_Length = RAW_TRACK_LEN;
        DiskExtIO->iota_Req.io_Flags = (BYTE) IOTDF_INDEXSYNC;
        DiskExtIO->iotd_Req.io_Actual = 0L;
        Do_Command(DiskExtIO, (UWORD) ETD_RAWREAD);
    }
}
/*********************************************
*           TrackDisk_RawWriteSector()  (Track_Support)*
*
* Fonction: Ecrire dans un track (non traité)
*-----*
```

```

* Paramètres d'entrée: *
* DiskExtIO: Device-Bloc
* TrackBuffer: Données du track
* Offset: Numéro de track
*****
VOID TrackDisk_RawWriteSector(DiskExtIO, TrackBuffer, Offset)
struct IOExtTD           *DiskExtIO;
APTR                      TrackBuffer;
ULONG                     Offset;
{
    if (Offset > 159) DiskExtIO->iotd_Req.io_Error = 0xfc;
    else
    {
        DiskExtIO->iotd_Count = TrackDisk_GetDiskChangeCount(DiskExtIO);
        DiskExtIO->iotd_Req.io_Offset = Offset;
        DiskExtIO->iotd_Req.io_Data = (APTR) TrackBuffer;
        DiskExtIO->iotd_Req.io_Length = (ULONG) RAW_TRACK_LEN;
        DiskExtIO->iotd_Req.io_Flags = (BYTE) IOTDF_INDEXSYNC;
        Do_Command(DiskExtIO, (UWORD) ETD_RAWWRITE);
    }
}
}

```

Il faut noter qu'un track a besoin de 0x397c octets au lieu de $11 \times 512 = 0x1600$ octets. A l'aide de cette commande, il doit être possible d'attendre l'apparition du trou d'index, et de commencer alors seulement l'écriture et la lecture des données. Pour cela, on pose le flag IOTDF_INDEXSYNC. Malheureusement, une erreur de hardware s'est glissée ici, effaçant le bit du registre de hardware avant l'accès à la disquette.

Puisque ces commandes ne respectent pas le format des données, seuls les tracks (0-160) peuvent être lus et écrits. Cela tient au fait que les positions de la tête de lecture/écriture sont définies pour chaque track. Vous n'avez pas besoin de multiplier ici le numéro de secteur par 512, comme pour ETD_READ et ETD_WRITE. Il suffit d'indiquer le numéro du track à lire.

4.14.3. Formatage d'une disquette

Nous avons présenté toutes les commandes permettant de travailler utilement avec le device Trackdisk. Mais le device Trackdisk a encore d'autres possibilités. Il permet par exemple de formater les différents tracks pris un à un. Ceci est particulièrement utile par exemple lorsque le bloc de bootage a été détruit pour une raison quelconque, et donc que le DOS refuse l'accès à cette disquette. Supposons qu'il n'y ait pas de données importantes sur le track 0 (secteur 0 à 10 compris); vous pouvez formater le track 0, et copier le bloc de bootage d'une disquette intacte sur les secteurs fraîchement formatés, à l'aide de READ et WRITE. Il est possible ensuite de sauver les fichiers les plus importants sur une disquette, en supposant que le reste de la disquette est intact :

```

#define MEMTYPE (ULONG) MEMF_CLEAR|MEMF_CHIP
*****
*             TrackDisk_Format()      (Track_Support)*
*
* Fonction: Formater un track

```

```

*-----*
* Paramètres d'entrée:          *
*-----*
* DiskExtIO: Device-Bloc        *
* Offset:   Numéro de track    *
*****/                           *
VOID TrackDisk_Format(DiskExtIO, Offset)
struct IOExtTD      *DiskExtIO;
ULONG               Offset;
{
    BYTE *FormatData=OL;
    WORD i;
    FormatData = (BYTE *) AllocMem(NUMSECS*TD_SECTOR, MEMTYPE);
    if (FormatData == OL)
        CloseIt("No FormatData Buffer !!!");
    for (i=0; i<NUMSECS*TD_SECTOR; i+= 4)
    {
        FormatData[i] = ' ';
        FormatData[i+1] = 'd';
        FormatData[i+2] = 'b';
        FormatData[i+3] = '\0';
    }
    DiskExtIO->iotd_Count = TrackDisk_GetDiskChangeCount(DiskExtIO);
    DiskExtIO->iotd_Req.io_Data = (APTR) FormatData;
    DiskExtIO->iotd_Req.io_Length = (ULONG) (NUMSECS*TD_SECTOR);
    DiskExtIO->iotd_Req.io_Offset = Offset;
    Do_Command(DiskExtIO, (WORD)ETD_FORMAT);
    FreeMem(FormatData, NUMSECS*TD_SECTOR);
}

```

Au contraire de ce qui se passe avec READ et WRITE, le numéro du track à formater (0-160) est transmis dans io_Offset. Une multiplication par 512 n'est donc pas nécessaire. Pour cette commande, on réserve de la mémoire pour un track entier, et on écrit dedans avec "db". Ensuite, le tampon de formatage est transmis au pointeur io_Data. Notez que vous pouvez formater seulement un ou plusieurs tracks entiers à l'aide de TD_FORMAT et ETD_FORMAT. Avec un track à formater, la taille du FormatBuffer est de NUMSECS * TD_SECTOR (11*512) octets. Pour plusieurs tracks à formater, la taille de la mémoire augmente en conséquence.

Comme vous le savez sûrement, l'écriture dans les tracks et les secteurs lors du formatage se fait selon un modèle déterminé. Nous ne décrirons pas ici ce modèle, car cela est hors de notre sujet. Mais si vous en avez le loisir et le désir, vous pouvez examiner une disquette nouvellement formatée à l'aide du moniteur de disquettes reproduit dans ce chapitre, et analyser le modèle de formatage.

4.14.4. Commandes d'état

Il existe également des commandes qui vous renseignent sur l'état de la disquette et sur celui du lecteur de disquettes :

```

*****                                                 *
*           TrackDisk_GetProtStatus()  (Track_Support)*
*

```

```

* Fonction: protection contre l'écriture active/non active?      *
*-----*-----*-----*-----*-----*-----*-----*-----*-----*
* Paramètres d'entrée:                                         *
*-----*-----*-----*-----*-----*-----*-----*-----*-----*
* DiskExtIO: Device-Bloc                                         *
*-----*-----*-----*-----*-----*-----*-----*-----*-----*
* Valeur en retour:                                              *
*-----*-----*-----*-----*-----*-----*-----*-----*-----*
* 0: Non protégé contre l'écriture <>0: Protégé contre          *
*                               l'écriture                         *
*****/*****/*****/*****/*****/*****/*****/*****/*****/*****/*****/
ULONG TrackDisk_GetProtStatus(DiskExtIO)
struct IOEXTD           *DiskExtIO;
{
    Do_Command(DiskExtIO, (UWORD) TD_PROTSTATUS);
    return(DiskExtIO->iotd_Req.io_Actual);
}
*****/*****/*****/*****/*****/*****/*****/*****/*****/*****/*****/
*             TrackDisk_GetChangeState()   (Track_Support)*
*-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
* Fonction: Disquette introduite?                                *
*-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
* Paramètres d'entrée:                                         *
*-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
* DiskExtIO: Device-Bloc                                         *
*-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
* Valeur en retour:                                              *
*-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
* 0: Disquette introduite <>0: Disquette ôtée                  *
*****/*****/*****/*****/*****/*****/*****/*****/*****/*****/*****/
ULONG TrackDisk_GetChangeState(DiskExtIO)
struct IOEXTD           *DiskExtIO;
{
    Do_Command(DiskExtIO, (UWORD) TD_CHANGESTATE);
    return(DiskExtIO->iotd_Req.io_Actual);
}

```

Avec TD_PROTSTATUS, vous obtenez en io_Actual la position de la protection contre l'écriture. Si io_Actual est égal à 0, la disquette n'est pas protégée contre l'écriture. Il en va de même avec TD_CHANGESTATE. Si io_Actual est différent de 0 après cette commande, il n'y a pas de disquette dans le lecteur. Les commandes suivantes retournent leurs valeurs dans io_Actual :

TD_GETDRIVETYPE

A l'aide de cette commande, vous pouvez savoir quel est le type du lecteur qui sera adressé à l'aide du device Trackdisk. Si io_Actual a la valeur 1, c'est un lecteur de 3"1/2. S'il a la valeur 2, c'est un lecteur de 5"1/4 qui a été connecté.

TD_GETNUMTRACKS

Cette commande vous permet de connaître le nombre de tracks que vous pouvez gérer avec le lecteur connecté. Pour les lecteurs de 3"1/2 de l'Amiga, on peut gérer 160 tracks.

TD_SEEK et ETD_SEEK

Ces commandes permettent de définir et de tester la tête de lecture/écriture, pour voir si elle positionnée sur un track déterminé.

Il faut pour cela indiquer le numéro du premier secteur du track en question (multiple de 11). Si on n'a pas pu atteindre le track, on reçoit en retour une erreur (io_Error != 0) :

```
*****
*           TrackDisk_SeekSector()  (Track_Support)*
*
* Fonction: Positionner la tête de lecture
*-----*
* Paramètres d'entrée:
*
* DiskExtIO: Device-Bloc
* Offset:   Numéro de secteur
*****
VOID TrackDisk_SeekSector(DiskExtIO, Offset)
struct IOExtTD          *DiskExtIO;
ULONG                   Offset;
{
    DiskExtIO->iotd_Req.io_Offset = Offset*512L;
    Do_Command(DiskExtIO, (UWORD) TD_SEEK);
}
```

Si cette commande retourne une erreur, cela provient le plus souvent du fait que votre lecteur de disquettes est endommagé!

4.14.5. Disk-Interrupts

Une autre fonctionnalité intéressante du device Trackdisk tient dans la possibilité d'installer une interruption, qui sera exécutée à chaque introduction ou à chaque retrait d'une disquette. La commande qui en est responsable s'appelle TD_REMOVE.

```
*****
*           TrackDisk_InterruptOn()    (Track_Support)*
*
* Fonction: Installer l'interruption pour les changements
*           de disquette
*-----*
* Paramètres d'entrée:
*
* DiskExtIO:       Device-Bloc
* TrackDisk_DiskRemove: Interrupt-Routine
*****
VOID TrackDisk_InterruptOn(DiskExtIO, TrackDisk_DiskRemoved)
struct IOExtTD          *DiskExtIO;
VOID                   (*TrackDisk_DiskRemoved)();
{
    Interrupt = (struct Interrupt *) AllocMem((ULONG)(sizeof(struct
    Interrupt)), MEMTYPE);
    if (Interrupt == 0L) CloseIt("No Interrupt !");
```

```

    Interrupt->is_Code           = (VOID (*)()) TrackDisk_DiskRemoved;
    DiskExtIO->iotd_Req.io_Data = (APTR) Interrupt;
    DiskExtIO->iotd_Req.io_Command = (UWORD) TD_REMOVE;
    DoIO(DiskExtIO);
}

```

Cette routine réserve d'abord de la mémoire pour une structure Interrupt. Vous ne devez pas utiliser le nom "Interrupt" dans vos programmes, si vous voulez que cette routine marche. Vous transmettez ensuite au pointeur is_Code de la structure Interrupt l'adresse de votre routine d'interruption. Il n'y a pas de valeur à transmettre au champ is_Data de la structure Interrupt. Ce champ est effacé par l'allocation de mémoire (MEMF_CLEAR). Si vous indiquez ici un secteur de données, celui-ci sera transmis à la routine d'interruption en A1. Une fois que la structure d'interruption a été initialisée, son adresse est transmise au pointeur io_Data du bloc de device, et TD_REMOVE est appelé. L'interruption est maintenant installée. Pour désactiver à nouveau l'interruption, vous pouvez utiliser la routine suivante :

```

/*****
*          TrackDisk_InterruptOff()      (Track_Support)*
*
* Fonction: Désactiver le Disk-Interrupt
*-----*
* Paramètres d'entrée:
*
* DiskExtIO: Device-Bloc
*****/
VOID TrackDisk_InterruptOff(DiskExtIO)
struct IOExTD             *DiskExtIO;
{
    DiskExtIO->iotd_Req.io_Data = 0L;
    DiskExtIO->iotd_Req.io_Command = (UWORD) TD_REMOVE;
    DoIO(DiskExtIO);
    if (Interrupt != 0L)
        FreeMem(Interrupt, (ULONG)(sizeof (struct Interrupt)));
    Interrupt = (struct Interrupt *) 0L; /* pour CloseIt() */
}

```

Cette routine place sur 0 le pointeur io_Data du bloc de device, dirigé auparavant sur la structure interrupt, et appelle une seconde fois TD_REMOVE. La mémoire réservée pour la structure Interrupt est alors libérée. C'est pourquoi Interrupt a été définie de façon globale, et pourquoi il ne doit plus être utilisé dans vos programmes. TrackDisk_InterruptOn() et TrackDisk_InterruptOff() accèdent à Interrupt. Outre TD_REMOVE, il existe à partir de Kick1.2 deux autres commandes pour le traitement du Disk-Interrupt: TD_ADDCHANGEINT et TD_REMCHANGEINT. Avec TD_ADDCHANGEINT, on peut installer une interruption comme avec TD_REMOVE :

```

/*****
*          TrackDisk_AddChangeInt()     (Track_Support)*
*
* Fonction: Installer Disk-Interrupt
*           Attention : TD_REMCHANGEINT ne fonctionne pas !!!
*-----*
* Paramètres d'entrée:
*
* DiskExtIO:       Device-Bloc
*****/

```

```

* TrackDisk_DiskRemoved: Interrupt-Routine *
*****
VOID TrackDisk_AddChangeInt(DiskExtIO, TrackDisk_DiskRemoved)
struct IOExTD           *DiskExtIO;
VOID                      (*TrackDisk_DiskRemoved)();

{
    InterruptIO = (struct IOExTD *)GetDeviceBloc(TD_LEN);
    TrackDisk_Copy(DiskExtIO, InterruptIO);
    Interrupt = (struct Interrupt *)
        AllocMem((ULONG)(sizeof(struct Interrupt)), MEMTYPE);
    if (Interrupt == 0L) CloseIt("NoInterrupt");
    Interrupt->is_Code      = (VOID(*)()) TrackDisk_DiskRemoved;
    InterruptIO->iotd_Req.io_Data = (APTR) Interrupt;
    InterruptIO->iotd_Req.io_Command = (UWORD) TD_ADDCHANGEINT;
    SendIO(InterruptIO);
}

```

Dans cette routine, on alloue de la mémoire pour la structure Interrupt. Mais on crée aussi un autre bloc de device. Cela tient au fait que la commande TD_ADDCHANGEINT ne revient au programme que lorsque l'interruption est désactivée. Mais comme vous voulez sûrement continuer à travailler entre-temps avec le device Trackdisk, il faut un second bloc de device. Dans ce dernier, les variables les plus importantes sont copiées à l'aide de TrackDisk_Copy() :

```

***** TrackDisk_Copy() (Track_Support) *****
*
* Fonction: Copier un bloc de device
*-----*
* Paramètres d'entrée:
*
* OldExtIO: Original
* NewExtIO: Copie
*****/
VOID TrackDisk_Copy(OldExtIO, NewExtIO)
struct IOExTD      *OldExtIO,*NewExtIO;
{
    NewExtIO->iotd_Req.io_Device =
        OldExtIO->iotd_Req.io_Device;
    NewExtIO->iotd_Req.io_Unit =
        OldExtIO->iotd_Req.io_Unit;
    NewExtIO->iotd_Count =
        OldExtIO->iotd_Count;
}

```

Malheureusement, l'utilisation de TD_ADDCHANGEINT ou de TD_REMCHANGEINT pose un problème. En effet, l'interruption doit évidemment être supprimée au plus tard avant la fermeture du device Trackdisk. La commande proposée pour ce faire (TD_REMCHANGEINT) ne fonctionne malheureusement pas. C'est pourquoi nous vous conseillons plutôt d'installer un Disk-Interrupt au moyen de TrackDisk_AddChangeInt(), car sinon vous ne pourrez plus jamais le supprimer, et cela conduira à de graves problèmes après la fermeture du device Trackdisk.

4.14.6. Traitement des erreurs de disque

Maintenant que toutes les commandes Trackdisk ont été décrites, nous ne vous cacherons pas qu'il se produit très souvent des erreurs, notamment avec les programmeurs peu expérimentés dans ce domaine, à moins que vous n'utilisiez les routines de support du track que nous vous proposons. Outre les erreurs normales de device, il y a quelques erreurs spécifiques :

TDERR_NotSpecified	(20) Erreur non spécifiée
TDERR_NotSecHdr	(21) En-tête de secteur non trouvée
TDERR_BadSecPreamble	(22) Erreur dans le préambule du secteur
TDERR_BadSecID	(23) Erreur dans l'ID du secteur
TDERR_BadHdrSumm	(24) Erreur dans la somme de vérification du header
TDERR_BadSecSum	(25) Erreur dans la somme de vérification du secteur
TDERR_TooFewSecs	(26) Trop peu de secteurs sur le track
TDERR_BadSecHdr	(27) Tête de secteur illisible!
TDERR_WriteProt	(28) Disquette protégée contre l'écriture
TDERR_DiskChanged	(29) Pas de disquette insérée
TDERR_SeekError	(30) Position fausse de la tête de lecture (TD_SEEK)
TDERR_NoMem	(31) Trop peu de mémoire
TDERR_BadUnitNum	(32) Lecteur adressé non connecté
TDERR_BadDriveTyp	(33) Mauvais lecteur de disquettes
TDERR_DriveInUse	(34) Lecteur déjà en cours d'utilisation
TDERR_PostReset	(35) Pas d'accès autorisé après reset

La routine suivante fournit le numéro d'erreur (en cas d'erreur non spécifique du device), ou l'erreur en clair :

```

BYTE          HTab[] = {0, 1, 2, 3, 4, 5, 6, 7,
                      8, 9, A, B, C, D, E, F};
BYTE *ErrorStrings[] = {"Not Specified\012\015",
                        "No Sector Header\012\015",
                        "Bad Sector Preamble\012\015",
                        "Bad Sector ID\012\015",
                        "Bad Header Sum\012\015",
                        "Bad Sector Sum\012\015",
                        "Too Few Sectors\012\015",
                        "Bad Sector Header\012\015",
                        "Write Protected\012\015",
                        "Disk Changed\012\015",
                        "Seek Error\012\015",
                        "Not enough memory\012\015",
                        "Bad Unit Number\012\015",
                        "Bad Drive Type\012\015",
                        "Drive In Use\012\015",
                        "Post Reset\012\015"};

struct StrPack
{
    BYTE  *String;
    ULONG Len;
};

/*********************************************
*           TrackDisk_ProcessError()      (Track_Support)*
*

```

```

* Fonction: Traiter Trackdisk-Error
*-----*
* Paramètres d'entrée:
*
* DiskExtIO: Device-Bloc
* StrPack: String-Packet pour Error-String
*****/
VOID TrackDisk_ProcessError(DiskExtIO, StrPack)
struct IOExtTD           *DiskExtIO;
struct StrPack            *StrPack;
{
    BYTE   Error;
    static BYTE  *ErrStr;
    Error = DiskExtIO->iotd_Req.io_Error;
    StrPack->String = OL;
    StrPack->Len    = OL;
    if (Error != (BYTE) 0)
    {
        if ((Error >= (BYTE) 20) && (Error <= (BYTE) 35))
        {
            StrPack->String = ErrorStrings[(Error-(BYTE)20)];
            StrPack->Len    = (ULONG) strlen (StrPack->String);
        }
        else
        {
            ErrStr     = "\012\015Error # \012\015";
            ErrStr[9]  = HTab[(Error>>4)&15];
            ErrStr[10] = HTab[Error&15];
            StrPack->String = ErrStr;
            StrPack->Len    = 13L;
        }
    }
}
}

```

Si vous appelez cette routine, vous devez transmettre en tant que paramètre l'adresse sur un paquet de chaînes. Dans ce paquet, vous obtiendrez soit le texte en clair de l'erreur, soit encore une chaîne de la forme Error #nn. Vous pouvez demander alors l'affichage de cette chaîne.

Notez que vous devez appeler cette routine directement après exécution par exemple de READ ou WRITE (TD_MOTOR est exécuté le plus souvent sans erreur).

Avec la routine TrackDisk_WriteSector(), on rencontre un petit problème, car la commande UPDATE est exécutée après WRITE. Vous devez appeler ici vous-même UPDATE après ProcessError ou encore intégrer ProcessError dans WriteSector().

4.14.7. L'éditeur de disquette

Y a-t-il quelque chose de plus naturel que d'écrire un éditeur de disquette avec les commandes TrackDisk?

Le programme qui suit vous permet de lire et d'écrire dans des secteurs, de formater des tracks et le connaître l'état de la disquette. En outre, vous avez la possibilité d'écrire

dans un fichier les contenus d'un secteur, puis de demander son affichage par exemple avec type.

```
*****
*                               DiskEd.c
*                               (c) Bruno Jennrich
*                               Août 1988
*
 ****
/*
* Compile-Info:
*
* cc DiskEd.c
* ln DiskEd.o Track_Support.o Con_Support.o Devs_Support.o -lc
*/
#include "exec/types.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "exec/interrupts.h"
#include "graphics/gfxbase.h"
#include "libraries/dos.h"
#include "devices/trackdisk.h"
#include "devices/console.h"
#include "devices/keymap.h"
#include "intuition/intuitionbase.h"
#include "intuition/intuition.h"
#define MEMTYPE  (MEMF_CHIP | MEMF_CLEAR)
#define CON_LEN (ULONG) (sizeof (struct IOStdReq))
#define TD_LEN (ULONG) (sizeof (struct IOExtTD))
#define RAW_TRACK_LEN 0x397c1
VOID *CreatePort();
VOID *CreateExtIO();
VOID *Open();
VOID *AllocMem();
VOID *GetDeviceBloc();
VOID *OpenLibrary();
VOID *OpenScreen();
VOID *OpenWindow();
extern VOID TrackDisk_DiskRemoved();
struct IntuitionBase *IntuitionBase = 0L;
struct Screen      *Screen      = 0L;
struct Window      *Window      = 0L;
struct NewScreen   NewScreen = {
                                0, 0, 640, 255, 4,
                                0, 1,
                                HIRES,
                                CUSTOMSCREEN,
                                0L,
                                (UBYTE*) "No Name",
                                0L,
                                0L
};
struct NewWindow   NewWindow = {
                                0, 0,
                                640, 255,
                                0, 1,
                                0L,
                                (ULONG) ACTIVATE,
```

```

        OL,
        OL,
        (UBYTE*) "Trackdisk-Editor (c) Bruno Jennrich",
        OL,
        OL,
        0, 0,
        0, 0,
        CUSTOMSCREEN
    };
extern struct Interrupt *Interrupt;
extern struct IOExtTD  *InterruptIO;
struct IOExtTD  *DiskExtIO = OL;
struct IOStdReq  *ConsoleRead = OL,
                  *ConsoleWrite = OL;
UWORD           *File     = OL;
BYTE            *SectorBuffer = OL;
BYTE            *TrackBuffer = OL;
BYTE            *LabelBuffer = OL;
BYTE            ReadBuffer[256]; /* Tampon de clavier */
BYTE            HexTab[] = {0, 1, 2, 3, 4, 5, 6, 7,
                           8, 9, A, B, C, D, E, F};
BYTE            ConversionTable[256];
struct StrPack
{
    BYTE  *String;
    ULONG Len;
};
/*********************************************
*                         CloseIt()          (User)*
*
* Fonction: Tout fermer en cas d'erreur
*-----
* Paramètres d'entrée:
*
* String: Error-Message
*
*****************************************/
VOID CloseIt(String)
char      *String;
{
    UWORD i;
    UWORD *dff180 = (UWORD *)0xdff180;
    UWORD Error = 0;
    if (strlen(String) > OL)
    {
        for (i=0;i<0xffff;i++) *dff180 = i;
        puts(String);
        Error = 10;
    }
    if (Window != OL) CloseWindow(Window);
    if (Screen != OL) CloseScreen(Screen);
    if (IntuitionBase != OL) CloseLibrary(IntuitionBase);
    if (Interrupt != OL) TrackDisk_InterruptOff(DiskExtIO);
    if (DiskExtIO->iotd_Req.io_Device != -1L) Close_A_Device(DiskExtIO);
    else FreeDeviceBloc(DiskExtIO);
    if (SectorBuffer != OL)
        FreeMem(SectorBuffer, TD_SECTOR);
    if (TrackBuffer != OL)

```

```

        FreeMem(TrackBuffer, RAW_TRACK_LEN);
if (LabelBuffer != 0L)
    FreeMem(LabelBuffer, (ULONG)(NUMSECS));
if (ConsoleRead != 0)    Close_A_Device(ConsoleRead);
if (ConsoleWrite != 0)   FreeDeviceBloc(ConsoleWrite);
if (File != 0L)          Close (File);
exit(Error);
}
//*********************************************************************
*                         TrackDisk_DiskRemoved()           (User)*
*
* Fonction: Routine d'interruption, appelée lors d'un changement
* de disquette.
//*****************************************************************/
#asm
public _TrackDisk_DiskRemoved
_TrackDisk_DiskRemoved:
move.w #$ffff, d0
loopa:
move.w d0, $dff180
dbra d0, loopa
rts
#endifasm
//*********************************************************************
*                         ReadCommand()           (User)*
*
* Fonction: Lire les saisies au clavier (Console-Device)
*-----
* Paramètres d'entrée:
*
* Buffer: Où vont les saisies?
* MaxLength: Nombre maximal de pressions sur les touches
//*****************************************************************/
VOID ReadCommand(Buffer, MaxLength)
BYTE      *Buffer;
ULONG     MaxLength;
{
BYTE *BufPointer;
BYTE Character;
ULONG Length;

Character = (BYTE) 0;
Length = (ULONG)0;
BufPointer = Buffer;
while (Character != (BYTE)13)                                /* <Return> */
{
    Console_Read(ConsoleRead, &Character, 1L);
    if (Character == (BYTE)8) /* Backspace */
    {
        if (Length > 0L)
        {
            *BufPointer-- = (BYTE) \000; /* Marquer la fin de la chaîne
*/
            Console_Write(ConsoleWrite, &Character, 1L); /* Backspace */
            Console_Write(ConsoleWrite, " ", 1L); /* Effacer des
caractères */
            Console_Write(ConsoleWrite, &Character, 1L);/* Backspace */
Length--;
        }
    }
}

```

```

    }
else
if (Length < MaxLength)
{
    if (Character == (BYTE)13) Console_Write(ConsoleWrite, "\012",
1L);
    if (((Character | (BYTE) 0x20) >= a) &&
        ((Character | (BYTE) 0x20) <= z))

        *BufPointer++ = (Character | (BYTE) 0x20);
    else
        *BufPointer++ = Character;
    Length++;
    Console_Write(ConsoleWrite, &Character, 1L);
}
}
*BufPointer = (BYTE)0;
}
/********************************************* (User)*****
*
* Fonction: Convertir Hex-String en ULONG
* -----
* Paramètres d'entrée:
*
* BufPointer: Adresse du Hex-String
* -----
* Valeur en retour:
*
* Valeur du Hex-String ("ABC" = 0xabc)
*****/
```

HEXAtoULONG()

```

ULONG HEXAtoULONG(BufPointer)
BYTE           *BufPointer;
{
    WORD i;
    ULONG Val = 0L;
    WORD Len;
    Len = strlen(BufPointer);
    for (i=0;i<Len;i++)
    {
        if ((*BufPointer>= 'A') && (*BufPointer <= 'F'))
        {
            Val *= 16L;
            Val += ((*BufPointer && (0xff-0x20))- 'A'+(BYTE)10);
        }
        else
        if ((*BufPointer>= '0') && (*BufPointer <= '9'))
        {
            Val *= 16L;
            Val += (ULONG)(*BufPointer-'0');
        }
        BufPointer++;
    }
    return(Val);
}
/********************************************* (User)*****
*
```

DEZAtoULONG()

```

* Fonction: Convertir string décimal en ULONG
*-----*
* Paramètres d'entrée:
*-----*
* BufPointer: Adresse de la chaîne décimale
*-----*
* Valeur en retour:
*-----*
* Valeur de la chaîne décimale ("123" = 123)
*****/
```

ULONG DEZtoULONG(BufPointer)

BYTE *BufPointer;

{

UWORLD i;

ULONG Val = 0L;

UWORLD Len;

Len = strlen(BufPointer);

for (i=0;i<Len;i++)

{

if ((*BufPointer)>= '0') && (*BufPointer <= '9')

{

Val *= 10L;

Val += (ULONG)(*BufPointer-'0');

BufPointer++;

}

}

return(Val);

}

```

*-----*
*           LONGtoA()          (User)*
*-----*
```

* Fonction: Convertir valeur LONG en ASCII et afficher

* Paramètres d'entrée:

* Val: Valeur LONG

*****/

VOID LONGtoA(Val)

ULONG Val;

{

ULONG Start = 1000000000L;

BYTE ASCII[11];

UWORLD i;

i=0;

do

{

ASCII[i] = 0;

while (Val >= Start)

{

Val -= Start;

ASCII[i]++;

```

}

/*********************************************
*                                     (User)*
*
* Fonction: Convertir valeur UWORLD en Hex-String
*-----
* Paramètres d'entrée:
*
* Val: Valeur UWORLD
* Buffer: Où reporter les chaîne hexadécimales?
/*****************************************/
VOID UWORLDtoHex(Val, Buffer)
UWORD      *Val;
BYTE       *Buffer;
{
    UWORLD Hex;
    Hex     = *Val & 0xf000;
    Hex     = Hex >> 12;
    Buffer[0] = HexTab[Hex];
    Hex     = *Val & 0x0f00;
    Hex     = Hex >> 8;
    Buffer[1] = HexTab[Hex];
    Hex     = *Val & 0x00f0;
    Hex     = Hex >> 4;
    Buffer[2] = HexTab[Hex];
    Hex     = *Val & 0x000f;
    Buffer[3] = HexTab[Hex];
}
/*********************************************
*                                     (User)*
*
* Fonction: Présenter le contenu du secteur
*-----
* Paramètres d'entrée:
*
* Offset: Numéro de secteur (Seulement pour l'affichage!)
/*****************************************/
VOID Display(SectorBuffer, LabelBuffer, Offset)
BYTE      *SectorBuffer;
BYTE      *LabelBuffer;
ULONG     Offset;
{
    UWORLD i, j;
    UWORLD BufPos = 0;
    BYTE String[20];
    BYTE HexBuffer[72*16];
    BYTE ASCIIBuffer[72*8];
    BYTE LabelBuff[8*5];
    BYTE LabelASCII[17];
    BYTE OffsetBuffer[14];
    UWORLD Offs;
    OffsetBuffer[ 0] = 'B';
    OffsetBuffer[ 1] = 'l';
    OffsetBuffer[ 2] = 'o';
    OffsetBuffer[ 3] = 'c';
    OffsetBuffer[ 4] = 'k';
    OffsetBuffer[ 5] = ':';
    OffsetBuffer[ 6] = '.';
    OffsetBuffer[ 7] = '$';
}

```

```

OffsetBuffer[ 8] = ' ';
OffsetBuffer[ 9] = ' ';
OffsetBuffer[10] = ' ';
OffsetBuffer[11] = ' ';
OffsetBuffer[12] = '\012';
OffsetBuffer[13] = '\015';
Offs = (UWORD) Offset;           /* Conversion en UWORD pour */
                                  /* UWORDDtoHex() */          /*
UWORDDtoHex(&Offs, &OffsetBuffer[8]);
Console_Write(ConsoleWrite, OffsetBuffer, 14L);
Console_Write(ConsoleWrite, "LabelBuffer:\012\015", -1L);
for (i=0;i<8;i++)
{
    LabelBuff [i*5] = ' ';
    UWORDDtoHex((LabelBuffer+i*2), &LabelBuff[i*5+1]);
}
Console_Write(ConsoleWrite, LabelBuff, 81*51);
Console_Write(ConsoleWrite, "\012\015", -1L);
LabelASCII[0] = ' ';
for (i=0;i<16;i++)
{
    LabelASCII [i+1] = ConversionTable[(UBYTE)*(LabelBuffer+i)];
}
Console_Write(ConsoleWrite, LabelASCII, 171);
Console_Write(ConsoleWrite, "\012\015\012\015", -1L);
for (i=0;i<16;i++)      /* 16 caractères */

{
    HexBuffer[i*72] = ' ';
    UWORDDtoHex(&BufPos, &HexBuffer[i*72+1]);
    HexBuffer[i*72+5] = ' ';
    for (j=0;j<16;j++) /* 32 Bytes (16 UWORDS) = 64 caractères */
    {
        UWORDDtoHex(&SectorBuffer[BufPos], &HexBuffer[i*72+6+j*4]);
        BufPos += 2;
    }
    HexBuffer[i*72+70] = '\012';
    HexBuffer[i*72+71] = '\015';
}
Console_Write(ConsoleWrite, HexBuffer, 16L*72L);
BufPos = 0;
for (i=0;i<8;i++)
{
    ASCIIIBuffer[i*72] = ' ';
    UWORDDtoHex(&BufPos, &ASCIIIBuffer[i*72+1]);
    ASCIIIBuffer[i*72+5] = ' ';
    for (j=0;j<64;j++)
    {
        ASCIIIBuffer[i*72+6+j] =
            ConversionTable[(UBYTE)SectorBuffer[i*64+j]];
        BufPos++;
    }
    ASCIIIBuffer[i*72+70] = '\012';
    ASCIIIBuffer[i*72+71] = '\015';
}
Console_Write(ConsoleWrite, ASCIIIBuffer, 8L*72L);
if (File != 0)
{

```

```

Write(File, "\012", -1L);
Write(File, OffsetBuffer, 13L);
Write(File, "LabelBuffer: \012", 14L);
Write(File, LabelBuff, 8L*5L);
Write(File, "\012", 1L);
Write(File, LabelASCII, 17L);
Write(File, "\012-----\012", 73L);
for (i=0;i<16;i++)
{
    HexBuffer[i*72+70] = ' ';
    HexBuffer[i*72+71] = '\012';
}
for (i=0;i<8;i++)
{
    ASCIIBuffer[i*72+70] = ' ';
    ASCIIBuffer[i*72+71] = '\012';
}
Write(File, HexBuffer, (72L*16L));
Write(File, "\012", 1L);
Write(File, ASCIIBuffer, (72L*8L));
}

} ****
*          GetOffset()           (User)*
*
* Fonction: Calculer Offset pour READ/WRITE à partir de
*           la chaîne saisie
* -----
* Paramètres d'entrée:
*
* BufPointer: Adresse de la chaîne saisie (HEX ou DEC)
* -----
* Valeur en retour:
*
* Offset calculé
****

ULONG GetOffset(BufPointer)
BYTE      *BufPointer;
{
    BYTE *SecBuf;
    SecBuf = BufPointer;
    while ((*BufPointer != (BYTE)0) && (*BufPointer != '$') && (*BufPointer != '#'))
        BufPointer++;
    if (*BufPointer == (BYTE) 0) return(-1L);
    else
    if (*BufPointer == '$') return(HEXAtoULONG(BufPointer+1));
    else
    if (*BufPointer == '#') return(DEZAtoULONG(BufPointer+1));
    else
    if ((*BufPointer >= '0') && (*BufPointer <= '9'))
        return(DEZAtoULONG (SecBuf));
}
****

*          HandleCommands()         (User)*
*
* Fonction: Traiter les commandes saisies
****

VOID HandleCommands()

```

```

{
    BYTE *BufPointer;
    BYTE Command;
    BOOL QuitFlag;
    struct StrPack StrPack;
    ULONG Count;
    ULONG Offset = 0L;
    QuitFlag = FALSE;
    while (!QuitFlag)
    {
        ReadCommand(ReadBuffer, 256L);
        BufPointer = ReadBuffer;

        while (*BufPointer == ' ') BufPointer++;
        /* Sauter les blancs*/
        Command = *BufPointer;
        /* Le premier caractère différent de l'espacement est celui de la
        commande*/
        switch (Command) /* Quelle commande? */
        {
            case (BYTE)'h':
                /* help */
                Console_Write(ConsoleWrite, "\012\015r#/[$[Bloc]
                    - Read Sector\012\015", 43L);
                Console_Write(ConsoleWrite, "w#/[$[Bloc]
                    - Write Sector\012\015", 41L);
                Console_Write(ConsoleWrite, "f[Track]
                    - Format Track\012\015", 41L);
                Console_Write(ConsoleWrite, "s
                    - Disk Status\012\015", 40L);
                Console_Write(ConsoleWrite, "d
                    - Display Sector\012\015", 43L);
                Console_Write(ConsoleWrite, "q
                    - Quit\012\015", 34L);
                Console_Write(ConsoleWrite, "h
                    - This Reference\012\015", 43L);
            break;
            case (BYTE)'r':
                /* Read */
                Offset = GetOffset(BufPointer);
                TrackDisk_Motor(DiskExtIO, TRUE);
                TrackDisk_ReadSector(DiskExtIO, SectorBuffer, LabelBuffer, Offset);
                TrackDisk_ProcessError(DiskExtIO, &StrPack);
                TrackDisk_Motor(DiskExtIO, FALSE);
                Console_Write(ConsoleWrite, StrPack.String, StrPack.Len);
            break;
            case (BYTE)'w':
                /* Write */
                Offset = GetOffset(BufPointer);
                TrackDisk_Motor(DiskExtIO, TRUE);
                TrackDisk_WriteSector(DiskExtIO, SectorBuffer, LabelBuffer,
                    Offset);
                TrackDisk_ProcessError(DiskExtIO, &StrPack);
                TrackDisk_Motor(DiskExtIO, FALSE);
                Console_Write(ConsoleWrite, StrPack.String, StrPack.Len);
            break;
            case (BYTE)'f':
                Offset = GetOffset(BufPointer);

```

```

        TrackDisk_Motor(DiskExtIO, TRUE);
        TrackDisk_Format(DiskExtIO, Offset);
        TrackDisk_ProcessError(DiskExtIO, &StrPack);
        TrackDisk_Motor(DiskExtIO, FALSE);
        Console_Write(ConsoleWrite, StrPack.String, StrPack.Len);
    break;
case (BYTE)'s':
    Console_Write(ConsoleWrite, "\012\015DiskChangeCount : ", -1L);
    DiskExtIO->iotd_Count = TrackDisk_GetDiskChangeCount(DiskExtIO);
    LONGtoA(DiskExtIO->iotd_Count);
    Console_Write(ConsoleWrite, "\012\015", -1L);
    if (TrackDisk_GetProtStatus(DiskExtIO) != 0L)
        Console_Write(ConsoleWrite, "Disk protected\012\015", -1L);
    else
        Console_Write(ConsoleWrite, "Disk not protected\012\015", -1L);
        if (TrackDisk_GetChangeState(DiskExtIO) == 0L)
            Console_Write(ConsoleWrite, "Disk inserted\012\015", -1L);
        else
            Console_Write(ConsoleWrite, "Disk removed\012\015", -1L);
        /* Status */
    break;
case (BYTE)'d':
    /* Display */
    Display(SectorBuffer, LabelBuffer, Offset);
    break;
case (BYTE)'q':
    OuitFlag = (BOOL)TRUE;
    break;
case (BYTE)'13':
    /* Touche RETURN */
    break;
default :
    Console_Write(ConsoleWrite, "\012\015Bad Command
!!!\012\015", -1L);
    break;
}
}
}*****
*           The_TrackDisk_Device()          (User)*
*
* Fonction: Utiliser le device Trackdisk
*-----
* Paramètres d'entrée:
* Unit: Quel lecteur?
*****/
VOID The_TrackDisk_Device(Unit)
ULONG
{
    ULONG Offset;
    Open_A_Device("trackdisk.device", Unit, &DiskExtIO, 0L, TD_LEN);
    TrackDisk_InterruptOn(DiskExtIO, TrackDisk_DiskRemoved);
    HandleCommands();
    TrackDisk_InterruptOff(DiskExtIO);
    Close_A_Device(DiskExtIO);
}
*****/
*           Open_Screen_and_Window()          (User)*
* Fonction: Ouvrir Ecran Editor et Window Editor
*
```

```
*****
VOID Open_Screen_and_Window()
{
    IntuitionBase = (struct IntuitionBase*)
                    OpenLibrary("intuition.library", OL);
    if (IntuitionBase == OL) CloseIt("No IntuitionBase !");
    Screen = (struct Screen *) OpenScreen(&NewScreen);
    if (Screen == OL) CloseIt("No Screen !");
    NewWindow.Screen = Screen;
    Window = (struct Window *) OpenWindow(&NewWindow);
    if (Window == OL) CloseIt("No Window !");
}
 *****
*                         Open_Screen_and_Window()          (User)*
*
* Fonction: Fermer écran et fenêtre Editor
*****
VOID Close_Screen_and_Window()
{
    CloseWindow(Window);
    CloseScreen(Screen);
    CloseLibrary(IntuitionBase);
}
 *****
*                         main()                      (User)*
*
*-----*
* Paramètres d'entrée:
*
* argv[1]: Lecteur(df0:, df1: etc.)
* argv[2]: Fichier de sortie
*****
main (argc, argv)
UWORD argc;
BYTE      *argv[];
{
    UWORD i;
    ULONG Unit=OL;
    BYTE   *InputString;
    UWORD Len;
    InputString = argv[1];
    if (argc >= 2)
    {
        Len = strlen(InputString);
        for (i=0;i<Len;i++)
        {
            if ((*InputString >= 'A') && (*InputString <= 'Z'))
                *InputString |= (BYTE) 0x20; /* minuscules */
            InputString++;
        }
        if (strcmp("df0:", argv[1]) == OL) Unit = 0L;
        else
            if (strcmp("df1:", argv[1]) == OL) Unit = 1L;
            else
                if (strcmp("df2:", argv[1]) == OL) Unit = 2L;
                else
                    if (strcmp("df3:", argv[1]) == OL) Unit = 3L;
    }
    if (argc == 3)
```

```

{
    File = Open(argv[2], MODE_NEWFILE);
    if (File == 0L)
    {
        printf("Cant open %s!", argv[2]);
        CloseIt(" ");
    }
}
if (argc > 3)
{
    printf("USAGE: %s [[DF?:] [ListFile]]!\n", argv[0]);
    exit(0);
}
Open_Screen_and_Window();
SectorBuffer = (BYTE*) AllocMem((ULONG)(TD_SECTOR), MEMTYPE);
/* pour un secteur (Read/Write) */
if (SectorBuffer == 0) CloseIt("No SectorBuffer !");
TrackBuffer = (BYTE*) AllocMem(RAW_TRACK_LEN, MEMTYPE);
/* pour un track (RAWREAD/WRITE) */
if (TrackBuffer == 0) CloseIt("No TrackBuffer !");
LabelBuffer = (BYTE*) AllocMem((ULONG)(NUMSECS), MEMTYPE);
/* pour Label-Area */
if (LabelBuffer == 0) CloseIt("No LabelBuffer !");
for (i=0 : i<32 ; i++) ConversionTable[i] = (BYTE)'.';
for ( : i<128; i++) ConversionTable[i] = (BYTE)i;
for ( : i<160; i++) ConversionTable[i] = (BYTE)',';
for ( : i<256; i++) ConversionTable[i] = (BYTE)i;
ConsoleRead = (struct IOStdReq *)GetDeviceBloc(CON_LEN);
ConsoleWrite = (struct IOStdReq *)GetDeviceBloc(CON_LEN);
ConsoleRead->io_Data = (APTR) Window;
ConsoleRead->io_Length = (ULONG) (sizeof(struct Window));
Open_A_Device("console.device", 0L, &ConsoleRead, 0L, 0L);
Console_Copy(ConsoleRead, ConsoleWrite);
Console_Write(ConsoleWrite, "Welcome to the wonderful World of
TrackDisk !\n", -1L);
The_TrackDisk_Device(Unit);
Close_A_Device(ConsoleRead);
FreeDeviceBloc(ConsoleWrite);
if (File != 0L) Close(File);
FreeMem(SectorBuffer, (ULONG)(TD_SECTOR));
FreeMem(TrackBuffer, RAW_TRACK_LEN);
FreeMem(LabelBuffer, (ULONG)(NUMSECS));
Close_Screen_and_Window();
}

```

On peut appeler le programme par exemple avec DiskEd df1: listfile. Dans ce cas, c'est la disquette qui se trouve dans DF1: qui est examinée, et les contenus de secteur affichés à l'écran par d<Return> sont écrits dans le fichier listfile (vous obtiendrez un tableau des commandes avec h<Return>). L'appel DiskEd permet d'examiner la disquette dans le lecteur 0, et écrit les données de secteur uniquement à l'écran.

5. Le format d'échange standard

Le format d'échange standard est né d'une longue expérience ; il est tout à fait naturel que l'on mette en oeuvre plusieurs logiciels différents pour parvenir à un même but. Il existe donc beaucoup de logiciels graphiques, de logiciels de dessin, et encore plus de traitements de texte. Il en va de même dans tous les domaines de l'informatique.

5.1. Les formats IFF de Electronic Arts

Le problème

Cette situation présente un grand avantage : l'utilisateur et le programmeur se trouvent devant un choix plus large ; l'inconvénient est évidemment que chaque société développe son propre format de sauvegarde. Cela coûte de l'argent, puisque les acquéreurs doivent payer le prix du développement, et d'autre part, il est impossible de cette façon d'échanger les données entre deux programmes différents, si la possibilité d'échange n'a pas été explicitement prévue.

Cette possibilité serait fortement souhaitable, car chaque logiciel offre souvent des avantages qui lui sont propres, et les logiciels pourraient se compléter avantageusement si l'on pouvait échanger les données entre eux.

La solution

Les membres de la société Electronic Arts ont décidé de prendre ce problème à bras le corps. A la naissance du Macintosh de chez Apple, un format a été développé, prétendant à l'universalité. Il devait pouvoir être utilisé par chaque logiciel, et être lu par tous. Le grand avantage de ce format était sa clarté et le fait qu'il était né avant les premiers grands programmes. Il pouvait donc s'imposer sans difficulté.

Ce format s'appelle IFF, abréviation de Interchange File Format. Il a été développé en 1984, et mis officiellement sur la place publique en janvier 1985. Le format IFF comprend les domaines du texte, du graphique et du son dans presque toutes leurs variantes. Ce qu'il y a de meilleur dans ce format, c'est qu'il peut être étendu, et répondre à tout moment aux exigences d'un logiciel. Bien qu'il y ait parfois dans un fichier de

données plus de données que n'en attendent d'autres logiciels au moment de la lecture, il n'en résulte pas d'erreur pour autant.

Les perspectives

Nous avons donc affaire à un produit de pointe de la technique de programmation. Dans ce chapitre, vous apprendrez tout ce qu'il faut savoir sur ces formats. Nous avons divisé le chapitre en 4 parties, dont chacune traite d'un domaine particulier. A la suite de quoi nous avons ajouté le format Aegis-Anim élaboré à partir du format graphique IFF. Voyons d'abord la structure générale de tous les formats.

La structure

Tous les formats IFF sont conçus selon une norme bien définie. Celle-ci facilite le traitement. Ainsi, au début de chaque fichier de données, on trouve un *en-tête*, dans lequel sont déposées toutes les informations indiquant au programme quelles sont les données qui se trouvent dans le fichier. On peut aussi savoir s'il s'agit de graphique, de son ou de texte, ce qui n'est pas inutile pour la prise en compte des données.

La suite des fichiers de données est constituée de ce qu'on appelle des "chunks". Ce sont des blocs de données, qui contiennent toujours un groupe de données indispensables. Ainsi, pour le format IFF musical, il y a toujours un chunk pour chaque instrument, et toutes les propriétés y sont décrites. Les fichiers IFF graphique déposent dans un chunk Color toutes les données concernant la couleur de l'image. Nous avons donc affaire à un système de pièces détachées subtilement conçu, et pouvant être complété à l'infini. Cette ouverture du système a été explicitement voulu, car c'est le degré d'ouverture d'un format qui conditionne sa durée de vie. Chacun des chunks dont nous venons de parler est identifié par ses quatre premiers caractères dans l'*en-tête*. Puis viennent les données, différentes selon le chunk. Nous allons maintenant parler de ces données, dans les sections suivantes.

5.1.1. Le format graphique IFF-ILBM

Le format graphique IFF est sans doute le plus connu de tous les formats IFF. Cela est dû en grande partie au logiciel de dessin DeluxePaint, que l'on peut considérer comme le logiciel standard pour tous les logiciels de dessin. C'est aussi pourquoi tous les logiciels graphiques venant ensuite ont dû utiliser ce format. Si, en effet, le logiciel n'est pas compatible avec ce programme, il n'attire pas la foule à cause des risques de limitation.

Le cadre général d'un fichier graphique IFF est tracé par la "FORM" (c'est un chunk spécial). Il s'agit du rassemblement d'un grand nombre de chunks - nos blocs de données - en un seul ensemble. Puisque tous les chunks d'un fichier graphique sont reliés l'un à l'autre, ils sont situés dans une même FORM. Cette FORM contient pour seule information la longueur du fichier de données et le type de données.

Ainsi, la routine de lecture pourra ultérieurement vérifier si toutes les données ont réellement été saisies :

```
#define ID_ILBM MakeID('I','L','B','M')
```

Ensuite vient la caractérisation du format ILBM. S'il y a ici les quatre caractères ILBM, on est sûr d'avoir affaire précisément à ce format. Ensuite viennent l'un après l'autre les différents chunks de notre fichier de données. Nous allons tous les envisager successivement :

BMHD (BitMapHeaDer)

```
#define ID_BMHD MakeID('B','M','H','D')
```

Ce chunk contient des données qui se réfèrent principalement aux propriétés de notre graphique.

Il s'agit ici d'une structure qui va se révéler ultérieurement très importante pour l'ouverture des écrans, car c'est là que se trouvent les dimensions, la profondeur et toutes les autres indications nécessaires. Voici cette structure :

```
typedef struct
{
    WORD w, h;           /* Largeur et Hauteur en pixels */
    WORD x, y;           /* Position */
    UBYTE nPlanes;       /* Profondeur de l'écran */
    Masking masking;     /* contient des flags de masque */
    Compression compression; /* Type de compression */
    UBYTE pad1;           /* octet de remplissage */
    WORD transparentColor; /* numéro de la couleur transp. */
    UBYTE xAspect, yAspect; /* Rapport entre les côtés */
    WORD pageWidth, pageHeight; /* Largeur & Hauteur de l'écran en
pixels*/
}
BitMapHeader;
```

Avec w et h, on définit la largeur (width) et la hauteur (height) du graphique en pixels. Il n'est pas du tout nécessaire de conserver un graphique dans la taille de l'écran. Les "brushes" générées par les programmes de dessin sont également conservées sous la forme ILBM, et elles n'ont pas le plus souvent la taille de l'écran. Avec x et y, on indique la position de la section graphique. Si l'on a mis en mémoire tout un écran, on trouve toujours ici la valeur 0.

nPlanes indique le nombre de bitplanes constituant le graphique. Ce sera très important pour la ColorMap, car le nombre de couleurs est calculé, comme on le sait, à partir du nombre de bitplanes.

Le paramètre suivant (Masking) désigne le type de masque qui se trouve à la base du graphique. On peut choisir entre mskNone, sans aucun masque, ou mskHasMask. Dans ce cas, un MaskPlane viendra s'ajouter aux bitplanes normaux, pour déterminer exactement le type de masque correspondant au graphique. Le flag mskHasTransparentColor permet d'indiquer que le graphique possède une couleur

devant être représentée de manière transparente. Dans la valeur TransparentColor, on trouve le numéro de la couleur à effacer. Enfin, on trouve la spécification mskLasso, reprise du Macintosh d'Apple. Ici, le graphique est saisi comme avec un lasso. Il est ainsi possible de supprimer le bord du graphique qui peut être pourvu de la couleur d'arrière-plan, ce qui d'une part réduit le graphique, et d'autre part économise de la mémoire.

```
typedef UBYTE Masking;
#define mskNone 0L
#define mskHasMask 1L
#define mskHasTransparentColor 2L
#define mskLasso 3L
```

Sous "compression", on indique le type de compression des données bitmap. On peut demander que les données soient prises dans la mémoire de l'ordinateur et conservées exactement sous la même forme, mais on peut aussi définir à l'aide d'un numéro la méthode de codage, et donc de compression. Vous trouverez plus loin les méthodes existantes de codage et de décodage.

```
typedef UBYTE Compression;
#define cmpNone 0L
#define cmpByteRun1 1L
```

Pad1 est utilisé comme octet de remplissage, pour que la structure possède un nombre pair d'octets. Pour le moment, cet octet est inutilisé, et il doit être placé sur 0. Il faut observer cette règle dans tous les cas, car dans les versions suivantes, il se peut qu'il soit utilisé, et une valeur différente de 0 ne conviendrait pas.

Les variables xAspect et yAspect contiennent chacun le rapport avec les côtés x et y du graphique. Ces variables sont importantes pour les programmes qui transportent par exemple le graphique dans une autre résolution, ou encore qui le transfèrent sur un autre ordinateur.

```
#define x320x200Aspect 10L
#define y320x200Aspect 11L
#define x320x400Aspect 20L
#define y320x400Aspect 11L
#define x640x200Aspect 5L
#define y640x200Aspect 11L
```

Finalement, on trouve dans pageWidth et pageHeight la largeur et la hauteur de l'écran en son entier. Cela peut fournir de nouveaux renseignements sur le graphique, ce dernier pouvant être plus grand ou plus petit que l'écran sur lequel il est représenté.

CMAP (ColorMAP)

```
#define ID_CMAP MakeID('C','M','A','P')
```

Le ColorMap , contrairement au BitMap-Header, est un chunk qui n'a pas toujours la même longueur. La longueur dépend du nombre de bitplanes dans le graphique, car c'est à partir de là que l'on calcule le nombre de couleurs qui seront ensuite conservées.

Pour chaque registre de couleurs, on garde en mémoire trois valeurs d'octet, pour le rouge, le vert et le bleu. Ces composantes de la couleur peuvent prendre des valeurs allant de 0 à 255. Evidemment, l'Amiga n'a pas tant de nuances. Mais comme le format IFF a été développé pour tous les ordinateurs, on a laissé ici de la marge. Il faut donc décaler toutes les valeurs des couleurs dans les bits définis plus haut, c'est-à-dire les multiplier par 16. Pour chaque registre de couleur, il existe aussi une structure :

```
typedef struct
{
    UBYTE red, green, blue;
}
ColorRegister;
```

Pour la création et la lecture d'un fichier IFF, notez bien que le ColorMap peut également contenir un nombre impair d'octets. La liste doit alors être remplie par un octet nul !

CRNG (ColorRnNGe)

```
#define ID_CRNG MakeID('C','R','N','G')
```

Les programmes de peinture vous ont familiarisé avec la possibilité de cybler une palette de couleurs, et de faire naître ainsi des effets très intéressants. Cette fonction est soutenue grâce au chunk CRNG. Celui-ci indique dans le tableau de couleurs une palette qui peut être balayée. La structure prévue à cet effet est la suivante :

```
typedef struct
{
    WORD pad1;
    WORD rate;
    WORD active;
    UBYTE low, high;
} CRange
```

Avec Pad1, on retrouve le caractère de remplissage, prévu ici pour les extensions. En effet, on n'est pas sûr qu'il n'y aura pas d'autres données à conserver.

"rate" représente la vitesse avec laquelle les couleurs seront échangées. On prendra l'exemple suivant : on reporte la valeur 16384 pour 60 changements par seconde. Pour 30 par seconde, ce sera la valeur 8192. La règle est la suivante : le nombre de pas par seconde augmente avec la valeur.

CCRT (Color Cycling Range and Timing)

```
#define ID_CCRT MakeID('C','C','R','T')
```

Le chunk CCRT est responsable du ColorCycling, comme le chunk CRNG. Tous les deux ont été conçus pour la même tâche, mais il s'agit ici de ce qu'on appelle des "private chunks", développés par les sociétés productrices de logiciels. Le chunk CCRT appartient au Graficraft de Commodore, et le chunk CRNG à DeluxePaint.

Il ne faut pas négliger ce détail. Nous aurons en effet l'occasion dans nos programmes de lire et de traiter les deux chunks, pour être entièrement compatible dans le

ColorCycling. Comme les deux chunks sont légèrement différents, il existe évidemment ici une autre structure. La voici :

```
typedef struct {
{
WORD direction;
UBYTE start, end;
LONG seconds;
LONG microseconds;
WORD pad;
} CycleInfo;
```

La forme de cette structure est quelque peu différente de la précédente, mais elle est néanmoins facilement compréhensible. Avec "direction", on indique effectivement la direction dans laquelle les couleurs seront "cyclées". 0 veut dire "ne pas bouger", 1 signifie "vers l'avant", et 2 "vers l'arrière". "start" et "end" (comme low et high) indiquent les numéros de départ et de fin des deux registres de couleur, c'est-à-dire ceux entre lesquels l'échange des couleurs doit être effectué.

L'indication du temps est traitée différemment de ce qui précède. Commodore indique les secondes et les micro-secondes. Cela permet d'obtenir une continuité avec toutes les autres fonctions des librairies Amiga. En effet, cette division existe aussi dans ces librairies (cf. la structure Preferences et les fonctions Intuition). "pad" a été utilisé à nouveau comme octet de remplissage, pour recevoir éventuellement d'autres valeurs dans la structure.

Avant de nous tourner vers le plus important de tous les chunks, le chunk BODY, nous allons mentionner encore quelques chunks plus rares. Ceux-là aussi interviennent parfois dans un fichier ILBM, et vous devez savoir que les données se trouvent dans ces fichiers sous la forme dans laquelle on les utilise.

GRAB (GRAB position)

```
#define ID_GRAB MakeID('G','R','A','B')
```

Avec ce chunk supplémentaire GRAB, on indique la position relative du curseur dans le graphique. On trouve ce chunk surtout avec les "brushes", qui peuvent être situées en n'importe quel point.

```
typedef struct
{
WORD x, y;
} Point2D;
```

Les coordonnées x et y se réfèrent au bord supérieur gauche du graphique. Le chunk contient en tout et pour tout cette structure Point2D.

DEST (DESTination bitplanes)

```
#define ID_DEST MakeID('D','E','S','T')
```

Ce chunk assez rare permet de placer les bitplanes existant du chunk BODY dans d'autres bitplanes du graphique, et de remplir les bitplanes non utilisés avec des valeurs de bits 0 ou 1. On peut de cette façon altérer facilement les couleurs de l'image originale. Ce chunk est constitué lui aussi d'une structure comprenant toutes les instructions :

```
typedef struct
{
    UBYTE depth;
    UBYTE pad1;
    UWORLD planePick;
    UWORLD planeOnOff;
    UWORLD planeMask;
}
DestMerge;
```

"depth" désigne à nouveau la profondeur de l'écran dans lequel les données doivent être reportées. "pad1" est inutilisé, et représente ici aussi un octet de remplissage.

La variable "planePick" est envisagée d'après les bits définis. Chaque bit posé signifie : "Comprimez le bitplane suivant du fichier dans le bitplane de l'écran avec le numéro du bit posé". Si un bit n'est pas posé, on envisage le même bit sous planeOnOff. Si celui-ci est posé, tout le bitplane est rempli de 1 ; sinon, on le laisse vide.

"planeMask" a la tâche d'interdire l'écriture dans un bitplane. On place pour cela le bit correspondant dans un bitplane, si l'on veut écrire dans celui-ci. Dans le cas contraire, on efface ce bit, et on ne peut pas écrire dans ce bitplane. Dans le cas normal, planePick et planeMask doivent avoir la valeur $2^{n\text{Planes}} - 1$. De cette façon, tous les bits sont posés pour chaque bitplane, et tous les bitplanes existants sont utilisés.

SPRT (SPRiTé)

```
#define ID_SPRT MakeID('S','P','R','T')
```

Dans les fichiers ILBM, on ne conserve pas seulement des graphiques normaux. Il est parfaitement possible de conserver un sprite. Celui-ci est alors indiqué à l'aide de ce chunk. Le chunk contient une seule valeur, indiquant la priorité du sprite. La valeur 0 représente la priorité supérieure, et chaque valeur plus élevée diminue la priorité. Le sprite de plus haute priorité se trouve à l'écran devant tous les autres.

```
typeUWORD SpritePrecedence;
```

CAMG (Commodore AMIGa computer)

```
#define ID_CAMG MakeID('C','A','M','G')
```

Il y a sur l'Amiga une particularité qui le distingue des autres ordinateurs travaillant également avec le format IFF. En effet, l'Amiga possède ce qu'on appelle les "viewModes", dans lesquels différents modes de présentation sont spécifiés, comme par exemple HAM. Mais comme ceux-ci ne sont pas soutenus par les chunks IFF-ILBM normaux, il fallait pour cela introduire un nouveau chunk. Ce chunk CAMG contient uniquement le registre ViewMode :

```
typedef struct
{
    ULONG ViewModes;
}
CamgChunk;
```

BODY (all Bitplanes and Optional mask, interleaveD bY row)

```
#define ID_BODY MakeID('B','O','D','Y')
```

La partie essentielle de tout le fichier graphique IFF est le bitmap. Il est conservé dans le chunk BODY. Ici aussi, on procède selon des règles bien définies, que l'on déduit des autres données.

Selon le nombre de couleurs et de bitplanes, on peut définir l'ensemble des données dans le chunk BODY. Celui-ci contient le bitmap dans une présentation ligne par ligne. On mémorise donc d'abord la première ligne du premier bitplane, puis la première série du second bitplane, etc... Lorsque tous les bitplanes et le masque optionnel sont mémorisés, on écrit la ligne suivante dans le même ordre. On va ainsi jusqu'à la dernière ligne. Le chunk BODY est prêt.

Puisque ce procédé de mémorisation est très complexe, surtout pour les graphiques très étendus, on s'est décidé à mettre en place quelques procédés de compression. Ces procédés doivent aider à représenter les mêmes octets par un octet de nombre et un octet de données. Ce simple fait permet d'économiser beaucoup de place.

```
/*****************************************/
/*           Simpler IFF-Read and Display      */
/*           (SANS SPEEDUP)                      */
/*           (c) Bruno Jennrich                  */
/*****************************************/
#include "fcntl.h"
#include <exec/types.h>
#include <exec/memory.h>
#include <exec/devices.h>
#include <devices/keymap.h>
#include <graphics/gfxmacros.h>
#include <graphics/regions.h>
#include <graphics/copper.h>
#include <intuition/intuition.h>
#include <graphics/gfxbase.h>
#include <graphics/gels.h>
#include <hardware/custom.h>
#include <hardware/blit.h>
struct IntuitionBase *IntuitionBase;
struct GfxBase      *GfxBase;
long               *DosBase;
void              *OpenLibrary();
char Mask[9] = {0,1,2,4,8,16,32,32,32};          /* NombCouleurs */
typedef struct BitMapHeader {
    WORD w,h;
    WORD x,y;
    UBYTE Bitplanes;
    UBYTE Masking;
    UBYTE Compression;
```

```

        UBYTE PadByte;
        WORD TransCol;
        UBYTE XAspect,YAspect;
        WORD Width,Height;
    }:
typedef struct ColorRegister {
    UBYTE rouge;
    UBYTE vert;
    UBYTE bleu;
}:
typedef struct CommodoreAmiga {
    WORD PadWord;
    WORD ViewModes;
}:
struct BitMapHeader BMHD;
struct ColorRegister Colors[32];
struct CommodoreAmiga CAMG;
struct Screen *Screen, *OpenScreen();
struct NewScreen NewScreen;
int FileHandle;
LONG Len;
ULONG ChunkLen;
BOOL BMHDFlag;
BOOL CMAPFlag;
BOOL CAMGFlag;
BOOL BODYFlag;
BOOL FoundChunk;
BOOL ShowFlag = FALSE;
UBYTE Buffer [300];
LONG i; /* Compteur général */
LONG x; /* Compteur colonnes */
LONG y; /* Compteur lignes */
LONG b; /* Compteur bitplane */
UBYTE ByteCount;
UBYTE BytesPerRow;
char *WohinDamit; /* Adresses bitplane */
char *MouseButton = (char *) 0xBFE001;
CloseIt(s)
char *s;
{
    printf ("%s\n",s);
    if (FileHandle) Close (FileHandle);
    if (Screen) CloseScreen (Screen);
    if (DosBase) CloseLibrary (DosBase);
    if (IntuitionBase) CloseLibrary (IntuitionBase);
    if (GfxBase) CloseLibrary (GfxBase);
    exit(TRUE);
}
Lire(Buffer, Nomb, Flag)
char *Buffer;
WORD Nomb;
BOOL Flag;
{
    Len = read(FileHandle, Buffer, Nomb);
    if ((Flag == TRUE) && (Len < 0))
        CloseIt("File-Error !!!!!!!\n");
}
main (argc, argv) /* Compteur argument */
int argc;

```

```

char **argv;           /* Argument Value */
{
    if (argc != 2)
    {
        printf (" USAGE: %s IFF-Filename\n", argv[0]);
    }
    else
    {
        printf (" CLIQUER EN HAUT A GAUCHE POUR METTRE FIN A
L'AFFICHAGE.\n");
        DosBase      = OpenLibrary("dos.library", 0L);
        GfxBase      = OpenLibrary("graphics.library", 0L);
        IntuitionBase = OpenLibrary("intuition.library", 0L);
        if ((DosBase == 0) || (GfxBase == 0) || (IntuitionBase == 0))
            CloseIt("Librarys ??!!?!!?!?!?!?!?!?!?\n");
        FileHandle = open(argv[1], O_RDONLY);      if (FileHandle < 0)
            CloseIt("File OPEN Error !?!?!?!?!?!?\n");
        Lire(&Buffer[0], 12, TRUE);
        if (strncmp(&Buffer[0], "FORM", 4) != 0)
            CloseIt("Pas de IFF-File !!!\n");
        if (strncmp(&Buffer[8], "ILBM", 4) != 0)
            CloseIt("Pas de fichier IFF !!!\n");
        BMHDFlag = FALSE;
        CMAPFlag = FALSE;
        CAMGFlag = FALSE;
        BODYFlag = FALSE;
Loop:
    FoundChunk = FALSE;
    Lire(&Buffer[0], 8, FALSE);
    if (Len <= 0)
        if ((BMHDFlag == TRUE) && (BODYFlag == TRUE) && (CMAPFlag == TRUE))
        {
            if (CAMGFlag == FALSE)
                printf (" Pas de CAMG !!!\n");
LoopA:
            while ((*MouseButton & 0x40) == 0x40);
            if ((Screen->MouseX == 0) && (Screen->MouseY == 0))
                CloseIt ("");
            else
                goto LoopA;
        }
        else
        {
            if ((BMHDFlag != TRUE) || (BODYFlag != TRUE) ||
                (CMAPFlag != TRUE))
            {
                if (BMHDFlag == FALSE) CloseIt(" Pas de BMHD !!!\n");
                if (BODYFlag == FALSE) CloseIt(" Pas de BODY !!!\n");
                if (CMAPFlag == FALSE) CloseIt(" Pas de CMAP !!!\n");
            }
        }
        ChunkLen = Buffer[4]*16777216+Buffer[5]*65536+
                    Buffer[6]*256+Buffer[7];
        if (strcmp (Buffer, "BMHD", 4) == 0)
        {
            if (BMHDFlag == TRUE) CloseIt (" deux BMHD's ?????\n");
            Lire(&BMHD, ChunkLen, TRUE);          /* BMHD Lire */
            NewScreen.LeftEdge = 0;
            NewScreen.TopEdge = 0;
        }
    }
}

```

```

NewScreen.Width      = BMHD.Width;
NewScreen.Height     = BMHD.Height;
NewScreen.Depth       = BMHD.Bitplanes;
NewScreen.DetailPen  = 1;
NewScreen.BlockPen   = 0;
if (CAMGFlag == TRUE)
    NewScreen.ViewModes = CAMG.ViewModes;
else
{
    NewScreen.ViewModes = 0;
    if (NewScreen.Width > 320)
        NewScreen.ViewModes |= HIRES;
    if (NewScreen.Height > 256)
        NewScreen.ViewModes |= LACE;
}
NewScreen.Type        = CUSTOMSCREEN;
NewScreen.Font         = Null;
NewScreen.DefaultTitle = (char *)argv[1];
NewScreen.Gadgets      = Null;
NewScreen.CustomBitMap = Null;
    Screen = OpenScreen(&NewScreen);
if (Screen == 0)
    CloseIt (" Pas de Screen !!!\n");
BytesPerRow = BMHD.Width/8;
ShowTitle(Screen, FALSE);
BMHDFlag = TRUE;
FoundChunk = TRUE;
}
if (strncmp(Buffer,"CMAP",4) == 0)
{
    if (CMAPFlag == TRUE)
        CloseIt("Deux CMAP's ???\n");
    if (BMHDFlag == FALSE)
        CloseIt("BMHD doit venir avant CMAP !!!\n");
    Lire(Colors, ChunkLen, TRUE); /* CMAP Lire */
    for (i=0; i<Mask[BMHD.BitPlanes]; i++)
        SetRGB4(&Screen->ViewPort, i, Colors[i].rouge>4,
                Colors[i].vert>4,
                Colors[i].bleu>4);
    CMAPFlag = TRUE;
    FoundChunk = TRUE;
}
if (strncmp (Buffer,"BODY",4) == 0)
{
    if (BODYFlag == TRUE)
        CloseIt ("Deux Body's ???\n");
    if (BMHDFlag == FALSE)
        CloseIt ("Avant BODY doit venir BMHD !!!\n");
    for (y=0; y<BMHD.Height; y++)
        for (b=0; b<BMHD.BitPlanes; b++)
        {
            ByteCount = 0;
            WohinDamit =
            (char *) Screen->RastPort.BitMap->Planes[b]+y*BytesPerRow;
            if (BMHD.Compression == 0)
                Lire (OuDonc,BytesPerRow,TRUE);
            if (BMHD.Compression == 1)
                while (ByteCount<BytesPerRow)
                {

```

```

        Lire (&Buffer[0],1,TRUE);
        if (Buffer[0] < 128)
        {
            Lire(OuDonc+ByteCount, Buffer[0]+1, TRUE);
            ByteCount += Buffer[0]+1;
        }
        /* Buffer[0] == 128 => Nop */
        if (Buffer[0] > 128)
        {
            Lire(&Buffer[1], 1, TRUE);
            for (i=ByteCount; i<(ByteCount+257-Buffer[0]); i++)
                *(OuDonc+i) = Buffer[1];
            ByteCount += 257-Buffer[0];
        }
    }
    BODYFlag = TRUE;
    FoundChunk = TRUE;
}
if (strcmp (Buffer,"CAMG",4) == 0)
{
    if (CAMGFlag == TRUE)
        CloseIt("Deux CAMG's !!!\n");
    if (BMHDFlag == FALSE)
        CloseIt("BMHD doit venir avant CAMG !!!\n");
    Lire (&CAMG,ChunkLen,TRUE);
    Screen->ViewPort.Modes = CAMG.ViewModes;
    RemakeDisplay();
    CAMGFlag = TRUE;
    FoundChunk = TRUE;
}
if (FoundChunk == FALSE)
{
    Lire (Buffer,ChunkLen,FALSE);
    if ((ChunkLen & 1) == 1) Lire (Buffer,1,FALSE);
}
goto Loop;
}
}

```

Description du programme

Le programme utilise pour son travail deux routines de soutien. Il s'agit d'abord de la fonction `CloseIt()`, à l'aide de laquelle on met fin au programme en cas d'erreur ou en fin de programme. Cette routine examine ce qui est déjà ouvert, et le ferme pour éviter les erreurs système. La seconde routine `Lire()` est utilisée pour la lecture d'un certain nombre de données.

Le programme principal commence après ouverture des librairies et des fichiers avec la lecture des données. On effectue d'abord un test du fichier IFF-ILBM, et on examine ensuite le premier chunk dans la boucle principale. S'il est connu, ses données sont lues d'après les règles précisées plus haut. Les chunks inconnus ou ceux qui ne sont pas soutenus par le programme sont simplement sautés.

Si les données nécessaires existent, BMHD ouvre un écran dans lequel les données bitmap sont alors reportées avec le chunk BODY. En fin de compte, le programme attend un clic sur la souris dans le coin supérieur gauche, pour fermer le tout, et conclure son travail. Ensuite, vous voyez pour le format ILBM une liste de tous les chunks qui peuvent se présenter si vous lisez une forme ILBM. Notez au moment de l'écriture que tous les chunks lus sont à nouveau écrits, même ceux qui n'ont pas été traités par votre programme.

```
#ifndef ILBM_H
#define ILBM_H
#define ID_ANFR MakeID('A','N','F','R')
#define ID_MAHD MakeID('M','A','H','D')
#define ID_MFHD MakeID('M','F','H','D')
#define ID_CM16 MakeID('C','M','I','6')
#define ID_ILBM MakeID('I','L','B','M') /* Interleaved BitMap */
#define ID_ILBM MakeID('S','H','A','K') /* Shakespeare-Chunk, contenant
des ILBM's */
#define ID_ANIM MakeID('A','N','I','M') /* Format animation */
#define ID_BMHD MakeID('B','M','H','D') /* BitMap Header */
#define ID_ANHD MakeID('A','N','H','D') /* Header animation*/
#define ID_CMAP MakeID('C','M','A','P') /* Color Map */
#define ID_GRAB MakeID('G','R','A','B') /* Hot Spot du BitMap */
#define ID_DEST MakeID('D','E','S','T') /* Division du bitplane */
#define ID_SPRT MakeID('S','P','R','T') /* Identification du Sprite */
#define ID_CAMG MakeID('C','A','M','G') /* Commodore Amiga View */
#define ID_BODY MakeID('B','O','D','Y') /* Données BitMap */
#define ID_ATXT MakeID('A','T','X','T') /* parfois */
#define ID_PTXT MakeID('P','T','X','T') /* utilisé */
#define ID_DLTA MakeID('D','L','T','A') /* Décalage Anim Delta */
#define ID_CRNG MakeID('C','R','N','G') /* Color Cycling Chunk */
```

Pour finir, nous allons mentionner quelques problèmes qui peuvent surgir du fait que certains programmes utilisant les fichiers ILBM-IFF ne se tiennent pas rigoureusement à ces règles.

1er problème : La taille de l'écran de présentation

Normalement, on doit trouver dans le header de bitmap la taille du graphique et la taille de l'écran sur lequel le graphique est présenté. Les valeurs dans w et h correspondent donc toujours au nombre de pixels contenus dans le graphique selon les deux directions x et y.

Les valeurs dans pageWidth et pageHeight se réfèrent au nombre de pixels dans les directions x et y. Ces valeurs sont écrites par DpaintII dans la structure. Ce n'est que logique, car on peut mémoriser ainsi entièrement une image bien plus grande que l'écran, et la représenter néanmoins sur un écran de 350x256 points.

Mais il y de plus en plus de programmes qui utilisent aussi le mode OverScan et contiennent maintenant dans les variables pageWidth et pageHeight les valeurs qui ont été sélectionnées à cause de l'OverScan. Cela provoque certains problèmes, car bien que la valeur soit nettement supérieure à 320, on n'est pas obligé d'ouvrir des écrans HIRES ! Pour faire face à ce problème, il est recommandé de mémoriser également le

ViewMode, et de pouvoir le lire. En effet, dans ce mode, la résolution sélectionnée est clairement indiquée. Familiarisez donc votre programme avec le chunk CAMG !

2ème problème : Chunks oubliés

Certains programmes oublient simplement le chunk CAMG, tout en n'utilisant pas les résolutions normales.

Ainsi il arrive que l'on mémorise des graphiques HAM ou HALFBRITE, et les programmeurs pensent que l'on reconnaît ce fait aux 6 bitplanes. Mais personne ne pense à l'existence de l'autre mode, car c'est maintenant les programmes qui décident s'il s'agit d'une image HAM ou HALFBRITE.

Un autre problème surgit avec le fait que certains programmes écrivent les ViewModes directement dans le registre d'écran lorsqu'ils doivent être mémorisés. Cela risque de provoquer des problèmes lorsque les flags SPRITES, VP_HIDE, GENLOCK_AUDIO et GENLOCK_VIDEO sont posés. Effacez donc ces flags si vous mémorisez le chunk CAMG.

3ème problème : Color Cycling

Malheureusement, notre modèle DPaintII ne mémorise par les chunks Color-Cycling correctement. Le chunk CRNG n'est pas manipulé comme il faut, à cause d'un petit détail. DPaintII écrit tous les "cycling ranges" sur la disquette comme étant actifs, sans se préoccuper du fait que l'image est "cyclée" ou non.

Cela a pour conséquence le fait que d'autres programme ne peuvent absolument pas vérifier si une image doit être "cyclée" activement ou non, car toutes images qui sont passées dans les mains de DPaintII sont infestées par ce défaut. Il ne reste plus qu'à recourir à une spécification spéciale du programme Slide-Show: soit un paramètre dans le CLI, soit encore un ToolType avec la mention CYCLING=ON.

4ème problème : Le nombre de couleurs d'un chunk CMAP

Lorsqu'on utilise des images HAM, on voit surgir un problème avec la ColorMap. Celle-ci doit contenir toutes les couleurs utilisées pour l'image. Mais bien que l'on fasse usage de 6 bitplanes, ce type de graphique n'utilise que 16 couleurs. Il y a cependant là-dessus un désaccord, car certains programmes mémorisent 16 couleurs, d'autres 32 et beaucoup 64 couleurs. En effet, la règle IFF stipule que le nombre de couleurs soit calculé par l'intermédiaire de $1 << \text{bitmap-depth}$, et cela donne 64.

Soyez donc prudent ! Un chunk CMAP n'a pas forcément le nombre de couleurs préférés par votre chunk BMHD. Il vaut mieux lire à nouveau l'indication de l'octet au début du chunk ! Et ne placez jamais le nombre de registres couleur après la mention à l'intérieur du CMAP !

5.1.2. Le format texte IFF-FTXT

Ce format IFF a été utilisé pour imposer l'idée d'un transfert de données sans problème. Malheureusement, ce format ne s'est pas répandu, ce qui est très facile à voir si l'on sait que par exemple Beckertext et WordPerfect sont incompatibles. Le seul logiciel qui utilise le format FTXT est TextCraft. Il a été distribué au tout début dans un paquet de logiciels avec l'Amiga. La format FTXT se présente ainsi :

```
FORM#####
FTXT
[FONS]####FontData
CHRS####Characters....<END>
(#### soit File ou longueur du chunk)
```

FTXT est la chaîne d'identification qui signale qu'il s'agit d'un fichier texte IFF formaté (Formated Text = FTXT). Les deux chunks soutenus par le chunk FTXT sont CHRS et FONS. Le chunk FONS est constitué d'une structure FontSpecifier, qui définit la fonte à utiliser :

Offset	Structure
-----	-----
	FontSpecifier
	{
0 0x00	UBYTE id; /* Numéro de fonte 0 - 9 */
1 0x01	UBYTE pad1; /* Toujours égal à 0 */
2 0x02	UBYTE proportional; /* Fonte proportionnelle ? */ /* 0 - inconnu, 1 - non, 2 - oui */
3 0x03	UBYTE serif; /* Sérifs ? */ /* 0 - inconnu, 1 - non, 2 - oui */
4 0x04	char name[]; /* Nom de fonte (p.ex. "topaz/8") */
...	}

La longueur de ce chunk dépend du nombre de caractères dans le nom de la fonte. Les crochets droits autour du mot-clé FONS sont destinés à rappeler que ce chunk n'est pas nécessaire pour constituer un fichier IFF complet. Le chunk CHRS comprend le texte proprement dit (codes ASCII 0x20 à 0x7f). La caractéristique CHRS est ici suivie du nombre de caractères contenus dans ce chunk. Les chunks CHRS et FONS peuvent être échangés dans un fichier FTXT.

5.1.3. Le format musical IFF-SMUS

A l'aide de ce format, il est possible d'échanger des morceaux de musique entre deux programmes. On définit pour cela les différentes voix, ainsi que les instruments avec lesquelles ces voix sont jouées. Un fichier SMUS a le format suivant :

```
FORM#####
SMUS
SHDR####SScoreHeader
[NAME]####" .. "
[(c) ]####" .. "
[AUTH]####" .. "
[IRev]####????
```

```

ANNO#####...
INSI####RefInstrument
TRAK####SEvents...<End>

```

Les chunks

SMUS est la chaîne d'identification qui indique que ce fichier IFF est un fichier SMUS (Simple MUsical Score). Le chunk SHDR contient une structure SScoreHeader :

Offset	Structure
-----	-----
	struct SScoreHeader
	{
0 0x00	UWORD tempo;
2 0x02	UBYTE volume; /* Volume (0-127) */
3 0x03	UBYTE ctTrack; /* Nombre de voix */
4 0x04	}

Le tempo d'un morceau de musique est indiqué d'une manière inhabituelle : par 1/4 de note pour 128 minutes. Si tempo = 1, un 1/4 de note est joué toutes les 128 minutes.

- NAME** contient le nom du morceau (p.ex. "Fugue en ré")
- (c)** contient le copyright (p.ex. Beethoven 1988")
- AUTH** contient le nom de l'auteur (p.ex. "Frédéric") et on peut utiliser IRev pour les données personnelles.
- ANNO** contient les annotations sur le morceau. Ce chunk doit exister nominalement, mais il peut contenir des octets nuls (Ne pas dépasser 32767 octests).
- INS1** contient des données sur l'instrument à utiliser. Pour définir un instrument, vous disposez de la structure suivante :

Offset	Structure
-----	-----
	struct RefInstrument
	{
0 0x00	UBYTE register;
1 0x01	UBYTE type;
2 0x02	UBYTE data1,
3 0x03	data2;
4 0x04	char name[];
	}

- Register** contient le numéro de l'instrument. Grâce à ce numéro, vous pourrez ultérieurement sélectionner de nouveaux instruments pour jouer la même voix. Avec "type", vous pouvez indiquer si un instrument doit être utilisé avec le nom "name" (type = 0), ou s'il faut utiliser un canal MIDI (type = 1). Dans le dernier cas, les octets data1 et data2 déterminent le canal MIDI et le preset MIDI à utiliser. Le champ "name" n'est pas utilisé ici.

Comme vous le voyez, INS1 ne définit pas comment on doit entendre un instrument, mais uniquement quel est l'instrument que l'on doit entendre. L'instrument 0 est affecté à la voix 0, l'instrument 1 à la voix 1, etc. Cet ordre peut être ultérieurement modifié.

TRAK contient les notes à jouer et d'autres informations. Chaque mention dans TRAK est longue de deux octets, et le premier de ces deux octets définit comment le second octet doit être interprété :

Offset	Structure
-----	-----
	struct SEvent /* Simple Musical Event */
	{
0 0x00	UBYTE SID;
1 0x01	UBYTE data;
2 0x02	}

Voici les interprétations de Data pour les valeurs suivantes de SID :

0-127 (Note)

Ces valeurs indiquent la hauteur du son à jouer. L'octet Data détermine la longueur du son :

- Bit 7** Si ce bit est posé, la note présente et la suivante seront jouées sous forme d'accord.
- Bit 6** Ce bit sert à obtenir la note présente et la suivante sans interruption.
- Bit 4** Ces deux bits indiquent si la note est une tierce (1), une quinte (2) ou une septième (3). Si les deux bits sont effacés, il s'agit d'une note normale.
- Bit 5** Ces deux bits indiquent si la note à jouer est pointée.
- Bits 2-0** Ces bits indiquent s'il s'agit d'une note entière (0), d'un quart de note (2), d'un huitième (3), d'un seizième (4), d'un trente-deuxième (5), d'un soixante-quatrième (6) ou d'un cent-vingt-huitième (7).

128 (Pause)

Si SID a la valeur 128, une pause est ménagée. L'octet Data est interprété ici exactement comme ci-dessus.

129 (Instrument)

A l'aide de cet événement, vous pouvez modifier l'instrument de cette voix. L'octet Data contient ici le numéro de l'instrument.

130 (Rythme)

A l'aide de cet événement, vous pouvez indiquer le rythme. Le rythme est fourni par le quotient entre les cinq bits supérieurs et les trois inférieurs. Les cinq bits supérieurs sont indiqués en coups par seconde (1-32), alors que les 3 bits inférieurs sont indiqués comme pour SID = Note. Pour générer un rythme 4/4, il faut placer dans les 5 bits supérieurs la valeur 3 et dans les 3 bits inférieurs la valeur 2 (quart de note) (Une valeur de 0 dans les 5 bits supérieurs signifie un coup par seconde).

131 (Tonalité)

Avec cet événement, vous déterminez la tonalité des notes suivantes :

Data	Tonalité
0	Do (majeur) C (maj)
1	Sol G
2	Ré D
3	La A
4	Mi E
5	Si B
6	Fa # F#
7	Do # C#
8	Fa F
9	Sib Bd
10	Mib Ed
11	Lab Ad
12	Réb Dd
13	Solb Gd
14	Dob Cd

132 (Volume)

A l'aide de cet événement, vous pouvez indiquer un nouveau volume (0-127) pour cette voix.

133 (Midi Channel)

Cet événement permet de sélectionner un nouveau canal MIDI pour les notes suivantes (data = 0-255).

134 (MIDI Presets)

Cet événement permet de sélection de nouveaux presets MIDI (data = 0-255).

5.1.4. Le format IFF-8SVX-Sample

Ce format 8SVX IFF permet d'échanger des bruits digitalisés. Un 8SVX (8-bit Sampled Voice) se présente ainsi :

```
FORM#####  
8SVX  
VHDR#####Voice8Header  
[NAME]#####".."  
[(c)]#####".."  
[AUTH]#####".."  
ANNO#####".."  
ATAK#####EGPoint  
RLSE#####EGPOINT  
BODY#####Samples...<End>
```

Voici les différents chunks :

8SVX est la chaîne d'identification pour le fichier 8-bit Sampled Voices. VHDR contient une structure Voice8Header :

Offset	Structure
0x00	-----
0x04	-----
0x08	-----
0x0C	-----
0x10	-----
0x14	-----
0x18	-----
0x1C	-----
0x20	-----

```
struct Voice8Header
{
    ULONG oneShotHiSamples,
    ULONG repeatHiSamples,
    ULONG samplesPerHiCycle;
    WORD samplesPerSec;      /* Fréquence sampling */
    BYTE ctOctave;          /* Nombre d'octaves */
    WORD sCompression;       /* Compression des données ? */
    LONG volume;
}
```

- NAME** contient le nom du bruit (p.ex. "AAAAHHHH")
- (c)** contient le copyright (p.ex. "(c) Frédéric")
- AUTH** contient le nom de l'auteur (p.ex. "Frédéric")
- ANNO** contient les annotations sur le bruit. Ce chunk doit exister explicitement, mais on peut le réduire à des octets nuls (ne pas dépasser 32767 caractères).
- ATAK** contient une structure EGPoint, grâce à laquelle on peut augmenter dans un temps donné le volume jusqu'à un certain niveau :

Offset	Structure
0x00	-----
0x02	-----
0x06	-----

```
struct EGPoint
{
    WORD duration; /* durée en millisecondes */
    LONG dest;     /* facteur de volume */
}
```

"duration" indique le temps dans lequel le volume doit atteindre la nouvelle valeur. "dest" indique le facteur d'augmentation du volume. Il faut noter que cette variable est à virgule flottante. Les 16 bits supérieurs fournissent la partie qui précède la virgule, les 16 bits inférieurs fournissent la partie qui la suit. Une valeur de 0x00015000 fournit un facteur de 1,5. RLSE contient également une structure EGPoint. Celle-ci est nécessaire pour faire baisser le volume.

5.2. Le format ANIM

Le format ANIM a été créé par les sociétés Sparta Inc. et Aegis, pour réaliser des séquences d'images animées sur l'Amiga. Le premier programme utilisant ce format qui vient à l'esprit est évidemment VideoScape 3D, mais il en existe aussi d'autres.

L'idée de base qui se tient derrière le format ANIM a été de procurer une possibilité unitaire pour mémoriser des images sur disquette. Il fallait pour cela se montrer très économique en ce qui concerne la place sur la disquette, mais aussi permettre un déroulement des images aussi rapide que possible. En outre, il fallait avoir recours le plus largement possible aux formats IFF existants, pour ne pas créer un nouveau standard.

Le format ANIM utilise donc avant tout le format ILBM pour mémoriser la première image d'un film. Toutes les autres images ne sont pas mémorisées directement, mais seulement à travers leurs différences par rapport à la première image. Il existe ici diverses méthodes de compression, qui ont été conçues pour des situations différentes, et sur lesquelles nous reviendrons en détail un peu plus loin.

Venons-en d'abord à la structure fondamentale du format ANIM. Au début se trouve une image ILBM normale, qui informe entre autres sur la taille de l'image et donc du film tout entier, et sur le mode de résolution utilisé. Ces valeurs ne doivent plus être modifiées pendant tout le film, alors qu'on peut utiliser par exemple pour chaque image une palette de couleurs différente.

Ensuite viennent les données qui contiennent uniquement les différences par rapport aux images précédentes. Le bloc de données contient ici les différences de l'image 2 par rapport à l'image 1, mais tous les blocs ultérieurs contiennent les différents par rapport à l'avant-dernière image: le bloc 3 contient les différences de l'image 3 par rapport à l'image 1, le bloc 4 celles de l'image 4 par rapport à l'image 2, etc. On en comprendra aisément la raison, si l'on réfléchit à la façon dont ces images sont affichées.

On utilise en effet pour cela ce qu'on appelle le double-buffering, donc deux mémoires d'images, entre lesquelles on fait le va-et-vient. L'image 1 est d'abord chargée dans le tampon A, et copiée dans le tampon B. Puis les modifications pour l'image 2 sont entreprises dans le tampon B, pendant que le tampon A est affiché. Ensuite on passe sur tampon B, et les modifications pour l'image 1 sont entreprises dans le tampon A, dans lequel se trouve en effet toujours l'image 1. Puis le tampon A est affiché, les modifications pour l'image 2 sont exécutées dans le tampon dans lequel se trouve encore l'image 1, etc. Cette méthode permet d'obtenir une vitesse maximale de débit, même

si elle paraît au premier abord tirée par les cheveux. Le format ANIM se présente comme suit :

```

FORM ANIM
  FORM ILBM    première image
    BMHD
    ANHD
    CMAP
    BODY
  FORM ILBM    seconde image
    ANHD
    DLTA
  FORM ILBM    troisième image
    ANHD
    DLTA
...

```

La première image représente une image ILBM tout à fait normale, qui peut contenir des informations supplémentaires comme CRNG, qui ne sont cependant pas utilisées. Si l'image contient un bloc de données ANHD, contenant des indications sur le temps passé, son champ d'opération (cf. ci-dessous) doit être placé sur 0.

Dans ce qui suit, nous présentons le début d'un fichier ANIM, comme exemple de ce format. Les données proprement dites n'ont pas été reportées :

Adr.	Contenu (hex)	Type	Taille (décimal)
0000:	464F524D 00038E80	FORM	233088
0008:	414E494D	ANIM	
000C:	464F524D 0000409C	: Image 1	
0014:	494C424D	FORM	16540
0018:	424D4844 00000014	ILBM	
0020:	016000DC 00000000 05000100 00000A0B	BMHD	20
0030:	016000DC		
0034:	434D4150 00000060	CMAP	96
003C:	000000E0 COA00020 50004070 00609000		
004C:	80B010A0 D030C0FO 00400000 60000080		
005C:	0000A000 10C01030 F0305000 00700000		
006C:	90000080 0000D020 20F04040 50300070		
007C:	50009070 00B09000 D0C010F0 F0303030		
008C:	30505050 70707090 9090COC0 COFOFOFO		
009C:	424F4459 0000400B		
00A4:	D500D5FF D500D500 D500D500 D5FFD500	BODY	16395
...			
40A0:	00D5FFD5 00D5FFD5 FFD500D5 FFD50000	: Image 2	
40B0:	464F524D 00001E5E	FORM	7774
40B8:	494C424D	ILBM	
40B8:	414E4844 00000028	ANHD	40
40C4:	03FF0000 00000000 00000000 00000000		
40D4:	00000022 4E6A0021 E85E0021 E8660021		
40E4:	CEBA0022 4E7A0022		
40EC:	444C5441 00001E22		
40F4:	00000020 000006D0 00000D8C 00001370		
...		DLTA	7714
5F06:	FFFFEFFEA 0002007F FFFE0016 01FEFFFF	: Image 3	

5F16: 464F524D 0000295E	FORM 10590
5F1E: 494C4240	I LBM
5F22: 414E4844 00000028	ANHD 40
5F2A: 03FF0000 00000000 00000000 00000000	
5F3A: 00000022 4E6A0021 E85E0021 E8660021	
5F4A: CEBA0022 4E7A0022	
5F52: 444C5441 00002922	DLTA 10530
5F5A: 00000020 00000938 00001296 000019C6	

Ceci permet de voir clairement la structure de base d'un fichier ANIM, et nous pouvons donc nous tourner maintenant vers les problèmes qui surgissent lorsqu'on enregistre un film, c'est-à-dire lorsqu'on veut créer un fichier ANIM.

Pour enregistrer un film, on a besoin de trois bitmaps, donc un de plus que pour le présenter à l'écran. L'un de ces bitmaps est utilisé pour l'affichage de l'image actuelle, et deux autres pour les deux images qui ont été générées à partir de là, et dont nous avons besoin pour la compression. Nous avons ici le choix entre 5 méthodes de compression différentes. La première méthode n'a plus qu'une valeur historique: elle relie l'image actuelle à l'avant-dernière par une relation OU exclusif. Le résultat de cette opération est considéré comme une image à proprement parler, et mémorisée comme telle. Mais comme elle contient partout des 0 là où les deux images étaient identiques, elle peut être sérieusement compressée. Cette méthode n'est cependant plus utilisée, car les méthodes créées depuis permettent une bien meilleure utilisation de la mémoire, et une vitesse plus élevée.

La seconde méthode compare les deux images bitplane par bitplane, et mot long par mot long. Si plusieurs mots longs successifs sont identiques, seul le nombre de mots longs identiques est mémorisé. Cette méthode est intéressante surtout pour les Amiga construits autour du 68020, car celui-ci est particulièrement rapide dans son traitement des mots longs. Mais comme tout le monde ne dispose pas de ce processuer, il existe une troisième méthode qui ne se distingue de la précédente que sur un point: elle compare des mots (16 bits) à la place des mots longs. Cette méthode a été utilisée également par VideoScape 3D V1.0.

La méthode suivante (la quatrième) est en fait une combinaison entre la seconde et la troisième. Elle permet aussi une compression verticale: les deux méthodes précédentes compriment horizontalement. Enfin, la méthode 5 a été développée par Jim Kent et elle est utilisée dans VideoScapeV1.1. Ici , les images sont comparées par colonnes (donc verticalement), et en fait octet par octet. Cette méthode est la plus économique pour la mémoire, surtout avec les enregistrement digitalisés qui présentent un léger brouillage de l'image, ce qui rend difficile la compression.

Mais avant de revenir plus en détail sur ces méthodes de compression, nous allons décrire de près la présentation d'un film à l'écran. Comme nous l'avons mentionné au début, nous avons besoin pour cela de deux mémoires d'images: l'une permet d'afficher l'image actuelle, pendant que l'autre calcule la suivante. La table des couleurs reste la même, tant qu'elle n'est pas remplacée par une nouvelle.

Avec les films cycliques, dans lesquels la fin revient au début, les deux dernières images correspondent exactement aux deux premières, si bien qu'après présentation de la

dernière image, on peut passer à la troisième. Les deux premières images sont écartées, car elles sont codées différemment des autres: la première image est une "véritable" image, et la seconde est compressée relativement à la première, alors que toutes les autres sont compressées relativement à l'avant-dernière.

Les versions les plus récentes des logiciels de jeux, donc les programmes qui présentent à l'écran des fichiers ANIM, soutiennent aussi la division d'un film en plusieurs fichiers, si bien que les passionnés d'animation qui ne possèdent pas de disque dur peuvent obtenir néanmoins de très bons résultats. Ici, les fichiers sont chargés dans l'ordre, et les deux premières images du fichier (sauf naturellement pour le premier) sont à chaque fois écartées. Ceci présente un grand avantage: chaque fichier peut être présenté à l'écran ou enregistré à lui tout seul, ceci qui simplifie le traitement d'une animation complexe.

Pour obtenir une présentation régulière d'un film, le logiciel doit conserver des intervalles de temps aussi constants que possible entre l'affichage de deux images successives. Pour mesurer le temps, on utilise le retour vertical du faisceau, avec lequel on peut atteindre une précision de 1/50 de seconde (1/60 avec l'Amiga NTSC). On peut spécifier par là également la vitesse de présentation des images.

Voyons maintenant le format exact du bloc de données ANHD. Les formats exacts des autres blocs de données (BMHD, CMAP, etc.) ont été décrits dans les sections précédentes.

UBYTE operation Méthode de compression :

0	Image ILBM normale
1	Format XOR
2	Compression horizontale par mots longs
3	Compression horizontale par mots
4	Compression général
5	Compression verticale par octets
74	Réserve pour la méthode de compression Eric Graham (74 = ASCII 'J')

UBYTE mask (seulement format XOR)

Les bits posés renseignent sur les bitplanes dans lesquels quelque chose s'est modifié (bit 0 pour bitplane 0)

UWORD *w,h* (seulement format XOR)

Largeur et hauteur du secteur dans lequel quelque chose s'est modifié.

WORD *x,y* (seulement format XOR)

Position en X et Y du secteur dans lequel quelque chose s'est modifié.

ULONG abstime

Actuellement non utilisé. Mesure du temps au 1/50 de secondes, à partir du début de l'animation, jusqu'au moment où cette image doit être affichée.

ULONG reltime

Mesure du temps en 1/50 de secondes, entre l'image précédente et l'image actuelle.

UBYTE interleave (non utilisé)

Indique l'image par rapport à laquelle l'image présente a été compressée. Un 0 signifie l'avant-dernière image; toutes les autres valeurs indiquent le nombre d'images entre cette avant-dernière et l'image désignée. La raison principale de la mise en place de ce paramètre est de fournir la possibilité de travailler uniquement avec une mémoire d'images, ce qui permettrait de placer le facteur Interleave sur 1.

UBYTE pad0 (non utilisé)***ULONG bits***

Divers flags pour les méthodes de compression 4 et 5. Actuellement, seuls 6 bits sont utilisés sur les 32 possibles. Tous les autres sont placés sur 0 pour des raisons de compatibilité. Un programme doit pour cette même raison tester tous les bits qu'il n'utilise pas, et afficher au besoin un message d'erreur. Voici les significations des 6 bits définis :

- Bit 0** Indique si l'on a utilisé une compression par mots (0) ou par mots longs (1).
- Bit 1:** Prend la valeur 1 si l'on utilise le mode XOR, et 0 si les données doivent remplacer les anciennes valeurs.
- Bit 2:** Est nul lorsqu'il existe une information propre à chaque bitplane, et 1 si une seule Info suffit pour tous les bitplanes.
- Bit 3:** Est égal à 1 lorsque les données ont le code Run-Length-Code (RLC); sinon 0. Nous y reviendrons plus loin.
- Bit 4:** Si les données sont compressées verticalement, ce bit est égal à 1, et 0 représente une compression horizontale.
- Bit 5:** Nul si les offsets des infos sont des mots, et 1 si ce sont des mots longs.

UBYTE pad[16] (non utilisé)

Ce bloc de données ANHD est suivi, comme vous l'avez déjà vu dans l'exemple précédent, d'un bloc DLTA (sauf pour la méthode de compression 1). Ce dernier bloc se présente comme suit pour les méthodes de compression 2 et 3: au début se trouvent 8 mots longs, qui pointent sur les données concernant chacun des bitplanes, relative au début du bloc de données. Cette méthode permet donc de traiter des films dont la longueur maximale est de 8 bitplanes (256 couleurs). Le premier pointeur utilisé a la valeur 32, puisque le premier champ de données commence directement derrière ces 8 pointeurs ($8 \times 4 = 32$). Un pointeur placé sur 0 indique qu'il n'y a pas eu de modification dans ce bitplane.

Les données proprement dites sont constituées par un offset de mot. Si celui-ci est positif, il faut sauter le nombre correspondant de mots ou de mots longs (selon que l'on travaille avec la méthode 3 ou 2) dans le bitplane. Le mot ou le mot long suivant est ensuite écrit dans le bitplane.

Si l'on a Offset = -1, cela indique la fin des données pour ce bitplane. Une autre valeur négative, en revanche, indique que les données suivantes dans le bitplane doivent être copiées. La valeur effective de l'offset est alors (Offset+2). Derrière l'offset se trouve le nombre de mots ou de mots longs à copier dans le bitplane, suivis par ceux-ci.

Pour la méthode 4, le format du bloc de données DLTA est un peu différent. La différence essentielle tient dans le fait que les offsets et les données d'images proprement dites sont séparés. Pour chaque bitplane, il existe donc une liste avec les offsets et le nombre de données à copier (les Infos mentionnées plus haut), ainsi qu'une liste avec les données à copier dans le bitplane correspondant. Au début du bloc de données DLTA se trouvent pour cela 8 pointeurs sur les données d'images des différents bitplanes, suivis par 8 autres pointeurs sur les Infos, donc les listes contenant les offsets et le nombres de données-images à copier. Si, dans le bloc de données ANHD, le bit 2 du champ de bits est placé sur 1, il existe seulement une liste Info pour les 8 bitplanes (possibles), ce qui réduit encore la place occupée en mémoire. Il existe toutefois 8 pointeurs, qui pointent tous en réalité sur la même liste Info.

La compression est également différente dans cette méthode, par rapport à ce qu'elle était dans les deux méthodes précédentes. La liste Info comprend essentiellement des offsets, suivis de mots pour le nombre. L'offset indique la position des données à modifier relativement au début du bitplane. Celui-ci est toujours positif. Un -1 désigne la fin de la liste. Le mot suivant peut en revanche être positif ou négatif. S'il est positif, il indique le nombre de données (mot ou mot long), à copier dans la liste correspondante du bitplane. Un nombre négatif -x indique en revanche que la donnée suivante doit être copiée x fois dans le bitplane. Le bit 4 du champ de bits indique ici si ces données s'écrivent l'un en dessous de l'autre (bit posé => vertical), ou l'un à côté de l'autre (bit non posé => horizontal).

La cinquième méthode, qui est aussi la plus performante, pour comprimer des fichiers ANIM, permet de générer des fichiers particulièrement réduits. Avec cette méthode, on trouve au début du bloc de données DLTA 8 pointeurs, qui ont la même fonction que

dans la méthode 4. Ils sont suivis par 8 autres pointeurs, qui sont actuellement inutilisés. On pourrait s'en servir pour augmenter le nombre possible de bitplanes.

Les données elles-mêmes, par contre, sont organisées de manière tout à fait différente. Le bitplane est divisé en colonnes, et chaque colonne est large de huit bits, donc d'un octet. Il s'agit donc d'une méthode de compression verticale. Pour chaque colonne, il existe un octet qui indique le nombre des opérations à exécuter sur cette colonne. Après les données de la ligne actuelle, vient l'octet contenant le nombre pour la colonne suivante, etc.

L'opération est définie elle aussi par un octet. Si dans cet octet le septième bit est effacé, cet octet indique le nombre de lignes inchangées. Si le septième bit est posé, l'octet indique avec 0x7F, lié par le ET logique, le nombre d'octets à copier dans les lignes suivantes. Les octets à copier viennent tout de suite après cet octet d'opération. L'octet d'opération a une signification toute particulière s'il est placé sur 0. Dans ce cas, l'octet suivant contient un pointeur qui indique combien de fois l'octet suivant le pointeur doit être copié dans le bitplane.

Avec cette méthode, vous devez absolument noter (comme déjà dans la méthode 4) que les données successives sont écrites verticalement dans le bitplane. Lorsque votre logiciel utilise un pointeur dirigé sur l'emplacement du bitplane dans lequel il faut écrire l'octet suivant, il faut incrémenter sa valeur non pas d'une unité, mais du nombre d'octets contenu dans la largeur du bitplane.

Pour mieux comprendre, nous allons présenter ici les méthodes de compression 3 et 5 sur un exemple. Comme il n'est pas question de comprimer une image de 320x200 pixels, nous nous contenterons de 32x8 pixels. Nous supposons de plus que l'image n'a que deux couleurs, et que le film n'a que 4 images. La figure ci-après montre à gauche les 4 images, et à droite les données des blocs de données DLTA avec la méthode 3.

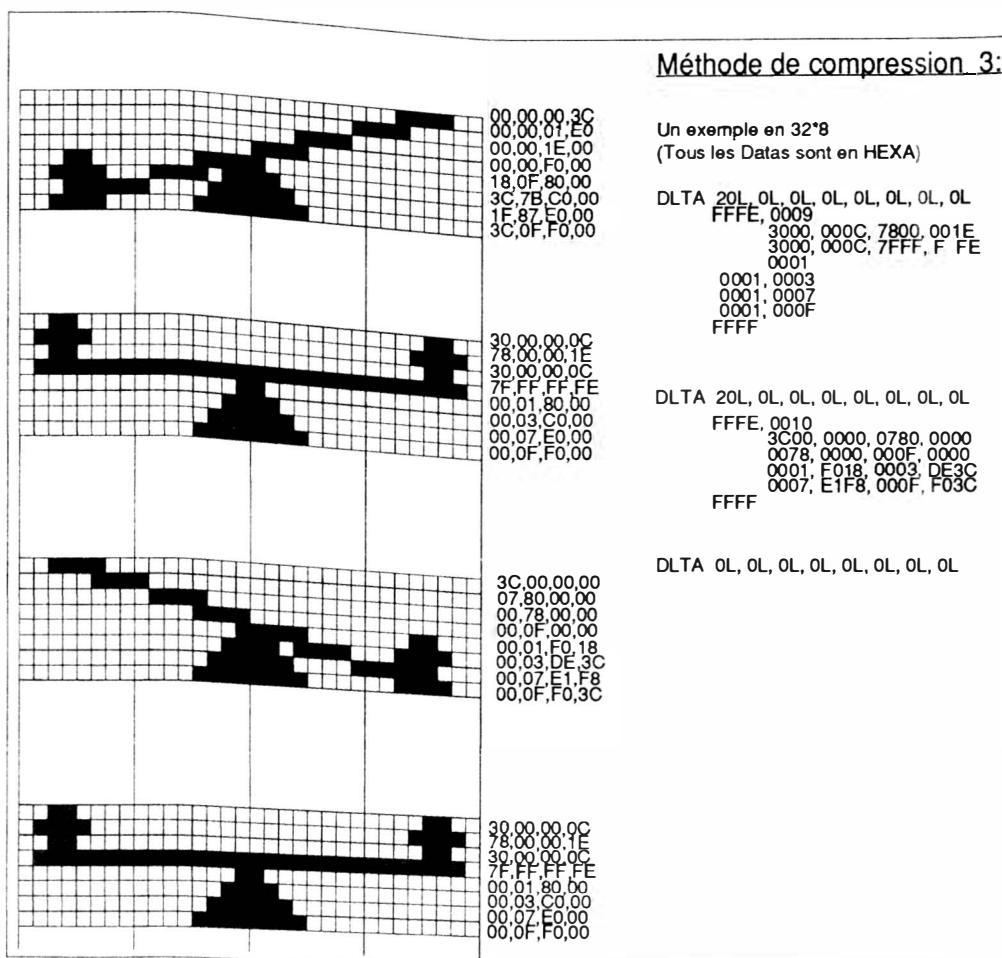


Figure 5 - 55

Le bloc DLTA de la seconde image se réfère aux données du premier bloc, celui de la troisième image au premier, et celui de la quatrième image au second. C'est ce que nous avons expliqué plus haut. La caractérisation 'DLTA' est immédiatement suivie de 8 pointeurs, dont le premier seulement est utilisé pour une image à deux couleurs.

Puis viennent les données proprement dites. Dans l'image 2, on a d'abord 9 mots à copier dans le bitplane. Mais auparavant, l'offset doit être ajouté au pointeur du bitplane. Celui-ci est $(0xFFFF + 2) = 0$. Les données sont donc enregistrées directement au début du bitplane. Puis on saute le mot suivant (Offset = 1), et on écrit le mot 0x0003 dans le bitplane.

Ensuite, on saute à nouveau un mot, et on écrit 0x0007, puis on saute un mot, et on écrit 0x0007. Le 0xFFFF indique que ce bitplane est terminé, et avec lui également l'image.

Dans l'image 3, la compression est beaucoup moins avantageuse. Ici, l'image est différente de la première image dans chacun de ses mots, de sorte qu'il faut écrire au total 16 mots dans le bitplane à partir de l'offset 0 (0xFFFFE+2)).

En revanche, l'image 4 présente des conditions idéales, puisqu'elle est identique à l'image 2. Il suffit ici de placer sur 0 le pointeur dirigé sur l'ensemble des données.

Comparons maintenant ce résultat avec celui de la méthode 5, représenté dans la figure ci-dessous :

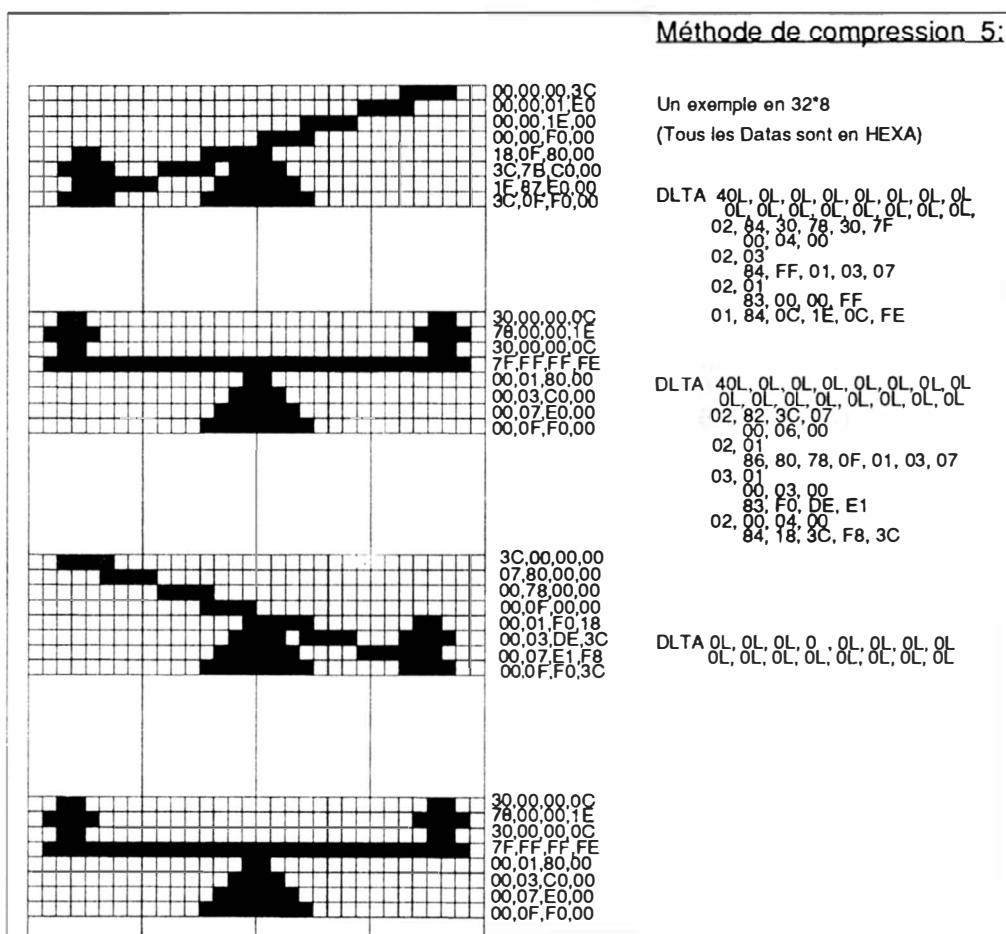


Figure 5 - 56

Au lieu des 8 pointeurs suivant la caractérisation du bloc de données 'DLTA', on a ici 16 pointeurs. En outre, on comprime cette fois verticalement octet par octet, au lieu de comprimer horizontalement mot par mot, comme dans la méthode 3. Considérons

d'abord les données de l'image 2: le premier octet indique le nombre d'opérations dans la colonne actuelle, c'est-à-dire dans la colonne 1; il y a en 2. La première opération a un bit 7 posé, ce qui signifie qu'on a là le nombre des octets devant être écrit dans le bitplane. Dans ce cas, il faut donc écrire 4 octets à écrire, de la ligne 1 à la ligne 4, puisqu'il n'y a pas d'offset posé.

La seconde opération pour la première colonne commence avec un 0. Puis l'octet suivant indique combien de fois il faut copier l'octet suivant dans le bitplane: ici il faut copier 4 fois l'octet de valeur 0 dans le bitplane. Si vous considérez la colonne de gauche des données du bitplane, vous constaterez que toutes les données ont été posées correctement.

Sur la colonne 2, il y a aussi deux opérations à exécuter. Il faut d'abord sauter trois lignes, puis copier 4 octets de la ligne 4 à la ligne 7. Si vous comparez les opérations pour l'image 2 avec cette image, vous pourrez comprendre vous-même comment ces données ont été placées, et comment fonctionne la méthode 5.

Les données pour les autres images ne sont pas très différentes de l'image 2. Dans l'image 3, nous savons même trois commandes par ligne: d'abord sauter une ligne (0x01), puis écrire trois octets nuls dans les lignes suivantes (0x00,0x03,0x00), et enfin trois octets différents (0x83,0xF0,0xDE,0xE1). L'image 4, comme nous le savons, est identique à l'image 2, et les 16 pointeurs sont donc placés sur 0.

Si nous comparons maintenant les deux méthodes de compression, le petit exemple que nous avons pris n'est sans doute pas tout à fait convaincant. Nous constatons en tout cas que les données de compression de la méthode 3 sont pour l'image 2 au nombre de 18 mots = 36 octets; sans les pointeurs, puisque leur besoin en mémoire est constant pour chaque image. Dans la méthode 5, nous obtenons pour la même image seulement 28 octets. Avec des images plus grandes, habituelles dans l'animation, cet avantage augmente considérablement. La méthode 5 est donc excessivement avantageuse.

6. Les librairies de l'Amiga

Dans ce chapitre, nous allons vous présenter les différentes librairies internes et externes de l'Amiga. De même, nous vous donnons la syntaxe adoptée dans tous le chapitre:

Librairie

Nom de la librairie contenant la fonction décrite.

Nom

Nom de la fonction telle qu'elle apparaît dans les descriptions de Commodore.

Syntaxe

Indique la syntaxe exacte de la fonction, ainsi que les registres affectés par celle-ci. De même, il est donné les types des paramètres devant être utilisés par la fonction.

Description

Explication de la fonction et de l'utilisation de celle-ci dans un code source.

Paramètres

Explication et description des paramètres.

Structure

On indique, dans cette partie, la description exacte de la structure de données utilisée par la fonction quand elle existe.

Résultat

Données renvoyées par la fonction après appel de celle-ci.

Avertissement

Nous préciserons dans cette partie les difficultés et problèmes rencontrées.

Exemple

Petit programme explicatif pour mieux préciser une fonction ou une structure de données.

Voir aussi

Renvoie sur d'autres fonctions pour plus de renseignements.

6.1. La librairie Exec

Avant toute chose, nous devons vous rappeler que la plupart des librairies décrites dans ce chapitre doivent être ouvertes avant de pouvoir accéder et utiliser leurs fonctions associées.

Pour ouvrir une librairie, vous devez utiliser la fonction Openlibrary() dont voici la description.

```
struct LibrBase *LibrBase  
LibrBase = Openlibrary("Nom_de_la_librarie",version);  
"Nom_de_la_librarie"
```

chaîne de caractère contenant le nom de la librairie devant être ouverte.

Version

numéro de version de la librairie que vous voulez ouvrir. Une valeur de 0 permet d'ouvrir toutes les versions d'une même librairie. Pour information, la librairie exec du kickstart 1.3 possède un numéro de version égal à 34.2.

En retour, la fonction OpenLibrary() renvoie l'adresse de base de la librairie. Si vous utilisez un linker, vous devrez associer la valeur de retour à une variable bien définie.

Voici le nom des principaux pointeurs de base.

Librairies	Pointeurs de base
diskfont.library	DiskfontBase
dos.library	DosBase
exec.library	ExecBase
graphics.library	GfxBase
icon.library	IconBase
intuition.library	IntuitionBase
layers.library	LayersBase
mathffp.library	MathBase
mathtrans.library	MathTransBase
Mathieeedoubbas.library	MathleeeDoubBasBase

Mathieeedoubtrans.library
transltor.library

MathleeeDoubTransBase
TranslatorBase

Remarque Toutes les librairies acceptent des paramètres dans les registres et les fonctions retournent leur résultat (quand il existe) dans le registre de données D0. La valeur de retour est toujours un mot long de 32 bit, et ce même si certains bits ne sont pas significatifs.

Les fonctions de la librairie exec

Les fonctions suivantes ne peuvent pas être expliquées faute de documentation !

Supervisor	Non détaillée
ExitIntr	Non détaillée
Schedule	Non détaillée
Reschedule	Non détaillée
Switch	Non détaillée
Dispatch	Non détaillée
Exception	Non détaillée

1. Fonctions spéciales

*InitCode	749
InitStruct	750
*FindResident	750
*InitResident	750
*Alert	751
*Debug	751

2. Fonctions de gestion des interruptions

AddIntServer	755
Cause	756
Disable	752
Enable	752
Forbid	753
Permit	753
RemIntServer	756
SetIntVector	755
SetSR	753
SuperState	754
UserState	754

3. Fonctions de mémoire

AllocAbs	759
Allocate	757
AllocEntry	761
AllocMem	758
AvailMem	760
Deallocate	758
FreeEntry	762
FreeMem	760

4. Fonctions de gestion de liste

AddHead	764
AddTail	764
Enqueue	766
FindName	766
Insert	763
RemHead	765
Remove	765
RemTail	765

5. Fonctions des tâches

AddTask	768
AllocSignal	772
AllocTrap	773
FindTask	769
FreeSignal	773
FreeTrap	774
RemTask	768
SetExcept	771
SetSignal	770
SetTaskPri	769
Signal	772
Wait	771

6. Fonctions des messages

AddPort	775
FindPort	778
GetMsg	777
PutMsg	776
RemPort	776
ReplyMsg	777
WaitPort	778

7. Fonctions des librairies

*MakeFonctions	780
*OldOpenLibrary	782
AddLibrary	779
CloseLibrary	780
MakeLibrary	781
OpenLibrary	783
RemLibrary	782
SetFunction	784
SumLibrary	784

8. Fonctions des devices

*AbortIO	789
AddDevice	785
CheckIO	788
CloseDevice	787
DoIO	787
OpenDevice	786
RemDevice	786
SendIO	788
WaitIO	789

9. Fonctions des resources

AddResource	791
OpenResource	792
RemResource	791

10. Fonctions des Sémaphores

*AddSemaphore	795
*AttemptSemaphore	794
*FindSemaphore	795
*InitSemaphore	792
*ObtainSemaphore	793
*ObtainSemaphoreList	794
*ReleaseSemaphore	793
*ReleaseSemaphoreList	795
*RemSemaphore	796
Vacate	793

11. Fonctions du KickStart

*AddMemList	797
*CopyMem	797
*CopyMemQuick	798

Nous vous donnons maintenant la description des structures de cette librairie :

```
struct ExecBase <exec/execbase.h>
{
0x00 00 struct Library LibNode;
0x22 34 WORD SoftVer;
0x24 36 WORD LowMemChkSum;
0x26 38 ULONG ChkBasis;
0x2A 42 APTR ColdCapture;
0x2E 46 APTR CoolCapture;
0x32 50 APTR WarmCapture;
0x36 54 APTR SysStkUpper;
0x3A 58 APTR SysStkLower;
0x3E 62 ULONG MaxLocMem;
0x42 66 APTR DebugEntry;
0x46 70 APTR DebugData;
0x4A 74 APTR AlertData;
0x4E 78 APTR MaxExtMem;
0x52 82 WORD ChkSum;
0x54 84 struct IntVector IntVects[16];
0x114 276 struct Task *ThisTask;
0x118 280 ULONG IdleCount;
0x11C 284 ULONG DispCount;
0x120 288 WORD Quantum;
0x122 290 WORD Elapsed;
0x124 292 WORD SysFlags;
0x126 294 BYTE IDNestCnt;
0x127 295 BYTE TDNestCnt;
0x128 296 WORD AttnFlags;
0x12A 298 WORD AttnResched;
0x12C 300 APTR ResModules;
0x130 304 APTR TaskTrapCode;
0x134 308 APTR TaskExceptCode;
0x138 312 APTR TaskExitCode;
0x13C 316 ULONG TaskSigAlloc;
0x140 320 WORD TaskTrapAlloc;
0x142 322 struct List MemList;
0x146 336 struct List ResourceList;
0x15E 350 struct List DeviceList;
0x16C 364 struct List IntrList;
0x17A 378 struct List LibList;
0x188 392 struct List PortList;
0x196 406 struct List TaskReady;
0x1A4 420 struct List TaskWait;
0x1B2 434 struct SoftIntList SoftInts[5];
0x202 514 LONG LastAlert[4];
0x212 530 UBYTE VBlankFrequency;
0x213 531 UBYTE PowerSupplyFrequency;
0x214 532 struct List SemaphoreList;
0x222 546 APTR KickMemPtr;
0x226 550 APTR KickTagPtr;
```

```

0x22A 554 APTR KickCheckSum;
0x22C 558 UBYTE ExecBaseReserved[10];
0x238 568 UBYTE ExecBaseNewReserved[20];
0x24C 588
};

SYSBASESIZE ((long)sizeof(struct ExecBase))

Processor_Bitnumber:
AFB_68010 0L
AFB_68020 1L
AFB_68881 4L

Processor_Flags:
AFF_68010 (1L<<0)
AFF_68020 (1L<<1)
AFF_68881 (1L<<4)

Bytes:
AFB_RESERVED8 8L
AFB_RESERVED9 9L

```

Description des fonctions

1. Fonctions spéciales

*InitCode	Initialiser un module de code résident
-----------	--

Syntaxe

```

InitCode(StartClass, Version);
-72          D0          D1
ULONG StartClass;
ULONG Version;

```

Description

Cette fonction initialise tous les modules résidents avec la classe de départ donnée (StartClass) et avec une version égale ou supérieure à celle spécifiée (Version). Les modules sont initialisés dans un ordre de priorité.

Paramètres

StartClass: Classe du code à initialiser (coldstart, coolstart, warmstart ...).
 Version: Numéro de version.

InitStruct**Initialiser la mémoire à partir d'une table****Syntaxe**

```
InitStruct(InitTable, Memory, Size);
    -78      A1      A2      D0
    ULONG *InitTable;
    ULONG *Memory;
    ULONG Size;
```

Description

Cette fonction initialise une partie de la mémoire à partir d'une table de données.

Paramètres

- InitTable: Début de la table d'initialisation. Celle-ci doit se trouver à une adresse paire.
- Memory: Début de la mémoire à initialiser.
- Size: Taille de la mémoire (nombre paire).

FindResident*Trouver un module résident par son nom****Syntaxe**

```
Resident = FindResident(Name);
    D0      -96      A1
    ULONG *Resident;
    char  *Name;
```

Description

Cette fonction permet de trouver un module résident en utilisant le nom donné. Si la fonction trouve, un pointeur sur la structure est renvoyé, sinon c'est un 0.

Paramètres

- Name: Pointeur sur le nom.

Retour

- Resident: Pointeur sur la structure Tag ou 0 si pas trouvé.

InitResident*Initialiser un module résident****Syntaxe**

```
InitResident(Resident, SegList);
    -102      A1      D1
    ULONG *Resident;
    SegList SegList;
```

```
ULONG      *Resident;
struct List *SegList;
```

Description

Cette fonction permet d'initialiser un module résident (aussi appelé "ROM-Tags").

Paramètres

Resident: Pointeur sur le module.

SegList: Pointeur sur la liste.

*Alert	Alerter l'utilisateur d'une erreur
---------------	---

Syntaxe

```
Alert(AlertNum, Parameters);
-108      D7      A5
ULONG AlertNum;
ULONG *Parameters;
```

Description

Cette fonction alerte l'utilisateur d'un problème grave dans le système. Si l'alerte est non recouvrable, un reset est exécuté. L'appel peut être exécuté à n'importe quel moment, y compris pendant une interruption.

Paramètres

AlertNum: Nombre indiquant l'alerte

Parameters: Pointeur sur la tâche devant être exécutée au moment du problème

*Debug	Lancer le debugger système
---------------	-----------------------------------

Syntaxe

```
Debug();
-114
```

Description

Cette fonction lance le debugger système. Celui-ci est de type "ROM-WACK" par défaut.

Structure

```
struct Resident <exec/resident.h>
{
0x00 00 UWORLD rt_MatchWord;
0x02 02 struct Resident *rt_MatchTag;
0x06 06 APTR rt_EndSkip;
0xA 10 UBYTE rt_Flags;
```

```

0x0B 11 UBYTE rt_Version;
0x0C 12 UBYTE rt_Type;
0x0D 13 BYTE rt_Pri;
0x0E 14 char *rt_Name;
0x12 18 char *rt_IdString;
0x16 22 APTR rt_Init;
0x1A 26
};

Resident_Flags
RTC_MATCHWORD 0x4AFCL
RTF_AUTOINIT (1L<<7)
RTF_COLDSTART (1L<<0)
RTM_WHEN 3L
RTW_NEVER 0L
RTW_COLDSTART 1L

```

2. Fonctions de gestion des interruptions

Disable

Interdire toutes les interruptions

Syntaxe

```
Disable();
-120
```

Description

Cette fonction interdit toutes les interruptions et incrémente IDNestCnt. Attention, il faut l'utiliser avec précaution.

Avertissement

Cette fonction doit être utilisée avec beaucoup de précautions, car le multi-tâches de l'Amiga est géré par les interruptions.

Voir aussi

[Enable\(\)](#), [Forbid\(\)](#), [Permit\(\)](#)

Enable

Réactiver les interruptions

Syntaxe

```
Enable();
-126
```

Description

Cette fonction décrémente IDNestCnt. Lorsque celui-ci est négatif, les interruptions sont de nouveau possibles.

S'utilise avec la fonction [Disable\(\)](#).

Voir aussi

Disable(), Forbid(), Permit()

Forbid**Interdire le Task-Switching****Syntaxe**

```
Forbid();
-132
```

Description

Cette fonction permet d'interdire le Task-Switching et incrémente TDNestCnt.

Voir aussi

Permit(), Disable(), Enable()

Permit**Autoriser le Task-Switching****Syntaxe**

```
Permit();
-138
```

Description

Cette fonction décrémente TDNestCnt et autorise le Task-Switching lorsque TDNestCnt est négatif.

Voir aussi

Forbid(), Disable(), Enable()

SetSR**Obtenir ou fixer le registre d'état du processeur****Syntaxe**

```
OldSR = SetSR(NewSR, Mask)
DO      -144    DO      D1
LONG OldSR;
LONG NewSR;
LONG Mask;
```

Description

Cette fonction permet d'obtenir ou bien de fixer le registre d'état d'une manière dite 'saine'.

Paramètres

NewSR: Nouvelles valeurs pour les bits spécifiés dans le masque.

Mask: Bits devant être changés.

Retour

OldSR: Registre d'état avant l'appel.

Exemple

Pour obtenir le registre d'état

```
CurrentSR = SetSR(0,0);
```

SuperState**Passer en mode Superviseur avec la pile utilisateur**

Syntaxe OldSysStack = SuperState()
 D0 -150

Description

Cette fonction permet le passage en mode Superviseur en utilisant la pile utilisateur. Cela signifie qu'il est possible d'avoir accès aux variables de cette même pile.

Avertissement

La pile utilisateur doit être suffisamment grande pour recevoir toutes les données des interruptions.

Retour

OldSysStack: Pointeur sur la pile système. Très utile lors du retour au mode Utilisateur. Si le système est déjà en mode Superviseur, OldSysStack est nul.

Voir aussi

UserState()

UserState**Retour au mode Utilisateur**

Syntaxe UserState(SysStack)
 -156 D0

Description

Cette fonction permet de revenir au mode Utilisateur avec la pile utilisateur depuis le mode Superviseur utilisant la pile utilisateur.

Avertissement

Cette fonction NE DOIT PAS être appelée depuis le mode Utilisateur.

Paramètres

SysStack: Pointeur de pile utilisateur.

Voir aussi

SuperState()

SetIntVector

Définir une interruption système

Syntaxe

```
OldInterrupt = SetIntVector(IntNumber, Interrupt)
              D0      -162      D0      A1
              struct Node *OldInterrupt;
              LONG      IntNumber;
              struct Node *Interrupt;
```

Description

Cette fonction permet de définir un vecteur d'interruption système. Elle ne sont pas partagable et une nouvelle définition déconnecte les anciens handlers.

Paramètres

IntNumer: Nombre des interruptions.

Interrupt: Pointeur de structure d'interruption contenant l'entrée du handler et le pointeur de données.

Retour

OldInterrupt: Pointeur qui a le contrôle de cette interruption.

AddIntServer

Ajouter une interruption serveur au système

Syntaxe

```
AddIntServer(IntNumber, Interrupt)
              -168      D0      A1
              LONG      IntNumber;
              struct Node *Interrupt;
```

Description

Cette fonction ajoute une nouvelle interruption serveur à une chaîne de serveur donnée. Le noeud est localisé sur la chaîne, la position déterminant la priorité. Si c'est la première sur cette chaîne, les interruptions seront possibles sur la chaîne.

Paramètres

- IntNumber: Niveau de l'interruption.
 Interrupt: Pointeur sur le noeud de la routine d'interruption.

Voir aussi

`RemIntServer()`, `Cause()`, `SetIntHandler()`

RemIntServer

Supprimer un vecteur d'interruption système

Syntaxe

```
RemIntServer(IntNumber, Interrupt)
-174          D0          A1
LONG          IntNumber;
struct Node *Interrupt;
```

Description

Cette fonction supprime une interruption dans la chaîne donnée.

Paramètres

- IntNumber: Nombre des interruptions.
 Interrupt: Pointeur d'interruption.

Voir aussi

`AddIntServer()`, `Cause()`, `SetIntHandler()`

Cause

Produire un "Software-Interrupt"

Syntaxe

```
Cause(Interrupt)
-180          A1
struct Node *Interrupt;
```

Description

Cette fonction produit une interruption. On dispose de 5 priorités d'interruptions (-32, -16, 0, +16, +32).

Paramètres

Interrupt: Pointeur sur l'initialisation de l'interruption.

Voir aussi

AddIntServer(), RemIntServer(), SetIntHandler()

Structure

```
struct Interrupt <exec/interrupts.h>
{
0x00 00 struct Node is_Node;
0x0E 14 APTR is_Data;
0x12 18 VOID (*is_Code)();
0x16 22
};

struct IntVector <exec/interrupts.h>
{
0x00 00 APTR iv_Data;
0x04 04 VOID (*iv_Code)();
0x08 08 struct Node *iv_Node;
0x0C 12
};

struct SoftIntList <exec/interrupts.h>
{
0x00 00 struct List sh_List;
0x0E 14 UWORLD sh_Pad;
0x10 16
};

Interrupt_Type
SIH_PRIMASK (0xf0L)
INTB_NMI 15L
INTF_NMI (1L<<15)
```

3. Fonctions de mémoire

Allocate	Définir une zone mémoire
----------	--------------------------

Syntaxe

```
MemoryBlock = Allocate(FreeList, ByteSize);
              D0      -186     A0      D0
              ULONG    *MemoryBlock;
struct MemHeader *FreeList;
ULONG           ByteSize;
```

Description

Cette fonction est utilisée pour allouer un bloc de mémoire.

Paramètres

FreeList: Pointe sur le "memory list header".

ByteSize: Taille du bloc désiré en octets.

Retour

MemoryBlock: Pointeur sur le bloc libre alloué.

Voir aussi

`Deallocate()`, `AllocAbs()`, `AllocMem()`, `AllocEntry()`, `AllocRemember()`

Deallocate**Retirer un bloc de mémoire****Syntaxe**

```
Deallocate(FreeList, MemoryBlock, ByteSize);
           -192      A0      A1      D0
           struct MemHeader *FreeList;
           ULONG          *MemoryBlock;
           ULONG          ByteSize;
```

Description

Cette fonction retire un bloc de mémoire déjà défini. De même, elle peut retirer un ancien bloc défini auparavant.

Paramètres

FreeList: Pointe sur la liste libre (`freeList`).

MemoryBlock: Bloc de mémoire à retourner.

ByteSize: Taille du bloc désiré en octets.

Voir aussi

`Allocate()`

AllocMem**Réserver une partie de la mémoire****Syntaxe**

```
MemoryBlock = AllocMem(ByteSize, Requirements);
           D0      -198      D0      D1
           ULONG *MemoryBlock;
           ULONG ByteSize;
           ULONG Requirements;
```

Description

Cette fonction réserve une partie de la mémoire. Celle-ci peut être de la mémoire-Chip, de la mémoire-Fast ou bien de la mémoire-publique.

Paramètres

ByteSize: Taille du bloc mémoire désiré.

Requirements: MEMF_CHIP:
Si la mémoire doit être utilisée par les processeurs spécialisés, vous DEVEZ définir de la mémoire CHIP. En effet, seule celle-ci est accessible par ceux-ci.

MEMF_FAST:

C'est ce type de mémoire qui sera défini par défaut par le système

MEMF_PUBLIC:

Mémoire publique option possible

MEMF_CLEAR:

Vous pouvez demander que le bloc mémoire soit initialisé par des 0.

Retour

MemoryBlock: Pointeur sur le bloc défini. Si la définition ne peut pas se faire, un 0 est retourné.

Voir aussi

[FreeMem\(\)](#), [Allocate\(\)](#), [AllocEntry\(\)](#), [AllocAbs\(\)](#), [AllocRemember\(\)](#)

AllocAbs

Réserver une partie de la mémoire

Syntaxe

```
MemoryBlock = AllocAbs(ByteSize, Location);
          D0      -204     D0      A1
          ULONG *MemoryBlock;
          ULONG ByteSize;
          ULONG Location;
```

Description

Cette fonction essaie de réservé une partie de la mémoire à partir d'une adresse absolue. Si le bloc est déjà utilisé, un 0 est retourné.

Paramètres

ByteSize: Taille du bloc désiré en octets.

Location: Adresse où le bloc de mémoire doit être.

Retour

MemoryBlock: Pointeur sur le nouveau bloc ou un 0 si la réservation n'est pas bonne.

Voir aussi

`FreeMem()`, `AllocMem()`, `AllocEntry()`, `Allocate()`, `AllocRemember()`

FreeMem

Libérer une zone mémoire

Syntaxe

```
FreeMem(MemoryBlock, ByteSize);
        -210      A1          D0
        ULONG *MemoryBlock;
        ULONG ByteSize;
```

Description

Cette fonction libère une partie de la mémoire déjà allouée.

Paramètres

MemoryBlock: Bloc mémoire à libérer.

ByteSize: Taille du bloc en octets.

Avertissement

Si vous essayez de libérer un bloc qui l'a déjà été, le système libérera un "Guru-meditation".

Voir aussi

`AllocMem()`, `freeentry()`, `FreeRemember()`, `Deallocate()`

AvailMem

Donner la taille de la mémoire libre

Syntaxe

```
Size = AvailMem(Requirements);
        DO      -216      D1
        ULONG Size;
        ULONG Requirements;
```

Description

Cette fonction renvoie le total de mémoire libre. Pour trouver le type du plus grand bloc de mémoire libre, utilisez l'argument `MEMF_LARGEST`.

Paramètres

Requirements: Masque de requête (le même que pour AllocMem).

Retour

Size: Espace libre total ou bien plus grand bloc libre(suivant paramètre dans Requirements).

Avertissement

A cause du multitâche, la valeur renournée par cette fonction ne sera peut-être pas exactement la valeur réelle.

AllocEntry

Réserver plusieurs régions de la mémoire

Syntaxe

```
MemList = AllocEntry(Entry);
          DO      -222     A0
struct MemList *MemListe;
struct MemList *Entry;
```

Description

Cette fonction alloue plusieurs parties de la mémoire comme précisé dans la liste.

Paramètres

Entry: Liste de description.

Retour

MemList: Liste différente de la précédente contenant la mémoire actuelle allouée.

Voir aussi

FreeEntry(), AllocMem(), AllocAbs(), Allocate(), AllocRemember()

Exemple

```
MemListDefin:
DC.B LN_SIZE      ;Réserve la place en mémoire pour la liste
DC.W 5      ;Nombre d'entrée
DC.L MEMF_CLEAR  ;entrée numéro 0
DC.L 2
DC.L MEMF_PUBLIC ;entrée numéro 1
DC.L 4
DC.L MEMF_CHIP|MEMF_CLEAR    ;entrée numéro 2
DC.L 8
DC.L MEMF_FAST|MEMF_CLEAR    ;entrée numéro 3
DC.L 16
DC.L MEMF_PUBLIC|MEMF_CLEAR  ;entrée numéro 4
```

```

DC.L 32
Début:
    LEA.L Memlistdefin(PC),A0
    JSR    _LV0AllocEntry(A6)
    BCLR.L      #31,D0
    BEQ.S success

```

FreeEntry**Libérer plusieurs parties de la mémoire****Syntaxe**

```

FreeEntry(Entry):
    -228      A0
    struct MemList *Entry;

```

Description

Cette fonction libère la mémoire allouée par la fonction AllocEntry.

Paramètres

Entry: Pointe sur la liste rentrée par la fonction MemEntry.

Voir aussi

AllocEntry(), FreeMem(), Deallocate(), FreeRemember()

Structure

```

struct MemHeader <exec/memory.h>
{
0x00 00 struct Node mh_Node;
0x0E 14 UWORD mh_Attributes;
0x10 16 struct MemChunk *mh_First;
0x14 20 APTR mh_Lower;
0x18 24 APTR mh_Upper;
0x1C 28 ULONG mh_Free;
0x20 32
};

struct MemChunk <exec/memory.h>
{
0x00 00 struct MemChunk *mc_Next;
0x04 04 ULONG mc_Bytes;
0x08 08
};

struct MemList <exec/memory.h>
{
0x00 00 struct Node ml_Node;
0x0E 14 UWORD ml_NumEntries;
0x10 16 struct MemEntry ml_ME[1];
0x18 24
};

ml_me ml_ME

```

```

Memory_Types:
MEMF_PUBLIC (1L<<0)
MEMF_CHIP (1L<<1)
MEMF_FAST (1L<<2)
MEMF_CLEAR (1L<<16)
MEMF_LARGEST (1L<<17)
MEM_BLOCKSIZE 8L
MEM_BLOCKMASK 7L

struct MemEntry <exec/memory.h>
{
union
{
    ULONG meu_Req;
    APTR meu_Addr;
}
0x00  me_Un;
0x04  04  ULONG me_Length;
0x08  08
};

Definition:
me_un      me_Un
me_Req     me_Un.meu_Req;
me_Addr   me_Un.meu_Addr

```

4. Fonction de gestion de liste

Insert

Insérer un noeud dans une liste

Syntaxe

```

Insert(Liste, Node, Précédent)
-234  A0  A1  A2
struct List *Liste;
struct Node *Node;
struct Node *Précédent;

```

Description

Cette fonction permet d'insérer un noeud dans une liste.

Paramètres

- | | |
|------------|---|
| Liste: | Pointeur sur la liste où le noeud doit être inséré. |
| Node: | Pointeur sur le noeud dans la liste où a lieu l'insertion. |
| Précédent: | Pointeur sur le noeud précédent l'insertion. Si ce pointeur a une valeur différente de 0, le paramètre Liste ne présente plus d'intérêt. Ce ne sera pas le pointeur du noeud qui sera employé, pour déterminer la position de l'insertion, mais le paramètre précédent. |

Voir aussi

`Remove()`, `AddHead()`, `RemHead()`, `AddTail()`, `RemTail()`

AddHead**Insérer un noeud en début de liste****Syntaxe**

```
AddHead(Liste, Node)
-240      A0      A1
struct List *Liste;
struct Node *Node;
```

Description

Cette fonction est utilisée pour insérer un noeud en tête d'une liste.

Paramètres

Liste: Pointeur sur la liste où le noeud doit être inséré.

Node: Pointeur sur le noeud à insérer.

Voir aussi

`Insert()`, `Remove()`, `RemHead()`, `AddTail()`, `RemTail()`

AddTail**Insérer un noeud en fin de liste****Syntaxe**

```
AddTail(Liste, Node)
-246      A0      A1
struct List *Liste;
struct Node *Node;
```

Description

Cette fonction permet d'insérer un noeud comme dernier membre d'une liste.

Paramètres

Liste: Pointeur sur la liste où le noeud doit être inséré.

Node: Pointeur sur le noeud qui doit être inséré.

Voir aussi

`Insert()`, `Remove()`, `AddHead()`, `RemHead()`, `RemTail()`

Remove**Eliminer un noeud d'une liste****Syntaxe**

```
Remove(Node)
      -252   A1
      struct Node *Node;
```

Description

Cette fonction permet d'éliminer facilement un noeud dans une liste.

Paramètres

Node: Pointeur sur le noeud qui doit être détruit. S'il ne s'agit pas d'un pointeur sur le noeud, Exec ne le reconnaît pas, mais il tente quand même de l'éliminer, ce qui peut provoquer une perte d'informations ou un effondrement du système.

Voir aussi

Insert(), AddHead(), RemHead(), AddTail(), RemTail()

RemHead**Eliminer le premier noeud d'une liste****Syntaxe**

```
Node = RemHead(Liste)
      D0      -258   A0
      struct List *Liste;
```

Description

Cette fonction permet d'éliminer le premier noeud d'une liste.

Paramètres

Liste: Pointeur sur la liste où le premier noeud doit être éliminé.

Voir aussi

Insert(), Remove(), AddHead(), AddTail(), RemTail()

RemTail**Eliminer la dernière entrée****Syntaxe**

```
Node = RemTail(Liste)
      D0      -264   A0
      struct List *Liste;
```

Description

Cette fonction permet d'éliminer la dernière entrée d'une liste.

Paramètres

Liste: Pointeur sur la liste où le dernier enregistrement doit être éliminé.

Voir aussi

`Insert()`, `Remove()`, `AddHead()`, `RemHead()`, `AddTail()`

Enqueue**Classer des entrées dans une liste****Syntaxe**

```
Enqueue(Liste, Node)
-270      A0      A1
struct List *Liste;
struct Node *Node;
```

Description

Cette fonction est utilisée pour classer des entrées dans une liste suivant leurs niveaux de priorité. Le noeud possédant le niveau de priorité le plus important sera mis au début de la liste. Si plusieurs noeuds ont la même priorité, le nouveau sera mis derrière ceux qui sont déjà présents.

Paramètres

Liste: Pointeur sur la liste où le noeud doit être inséré.

Node: Pointeur sur le noeud qui sera inséré.

FindName**Chercher un noeud à partir d'un nom****Syntaxe**

```
Node = FindName(Liste, Name)
D0      -276      A0      A1
struct Node *Node;
struct List *Liste;
char        *Name;
```

Description

Cette fonction permet de rechercher dans une liste indiquée un noeud ayant le nom mentionné.

Paramètres

Liste: Pointeur sur la liste dans laquelle s'effectue la recherche.

Name: Pointeur sur le nom recherché. Cette chaîne de caractères doit être terminée par un 0.

Retour

Node: La fonction renvoie un pointeur sur le noeud trouvé. Si aucune entrée ne correspond au nom fixé, la fonction renverra un 0.

Structure

```
struct List <exec/lists.h>
{
0x00 00 struct Node *lh_Head;
0x04 04 struct Node *lh_Tail;
0x08 08 struct Node *lh_TailPred;
0x0C 12 UBYTE lh_Type;
0x0D 13 UBYTE lh_pad;
0x0E 14
};

struct MinList <exec/lists.h>
{
0x00 00 struct MinNode *mlh_Head;
0x04 04 struct MinNode *mlh_Tail;
0x08 08 struct MinNode *mlh_TailPred;
0x0C 12
};

struct Node <exec/nodes.h>
{
0x00 00 struct Node *ln_Succ;
0x04 04 struct Node *ln_Pred;
0x08 08 UBYTE ln_Type;
0x09 09 BYTE ln_Pri;
0x0A 10 char *ln_Name;
0x0E 14
};

struct MinNode <exec/nodes.h>
{
0x00 00 struct MinNode *mln_Succ;
0x04 04 struct MinNode *mln_Pred;
0x08 08
};

Node_Type:
NT_UNKNOWN      0L
NT_TASK          1L
NT_INTERRUPT     2L
NT_DEVICE         3L
NT_MSGPORT        4L
NT_MESSAGE        5L
NT_FREEMSG        6L
NT_REPLYMSG       7L
NT_RESOURCE        8L
NT_LIBRARY         9L
NT_MEMORY          10L
NT_SOFTINT         11L
```

NT_FONT	12L
NT_PROCESS	13L
NT_SEMAPHORE	14L
NT_SIGNALSEM	15L

5. Fonctions des tâches

AddTask

Insérer une nouvelle tâche dans le système

Syntaxe

```
AddTask(Task, InitPC, FinalPC)
        -282    A1      A2      A3
        struct Task *Task;
        ULONG      InitPC;
        ULONG      FinalPC;
```

Description

Cette fonction permet d'insérer une nouvelle tâche dans le système.

Paramètres

- Task: Pointeur sur une structure de tâche. Les champs tc_SPUpper, tc_SPLower, tc_SPReg et tc_Node.In_Type doivent être initialisés correctement.
- InitPC: Adresse de début de traitement du programme du module.
- FinalPC: Adresse de retour qui, au début de la tâche, est mise sur la pile. Si la tâche exécute un RTS en sumombre, on saute à cette adresse. Si finalPC est à 0, Exec met en oeuvre sa routine standard finalPC.

Voir aussi

RemTask(), FindTask()

RemTask

Éliminer une tâche du système

Syntaxe

```
RemTask(Task)
        -288    A1
        struct Task *Task;
```

Description

Cette fonction sert à éliminer une tâche du système. Si le champ tc_MemEntry pointe sur une liste MemEntry, cette dernière sera libérée. Toutes les autres ressources système occupées par la tâche doivent avoir été restituées auparavant au système.

Paramètres

Task: Pointeur sur la structure Task de la tâche à éliminer. Si Task = 0, la tâche en cours sera éliminée. Une fois la tâche supprimée, on revient dans la fonction Switch de Exec.

Voir aussi

AddTask(), FindTask()

FindTask

Chercher une tâche

Syntaxe

```
Task = FindTask(Name)
      D0      -294     A1
      struct Task *Task;
      char      *Name;
```

Description

Cette fonction cherche une tâche portant un nom déterminé dans la liste des tâches. Si cette tâche est trouvée, un pointeur sur cette structure de tâche sera délivré. Si on indique 0 pour le nom, on obtiendra un pointeur sur la structure Task de la tâche en cours.

Paramètres

Name: Pointeur sur le nom de la tâche recherchée ou 0.

Retour

Task: Pointeur sur la structure Task de la tâche recherchée.

Voir aussi

AddTask(), RemTask()

SetTaskPri

Définir une priorité à une tâche

Syntaxe

```
OldPriority = SetTaskPri(Task, Priority)
      D0      -300     A1      D0
      BYTE    OldPriority;
      struct Task *Task;
      BYTE    Priority;
```

Description

Cette fonction renvoie l'ancienne priorité d'une tâche et donne à la tâche sa nouvelle priorité. On a en plus l'exécution d'un Rescheduling, ce qui signifie que le temps de

calcul de chaque tâche est redécoupé suivant les nouvelles priorités. Si on donne une haute priorité à une tâche avec cette fonction, cette dernière aura accès immédiatement au processeur.

Paramètres

- Task: Correspond au pointeur sur la structure Task de la tâche.
- Priority: Correspond à la nouvelle priorité de la tâche (dans les 8 bits inférieurs de D0).

Retour

- OldPriority: Correspond à l'ancienne priorité de la tâche (dans les 8 bits inférieurs de D0).

SetSignal

Indiquer l'état d'une tâche

Syntaxe

```
OldSignals = SetSignal(NewSignals, SignalSet)
          D0      -306      D0      D1
ULONG OldSignals;
ULONG NewSignals;
ULONG SignalSet;
```

Description

Cette fonction transfère l'état de chaque signal dont les bits correspondants sont posés dans le masque, en les prenant dans NewSignals et en les portant dans les signaux de la tâche (tc_SigRecv). Si un bit de NewSignals et de SignalSet est égal à 1, le signal correspondant sera activé. Si un bit de NewSignals = 0 alors que celui de SignalSet est égal à 1, le signal sera supprimé. Si un bit de SignalSet est égal à 0, le signal correspondant reste inchangé.

Paramètres

- NewSignal: Contient le nouvel état des signaux.
- SignalSet: Détermine le bit Signal à modifier.

Retour

- OldSignal: Indique l'état précédent des signaux de tâche.

Voir aussi

`AllocSignal()`, `FreeSignal()`

SetExcept**Déterminer les signaux d'exception****Syntaxe**

```
OldSignals = SetExcept(NewSignals, SignalSet)
      DO      -312      DO      D1
ULONG OldSignals;
ULONG NewSignals;
ULONG SignalSet;
```

Description

Cette fonction détermine les signaux qui peuvent être déclenchés par une exception. Le comportement de cette fonction est identique à celui de `SetSignal()`.

Paramètres

`NewSignals:` Nouvel état des signaux.

`SignalSet:` Etablir quels signaux doivent être influencés.

Retour

`OldSignals:` Etat des signaux avant la modification.

Wait**Attendre un signal de masque****Syntaxe**

```
Signals = Wait(SignalSet)
      DO      -318      DO
ULONG Signals;
ULONG SignalSet;
```

Description

Cette fonction attend un signal du masque des signaux indiqués. Ceci signifie qu'une tâche restera dans l'état Waiting aussi longtemps que le signal ne sera pas occupé par une autre tâche ou une autre interruption. Si un des signaux est activé avant l'appel de la fonction `Wait`, `Wait` retourne au programme. Le résultat se présente sous la forme d'un masque de signaux contenant les signaux attendus.

Paramètres

`SignalSet:` Signaux attendus par la tâche.

Retour

`Signals:` Signaux réceptionnés.

Voir aussi

`Signal()`

Signal**Allouer un signal****Syntaxe**

```
Signal(Task, SignalSet)
-324  A1      D0
struct Task *Task;
ULONG      SignalSet;
```

Description

Cette fonction permet d'allouer un signal d'une autre tâche. Elle est le centre du système des signaux car elle permet de communiquer entre tâches. Lorsqu'une tâche reçoit un signal attendu, elle retourne automatiquement dans le mode Ready ou Running.

Paramètres

Task: Pointeur sur la structure Task de la tâche réceptrice.

SignalSet: Masque de signaux contenant le bits Signal à communiquer.

Voir aussi

[Wait\(\)](#)

AllocSignal**Allouer un signal de tâche****Syntaxe**

```
SignalNum = AllocSignal(SignalNum)
DO          -330      D0
LONG SignalNum;
LONG SignalNum;
```

Description

Cette fonction permet d'allouer un des signaux de tâche. Si on donne comme numéro de signal -1, et non pas le numéro de signal à allouer, AllocSignal() cherche le premier signal libre et l'alloue. Si le signal souhaité est déjà occupé, AllocSignal() retourne la valeur -1.

Paramètres

SignalNum: Numéro du signal à occuper (0-31) ou -1 pour le prochain signal libre.

Retour

SignalNum: Numéro du signal occupé ou -1 lorsque le signal demandé est déjà occupé (ou tous les signaux avec AllocSignal(-1)).

Voir aussi[FreeSignal\(\)](#)**FreeSignal****Libérer un signal****Syntaxe**

```
FreeSignal(SignalNum)
    -336      DO
LONG SignalNum;
```

Description

Cette fonction est la fonction inverse de AllocSignal(). Le signal correspondant au numéro est libéré. De la même manière que AllocSignal(), FreeSignal() ne doit pas être employé à partir d'une exception.

Paramètres

SignalNum: Numéro du signal à libérer (0-31).

Voir aussi[AllocSignal\(\)](#)**AllocTrap****Allouer une instruction Trap****Syntaxe**

```
TrapNum = AllocTrap(TrapNum)
    DO      -342      DO
LONG TrapNum;
LONG TrapNum;
```

Description

Cette fonction permet d'allouer une des instructions Trap du 68000. Le numéro Trap peut être compris entre 0 et 15, initialisant ainsi l'instruction Trap correspondante. Avec le numéro -1, AllocTrap recherche la première instruction Trap libre.

Paramètres

TrapNum: Nombre compris entre 0 et 15 pour une instruction Trap déterminée (ou -1 pour la première instruction libre).

Retour

TrapNum: Contient l'instruction Trap allouée. Si numtrap = -1, c'est que l'instruction souhaitée n'est pas libre ou qu'il n'y a plus d'instruction libre.

Voir aussi[FreeTrap\(\)](#)**FreeTrap****Libérer une instruction Trap****Syntaxe**

```
FreeTrap(TrapNum)
    -348      D0
    LONG TrapNum;
```

Description

Cette fonction libère l'instruction Trap caractérisée par le numéro donné.

Paramètres

TrapNum: Numéro de l'instruction Trap (0-15).

Voir aussi[AllocTrap\(\)](#)**Structure**

```
extern struct Task <exec/tasks.h>
{
    0x00 00 struct Node tc_Node;
    0x0E 14 UBYTE tc_Flags;
    0x0F 15 UBYTE tc_State;
    0x10 16 BYTE tc_IDNestCnt;
    0x11 17 BYTE tc_TDNestCnt;
    0x12 18 ULONG tc_SigAlloc;
    0x16 22 ULONG tc_SigWait;
    0x1A 26 ULONG tc_SigRecv;
    0x1E 30 ULONG tc_SigExcept;
    0x22 34 UWORLD tc_TrapAlloc;
    0x24 36 UWORLD tc_TrapAble;
    0x26 38 APTR tc_ExceptData;
    0x2A 42 APTR tc_ExceptCode;
    0x2E 46 APTR tc_TrapData;
    0x32 50 APTR tc_TrapCode;
    0x36 54 APTR tc_SPReg;
    0x3A 58 APTR tc_SPLower;
    0x3E 62 APTR tc_SPUpper;
    0x42 66 VOID (*tc_Switch)();
    0x46 70 VOID (*tc_Launch)();
    0x4A 74 struct List tc_MemEntry;
    0x58 88 APTR tc_UserData;
    0x5C 92
};

Task_Bytes:
TB_PROCTIME      0L
TB_STACKCHK      4L
```

```

TB_EXCEPT      5L
TB_SWITCH      6L
TB_LAUNCH      7L

Task_Flags:
TF_PROCTIME    (1L<<0)
TF_STACKCHK    (1L<<4)
TF_EXCEPT      (1L<<5)
TF_SWITCH      (1L<<6)
TF_LAUNCH      (1L<<7)

Task_State:
TS_INVALID 0L
TS_ADDED    1L
TS_RUN      2L
TS_READY    3L
TS_WAIT     4L
TS_EXCEPT   5L
TS_REMOVED  6L

Signal_Byt:
SIGB_ABORT    0L
SIGB_CHILD    1L
SIGB_BLIT     4L
SIGB_SINGLE   4L
SIGB_DOS      8L

Signal_Flag:
SIGF_ABORT    (1L<<0)
SIGF_CHILD    (1L<<1)
SIGF_BLIT     (1L<<4)
SIGF_SINGLE   (1L<<4)
SIGF_DOS      (1L<<8)

```

6. Fonctions des messages

AddPort

Insérer une structure message-port

Syntaxe

```
AddPort(Port)
-354      A1
struct MsgPort *Port;
```

Description

Cette fonction insère une structure Message-Port donnée à la liste du port activé. Elle sera classée suivant son niveau de priorité. On peut accéder à la tête de liste par SysBase -> PortList. AddPort initialise aussi la structure mp_MsgList à l'intérieur du Message-Port.

Paramètres

Port: Pointeur sur la structure Message-Port.

Voir aussi

[RemPort\(\)](#), [FindPort\(\)](#)

RemPort**Eliminer un Message-Port****Syntaxe**

```
RemPort(Port)
-360      A1
struct MsgPort *Port;
```

Description

Cette fonction élimine un Message-Port de la liste des ports actifs. Il ne sera alors plus possible d'y accéder au moyen de FindPort.

Paramètres

Port: Pointeur sur le Message-Port.

Voir aussi

[AddPort\(\)](#), [FindPort\(\)](#)

PutMsg**Envoyer une information à un Message-Port****Syntaxe**

```
PutMsg(Port, Message)
-366      A0      A1
struct MsgPort *Port;
struct Message *Message;
```

Description

Cette fonction envoie une information à un Message-Port. Dans ce dernier, elle sera rajoutée à la liste des informations et le contenu du champ mp_Flag de l'action correspondante sera libéré.

Paramètres

Port: Adresse de la structure Message-Port du port de destination.

Message: Pointeur sur la structure Message du message.

Voir aussi

[GetMsg\(\)](#), [ReplyMsg\(\)](#)

GetMsg**Prendre en compte un message****Syntaxe**

```
Message = GetMsg(Port)
      D0      -372   A0
      struct Message *Message;
      struct MsgPort *Port;
```

Description

Cette fonction prend en compte la première information arrivée au port et transfert un pointeur dans la structure Message.

Paramètres

Port: Adresse de la structure Message-Port du port.

Retour

Message: Pointeur sur la structure Message.

Voir aussi

PutMsg(), ReplyMsg(), WaitPort()

ReplyMsg**Envoyer un message de retour au Reply-Port****Syntaxe**

```
ReplyMsg(Message)
      -378     A1
      struct Message *Message;
```

Description

Cette fonction envoie un message de retour à son Reply-Port. Si le champ mn_ReplyPort de la structure Message est égal à 0, cette fonction sera ignorée.

Paramètres

Message: Pointeur sur la structure Message.

Voir aussi

PutMsg(), GetMsg(), WaitPort()

WaitPort**Attendre la réception d'un message****Syntaxe**

```
Message = WaitPort(Port)
DO      -384   A0
struct Message *Message;
struct Port   *Port;
```

Description

Cette fonction attend la réception d'un message à un Port déterminé. Si un message y parvient ou s'il y en avait déjà un présent avant l'appel de la fonction, WaitPort() retournera l'adresse de ce message. Le message ne sera pas éliminé de la liste des messages réceptionnés. Pour ce faire, on devra utiliser GetMsg().

Paramètres

Port: Adresse du port.

Retour

Message: Pointeur sur le premier message de la liste.

Voir aussi

GetMsg(), PutMsg(), ReplyMsg()

FindPort**Chercher le prochain Message-Port****Syntaxe**

```
Port = FindPort(Name)
DO      -390   A1
struct MsgPort *Port;
char      *Name;
```

Description

Cette fonction cherche le prochain Message-Port, caractérisé par un nom, dans la liste du Port activé. Si ce port existe, cette fonction donne, en retour, un pointeur sur ce port.

Paramètres

Name: Nom du port recherché.

Retour

Port: Pointeur sur le Message-Port. Lorsque ce dernier n'existe pas, Port=0.

Structure

```

struct MsgPort <exec/ports.h>
{
0x00 00 struct Node mp_Node;
0x0E 14 UBYTE mp_Flags;
0x0F 15 UBYTE mp_SigBit;
0x10 16 struct Task *mp_SigTask;
0x14 20 struct List mp_MsgList;
0x22 34
};

mp_SoftInt mp_SigTask

mp_Flags:
PF_ACTION 3L
PA_SIGNAL 0L
PA_SOFTINT 1L
PA_IGNORE 2L

struct Message <exec/ports.h>
{
0x00 00 struct Node mn_Node;
0x0E 14 struct MsgPort *mn_ReplyPort;
0x12 18 WORD mn_Length;
0x14 20
};

```

7. Fonctions des librairies

AddLibrary

Ajouter une librairie

Syntaxe

```

AddLibrary(Library)
-396      A1
struct Library *Library;

```

Description

Cette fonction permet d'ajouter une librairie personnelle à celles déjà existantes.

Paramètres

Library: Pointeur sur la structure librairie mise en place par la fonction MakeLibrary().

Voir aussi

[MakeLibrary\(\)](#), [RemLibrary\(\)](#), [OpenLibrary\(\)](#), [CloseLibrary\(\)](#)

CloseLibrary**Fermer une librairie****Syntaxe**

```
CloseLibrary(Library)
    -414          A1
    struct Library *Library;
```

Description

Cette fonction permet de fermer une librairie. Attention, si vous essayez de refermer une librairie qui n'est pas ouverte, vous provoquerez un effondrement du système.

Paramètres

Library: Pointeur sur la bibliothèque ouverte que l'on désire fermer.

Voir aussi

[OpenLibrary\(\)](#)

MakeFonctions*Construire une table de saut****Syntaxe**

```
TableSize = MakeFonctions(Target, FonctionArray,
    FoncDispBase)
        D0          -90          A0          A1          A2
    ULONG TableSize;
    ULONG Target;
    ULONG *FonctionArray;
    ULONG FoncDispBase;
```

Description

Cette fonction permet la création d'une table de saut de fonctions du type de celles utilisées par les librairies et les devices.

Paramètres

Target: Pointeur sur l'adresse de base (plus haute position mémoire).

FonctionArray: Pointeur sur les pointeurs de fonctions.

FoncDispBase: Pointeur sur la base des fonctions dont les déplacements sont relatifs.

Retour

TableSize: Table de la nouvelle table en octets.

MakeLibrary**Mettre en place une librairie personnelle****Syntaxe**

```
Library = MakeLibrary(FuncInit, StructInit, LibInit, DataSize, CodeSize)
          D0      -84      A0       A1      A2      D0      D1
ULONG *Library;
ULONG *FuncInit;
ULONG *StructInit;
ULONG *LibInit;
ULONG  DataSize;
ULONG  *CodeSize;
```

Description

Cette fonction met en place une librairie personnelle. La place mémoire de la structure librairie sera déterminée par la fonction lorsque lib_MemSize et Lib_PosSize seront correctement initialisés.

Paramètres

- FuncInit: Pointeur sur la table des vecteurs de la librairie. Sur cette table se trouvent soit des pointeurs sur les différentes fonctions, soit les offsets qui seront additionnés à l'adresse de base pour obtenir le saut des fonctions. Si vous avez stocké des offsets sur la table, cette dernière devra débuter à \$FFFF (-1). La marque de fin d'une table est à nouveau à -1 avec la même longueur d'enregistrements (mot long pour table d'offsets, mot long comme pointeur de fonctions).
- StructInit: Pointeur sur la table d'initialisation. La structure librairie sera initialisée avec cette table, à la fin de la fonction MakeLibrary. Les enregistrements lib_NegSize et lib_PosSize ne doivent pas être initialisés avec l'aide de cette table, sinon la valeur prête à être insérée sera recalculée par la fonction.
- LibInit: Pointeur sur le programme qui sera appelé à la fin de la fonction MakeLibrary(), dès que le pointeur sera initialisé. Une routine personnelle pourra donc initialiser une structure librairie, lorsque celle-ci ne le sera pas à l'aide d'une table. Le pointeur de la structure mise en place sera dans D0, celui de liste segment étant dans A0.
- DataSize: Pointeur sur une liste segment (utilisée par l'AmigaDOS).
- CodeSize: Taille du code.

Retour

- Library: Pointeur de retour sur la structure librairie.

Voir aussi

InitStruct()

RemLibrary**Eliminer une structure librairie****Syntaxe**

```
Error = RemLibrary(Library)
      DO      -402      A1
      LONG      Error;
      struct Library *Library;
```

Description

Cette fonction élimine une structure librairie dans la liste des librairies de la structure ExecBase. Celle-ci ne pourra plus être ouverte par la suite à l'aide de la fonction OpenLibrary().

Paramètres

Library: Pointeur sur la structure librairie.

Retour

Error: Indique si une erreur est apparue lors de l'utilisation de la fonction. Si c'est le cas, un message d'erreur sera transmis dans D0. Autrement, l'erreur est initialisée à 0.

Voir aussi

AddLibrary(), MakeLibrary()

OldOpenLibrary*Ouvrir une librairie****Syntaxe**

```
Library = OldOpenLibrary(LibName)
      DO      -408      A1
      struct Library *Library;
      char      *LibName;
```

Description

Cette fonction est issue de la version KickStart 1.0; Elle permet d'ouvrir une librairie sans effectuer un test sur le numéro de version. Cette fonction a été conservée dans la librairie Exec afin que les programmes conçus avec la version KickStart 1.0 puissent aussi tourner avec la version 1.2 et 1.3.

Paramètres

LibName: Nom de la librairie à ouvrir.

Retour

Library: Pointeur sur la librairie ou 0 si l'ouverture n'a pas été obtenue.

Voir aussi

`OpenLibrary()`, `CloseLibrary()`

OpenLibrary**Ouverture d'une librairie****Syntaxe**

```
Library = OpenLibrary(LibName, Version)
          DO      -552     A1      DO
          struct Library *Library;
          char           *LibName;
          LONG           Version;
```

Description

Cette fonction permet d'ouvrir une librairie et de pouvoir utiliser toutes les fonctions qu'elle contient.

Paramètres

LibName: Pointeur sur le nom de la bibliothèque qui doit être ouverte, par exemple "intuition.library" (le nom sera toujours terminé par un 0).

Version: Donne la version de la bibliothèque que l'utilisateur veut ouvrir. Si plusieurs des bibliothèques disponibles portent le même nom, elles seront distinguées par leurs numéros de version. Si une nouvelle version est exigée, alors qu'elle n'est pas encore disponible, la fonction 'OpenLibrary' échouera. Si l'on transmet un zéro pour numéro de version, la version n'est pas vérifiée à l'ouverture.

Retour

Library: Contient, après l'appel de la fonction, l'adresse de base de la bibliothèque, si cette dernière a été trouvée. Sinon, c'est un 0 qui sera renvoyé.

Voir aussi

`OldOpenLibrary()`, `CloseLibrary()`

SetFunction**Modifier le saut d'offset d'une fonction****Syntaxe**

```
OldFunct = SetFunction(Library, Offset, FunctEntry)
    DO          -420      A1      A0      DO
    LONG        OldFunct;
    struct Library *Library;
    LONG        Offset;
    ULONG       FuntcEntry;
```

Description

Cette fonction permet de modifier le saut d'offset d'une fonction définie avec un offset négatif, de façon à ce que l'offset de la fonction pointe désormais sur la nouvelle routine. La somme de contrôle de la librairie est recalculée.

Paramètres

Library: Pointeur sur la librairie à modifier.

Offset: Indique l'offset de la fonction, qui doit être modifié.

FunctEntry: Pointeur sur la routine.

Retour

OldFunct: Pointeur sur l'ancienne adresse.

SumLibrary**Calculer le CheckSum d'une librairie****Syntaxe**

```
SumLibrary(Library)
    -426      A1
    struct Library *Library;
```

Description

Cette fonction calcule le CheckSum d'une librairie. Si le CheckSum ne correspond pas et que la librairie n'a pas été déclarée comme modifiée, le système plantera.

Paramètres

Library: Pointeur sur la librairie à calculer.

Structure

```
extern struct Library <exec/libraries.h>
{
0x00 00  struct Node lib_Node;
0x0E 14  UBYTE lib_Flags;
0x0F 15  UBYTE lib_pad;
0x10 16  UWORLD lib_NegSize;
```

```

0x12 18  UWORD lib_PosSize;
0x14 20  UWORD lib_Version;
0x16 22  UWORD lib_Revision;
0x18 24  APTR lib_IdString;
0x1C 28  ULONG lib_Sum;
0x20 32  UWORD lib_OpenCnt;
0x22 34

};

Lib_Flags:
LIBF_SUMMING (1L<<0)
LIBF_CHANGED (1L<<1)
LIBF_SUMUSED (1L<<2)
LIBF_DELEXP (1L<<3)

Definition:
lh_Node      lib_Node
lh_Flags     lib_Flagslh_pad lib_pad
lh_NegSize   lib_NegSize
lh_PosSize   lib_PosSize
lh_Version   lib_Version
lh_Revision  lib_Revision
lh_IdString  lib_IdString
lh_Sum       lib_Sum
lh_OpenCnt   lib_OpenCnt

LIB_VECTSIZE 6L
LIB_RESERVED 4L
LIB_BASE      (-LIB_VECTSIZE)
LIB_USERDEF   (LIB_BASE-(LIB_RESERVED*LIB_VECTSIZE))
LIB_NONSTD    (LIB_USERDEF)

lib_Vectors:
LIB_OPEN      (-6L)
LIB_CLOSE     (-12L)
LIB_EXPUNGE   (-18L)
LIB_EXTFUNC   (-24L)

```

8. Fonctions des devices

AddDevice

Ajouter un device au système

Syntaxe

```

AddDevice(Device)
-432      A1
struct Device *Device;

```

Description

Cette fonction ajoute un device à la liste des devices système. Le device doit être prêt à être ouvert après l'appel de cette fonction.

Paramètres

Device: Pointeur sur le noeud d'initialisation du device.

Voir aussi

`RemDevice()`, `OpenDevice()`, `CloseDevice()`

RemDevice**Supprimer un device au système****Syntaxe**

```
Error = RemDevice(Device)
DO      -438     A1
LONG      Error;
struct Device *Device;
```

Description

Cette fonction supprime un device au système. Elle peut être refusée par le système si le device est en cours d'utilisation.

Paramètres

`Device:` Pointeur sur le noeud du device.

Retour

`Error:` 0 si la suppression n'a pas été effectuée.

Voir aussi

`AddDevice()`, `OpenDevice()`, `CloseDevice()`

OpenDevice**Activer un device****Syntaxe**

```
Error = OpenDevice(DevName, Unit, IORequest, Flags)
DO      -444     A0      D0      A1      D1
LONG      Error;
char      *DevName;
ULONG      Unit;
struct IORequest *IORequest;
ULONG      Flags;
```

Description

Cette fonction ouvre et active un device. Celui-ci doit exister en mémoire ou sur le disque (disquette ou disque dur).

Paramètres

`DevName:` Nom du device.

Unit: Nombre pour ouvrir le device. Il est spécifique au device. Si le device n'a pas d'unités séparées, utilisez un 0.

IOResquest: I/O Request Block retourné une fois initialisé.

Flags: Information additionnelle spécifique au chemin.

Retour

Error: 0 si OK, sinon une autre valeur.

Voir aussi

AddDevice(), RemDevice(), CloseDevice()

CloseDevice

Fermer un device

Syntaxe

```
CloseDevice(IOResquest)
    -450          A1
    struct IOResquest *IOResquest;
```

Description

Cette fonction permet de refermer un device qui est déjà ouvert.

Paramètres

IOResquest: Pointeur sur la structure du device.

Voir aussi

AddDevice(), RemDevice(), OpenDevice()

DolO

Exécuter une commande

Syntaxe

```
Error = DoIO(IOResquest)
    DO      -456      A1
    LONG           Error;
    struct IOResquest *IOResquest;
```

Description

Cette fonction essaie d'exécuter la commande définie dans la structure I/O-Request.

Paramètres

IOResquest: Adresse de la structure I/O-Request initialisée.

Retour

Error: Si l'opération s'effectue sans problème, on obtient 0 en retour. S'il se produit une erreur, on obtient le numéro de l'erreur.

Voir aussi

SendIO(), WaitIO()

SendIO**Etendre le contrôle d'un driver****Syntaxe**

```
SendIO(IORequest)
-462      A1
struct IORequest *IORequest;
```

Description

Cette fonction étend le contrôle d'un driver de périphérique sur la commande indiquée dans la structure I/O-Request.

Paramètres

IORequest: Adresse de la structure I/O-Request.

Voir aussi

DoIO(), WaitIO()

CheckIO**Tester si un processus a été traité****Syntaxe**

```
Result = CheckIO(IORequest)
D0      -468      A1
struct IORequest *Result
struct IORequest *IORequest;
```

Description

Cette fonction teste si un processus I/O déterminé a été traité.

Paramètres

IORequest: Adresse de la structure I/O-Request.

Retour

Result: Si OK, le pointeur sur la structure I/O-Request est retourné, sinon c'est un 0.

WaitIO**Attendre la fin d'un traitement I/O****Syntaxe**

```
Error = WaitIO(IORequest)
DO      -474      A1
LONG          Error:
struct IORequest *IORequest;
```

Description

Cette fonction attend jusqu'à ce que la commande indiquée dans la structure I/O-Request soit exécutée.

Paramètres

IORequest: Adresse de la structure I/O-Request.

Retour

Error: 0 si OK ou message d'erreur.

Voir aussi

SendIO(), DoIO()

AbortIO*Annuler le dernier I/O-Request****Syntaxe**

```
AbortIO(IORequest)
-480      A1
struct IORequest *IORequest;
```

Description

Cette fonction tente d'annuler le dernier I/O-Request.

Paramètres

IORequest: Adresse d'une structure I/O-Request.

Voir aussi

SendIO(), WaitIO(), DoIO()

Structure

```
struct Device <exec/devices.h>
{
0x00 00 struct Library dd_Library;
0x22 34
};
```

```
struct IORequest <exec/io.h>
{
0x00 00 struct Message io_Message;
0x0E 14 struct Device *io_Device;
0x12 18 struct Unit *io_Unit;
0x16 22 UWORD io_Command;
0x18 24 UBYTE io_Flags;
0x19 25 BYTE io_Error;
0x1A 26
};

Unit_Flags:
UNITF_ACTIVE (1L<<0)
UNITF_INTASK (1L<<1)

struct Unit <exec/devices.h>
{
0x00 00 struct MsgPort *unit_MsgPort;
0x04 04 UBYTE unit_flags;
0x05 05 UBYTE unit_pad;
0x06 06 UWORD unit_OpenCnt;
0x08 08
};

I/O-Errors:
IOERR_OPENFAIL -1L
IOERR_ABORTED -2L
IOERR_NOCMD -3L
IOERR_BADLENGTH -4L

struct IOStdReq <exec/io.h>
{
0x00 00 struct Message io_Message;
0x0E 14 struct Device *io_Device;
0x12 18 struct Unit *io_Unit;
0x16 22 UWORD io_Command;
0x18 24 UBYTE io_Flags;
0x19 25 BYTE io_Error;
0x1A 26 ULONG io_Actual;
0x1E 30 ULONG io_Length;
0x22 34 APTR io_Data;
0x26 38 ULONG io_Offset;
0x2A 42
};

io_Command:
DEV_BEGINIO (-30L)
DEV_ABORTIO (-36L)

io_Flags:
IOB_QUICK 0L
IOF_QUICK (1L<<0)

io_Command:
CMD_INVALID 0L
CMD_RESET 1L
CMD_READ 2L
CMD_WRITE 3L
```

```
CMD_UPDATE      4L
CMD_CLEAR       5L
CMD_STOP        6L
CMD_START       7L
CMD_FLUSH       8L
CMD_NONSTD     9L
```

9. Fonctions des resources

AddResource

Ajouter un Resource au système

Syntaxe

```
AddResource(Resource)
-486          A1
struct Node *Resource;
```

Description

Cette fonction permet d'ajouter un Resource à la liste des Resources système et le rend accessible à toutes les tasks. Aucun paramètre en retour.

Paramètres

Resource: Adresse de la structure Resource initialisée.

Voir aussi

RemResource(), OpenResource()

RemResource

Supprimer un Resource au système

Syntaxe

```
RemResource(Resource)
-492          A1
struct Node *Resource;
```

Description

Cette fonction permet de supprimer un fichier auxiliaire dans la liste des Resources du système. Aucune valeur en retour.

Paramètres

Resource: Adresse de la structure Resource.

Voir aussi

AddResource(), OpenResource()

OpenResource**Ouvrir un fichier auxiliaire****Syntaxe**

```
Resource = OpenResource(ResName)
          D0           -498      A1      D0
          struct Node *Resource;
          char      *ResName;
          ULONG     Version;
```

Description

Cette fonction permet d'ouvrir un fichier auxiliaire déjà initialisé, et retourne l'adresse d'une structure Resource.

Paramètres

ResName: Nom du fichier Resource.

Retour

Resource: Adresse de la structure Resource ou 0.

10. Fonctions des Sémaphores***InitSemaphore****Initialiser un Semaphore****Syntaxe**

```
InitSemaphore(SignalSemaphore)
             -558          A0
             struct SignalSemaphore *SignalSemaphore;
```

Description

Cette fonction permet d'initialiser un Semaphore. Celle-ci n'alloue rien, mais elle initialise la liste des pointeurs et les compteurs Semaphores.

Paramètres

SignalSemaphore: Adresse de la structure Semaphore.

Voir aussi

ObtainSemaphore(), ReleaseSemaphore()

ObtainSemaphore*Obtenir un accès exclusif****Syntaxe**

```
ObtainSemaphore(SignalSemaphore)
    -564          A0
    struct SignalSemaphore *SignalSemaphore;
```

Description

Cette fonction est utilisée pour obtenir un accès exclusif sur un objet.

Paramètres

SignalSemaphore: Adresse de la structure Semaphore.

Voir aussi

ObtainSemaphoreList(), InitSemaphore(), ReleaseSemaphore(), TransferSemaphore(), AttemptSemaphore()

Vacate**Libérer un Semaphore****Syntaxe**

```
Vacate(Semaphore)
    -546          A0
    struct SignalSemaphore *SignalSemaphore;
```

Description

Cette fonction permet de libérer un Semaphore.

Paramètres

SignalSemaphore: Adresse de la structure Semaphore.

ReleaseSemaphore*Rendre viable un signal Semaphore****Syntaxe**

```
ReleaseSemaphore(SignalSemaphore)
    -570          A0
    struct SignalSemaphore *SignalSemaphore;
```

Description

Cette fonction permet de rendre viable un signal Semaphore aux autres tâches.

Paramètres

SignalSemaphore: Adresse de la structure Semaphore.

Avertissement Chaque ReleaseSemaphore doit être lié à un ObtainSemaphore.

Voir aussi

ObtainSemaphore(), ObtainSemaphoreList()

*AttemptSemaphore

Id ObtainSemaphore()

Syntaxe

```
Success = AttemptSemaphore(SignalSemaphore)
DO          -576           A0
BOLL          Success;
struct SignalSemaphore *SignalSemaphore;
```

Description

Cette fonction est similaire à la fonction ObtainSemaphore() à une exception. La fonction ne bloquera pas en cas d'impossibilité d'exécution.

Paramètres

SignalSemaphore: Adresse de la structure Semaphore.

Retour

Success: TRUE si OK, FALSE si une autre tâche est déjà en train de l'utiliser.

Voir aussi

ObtainSemaphore(), ObtainSemaphoreList(), ReleaseSemaphore()

*ObtainSemaphoreList

Obtenir une liste de Séaphore

Syntaxe

```
ObtainSemaphoreList(List)
-582           A0
struct SignalSemaphore List;
```

Description

Cette fonction permet d'obtenir une liste de Séaphore.

Paramètres

List: Liste de Séaphore.

Voir aussi

ObtainSemaphore(), ReleaseSemaphore(), ReleaseSemaphoreList()

ReleaseSemaphoreList*Libérer une liste de Sémafore****Syntaxe**

```
ReleaseSemaphoreList(List)
-588          A0
struct SignalSemaphore List;
```

Description

Cette fonction permet de libérer une liste de Sémafore.

Paramètres

List: Liste de Sémafore.

Voir aussi

ObtainSemaphore(), ObtainSemaphoreList(), ReleaseSemaphore()

FindSemaphore*Chercher un Semaphore à partir du nom****Syntaxe**

```
SignalSemaphore = FindSemaphore(Name)
D0           -594      A0
struct SignalSemaphore *SignalSemaphore;
char           *Name;
```

Description

Cette fonction permet de rechercher un Sémafore à partir de son nom.

Paramètres

Name: Nom du Sémafore à trouver.

Retour

SignalSemaphore: Pointeur sur la structure résidente ou 0 si pas trouvé.

AddSemaphore*Ajouter un Sémafore au système****Syntaxe**

```
AddSemaphore(SignalSemaphore)
-600          A0
struct SignalSemaphore *SignalSemaphore;
```

Description

Cette fonction permet d'ajouter un Sémaphore au système (à travers la liste de Sémaphore).

Paramètres

SignalSemaphore: Adresse de la structure Sémaphore.

Voir aussi

RemSemaphore(), InitSemaphore(), FindSemaphore()

*RemSemaphore

Supprimer un Sémaphore

Syntaxe

```
RemSemaphore(SignalSemaphore)
    -606          A0
    struct SignalSemaphore *SignalSemaphore;
```

Description

Cette fonction permet de supprimer un Sémaphore du système.

Paramètres

SignalSemaphore: Adresse de la structure Semaphore.

Voir aussi

AddSemaphore(), FindSemaphore()

Structure

```
struct SignalSemaphore <exec/semaphores.h>
{
 0x00 00 struct Node ss_Link;
 0x0e 14 SHORT ss_NestCount;
 0x10 16 struct MinList ss_WaitQueue;
 0x1c 28 struct SemaphoreRequest ss_MultipleLink;
 0x28 40 struct Task *ss_Owner;
 0x2c 44 SHORT ss_QueueCount;
 0x2e 46
};

struct Semaphore <exec/semaphores.h>
{
 0x00 00 struct MsgPort sm_MsgPort;
 0x22 34 WORD sm_Bids;
 0x24 36
};

struct SemaphoreRequest <exec/semaphores.h>
{
```

```

0x00 00 struct MinNode sr_Link;
0x08 08 struct Task *sr_Waiter;
0x0C 12
}:

```

11. Fonctions du KickStart

*AddMemList

Ajouter de la mémoire au système

Syntaxe

```

Error = AddMemList(Size, Attributes, Pri, Base, Name)
      D0      -618     D0      D1      D2   A0   A1
      LONG Error;
      ULONG Size;
      WORD Attributes;
      BYTE Pri;
      ULONG Base;
      char *Name;

```

Description

Cette fonction ajoute de la mémoire à la liste de mémoire libre. Le début de la zone mémoire sera utilisé pour la structure MemHeader.

Paramètres

Size: Taille de la zone mémoire (en octets).

Pri: Priorité de la mémoire. La mémoire CHIP à une priorité de -10, la mémoire d'expansion 16 bit à une priorité de 0.

Base: Base de la nouvelle zone mémoire.

Name: Nom utilisé dans le "memory header".

Retour

Error: Le nom si OK, sinon NULL.

Voir aussi

[Allocate\(\)](#), [AllocMem\(\)](#), [AllocEntry\(\)](#), [AllocRemember\(\)](#)

*CopyMem

Copier des zone mémoire

Syntaxe

```

CopyMem(Source, Dest, Size)
      -624    A0    A1    D0
      ULONG Source
      ULONG Dest;
      ULONG Size;

```

Description

Cette fonction permet de copier des zones mémoire.

Paramètres

Source: Pointeur sur la zone source des données.

Dest: Pointeur sur la zone de destination des données.

Size: Taille de la zone mémoire (en octets).

Voir aussi

[CopyMemQuick\(\)](#)

*CopyMemQuick

Copier des zones mémoire

Syntaxe

```
CopyMemQuick(Source, Dest, Size)
             -630      A0      A1      D0
    ULONG Source
    ULONG Dest;
    ULONG Size;
```

Description

Cette fonction permet de copier des zones mémoire. A la différence de la fonction [CopyMem\(\)](#), celle-ci est optimisée.

Paramètres

Source: Pointeur sur la zone source des données ("long aligned" = multiple de 4).

Dest: Pointeur sur la zone de destination des données ("long aligned").

Size: Taille de la zone mémoire (en octets).

Avertissement La taille de la zone mémoire doit être divisible par 4.

Voir aussi

[CopyMem\(\)](#)

6.2. La DOS-Librairie

Fonctions de la DOS-Librairie:

1. Entrées et Sorties

Close	800
Open	800
Read	801
Seek	802
Write	802

2. Gestion des données

CreateDir	803
CurrentDir	804
DeleteFile	804
Examine	805
ExNext	806
Info	806
ParentDir	807
Rename	807
SetComment	808
SetProtection	808

3. Fonctions et gestion des données

DupLock	810
Input	810
IoErr	811
IsInteractive	811
Lock	812
Output	812
UnLock	813

4. Gestion des procédés

CreateProc	814
DateStamp	815
Delay	815
DeviceProc	816
Exit	816
WaitForChar	816

5. Exécution des programmes

Execute	818
LoadSeg	819
UnLoadSeg	819

6. Directives internes du DOS

GetPacket	Non détaillée
QueuePacket	Non détaillée

1. Entrées et Sorties**Close****Fermer un fichier****Syntaxe**

```
Close(Handle)
-36  D1
struct FileHandle *Handle;
```

Description

Cette fonction permet de fermer un fichier ouvert avec la fonction Open(). Le pointeur transmis en D1 est celui qui a été obtenu par la fonction Open() et qui pointe sur le FileHandle.

Paramètres

Handle: Adresse de la structure FileHandle.

Voir aussi

Open()

Open**Ouvrir un fichier****Syntaxe**

```
Handle = Open(name,mode)
D0      -30  D1   D2
struct FileHandle *Handle;
UBYTE *name;
LONG mode;
```

Description

Cette fonction permet d'ouvrir un fichier aux accès en écriture et en lecture. Il retourne le numéro d'identification.

Paramètres

- name: Nom du fichier dont l'accès doit être ouvert.
- mode: Si on est en mode MODE_OLDFILE(1005), un fichier existant est ouvert en lecture et en écriture. Si on est en mode MODE_NEWFILE(1006), il y a création d'un nouveau fichier, ouvert à l'écriture. S'il existe un fichier du même nom, il est écrasé par le nouveau.

Retour

- Handle: Pointeur BCPL sur le fichier.

Voir aussi

[Close\(\)](#), [IoErr\(\)](#).

Read

Lire des données

Syntaxe

```
Retour = Read(Handle,buffer,long)
        D0      -42    D1     D2     D3
LONG Retour;
struct FileHandle *Handle;
UBYTE *buffer;
LONG long;
```

Description

Cette fonction permet de lire le fichier spécifié par 'Handle'.

Paramètres

- Handle: Adresse de la structure FileHandle.
- buffer: Adresse du secteur mémoire dans lequel les données doivent être écrites.
- long: Nombre d'octets devant être lus.

Retour

- Retour: Nombre d'octets réellement lus. Si cette valeur est 0, la fin du fichier est déjà atteinte. S'il se produit une erreur, la valeur renournée est -1.

Voir aussi

[Open\(\)](#), [Close\(\)](#), [Write\(\)](#)

Seek**Aller à un endroit précis dans un fichier****Syntaxe**

```
Retour = Seek(Handle,position,mode)
      D0      -66    D1     D2      D3
      LONG Retour;
      struct FileHandle *Handle;
      LONG position, mode;
```

Description

Cette fonction permet de modifier le pointeur interne d'un fichier pour se rendre à des endroits déterminés de ce même fichier. Bien entendu, les déplacements vers l'arrière du fichier sont possibles.

OFFSET_BEGINNING: -1 à partir du début du fichier.

OFFSET_END: 1 à partir de la fin du fichier.

OFFSET_CURRENT: 0 à partir de la position actuelle.

Paramètres

Handle: Adresse de la structure FileHandle du fichier.

position: Position précise pour le déplacement.

mode: Mode de position.

Retour

Retour: La valeur en retour est celle de l'ancienne position.

Exemple :

```
pos = Seek(Handle,0L,OFFSET_CURRENT);
```

Write**Ecrire des données****Syntaxe**

```
nombre = Write(Handle,buffer,long)
      D0      -48    D1     D2      D3
      LONG nombre;
      struct FileHandle *Handle;
      UBYTE *buffer;
      LONG long;
```

Description

Cette fonction permet d'écrire des données dans un fichier ouvert.

Paramètres

- Handle: Adresse de la structure FileHandle.
- buffer: Adresse du secteur mémoire dans lequel se trouvent les données à écrire.
- long: Longueur du secteur mémoire en octets.

Retour

- nombre: Nombre effectif d'octets ayant été écrits. S'il se produit une erreur, -1 est retourné.

Voir aussi

[Open\(\)](#), [Close\(\)](#), [Read\(\)](#).

Structure

MODE_READWRITE	1004
MODE_OLDFILE	1005
MODE_NEWFILE	1006
OFFSET_BEGINNING	-1
OFFSET_CURRENT	0
OFFSET_END	1
struct FileHandle <libraries/dosextens.h>	
{	
Offset:	
0x00 0	struct Message *fh_Link;
0x04 4	struct MsgPort *fh_Port;
0x08 8	struct MsgPort *fh_Type;
0x0C 12	LONG fh_Buf;
0x10 16	LONG fh_Pos;
0x14 20	LONG fh_End;
0x18 24	LONG fh_Funcs;
0x1C 28	LONG fh_Func2;
0x20 32	LONG fh_Func3;
0x24 36	LONG fh_Args;
0x28 40	LONG fh_Arg2;
sizeof(struct FileHandle)	
0x2C 44	
}	

2. Gestion des données

CreateDir

Créer un répertoire

Syntaxe

```
lock = CreateDir(name)
      D0      -120     D1
```

```
struct FileLock *lock;
UBYTE *name;
```

Description

Cette fonction permet la création d'un répertoire dans le répertoire en cours.

Paramètres

name: Nom du répertoire à créer.

Retour

lock: Si la création est impossible, on obtient un 0 en retour. La fonction IoErr fournira alors le message d'erreur correspondant.

Voir aussi

IoErr, Lock, UnLock

CurrentDir	Transformer un répertoire
-------------------	----------------------------------

Syntaxe

```
oldLock = CurrentDir(lock)
      D0      -126      D1
      struct FileLock *oldLock;
      struct FileLock *lock;
```

Description

Cette fonction permet de transformer un répertoire en répertoire courant.

Paramètres

lock: Adresse de la structure Lock du nouveau répertoire.

Retour

oldLock: Adresse de la structure Lock de l'ancien répertoire actuel.

Voir aussi

Lock, UnLock

DeleteFile	Effacer un fichier
-------------------	---------------------------

Syntaxe

```
ok = DeleteFile(name)
      D0      -72      D1
```

```
BOOL ok;
UBYTE *name;
```

Description

Cette fonction permet d'effacer un fichier ou un répertoire. La transmission du nom doit se terminer par l'octet 0. S'il s'agit d'un répertoire, celui-ci ne doit contenir aucun fichier ou sous-répertoire.

Paramètres

name: Nom du fichier ou du répertoire à supprimer.

Retour

ok: TRUE ou FALSE. On peut obtenir le numéro de l'erreur en utilisant la fonction IoErr.

Voir aussi

IoErr

Examine	Ecrire les informations d'un fichier
---------	--------------------------------------

Syntaxe	<pre>ok = Examine(lock,infoBlock) D0 -102 D1 D2 BOOL ok; struct FileLock *lock; struct FileInfoBlock *infoBlock;</pre>
----------------	---

Description

Cette fonction écrit dans un bloc File-Info les informations du fichier ou du répertoire qui sont indiqués dans la structure Lock. Ce bloc contient alors le nom, la taille, la date de création et le type du fichier ou du répertoire.

Paramètres

lock: Adresse de la structure Lock.

infoBlock: Adresse de la structure du bloc FileInfo.

Retour

ok: En cas de succès, la fonction retourne -1. Sinon, on peut rechercher l'erreur à l'aide de IoErr.

Voir aussi

ExNext

ExNext**Voir les entrées d'un répertoire****Syntaxe**

```
ok = ExNext(lock,infoBlock)
D0      -108    D1      D2
BOOL ok;
struct FileLock *lock;
struct FileInfoBlock *infoBlock;
```

Description

Cette fonction permet de passer en revue toutes les entrées d'un répertoire. Il faut appeler la fonction jusqu'à ce que toutes les entrées aient été traitées. Cette fonction fournit la première entrée du répertoire, et les autres sont examinées à l'aide de **ExNext**.

Paramètres

lock: Adresse de la structure Lock.

infoBlock: Adresse de la structure du bloc FileInfo.

Retour

ok: -1 si tous se passe bien.

Voir aussi

Examine.

Info**Lire les informations des disquettes****Syntaxe**

```
ok = Info(lock,parameterBlock)
D0      -114    D1      D2
BOOL ok;
struct FileLock *lock;
struct InfoData *parameterBlock;
```

Description

Cette fonction lit les informations sur toutes les disquettes en service et sur le disque dur. Ces informations comprennent: la taille de la disquette en Koctets, le nombre de blocs libres et occupés, et le nombre de Soft_Error.

Paramètres

lock: Adresse de la structure Lock contenant le nom de la disquette.

parameterBlock: Adresse du secteur mémoire dans lequel les informations doivent être copiées. Ce secteur doit se trouver nécessairement à une adresse mémoire divisible par 4.

Retour

ok: TRUE si OK ou FALSE si une erreur est intervenue.

ParentDir**Communiquer un répertoire****Syntaxe**

```
newLock = ParentDir(lock)
      D0      -210    D1
      struct FileLock *newLock;
      struct FileLock *lock;
```

Description

Cette fonction communique le répertoire d'où sont issus un répertoire ou un fichier donnés.

Paramètres

lock: Adresse de la structure Lock.

Retour

newLock: Adresse d'une structure Lock. Le nom du répertoire d'origine peut ensuite être lu avec la fonction Examine().

Voir aussi

CurrentDir

Rename**Renommer un fichier ou un répertoire****Syntaxe**

```
ok = Rename(ancName,newName)
      D0      78      D1      D2
      BOOL ok;
      UBYTE *ancName,*newName;
```

Description

Cette fonction permet de changer le nom d'un fichier ou d'un répertoire.

Paramètres

ancName: Ancien nom du fichier.

newName: Nouveau nom du fichier.

Retour

ok: TRUE si OK ou FALSE si une erreur est intervenue.

SetComment**Etablir un commentaire****Syntaxe**

```
ok = SetComment(name,comment)
DO      -180      D1      D2
BOOL ok;
UBYTE *name,comment;
```

Description

Cette fonction permet d'écrire un commentaire spécifique au fichier dans le répertoire des fichiers de la disquette.

Paramètres

name: Nom du fichier.

comment: Texte du commentaire, qui ne peut pas être plus long que 80 caractères.

Retour

ok: TRUE si OK ou FALSE si une erreur est intervenue.

SetProtection**Protéger un fichier ou un répertoire****Syntaxe**

```
ok = SetProtection(name,mask)
DO      -186      D1      D2
BOOL ok;
UBYTE *name;
LONG mask;
```

Description

Cette fonction permet de protéger un fichier ou un répertoire.

Paramètres

name: Nom du fichier.

mask: Valeur 32 bits, parmi lesquels seuls 4 bits peuvent être utilisés.
 Le bit 3 pose la protection contre la lecture
 Le bit 2 pose la protection contre l'écriture

Le bit1 pose la protection contre l'exécution
 Le bit 0 pose la protection contre l'effacement.

Retour

ok: TRUE si OK ou FALSE si une erreur est intervenue.

Structure

```

struct FileLock <libraries/dosextens.h>
{
 0x00 0 BPTR      fl_Link;
 0x04 4 LONG      fl_Key;
 0x08 8 LONG      fl_Access;

ACCESS_READ          -2
ACCESS_WRITE         -1
0x0C 12 struct MsgPort *fl_Task;
0x10 16 BPTR        fl_Volume;
sizeof(struct FileLock)
0x14 20
};

struct FileInfoBlock <libraries/dos.h>
{
 0x00 0 LONG fib_DiskKey;
 0x04 4 LONG fib_DirEntryType;
 0x08 8 char   fib_FileName[108];
 0x74 116 LONG   fib_Protection;

Protection-Flags:
FIBF_ARCHIVE    16
FIBF_READ       8
FIBF_WRITE      4
FIBF_EXECUTE    2
FIBF_DELETE     1
0x78 120 LONG   fib_EntryType;
0x7C 124 LONG   fib_Size;
0x80 128 LONG   fib_NumBlocks;
0x84 132 struct DateStamp fib_Date;
0x90 144 char   fib_Comment[116];
sizeof(struct FileInfoBlock)
0x104 260
};

struct InfoData <libraries/dos.h>
{

Offset:
 0x00 0 LONG
 0x04 4 LONG
 0x08 8 LONG id_DiskState;

Status disquette:
ID_WRITE_PROTECTED 80
ID_VALIDATING     81
ID_VALIDATED      82
Offset:
 0x0C 12 LONG   id_NumBlocks;
 0x10 16 LONG   id_NumBlocksUsed;

```

```

0x14 20  LONG   id_BytesPerBlock;
0x18 24  LONG   id_DiskType;

Type disquette :
ID_NO_DISK_PRESENT      -1
ID_UNREADABLE_DISK     'BAD'
ID_DOS_DISK             'DOS'
ID_NOT REALLY_DOS       'NDOS'
ID_KICKSTART_DISK       'KICK'

Offset:
0x1C 28  BPTR   id_VolumeNode;
0x20 32  LONG   id_InUse;
sizeof(struct InfoData)
0x24 36
}:

```

3. Fonctions et gestion des données

DupLock

Copier un Shared-Read-Lock

Syntaxe

```

newLock = DupLock(lock)
          D0      -96   D1
          struct FileLock *newLock;
          struct FileLock *lock;

```

Description

Cette fonction permet la création d'une copie d'un "shared-read-lock" et retourne l'adresse.

Paramètres

lock: Adresse de la structure Lock.

Retour

NewLock: Adresse de la structure Lock copiée.

Voir aussi

Lock, UnLock

Input

Donner des numéros d'identification

Syntaxe

```

Handle = Input()
        D0      -54
        struct FileHandle *Handle;

```

Description

Cette fonction permet de communiquer le numéro d'identification avec lequel un programme a été initialisé.

Retour

Handle: Adresse du FileHandle.

Voir aussi

Output

IoErr

Communiquer le numéro d'une erreur

Syntaxe

```
erreur = IoErr()
      DO      -132
      LONG erreur;
```

Description

Cette fonction communique le numéro de la dernière erreur IO à être survenue .

Retour

erreur : numéro de l'erreur.

Voir aussi

Open, Read, ExNext, DeviceProc

IsInteractive

Trouver un fichier

Syntaxe

```
status = IsInteractive(Handle)
      DO      -216      D1
      BOOL status;
      struct FileHandle *Handle;
```

Description

Cette fonction permet d'examiner et de voir si un fichier se trouve sur une disquette.

Paramètres

Handle: Adresse du FileHandle.

Retour

status: TRUE si OK ou FALSE si la recherche est négative.

Lock**Verrouiller des répertoires ou des fichiers****Syntaxe**

```
lock = Lock(name,smode)
D0      -84    D1      D2
struct FileLock *lock;
UBYTE *name;
LONG mode;
```

Description

Cette fonction permet de verrouiller des répertoires ou des fichiers contre certains accès.

Paramètres

name: Nom du fichier ou du répertoire.

mode: Si l'on est en mode ACCESS_READ (-2), il y a création d'un "shared-read-lock". L'accès à la lecture dans le fichier est alors possible à partir de plusieurs programmes. Si l'on est en mode ACCESS_WRITE (-1), il y a création d'un "write_lock". L'accès à l'écriture dans le fichier n'est alors possible qu'à partir de ce programme.

Retour

lock: Adresse de la structure Lock.

Voir aussi

UnLock, DupLock

Output**Donner le numéro d'identification d'un niveau de sortie****Syntaxe**

```
Handle = Output()
D0      -60
struct FileHandle *Handle;
```

Description

Cette fonction permet de fournir l'adresse du numéro d'identification du niveau actuel de sortie.

Retour

Handle: Adresse de la structure FileHandle.

Voir aussi

Input

UnLock**Déverrouiller un fichier ou un répertoire****Syntaxe**

```
UnLock(lock)
    -90   D1
    struct FileLock *lock;
```

Description

Cette fonction permet d'enlever le verrouillage interdisant l'accès à un fichier. Attention, il n'y a aucune valeur en retour.

Paramètres

lock: Adresse de la structure Lock.

Voir aussi

Lock, DupLock

Structure

```
struct FileLock <libraries/dosextens.h>
{
  Offset:
  0x00 0 BPTR      f1_Link;
  0x04 4 LONG     f1_Key;
  0x08 8 LONG     f1_Access;

  mode:
  ACCESS_READ       -2
  ACCESS_WRITE      -1

  Offset:
  0x0C 12 struct MsgPort *f1_Task;
  0x10 16 BPTR      f1_Volume;
  sizeof(struct FileLock) -
  0x14 20
};
```

Signification des erreurs

ERROR_NO_FREE_STORE	103
ERROR_TASK_TABLE_FULL	105
ERROR_LINE_TOO_LONG	120

ERROR_FILE_NOT_OBJECT	121
ERROR_INVALID_RESIDENT_LIBRARY	122
ERROR_NO_DEFAULT_DIR	201
ERROR_OBJECT_IN_USE	202
ERROR_OBJECT_EXISTS	203
ERROR_DIR_NOT_FOUND	204
ERROR_OBJECT_NOT_FOUND	205
ERROR_BAD_STREAM_NAME	206
ERROR_OBJECT_TOO_LARGE	207
ERROR_ACTION_NOT_KNOWN	209
ERROR_INVALID_COMPONENT_NAME	210
ERROR_INVALID_LOCK	211
ERROR_OBJECT_WRONG_TYPE	212
ERROR_DISK_NOT_VALIDATED	213
ERROR_DISK_WRITE_PROTECTED	214
ERROR_RENAME_ACROSS_DEVICES	215
ERROR_DIRECTORY_NOT_EMPTY	216
ERROR_TOO_MANY_LEVELS	217
ERROR_DEVICE_NOT_MOUNTED	218
ERROR_SEEK_ERROR	219
ERROR_COMMENT_TOO_BIG	220
ERROR_DISK_FULL	221
ERROR_DELETE_PROTECTED	222
ERROR_WRITE_PROTECTED	223
ERROR_READ_PROTECTED	224
ERROR_NOT_A_DOS_DISK	225
ERROR_NO_DISK	226
ERROR_NO_MORE_ENTRIES	232

4. Gestion des procédés

CreateProc

Créer un nouveau procédé

Syntaxe

```
process = CreateProc(name,pri,segment,stacksize)
          D0      -138    D1    D2    D3      D4
struct Process *process;
UBYTE *name;
LONG pri;
BPTR *segment;
LONG stacksize;
```

Description

Cette fonction permet de créer un nouveau procédé avec le nom indiqué par "name". Pour cela, il alloue une structure de contrôle dans la mémoire libre et il l'initialise.

Paramètres

- name: Adresse du premier caractère du nom terminé par un 0.
pri: Priorité de la nouvelle procédure.

segment: Pointeur sur un segment.

stacksize: Taille de la pile en octets.

Retour

process: Identification de la procédure.

Voir aussi

LoadSeg, UnLoadSeg

DateStamp

Donner la date et l'heure

Syntaxe

```
DateStamp(ptr)
    -192    D1
    LONG *ptr;
```

Description

Cette fonction permet de donner la date et l'heure dans le format AmigaDOS.

Paramètres

ptr: Adresse de trois mots longs. Dans le premier de ces mots longs se trouve le nombre indiquant le jour. Le second contient les secondes écoulées dans le jour indiqué, et le troisième les microsecondes écoulées dans cette seconde.

Delay

Stopper un processus pendant un moment

Syntaxe

```
Delay(time)
    -198    D1
    LONG time;
```

Description

Cette fonction stoppe le processus en cours pendant le temps indiqué dans "time".

Paramètres

time: Temps exprimé en 1/50 de seconde.

DeviceProc**Donner l'identification d'un processus I/O****Syntaxe**

```
process = DeviceProc(name)
DO      -174      D1
struct Process *process;
UBYTE *name;
```

Description

Cette fonction retourne l'indentification d'un processus I/O.

Paramètres

name: Adresse du premier caractère du nom (chaîne terminée par 0).

Retour

process: Pointeur sur le processus.

Exit**Terminer un programme****Syntaxe**

```
Exit(errorCode)
-144      D1
LONG errorCode;
```

Description

Cette fonction permet de mettre fin à un programme ou à un processus. Lorsqu'un programme a été mis en route directement à partir du CLI, Exit met fin à ce programme et revient au CLI. Pour un programme mis en route à partir du Workbench, Exit met fin à l'ensemble du processus et libère la pile aussi bien que la place correspondante en mémoire.

Paramètres

errorCode: Nombre à retourner.

WaitForChar**Examiner la réception d'un caractère****Syntaxe**

```
ok = WaitForChar(Handle,time)
DO      -204      D1      D2
BOOL ok;
struct FileHandle *Handle;
LONG time;
```

Description

Cette fonction permet d'examiner si un caractère a été reçu par le fichier spécifié dans le laps de temps indiqué.

Paramètres

- Handle: Adresse de la structure FileHandle.
 time: Laps de temps à l'intérieur duquel le caractère doit avoir été reçu.

Retour

- ok: TRUE si OK ou FALSE si rien n'est arrivé.

Voir aussi

IsInteractive

Structure

```
struct DateStamp <libraries/dos.h>
{
Offset:
0x00 0 LONG ds_Days;
0x04 4 LONG ds_Minute;
0x08 8 LONG ds_Tick;
sizeof(struct DateStamp) -
0x0C 12
};

struct Process <libraries/dosextens.h>
{
Offset:
0x00 0 struct Task pr_Task;
0x5C 92 struct MsgPort pr_MsgPort;
0x7E 126 WORD pr_Pad;
0x80 128 BPTR pr_SegList;
0x84 132 LONG pr_StackSize;
0x88 136 APTR pr_GlobVec;
0x8C 140 LONG pr_TaskNum;
0x90 144 BPTR pr_StackBase;
0x94 148 LONG pr_Result2;
0x98 152 BPTR pr_CurrentDir;
0x9C 156 BPTR pr_CIS;
0xA0 160 BPTR pr_COS;
0xA4 164 APTR pr_ConsoleTask;
0xA8 168 APTR pr_FileSystemTask;
0xAC 172 BPTR pr_CLI;
0xB0 176 APTR pr_ReturnAddr;
0xB4 180 APTR pr_PktWait;
0xB8 184 APTR pr_WindowPtr;
sizeof(struct Process) -
0xBC 188
};
-4 LONG ;
```

```
0 BPTR ;
4 Code
```

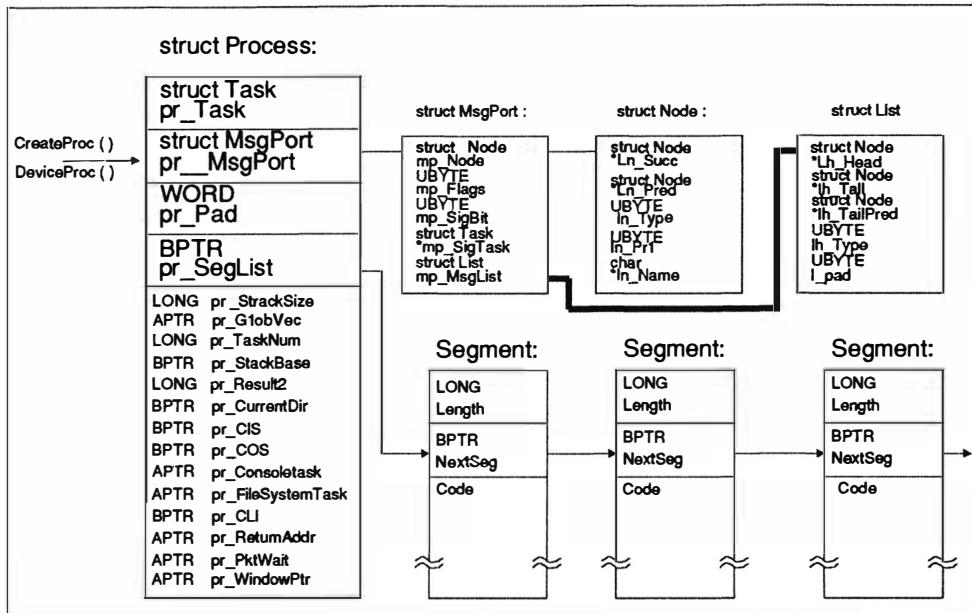


Figure 6 - 57

5. Exécution des programmes

Execute	Exécuter un programme
---------	-----------------------

Syntaxe	<pre>ok = Execute(command,in,out) D0 -222 D1 D2 D3 BOOL ok; UBYTE *command; struct FileHandle *in,*out;</pre>
----------------	--

Description

Cette fonction permet d'exécuter les fonctions CLI. Il faut pour cela que la commande CLI "Run" se trouve dans le répertoire C de la disquette de bootage.

Paramètres

command: La commande CLI dans une chaîne.

out: Numéro d'identification du fichier de saisie
 in: Numéro d'identification du fichier de saisie.

Retour

ok: TRUE ou FALSE.

LoadSeg

Charger un programme en mémoire

Syntaxe

```
segment = LoadSeg(name)
      DO      -150   D1
BPTR segment;
UBYTE *name;
```

Description

Cette fonction permet de charger un programme en mémoire et de le décomposer ses segments de code.

Paramètres

name: Nom du fichier

Retour

segment: Adresse à partir de laquelle la liste des segments se trouve en mémoire.

Voir aussi

UnLoadSeg, CreateProc

UnLoadSeg

Effacer un programme de la mémoire

Syntaxe

```
UnLoadSeg(segment)
      -156   D1
BPTR segment;
```

Description

Cette fonction permet d'effacer de la mémoire un programme chargé préalablement avec la fonction LoadSeg().

Paramètres

segment: Adresse de la liste des segments.

Voir aussi

LoadSeg

6.3. La librairie Intuition

Ce nom recouvre une bibliothèque du système d'exploitation dont la structure est tout à fait analogue à celle de la bibliothèque graphique. Intuition se charge des fenêtres, des écrans, des Requesters et des Alerts (par exemple celle du GOUROU-MEDITATION) entre autre chose.

L'activation de celle-ci se fait à l'aide la fonction Exec OpenLibrary("intuition.library", 0L).

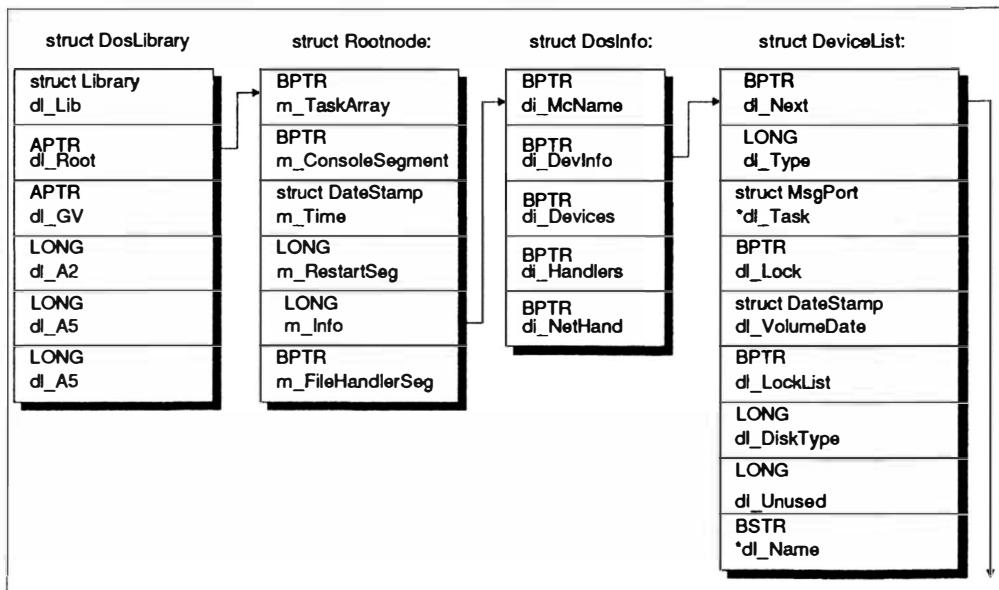


Figure 6 - 58

Les fonctions de la librairie Intuition

1. Les fonctions des fenêtres

*ActivateWindow	823
*RefreshWindowFrame	825
CloseWindow	823

ModifyIDCMP	824
MoveWindow	824
OpenWindow	825
SetWindowTitle	826
SizeWindow	826
WindowLimits	827
WindowToBack	827
WindowToFront	828

2. Les fonctions des gadgets

*ActivateGadget	830
*AddGList	831
*NewModifyProp	833
*RefreshGList	835
*RemoveGList	836
AddGadget	831
ModifyProp	832
OffGadget	834
OnGadget	834
RefreshGadgets	835
RemoveGadget	836

3. Les fonctions des menus

ClearMenuStrip	839
ItemAddress	840
OffMenu	840
OnMenu	841
SetMenuStrip	841

4. Les fonctions Requester et Alert

AutoRequest	843
BuildSysRequest	844
ClearDMRequest	844
DisplayAlert	845
EndRequest	846
FreeSysRequest	846
InitRequester	846
Request	847
SetDMRequest	847

5. Les fonctions d'écrans

*GetScreenData	850
CloseScreen	849
CloseWorkBench	849
DisplayBeep	850
MakeScreen	851
MoveScreen	851
OpenScreen	851
OpenWorkBench	852
ScreenToBack	852
ScreenToFront	853
ShowTitle	853
WBenchToBack	853
WBenchToFront	854

6. Les fonctions graphiques

ClearPointer	855
DrawBorder	856
DrawImage	856
IntuiTextLength	857
PrintText	857
SetPointer	858

7. Les fonctions de mise en mémoire

AllocRemember	860
FreeRemember	860

8. Les fonctions "Refresh"

BeginRefresh	861
EndRefresh	861
RemakeDisplay	862
RethinkDisplay	862

9. Les fonctions Internes

*LockIBase	863
*UnlockIBase	864
AlohaWorkbench	863
Intuition	863
ViewAddress	864
ViewPortAddress	865

10. Les fonctions spéciales

*SetPrefs	871
CurrentTime	869
DoubleClick	869
GetDefPrefs	870
GetPrefs	870
ReportMouse	871

Description des fonctions

1. Les fonctions des fenêtres

*ActivateWindow

Activer une fenêtre

Syntaxe

```
ActivateWindow(Window)
    -450      A0
    struct Window *Window;
```

Description

Cette fonction permet d'activer une fenêtre.

Paramètres

Window: Adresse de la structure Window.

Voir aussi

OpenWindow()

CloseWindow

Fermer une fenêtre

Syntaxe

```
CloseWindow(Window)
    -72      A0
    struct Window *Window;
```

Description

Cette fonction permet de fermer une fenêtre et de libérer la mémoire correspondante.
Attention, il n'y a aucune valeur en retour.

Paramètres

Window: Adresse de la structure Window.

Voir aussi

[OpenWindow\(\)](#), [CloseScreen\(\)](#)

ModifyIDCMP**Modifier les réglages IDCMP****Syntaxe**

```
ModifyIDCMP(Window, IDCMPFlags)
    -150      A0      D0
    struct Window *Window;
    ULONG      IDCMPFlags;
```

Description

Cette fonction permet de modifier les réglages IDCMP de la fenêtre indiquée. Pas de valeur en retour.

Paramètres

Window: Adresse de la structure Window.

IDCMPFlags: Nouveaux réglages IDCMP.

Voir aussi

[OpenWindow\(\)](#), [CloseWindow\(\)](#)

MoveWindow**Déplacer une fenêtre****Syntaxe**

```
MoveWindow(Window, DeltaX, DeltaY)
    -168      A0      D0      D1
    struct Window *Window;
    SHORT      DeltaX;
    SHORT      DeltaY;
```

Description

Cette fonction permet de déplacer la fenêtre d'une valeur indiquée dans DeltaX et DeltaY. Attention, il n'y a aucune valeur en retour.

Paramètres

Window: Adresse de la structure Window.

DeltaX: Nouvelle abscisse de la fenêtre, relative par rapport à l'ancienne.

DeltaY: Nouvelle ordonnée de la fenêtre, relative par rapport à l'ancienne.

Voir aussi

SizeWindow()

OpenWindow**Ouvrir une nouvelle fenêtre****Syntaxe**

```
Window = OpenWindow(&NewWindow);
DO      -204      A0
struct Window *Window;
struct NewWindow NewWindow;
```

Description

Cette fonction permet d'ouvrir une nouvelle fenêtre sur l'écran indiqué dans la structure NewWindow.

Paramètres

NewWindow: Adresse de la structure NewWindow qui contient les attributs de la fenêtre.

Retour

Windows: Si OK, pointeur sur la structure de la nouvelle fenêtre.

Voir aussi

CloseWindow(), ModifyIDCMP(), OpenScreen(), CloseScreen(), WindowTitles()

RefreshWindowFrame*Refaire l'encadrement d'une fenêtre****Syntaxe**

```
RefreshWindowFrame(Window);
-456      A0
struct Window *Window;
```

Description

Cette fonction permet de retracer l'encadrement d'une fenêtre. Attention, il n'y a aucune valeur en retour.

Paramètres

Window: Adresse de la structure Window.

Voir aussi

RefershGadgets(), RefreshGLList()

SetWindowTitle**Définir un titre dans une fenêtre****Syntaxe**

```
SetWindowTitle(Window, WindowTitle, ScreenTitle);
    -276      A0      A1      A2
    struct Window *Window
    UBYTE      *WindowTitle;
    UBYTE      *ScreenTitle;
```

Description

Cette fonction permet de créer un nouveau titre pour la fenêtre. On peut aussi indiquer dans la fonction un titre d'écran qui sera toujours affiché comme titre d'écran lorsqu'on activera la fenêtre. Pas de valeur en retour.

Paramètres

Window: Adresse de la structure Window.

WindowTitle: Nouveau titre de fenêtre.

ScreenTitle: Titre de l'écran spécifique à la fenêtre.

Voir aussi

`OpenWindow()`, `RefreshWindowFrame()`, `OpenScreen()`

SizeWindow**Définir la taille d'une fenêtre****Syntaxe**

```
SizeWindow(Window, DeltaX, DeltaY);
    -288      A0      D0      D1
    struct Window *Window;
    SHORT      DeltaX;
    SHORT      DeltaY;
```

Description

Cette fonction permet de définir la taille (agrandir ou réduire) d'une fenêtre. Pas de valeur en retour.

Paramètres

Window: Adresse de la structure Window.

DeltaX: Nouvelle abscisse relative par rapport à l'ancienne.

DeltaY: Nouvelle ordonnée relative par rapport à l'ancienne.

Voir aussi

[MoveWindow\(\)](#), [OpenWindow\(\)](#)

WindowLimits**Modifier les limites d'une fenêtre****Syntaxe**

```
Fonct = WindowLimits(Window, MinX, MinY, MaxX, MaxY);
      D0      -318     A0    D0    D1    D2    D3
      BOOL      Erfolg;
      struct Window *Window;
      USHORT      MinX;
      USHORT      MinY;
      USHORT      MaxX;
      USHORT      MaxY;
```

Description

Cette fonction permet de modifier les limites de la taille d'une fenêtre.

Paramètres

- | | |
|---------|---|
| Window: | Adresse de la structure Window. |
| MinX: | Nouvelles limites pour la taille de la fenêtre. |
| MinY: | Nouvelles limites pour la taille de la fenêtre. |
| MaxX: | Nouvelles limites pour la taille de la fenêtre. |
| MaxY: | Nouvelles limites pour la taille de la fenêtre. |

Retour

- | | |
|--------|--|
| Fonct: | TRUE si tout est en ordre ou FALSE si les dimensions ne sont pas bonnes. |
|--------|--|

Voir aussi

[GetScreenData\(\)](#)

WindowToBack**Repousser une fenêtre en arrière-plan****Syntaxe**

```
WindowToBack(Window);
      -306     A0
      struct Window *Window;
```

Description

Cette fonction permet de repousser une fenêtre à l'arrière-plan, de façon à être recouverte par d'autres fenêtres.

Paramètres

Window: Adresse de la structure Window.

Voir aussi

`WindowToFront()`, `MoveWindow()`, `SizeWindow()`

WindowToFront

Passer une fenêtre en avant plan

Syntaxe

```
WindowToFront(Window);
-312      A0
struct Window *Window;
```

Description

Cette fonction permet de ramener une fenêtre au premier plan.

Paramètres

Window: Adresse de la structure Window.

Voir aussi

`WindowToBack()`, `MoveWindow()`, `SizeWindow()`

Structure

```
struct NewWindow <intuition/intuition.h>
{
 0x00 00  SHORT LeftEdge;
 0x02 02  SHORT TopEdge;
 0x04 04  SHORT Width;
 0x06 06  SHORT Height;
 0x08 08  UBYTE DetailPen;
 0x09 09  UBYTE BlockPen;
 0x0A 10  ULONG IDCMPFlags;
 0x0E 14  ULONG Flags;
 0x12 18  struct Gadget *FirstGadget;
 0x16 22  struct Image *CheckMark;
 0x1A 26  UBYTE *Title;
 0x1E 30  struct Screen *Screen;
 0x22 34  struct Bitmap *Bitmap;
 0x26 38  SHORT MinWidth;
 0x28 40  SHORT MinHeight;
 0x2A 42  USHORT MaxWidth;
 0x2C 44  USHORT MaxHeight;
 0x2E 46  USHORT Type;
```

```

0x30 48
};

struct Window <intuition/intuition.h>
{
0x00 00 struct Window *NextWindow;
0x04 04 SHORT LeftEdge;
0x06 06 SHORT TopEdge;
0x08 08 SHORT Width;
0x0A 10 SHORT Height;
0x0C 12 SHORT MouseY;
0x0E 14 SHORT MouseX;
0x10 16 SHORT MinWidth;
0x12 18 SHORT MinHeight;
0x14 20 USHORT MaxWidth;
0x16 22 USHORT MaXHeight;
0x18 24 ULONG Flags;
0x1C 28 struct Menu *MenuStrip;
0x20 32 UBYTE *Title;
0x24 36 struct Requester *FirstRequest;
0x28 40 struct Requester *DMRequest;
0x2C 44 SHORT ReqCount;
0x2E 46 struct Screen *WScreen;
0x32 50 struct RastPort *RPort;
0x36 54 BYTE BorderLeft;
0x37 55 BYTE BorderTop;
0x38 56 BYTE BorderRight;
0x39 57 BYTE BorderBottom;
0x3A 58 struct RastPort *BorderRPort;
0x3E 62 struct Gadget *FirstGadget;
0x42 66 struct Window *Parent;
0x46 70 struct Window *Descendant;
0x4A 74 USHORT *Pointer;
0x4E 78 BYTE PtrHeight;
0x4F 79 BYTE PtrWidth;
0x50 80 BYTE XOffset;
0x51 81 BYTE YOffset;
0x52 82 ULONG IDCMPFlags;
0x56 86 struct MsgPort *UserPort;
0x5A 90 struct MsgPort *WindowPort;
0x5E 94 struct IntuiMessage *MessageKey;
0x62 98 UBYTE DetailPen;
0x63 99 UBYTE BlockPen;
0x64 100 struct Image *CheckMark;
0x68 104 UBYTE *ScreenTitle;
0x6C 108 SHORT GZZMouseX;
0x6E 110 SHORT GZZMouseY;
0x70 112 SHORT GZZWidth;
0x72 114 SHORT GZZHeight;
0x74 116 UBYTE *ExtData;
0x78 120 BYTE *UserData;
0x7C 124 struct Layer *WLayer;
0x80 128 struct TextFont *IFont;
0x84 132
};

Window_Flags:
WINDOWSIZING      0x0001L
WINDOWDRAG        0x0002L

```

WINDOWDEPTH	0x0004L
WINDOWCLOSE	0x0008L
SIZEBRIGHT	0x0010L
SIZEBBOTTOM	0x0020L
REFRESHBITS	0x00COL
SMART_REFRESH	0x0000L
SIMPLE_REFRESH	0x0040L
SUPER_BITMAP	0x0080L
OTHER_REFRESH	0x00COL
BACKDROP	0x0100L
REPORTMOUSE	0x0200L
GIMMEZEROZERO	0x0400L
BORDERLESS	0x0800L
ACTIVATE	0x1000L
WINDOWACTIVE	0x2000L
INREQUEST	0x4000L
MENUSTATE	0x8000L
RMBTRAP	0x00010000L
NOCAREREFRESH	0x00020000L
WINDOWREFRESH	0x01000000L
WBENCHWINDOW	0x02000000L
WINDOWTICKED	0x04000000L
SUPER_UNUSED	0xFCFC0000L

2. Les fonctions des gadgets

*ActivateGadget

Activer un gadget

Syntaxe

```

Succes = ActivateGadget(Gadget, Window, Requester);
      DO          -462      A0      A1      A2
      BOOL        Succes;
      struct Gadget *Gadget;
      struct Window *Window;
      struct Requester *Request;

```

Description

Cette fonction active un gadget sous forme de chaîne dans une fenêtre ou dans une boîte de dialogue (requester).

Paramètres

Gadget: Adresse de la structure String-Gadget.

Window: Adresse de la structure Window dans lequel se trouve le gadget.

Requester: Adresse de la structure Requester ou 0.

Retour

Succes: TRUE si OK ou FALSE s'il y a une erreur.

AddGadget**Ajouter un gadget****Syntaxe**

```
RealPosition = AddGadget(Window, Gadget, Position);
      D0          -42     A0      A1      D0
      USHORT      RealPosition;
      struct Window *Window;
      struct Gadget *Gadget;
      USHORT      Position;
```

Description

Cette fonction permet d'insérer un gadget dans la liste des gadgets d'un écran ou d'une fenêtre.

Paramètres

Window: Adresse de la structure Screen ou Window.

Gadget: Adresse de la structure Gadget.

Position: Numéro du gadget dans la liste. 0 si c'est le premier.

Retour

Real Postion: Position du gadget dans la liste.

Voir aussi

AddGList(), RemoveGadget(), RemoveGList()

AddGList*Insérer plusieurs gadgets****Syntaxe**

```
RealPosition = AddGList(Window, Gadget, Position, Numgad,
Requester);
      D0          -438     A0      A1      D0      D1
      D2
      USHORT      RealPosition;
      struct Window *Window;
      struct Gadget *Gadget;
      USHORT      Position;
      USHORT      Numgad;
      struct Requester *Requester;
```

Description

Cette fonction permet d'insérer plusieurs gadgets à la fois dans la liste des gadgets d'une fenêtre ou d'une boîte de dialogue.

Paramètres

- Window: Adresse de la structure de fenêtre.
 Gadget: Adresse du premier gadget de la liste des gadgets.
 Position: Numéro de gadget à l'endroit où la liste des gadgets est insérée.
 Numgad: Nombre des gadgets à insérer.
 Requester: Adresse de la structure Requester.

Retour

RealPosition: numéro de la liste des gadgets.

Voir aussi

[AddGadget\(\)](#), [RemoveGadget\(\)](#), [RemoveGList\(\)](#)

ModifyProp**Modifier les réglages d'un gadget**

Syntaxe

```
ModifyProp(Gadget,Window,Requester,Flags,HorizPot,VertPot,HorizBody,VertBody);
-156      A0      A1      A2      D0      D1      D2      D3      D4
struct Gadget *Gadget;
struct Window *Window;
struct Requester *Requester;
USHORT      Flags;
USHORT      HorizPot;
USHORT      VertPot;
USHORT      HorizBody;
USHORT      VertBody;
```

Description

Cette fonction modifie les réglages pour un gadget proportionnel. Le gadget est dessiné à neuf lors de l'appel de la fonction.

Paramètres

- Gadget: Adresse du gadget proportionnel.
 Window: Adresse de la structure Window.
 Requester: Adresse de la structure Requester. S'il s'agit d'un gadget de fenêtre, "requester" doit prendre la valeur 0.
 Flags: Nouveaux flags proportionnels.
 HorizPot: Nouveau pas horizontal.
 VertPot: Nouveau pas vertical.
 HorizBody: Nouvelle taille horizontale.

VertBody: Nouvelle taille verticale.

Voir aussi

NewModifyProp()

*NewModifyProp

Modifier un gadget proportionnel

Syntaxe

```
NewModifyProp(Gadget, Window, Requester, Flags, HorizPot,
468      A0      A1      A2      D0      D1
VertPot, HorizBody, VertBody);
D2      D3      D4
struct Gadget *Gadget;
struct Window *Window;
struct Requester *Requester;
USHORT   Flags;
USHORT   HorizPot;
USHORT   VertPot;
USHORT   HorizBody;
USHORT   VertBody;
```

Description

Cette fonction permet de modifier les paramètres d'un gadget proportionnel. Le gadget est alors recalculé et réaffiché.

Paramètres

- Gadget:** Pointeur sur la structure du gadget proportionnel.
- Window:** Pointeur sur la structure de l'élément affiché (fenêtre ou écran).
- Requester:** Pointeur sur la structure Requester (doit être à 0 si ce n'est pas un Requester).
- Flags:** Valeur devant être stockée dans la variable Flags de PropInfo.
- HorizPot:** Valeur devant être stockée dans la variable HorizPot de PropInfo.
- VertPot:** Valeur devant être stockée dans la variable VertPot de PropInfo.
- HorizBody:** Valeur devant être stockée dans la variable HorizBody de PropInfo.
- VertBody:** Valeur devant être stockée dans la variable VertBody de PropInfo.

Voir aussi

ModifyProp()

OffGadget**Désactiver un gadget****Syntaxe**

```
OffGadget(Gadget, Window, Requester);
-174      A0      A1      A2
struct Gadget    *Gadget;
struct Window   *Window;
struct Requester *Requester;
```

Description

Cette fonction permet de mettre hors service le gadget indiqué, et de le recouvrir par un raster de points (une trame) pour que la désactivation soit bien mise en évidence. Cette fonction appliquée à un gadget n'a de sens que si celui-ci est affiché en graphique ; dans le cas contraire, en effet, le quadrillage par un raster ne pourra plus être effacé.

Paramètres

Gadget: Adresse de la structure Gadget.

Window: Adresse de la structure Window.

Requester: Adresse de la structure Requester. S'il s'agit d'un gadget de fenêtre, la valeur de "Requester" peut être mise à 0.

Voir aussi

[OnGadget\(\)](#)

OnGadget**Activer un gadget****Syntaxe**

```
OnGadget(Gadget, Window, Requester);
-186      A0      A1      A2
struct Gadget    *Gadget;
struct Window   *Window;
struct Requester *Requester;
```

Description

Cette fonction permet d'activer à nouveau le gadget qui avait été désactivé par la fonction OffGadget().

Paramètres

Gadget: Adresse de la structure Gadget.

Window: Adresse de la structure Window.

Requester: Adresse de la structure Requester (seulement pour les gadgets des boîtes de dialogue).

Voir aussi

OffGadget()

RefreshGadgets**Redessiner des gadgets****Syntaxe**

```
RefreshGadgets(Gadgets, Window, Requester);
    -222      A0      A1      A2
    struct Gadget *Gadget;
    struct Window *Window;
    struct Requester *Requester;
```

Description

Cette fonction permet de redessiner la liste des gadgets d'une fenêtre ou d'une boîte de dialogue. Tous les gadgets sont redessinés à partir de celui qui est indiqué en paramètre et jusqu'à la fin de liste.

Paramètres

Gadget: Adresse de la structure Gadget.

Window: Adresse de la structure Window.

Requester: Adresse de la structure Requester, s'il s'agit d'un gadget de boîte de dialogue. Sinon 0.

Voir aussi

RefreshGList(), RemoveGadget(), RemoveGList(), AddGadget(), AddGList()

RefreshGList*Redessiner la liste des gadgets****Syntaxe**

```
RefreshGList(Gadgets, Window, Requester, Numgad);
    -432      A0      A1      A2      D0
    struct Gadget *Gadget;
    struct Window *Window;
    struct Requester *Requester;
    SHORT      Numgad;
```

Description

Cette fonction permet de redessiner la liste des gadgets d'une fenêtre ou d'une boîte de dialogue. Le nombre de gadgets redessinés, à partir de celui qui est indiqué en paramètre, est donné par Numgad.

Paramètres

Gadget: Adresse de la structure Gadget.

Window: Adresse de la structure Window.

Requester: Adresse de la structure Requester, s'il s'agit d'un gadget de boîte de dialogue. Sinon 0.

Numgad: Nombre de gadgets qui doivent être redessinés.

Voir aussi

`RefreshGadgets()`, `RemoveGadget()`, `RemoveGLList()`, `AddGadget()`, `AddGLList()`

RemoveGadget**Supprimer un gadget****Syntaxe**

```
Position = RemoveGadget(Window, Gadget);
          DO           -228      A0      A1
USHORT      Position;
struct Window *Window;
struct Gadget *Gadget;
```

Description

Cette fonction permet d'effacer un gadget dans la liste des gadgets d'une fenêtre.

Paramètres

Window: Adresse de la structure Window.

Gadget: Adresse de la structure Gadget.

Retour

Position: Position du gadget à l'intérieur de la liste.

Voir aussi

`AddGadget()`, `AddGLList()`, `RemoveGLList()`

RemoveGLList*Effacer des gadgets****Syntaxe**

```
Position = RemoveGLList(Window, Gadget, Numgad);
          DO           -444      A0      A1      D0
USHORT      Position;
```

```
struct Window *Window;
struct Gadget *Gadget;
SHORT      Numgad;
```

Description

Cette fonction permet d'effacer dans la liste des gadgets d'une fenêtre un certain nombre de gadgets fixé par Numgad.

Paramètres

Window: Adresse de la structure Window.

Gadget: Adresse de la structure Gadget.

Numgad: Nombre de gadgets à effacer.

Retour

Postion: Position du premier gadget à l'intérieur de la liste.

Voir aussi

`RemoveGadget()`, `AddGadget()`, `AddGList()`

Structure

```
struct Gadget <intuition/intuition.h>
{
 0x00 00 struct Gadget *NextGadget;
 0x04 04 SHORT LeftEdge;
 0x06 06 SHORT TopEdge;
 0x08 08 SHORT Width;
 0x0A 10 SHORT Height;
 0x0C 12 USHORT Flags;
 0x0E 14 USHORT Activation;
 0x10 16 USHORT GadgetType;
 0x12 18 APTR GadgetRender;
 0x16 22 APTR SelectRender;
 0x1A 26 struct IntuiText *GadgetText;
 0x1E 30 LONG MutualExclude;
 0x22 34 APTR SpecialInfo;
 0x26 38 USHORT GadgetID;
 0x28 40 APTR UserData;
 0x2C 44
};

Gadget_Flags
GADGHIGHBITS 0x0003L
GADGHCOMP    0x0000L
GADGHBBOX   0x0001L
GADGHIIMAGE 0x0002L
GADGHNONE   0x0003L
GADGIMAGE    0x0004L
GRELBOTTOM  0x0008L
GRELRIGHT   0x0010L
GRELWIDTH   0x0020L
```

```
GRELHEIGHT      0x0040L
SELECTED        0x0080L
GADGDISABLED    0x0100L

Gadget_Activation
RELVERIFY       0x0001L
GADGIMMEDIATE   0x0002L
ENDGADGET       0x0004L
FOLLOWMOUSE     0x0008L
RIGHTBORDER     0x0010L
LEFTBORDER      0x0020L
TOPBORDER       0x0040L
BOTTOMBORDER    0x0080L
TOGGLESELECT    0x0100L
STRINGCENTER    0x0200L

STRINGRIGHT     0x0400L
LONGINT         0x0800L
ALTKEYMAP       0x1000L

Gadget_GadgetType
BOOLEXTEND      0x2000L
GADGETTYPE      0xFC00L
SYSGADGET       0x8000L
SCRGADGET       0x4000L
GZZGADGET       0x2000L
REQGADGET       0x1000L
SIZING          0x0010L
WDRAGGING       0x0020L
SDRAGGING       0x0030L
WUPFRONT        0x0040L
SUPFRONT        0x0050L
WDOWNBACK       0x0060L
SDOWNBACK       0x0070L
CLOSE           0x0080L
BOOLGADGET      0x0001L
GADGET0002      0x0002L
PROPGADGET      0x0003L
STRGADGET       0x0004L

struct BoolInfo <intuition/intuition.h>
{
0x00 00 USHORT Flags;
0x02 02 UWORLD *Mask;
0x04 04 ULONG Reserved;
0x08 08
};

BoolInfo_Flags
BOOLMASK 0x00001

struct PropInfo
{
0x00 00 USHORT Flags;
0x02 02 USHORT HorizPot;
0x04 04 USHORT VertPot;
0x06 06 USHORT HorizBody;
0x08 08 USHORT VertBody;
0x0A 10 USHORT CWidth;
```

```

0x0C 12 USHORT CHeight;
0x0E 14 USHORT HPotRes;
0x10 16 USHORT VPotRes;
0x12 18 USHORT LeftBorder;
0x14 20 USHORT TopBorder;
0x16 22
};

PropInfo_Flags
AUTOKNOB      0x0001L
FREEHORIZ    0x0002L
FREEVERT     0x0004L
PROPBORDERLESS 0x0008L
KNOBHIIT     0x0100L

PropInfo_Valeur
KNOBHMIN      6L
KNOBVMIN      4L
MAXBODY       0xFFFFL
MAXPOT        0xFFFFL

struct StringInfo <intuition/intuition.h>
{
0x00 00 UBYTE *Buffer;
0x04 04 UBYTE *UndoBuffer;
0x08 08 SHORT BufferPos;
0x0A 10 SHORT MaxChars;
0x0C 12 SHORT DispPos;
0x0E 14 SHORT UndoPos;
0x10 16 SHORT NumChars;
0x12 18 SHORT DispCount;
0x14 20 SHORT CLeft;
0x16 22 SHORT CTop;
0x18 24 struct Layer *LayerPtr;
0x1C 28 LONG LongInt;
0x20 32 struct KeyMap *AltKeyMap;
0x24 36
};

```

3. Les fonctions des menus

ClearMenuStrip

Supprimer la bordure des menus

Syntaxe

```

ClearMenuStrip(Window);
-54      A0
struct Window *Window;

```

Description

Cette fonction permet de supprimer la barre de menu dans une fenêtre.

Paramètres

Window: Adresse de la structure Window.

Voir aussi[SetMenuStrip\(\)](#)**ItemAddress****Trouver l'entrée d'un menu****Syntaxe**

```
ItemAddress - ItemAddress(MenuStrip, MenuNumber);
      D0          -144      A0      D0
      struct MenuItem *ItemAddress;
      struct Menu    *MenuStrip;
      USHORT        MenuNumber;
```

Description

Cette fonction communique l'adresse à partir de laquelle on trouve l'entrée de menu recherchée.

Paramètres

MenuStrip: Numéro du menu dans lequel se trouve l'entrée de menu.

MenuNumber: Numéro de l'entrée dont l'adresse MenuItem doit être communiquée.

Retour

ItemAddress: Null ou adresse du MenuItem spécifié par MenuNumber.

OffMenu**Désactiver un menu****Syntaxe**

```
OffMenu(Window, MenuNumber);
      -180      A0      D0
      struct Window *Window;
      USHORT        MenuNumber;
```

Description

Cette fonction permet de désactiver un menu, de façon à ce qu'il reste affiché, mais ne puisse plus être sélectionné. Pas de valeur en retour.

Paramètres

Window: Adresse de la structure Window.

MenuNumber: Numéro du menu.

Voir aussi

OnMenu()

OnMenu**Activer un menu****Syntaxe**

```
OnMenu(Window, MenuNumber);
      -192    A0      D0
      struct Window *Window;
      USHORT      MenuNumber;
```

Description

Cette fonction permet d'activer à nouveau le menu qui avait été désactivé par la fonction OffMenu(). Pas de valeur en retour.

Paramètres

Window: Adresse de la structure Window.

MenuNumber: Numéro du menu.

Voir aussi

OffMenu()

SetMenuStrip**Rattacher la barre des menus à une fenêtre****Syntaxe**

```
Success = SetMenuStrip(Window, Menu);
      D0      -264      A0      A1
      struct Window *Window;
      struct Menu  *Menu;
```

Description

Cette fonction permet de rattacher la barre des menus à une fenêtre. Si l'utilisateur appuie sur le bouton droit de la souris, le menu sera alors affiché et accessible.

Paramètres

Window: Pointeur sur la structure Window.

Menu: Pointeur sur la structure du premier menu.

Retour:

Success: TRUE si aucun problème et pendant le temps d'attente.

Voir aussi**ClearMenuStrip()****Structure**

```
struct Menu <intuition/intuition.h>
{
 0x00 00 struct Menu *NextMenu;
 0x04 04 SHORT LeftEdge;
 0x06 06 SHORT TopEdge;
 0x08 08 SHORT Width;
 0x0A 10 SHORT Height;
 0x0C 12 USHORT Flags;
 0x0E 14 BYTE *MenuName;
 0x0F 18 struct MenuItem *FirstItem;
 0x16 22 SHORT JazzX;
 0x18 24 SHORT Jazzy;
 0x10 26 SHORT BeatX;
 0x12 28 SHORT BeatY;
 0x14B 30
};

Menu_Flags
MENUNABLED      0x0001L
MIDRAWN         0x0100L

struct MenuItem <intuition/intuition.h>
{
 0x00 00 struct MenuItem *NextItem;
 0x04 04 SHORT LeftEdge
 0x06 06 SHORT TopEdge;
 0x08 08 SHORT Width;
 0x0A 10 SHORT Height;
 0x0C 12 USHORT Flags;
 0x0E 14 LONG MutualExclude;
 0x12 18 APTR ItemFill;
 0x16 22 APTR SelectFill;
 0x1A 26 BYTE Command;
 0x1B 27 struct MenuItem *SubItem;
 0x1F 31 USHORT NextSelect;
 0x21 33
};

MenuItem_Flags
CHECKIT          0x0001L
ITEMTEXT         0x0002L
COMMSEQ          0x0004L
MENUTOGGLE       0x0008L
ITEMENABLED      0x0010L
HIGHFLAGS        0x00COL
HIGHIMAGE        0x0000L
HIGHCOMP          0x0040L
HIGHBOX           0x0080L
HIGHNONE          0x00COL
CHECKED          0x0100L
ISDRAWN          0x1000L
```

```
HIGHITEM    0x2000L
MENUTOGGLED 0x4000L
```

4. Les fonctions Requester et Alert

AutoRequest

Ouvrir une boîte de dialogue système

Syntaxe

```
Reponse = AutoRequest(Window, BodyText, PositiveText,
                      DO      -348     A0     A1     A2
                      NegativeText, PositiveFlags, NegativeFlags, Width, Height);
                      A3      DO      D1     D2     D3
BOOL           Response;
struct Window   *Window;
struct IntuiText *BodyText;
struct IntuiText *PositiveText;
struct IntuiText *NegativeText;
ULONG          PositiveFlags;
ULONG          NegativeFlags;
SHORT          Width;
SHORT          Height;
```

Description

Cette fonction ouvre une boîte de dialogue système et attend les réactions de l'utilisateur.

Paramètres

- Window: Adresse de la structure Window.
- BodyText: Adresse de la structure IntuiText.
- PositiveText: Adresse de la seconde structure IntuiText (Gadget "OK").
- NegativeText: Adresse de la troisième structure IntuiText (Gadget "Annuler").
- PositiveFlags: Flags IDCMP (TRUE).
- NegativeFlags: Flags IDCMP (FALSE).
- Width: Largeur de la boîte de dialogue en pixels.
- Height: Hauteur de la boîte de dialogue en pixels.

Retour

- Reponse: TRUE ou FALSE.

Voir aussi

[BuildSysRequest\(\)](#), [Request\(\)](#)

BuildSysRequest**Ouvrir une boîte de dialogue****Syntaxe**

```
ReqWindow = BuildSysRequest(Window, BodyText, PositiveText,
    DO          -360      A0      A1      A2
    NegativeText, IDCMPFlags, Width, Height);
    A3          DO      D2      D3
    struct Window *ReqWindow;
    struct Window *Window;
    struct IntuiText *BodyText;
    struct IntuiText *PositiveText;
    struct IntuiText *NegativeText;
    ULONG          IDCMPFlags;
    SHORT          Width;
    SHORT          Height;
```

Description

Cette fonction ouvre une boîte de dialogue dans une fenêtre et attend les réactions de l'utilisateur. S'il n'y a aucune adresse de fenêtre indiquée, Intuition crée une fenêtre pour la boîte de dialogue.

Paramètres

- Window: Adresse de la structure Window ou 0.
- BodyText: Adresse de la structure IntuiText.
- PositiveText: Adresse de la seconde structure IntuiText.
- NegativeText: Adresse de la troisième structure IntuiText.
- IDCMPFlags: Flags IDCMP.
- Width: Largeur de la boîte de dialogue.
- Height: Hauteur de la boîte de dialogue.

Retour

ReqWindow: Adresse de la structure Window.

Voir aussi

[AutoRequest\(\)](#), [FreeSysRequest\(\)](#), [DisplayAlert\(\)](#), [ModifyIDCMP\(\)](#)

ClearDMRequest**Effacer une boîte de dialogue****Syntaxe**

```
Reponse = ClearDMRequest(Window);
    DO          -48      A0
```

```
BOOL           Reponse;
struct Window *Window;
```

Description

Cette fonction permet d'effacer une boîte de dialogue DoubleMenu de la fenêtre.

Paramètres

Window: Adresse de la structure Window.

Retour

Reponse: TRUE si OK ou FALSE si la fonction ne peut pas effacer la boîte de dialogue.

Voir aussi

[SetDMRequest\(\)](#), [Request\(\)](#)

DisplayAlert

Ouvrir une boîte d'alarme à l'écran

Syntaxe

```
Reponse = DisplayAlert(AlertNumber, String, Height);
          DO      -90      D0      A0      D1
          BOOL   Reponse;
          ULONG  AlertNumber;
          UBYTE *String;
          LONG   Height;
```

Description

Cette fonction ouvre une boîte d'alarme à l'écran dans le style d'un GOUROU-MEDITATION

Paramètres

AlertNumber: Numéro du message d'alarme communiqué.

String: Chaîne contenant le message.

Height: Nombre de lignes.

Retour

Reponse: TRUE ou FALSE (si c'est une RECOVERY_ALERT).

EndRequest**Effacer une boîte de dialogue****Syntaxe**

```
EndRequest(Requester, Window);
      -120      A0      A1
struct Requester *Requester;
struct Window   *Window;
```

Description

Cette fonction permet d'effacer de la fenêtre la boîte de dialogue indiquée dans "requester". Pas de valeur en retour.

Paramètres

Requester: Adresse de la structure Requester.

Window: Adresse de la structure Window.

FreeSysRequest**Effacer des boîtes de dialogue****Syntaxe**

```
FreeSysRequest(Window);
      -372      A0
struct Window *Window;
```

Description

Cette fonction permet d'effacer toutes les boîtes de dialogue créées avec la fonction BuildSysRequest(). Pas de valeur en retour.

Paramètres

Window: Adresse de la structure Window.

Voir aussi

[BuildSysRequest\(\)](#), [AutoRequest\(\)](#)

InitRequester**Initialiser un Requester****Syntaxe**

```
InitRequester(Requester);
      -138      A0
struct Requester *Requester;
```

Description

Cette fonction permet d'initialiser une structure de boîte de dialogue. Cette fonction n'affiche pas la boîte à l'écran. Pas de valeur en retour.

Paramètres

Requester: Adresse de la structure Requester.

Voir aussi

Request(), EndRequest()

Request	Afficher une boîte de dialogue
Syntaxe	<pre>Success = Request(Requester, Window); DO -240 A0 A1 BOOL Success; struct Requester *Requester; struct Window *Window;</pre>
Description	Cette fonction permet de présenter une boîte de dialogue dans une fenêtre. La boîte indiquée doit avoir auparavant été initialisée avec la fonction InitRequester().
Paramètres	<p>Requester: Adresse de la structure Requester.</p> <p>Window: Adresse de la structure Window.</p>
Retour	Success: TRUE si le Requester est correctement ouvert ou FALSE si il ne peut pas être ouvert.
Voir aussi	EndRequest()

SetDMRequest	Afficher une boîte de dialogue
Syntaxe	<pre>Success = SetDMRequest(Window, DMRequester); DO -258 A0 A1 BOOL Success; struct Window *Window; struct Requester *DMRequester;</pre>

Description

Cette fonction permet de représenter à l'écran une boîte de dialogue à double menu.

Paramètres

Window: Adresse de la structure Window.

DMRequester: Adresse de la structure Requester.

Retour

Success: TRUE si le DMRequest n'est pas utilisé et FALSE s'il l'est.

Voir aussi

[ClearDMRequest\(\)](#), [Request\(\)](#), [EndRequest\(\)](#)

Structure

```
struct Requester <intuition/intuition.h>
{
    0x00 00 struct Requester *OlderRequest;
    0x04 04 SHORT LeftEdge;
    0x06 06 SHORT TopEdge;
    0x08 08 SHORT Width;
    0x0A 10 SHORT Height;
    0x0C 12 SHORT RelLeft;
    0x0E 14 SHORT RelTop;
    0x10 16 struct Gadget *ReqGadget;
    0x14 20 struct Border *ReqBorder;
    0x18 24 struct IntuiText *ReqText;
    0x1C 28 USHORT Flags;
    0x1E 30 UBYTE BackFill;
    0x20 32 struct Layer *ReqLayer;
    0x24 36 UBYTE ReqPad1[32];
    0x44 68 struct Bitmap *ImageBMap;
    0x48 72 struct Window *RWindow;
    0x4C 76 UBYTE ReqPad2[36];
    0x70 112
};

Requester_Flags
POINTREL      0x0001L
PREDRAWN     0x0002L
NOISYREQ     0x0004L
REQOFFWINDOW 0x1000L
REQACTIVE    0x2000L
SYSREQUEST   0x4000L
DEFERREFRESH 0x8000L
```

5. Les fonctions d'écrans

CloseScreen

Fermer un écran

Syntaxe

```
CloseScreen(Screen);
    -66      A0
    struct Screen *Screen;
```

Description

Cette fonction permet de fermer l'écran et de libérer la mémoire correspondante. Pas de valeur en retour.

Paramètres

Screen: Adresse de la structure Screen.

Voir aussi

OpenScreen()

CloseWorkBench

Fermer l'écran Workbench

Syntaxe

```
Success = CloseWorkBench();
    DO      -78
    BOOL Success;
```

Description

Cette fonction permet de fermer l'écran Workbench. Si l'écran est ouvert, un test sera effectué pour savoir s'il existe encore au moment de l'appel de la fonction des fenêtres sur l'écran Workbench. Si c'est le cas, FALSE sera retourné et l'écran ne sera pas refermé.

Retour

Success: TRUE si l'écran est fermé, FALSE si impossible.

Voir aussi

OpenWorkBench(), CloseScreen()

DisplayBeep**Faire clignoter l'écran**

Syntaxe

```
DisplayBeep(Screen);
      -96          A0
struct Screen *Screen;
```

Description

Cette fonction permet de faire clignoter l'écran et de prévenir l'utilisateur de manière visuelle. Pas de valeur en retour.

Paramètres

Screen: Adresse de la structure Screen. Si l'on transmet 0, tous les écrans se mettent à clignoter.

GetScreenData*Copier une structure d'écran**

Syntaxe

```
Succes = GetScreenData(Buffer, Size, Type, Screen);
      DO          -426          A0          D0      D1          A1
BOOL          Succes;
CPTR          Buffer;
USHORT        Size;
USHORT        Type;
struct Screen *Screen;
```

Description

Cette fonction permet de copier dans une partie de la mémoire une structure d'écran (Screen).

Paramètres

Buffer: Pointeur sur le tampon dans lequel la structure doit être copiée.

Size: Taille du tampon en octets.

Type: Type de l'écran (WBENCHSCREEN, CUSTOMSCREEN, ...).

Screen: Ignoré, sauf si le type est CUSTOMSCREEN.

Retour

Succes: TRUE si OK ou FALSE.

MakeScreen**Mette en place un écran d'affichage****Syntaxe**

```
MakeScreen(Screen);
    -378      A0
struct Screen *Screen;
```

Paramètres

Screen: Adresse de la structure d'écran (Screen).

MoveScreen**Déplacer un écran****Syntaxe**

```
MoveScreen(Screen, DeltaX, DeltaY);
    -162      A0      D0      D1
struct Screen *Screen;
SHORT        DeltaX;
SHORT        DeltaY;
```

Description

Cette fonction permet de déplacer l'écran d'une valeur indiquée dans deltaX et deltaY.
Pas de valeur en retour

Paramètres

Screen: Adresse de la structure Screen.

DeltaX: Nouvelles coordonnées de l'écran, relatives par rapport aux anciennes.

DeltaY: Nouvelles coordonnées de l'écran, relatives par rapport aux anciennes.

OpenScreen**Ouvrir un écran****Syntaxe**

```
Screen = OpenScreen(NewScreen);
    D0      -198      A0
struct Screen   *Screen;
struct NewScreen NewScreen;
```

Description

Cette fonction permet d'afficher un nouvel écran sur le moniteur.

Paramètres

NewScreen: Adresse de la structure NewScreen contenant les attributs de l'écran.

Retour

Screen: Adresse de la structure Screen ou 0.

Voir aussi

`CloseScreen()`, `MakeScreen()`

OpenWorkBench**Ouvrir l'écran Workbench**

Syntaxe

```
WBScreen = OpenWorkBench();
      DO           -210
      struct Screen *WBScreen;
```

Description

Cette fonction essaie de réouvrir le Workbench.

Retour

WBScreen: TRUE si l'écran a été ouvert ou s'il est déjà ouvert. FALSE si l'écran Workbench n'est ou ne peut pas être ouvert.

Voir aussi

`CloseWorkBench()`, `OpenScreen()`, `CloseScreen()`

ScreenToBack**Placer l'écran en arrière plan**

Syntaxe

```
ScreenToBack(Screen);
      -246          A0
      struct Screen *Screen;
```

Description

Cette fonction permet de repousser l'écran à l'arrière-plan, de façon à pouvoir être recouvert par d'autres. Pas de valeur en retour.

Paramètres

Screen: Adresse de la structure Screen.

Voir aussi

ScreenToFront(), WBenchToBack(), WBenchToFront()

ScreenToFront**Placer l'écran à l'avant plan**

Syntaxe

```
ScreenToFront(Screen);
      -252      A0
      struct Screen *Screen;
```

Description

Cette fonction permet de replacer l'écran à l'avant-plan, de façon à pouvoir recouvrir d'autres écrans. Pas de valeur en retour.

Paramètres

Screen: Adresse de la structure Screen.

Voir aussi

ScreenToBack(), WBenchToBack(), WBenchToFront()

ShowTitle**Afficher le barre de titre d'une fenêtre**

Syntaxe

```
ShowTitle(Screen, ShowIt)
      -282      A0      D0
      struct Screen *Screen;
      BOOL        ShowIt;
```

Description

Cette fonction permet de spécifier si la barre de titre d'une fenêtre doit apparaître en avant ou en arrière plan (visible ou grisé).

Paramètres

Screen: Pointeur sur la structure d'écran (Screen).

ShowIt: TRUE pour voir la barre ou FALSE pour la cacher.

WBenchToBack**Placer l'écran Workbench en arrière plan**

Syntaxe

```
WBenchToBack();
      -336
```

Description

Cette fonction permet de placer l'écran Workbench derrière tous les autres.

Voir aussi

WBenchToFront(), ScreenToBack(), ScreenToFront()

WBenchToFront

Placer l'écran Workbench en avant plan

Syntaxe

```
WBenchToFront();
-342
```

Description

Cette fonction permet de placer l'écran Workbench devant tous les autres.

Voir aussi

WBenchToBack(), ScreenToBack(), ScreenToFront()

Structure

```
struct NewScreen <intuition/intuition.h>
    <intuition/screens.h>      /* Version 1.3 */
{
0x00 00  SHORT LeftEdge;
0x02 02  SHORT TopEdge;
0x04 04  SHORT Width;
0x06 06  SHORT Height;
0x08 08  SHORT Depth;
0x0A 10  UBYTE DetailPen;
0x0B 11  UBYTE BlockPen;
0x0C 12  USHORT ViewModes;
0x0E 14  USHORT Type;
0x10 16  struct TextAttr *Font;
0x14 20  UBYTE *DefaultTitle;
0x18 24  struct Gadget *Gadgets;
0x1C 28  struct Bitmap *CustomBitmap;
0x20 32
};

struct Screen <intuition/intuition.h>
    <intuition/screens.h>      /* Version 1.3 */
{
0x000 00  struct Screen *NextScreen;
0x004 04  struct Window *FirstWindow;
0x008 08  SHORT LeftEdge;
0x00A 10  SHORT TopEdge;
0x00C 12  SHORT Width;
0x00E 14  SHORT Height;
0x010 16  SHORT MouseY;
0x012 18  SHORT MouseX;
0x014 20  USHORT Flags;
```

```

0x016 22 UBYTE *Title;
0x01A 26 UBYTE *DefaultTitle;
0x01E 30 BYTE BarHeight;
0x01F 31 BYTE BarVBorder;
0x020 32 BYTE BarHBorder;/
0x021 33 BYTE MenuVBorder;
0x022 34 BYTE MenuHBorder;
0x023 35 BYTE WBorTop;
0x024 36 BYTE WBorLeft;
0x025 37 BYTE WBorRight;
0x026 38 BYTE WBorBottom;
0x028 40 struct TextAttr *Font;
0x02C 44 struct ViewPort ViewPort;
0x054 84 struct RastPort RastPort;
0x0B8 184 struct Bitmap Bitmap;
0x0E0 224 struct Layer_Info LayerInfo;
0x13C 316 struct Gadget *FirstGadget;
0x140 320 UBYTE DetailPen
0x141 321 UBYTE BlockPen;
0x142 322 USHORT SaveColor0;
0x144 324 struct Layer *BarLayer;
0x148 328 UBYTE *ExtData;
0x14C 332 UBYTE *UserData;
0x150 336
};

Screen_Flags
SCREENTYPE 0x000FL
WBENCHSCREEN 0x0001L
CUSTOMSCREEN 0x000FL
SHOWTITLE 0x0010L
BEEPING 0x0020L
CUSTOMBITMAP 0x0040L
SCREENBEHIND 0x0080L
SCREENQUIET 0x0100L
STDSCREENHEIGHT -1L

```

6. Les fonctions graphiques

ClearPointer	Définir l'aspect du pointeur
Syntaxe	<pre> ClearPointer(Window); -60 A0 struct Window *Window; </pre>
Description	Cette fonction permet de définir l'aspect du pointeur de la souris conformément aux images contenues dans Preferences. Pas de valeur en retour.

Paramètres

Window: Adresse de la structure Window.

Voir aussi[SetPointer\(\)](#)**DrawBorder****Dessiner des structures de bord****Syntaxe**

```
DrawBorder(RastPort, Border, LeftOffset, RightOffset);
    -108      A0      A1      D0      D1
struct RastPort *RastPort;
struct Border   *Border;
SHORT        LeftOffset;
SHORT        RightOffset;
```

Description

Cette fonction permet de dessiner des structures de bord. L'enchaînement des structures y est prise en compte. Pas de valeur en retour.

Paramètres

- RastPort: Adresse de la structure RastPort.
- Border: Adresse de la première structure de bord (Border).
- LeftOffset: Coordonnées de l'offset qui doivent être ajoutées aux coordonnées du point de départ du bord.
- RightOffset: Coordonnées de l'offset qui doivent être ajoutées aux coordonnées du point de départ du bord.

Voir aussi[DrawImage\(\)](#), [PrintText\(\)](#)**DrawImage****Dessiner des images****Syntaxe**

```
DrawImage(RastPort, Image, LeftOffset, TopOffset);
    -114      A0      A1      D0      D1
struct RastPort *RastPort;
struct Image   *Image;
SHORT        LeftOffset;
SHORT        TopOffset;
```

Description

Cette fonction permet de dessiner des images sur un écran ou dans une fenêtre. Si les structures d'images sont chaînées, toute la liste des images sera dessinée. Pas de valeur en retour.

Paramètres

- RastPort: Adresse de la structure RastPort.
- Image: Adresse de la première structure d'image.
- LeftOffset: Coordonnées qui doivent être ajoutées aux coordonnées du point de départ des images.
- TopOffset: Coordonnées qui doivent être ajoutées aux coordonnées du point de départ des images.

Voir aussi

[DrawBorder\(\)](#), [PrintIText\(\)](#)

IntuiTextLength

Calculer la longueur d'une chaîne IntuiText

Syntaxe

```
Width = IntuiTextLength(IText);
      DO      -330      A0
USHORT      Width;
struct IntuiText *IText;
```

Description

Cette fonction permet de calculer la longueur d'une chaîne IntuiText, exprimée en pixels. La longueur de la chaîne se réfère aux indications contenues dans la structure TextAttr de IntuiText.

Paramètres

- IText: Adresse de la structure IntuiText.

Retour

- Width: longueur de la chaîne en pixels.

Voir aussi

[PrintIText\(\)](#)

PrintIText

Sortir des chaînes de texte

Syntaxe

```
PrintIText(RastPort, IText, LeftOffset, TopOffset);
      -216      A0      A1      D0      D1
struct RastPort *RastPort;
struct IntuiText *IText;
```

```
SHORT           LeftOffset;
SHORT           TopOffset;
```

Description

Cette fonction sort sur le rastport indiqué les chaînes de texte des structures IntuiText, dans la forme décrite par ces structures. Si la structure indiquée est chaînée avec d'autres structures, celles-ci seront également affichées.

Paramètres

- | | |
|-------------|--|
| RastPort: | Adresse de la structure RastPort sur laquelle le texte doit être sorti. |
| IText: | Adresse de la première structure IntuiText. |
| LeftOffset: | Coordonnées qui doivent être ajoutées aux coordonnées de départ de tous les IntuiTex à afficher. |
| TopOffset: | Coordonnées qui doivent être ajoutées aux coordonnées de départ de tous les IntuiTex à afficher. |

Voir aussi

[IntuiTextLength\(\)](#), [DrawBorder\(\)](#), [DrawImage\(\)](#)

SetPointer

Créer un pointeur de souris

Syntaxe	<code>SetPointer(Window, Pointer, Height, Width, XOffset, YOffset);</code>
	-270 A0 A1 D0 D1 D2 D3
	struct Window *Window;
	USHORT *Pointer;
	SHORT Height;
	SHORT Width;
	SHORT XOffset;
	SHORT YOffset;

Description

Cette fonction permet de créer un pointeur de souris spécifique à l'écran. Pas de valeur en retour.

Paramètres

- | | |
|----------|--|
| Window: | Adresse de la structure Window. |
| Pointer: | Adresse des données Sprite pour le nouveau pointeur de souris. |
| Height: | Hauteur du pointeur, exprimées en pixels |
| Width: | Largeur du pointeur, exprimées en pixels. |

XOffset: Coordonnées relatives du point d'action.

YOffset: Coordonnées relatives du point d'action.

Voir aussi

[ClearPointer\(\)](#)

Structure

```
struct Border <intuition/intuition.h>
{
0x00 00 SHORT LeftEdge /
0x02 02 SHORT TopEdge;
0x04 04 UBYTE FrontPen
0x05 05 UBYTE BackPen;
0x06 06 UBYTE DrawMode;
0x07 07 BYTE Count;
0x08 08 SHORT *XY;
0x0C 12 struct Border *NextBorder;
0x10 16
};

struct Image <intuition/intuition.h>
{
0x00 00 SHORT LeftEdge;
0x02 02 SHORT TopEdge;
0x04 04 SHORT Width;
0x06 06 SHORT Height;
0x08 08 SHORT Depth;
0x0A 10 USHORT *ImageData;
0x0E 14 UBYTE PlanePick
0x0F 15 UBYTE PlaneOnOff;
0x10 16 struct Image *NextImage;
0x14 20
};

struct IntuiText <intuition/intuition.h>
{
0x00 00 UBYTE FrontPen;
0x01 01 UBYTE BackPen;
0x02 02 UBYTE DrawMode;
0x04 04 SHORT LeftEdge;
0x06 06 SHORT TopEdge;
0x08 08 struct TextAttr *ITextFont;
0x0C 12 UBYTE *IText;
0x10 16 struct IntuiText *NextText;
0x14 20
};
```

7. Les fonctions de mise en mémoire

AllocRemember

Réserver de la mémoire pour une liste

Syntaxe

```
MemBlock = AllocRemember(RememberKey, Size, Flags);
          DO      -396   A0      D0      D1
CPTR      MemBlock;
struct Remember *RememberKey;
ULONG      Size;
ULONG      Flags;
```

Description

Cette fonction réserve un emplacement en mémoire et le rattache à une liste. Les différents secteurs de mémoire de la liste peuvent être entièrement libérés avec la fonction FreeRemember().

Paramètres

- RememberKey: Adresse de la structure Remember. 0 lors du premier appel.
- Size: Taille de l'emplacement mémoire, exprimée en octets.
- Flags: Attributs de la mémoire.

Retour

- MemBlock: Adresse de l'emplacement en mémoire ou 0.

Voir aussi

FreeRemember(), AllocMem(), FreeMem()

FreeRemember

Libérer la mémoire allouée par AllocRemember

Syntaxe

```
FreeRemember(RememberKey, ReallyForget);
          -408      A0      D0
struct Remember *RememberKey;
BOOL      ReallyForget;
```

Description

Cette fonction permet de libérer la mémoire allouée par la fonction AllocRemember().

Paramètres

- RememberKey: Pointeur sur la structure Remember.

ReallyForget: FALSE pour libérer la structure Remember ou TRUE pour libérer la structure et la mémoire allouée.

Voir aussi

AllocRemember(), AllocMem(), FreeMem()

Structure

```
struct Remember <intuition/intuition.h>
{
 0x00 00 struct Remember *NextRemember;
 0x04 04 ULONG RememberSize;
 0x08 08 UBYTE *Memory;
 0x0C 12
};
```

8. Les fonctions "Refresh"

BeginRefresh

Rafraîchissement d'une fenêtre

Syntaxe	BeginRefresh(Window); -354 A0 struct Window *Window;
----------------	--

Description

Cette fonction est nécessaire pour les fenêtres simple-refresh; elle annonce à Intuition la mise en route d'un renouvellement de la fenêtre. Elle appelle en outre la fonction BeginUpdate(). Pas de valeur en retour.

Paramètres

Window: Adresse de la structure Window.

Voir aussi

EndRefresh()

EndRefresh

Rafraîchissement d'une fenêtre

Syntaxe	EndRefresh(Window, Complete); -366 A0 D0 struct Window *Window; BOOL Complete;
----------------	---

Description

Il faut pouvoir mettre au tracé à neuf d'une fenêtre simple-refresh. C'est à cela que sert cette fonction, qui doit avoir été précédée par BeginRefresh(). Pas de valeur en retour.

Paramètres

Window: Adresse de la structure Window de la fenêtre qui vient d'être redessinée.

Complete: TRUE ou FALSE, selon que le bitmap de la fenêtre a été redessiné en entier ou seulement en partie.

Voir aussi

BeginRefresh()

RemakeDisplay**Refaire les viewports**

Syntaxe RemakeDiskplay()
 -384

Description

Cette fonction refait tous les viewports de l'écran représentés.

Voir aussi

MakeScreen(), RethinkDisplay(), MakeVPort()

RethinkDisplay**Mettre à jour l'écran et le Copper**

Syntaxe RethinkDisplay()
 -390

Description

Cette fonction permet de mettre à jour les relations de l'écran, ainsi que la liste de présentation du Copper.

Voir aussi

RemakeDiskplay(), MakeVPort(), MakeScreen()

9. Les fonctions internes

AlohaWorkbench

Activer la surveillance du Workbench

Syntaxe

```
AlohaWorkbench(WBPort);
    -402      A0
    struct MsgPort *WBPort;
```

Description

Cette fonction permet de mettre en service la surveillance du Workbench pour Intuition. Dès que l'on clique sur le Workbench, on obtient dans le port de messages indiqué le message (class) WBENCHMESSAGE. Pas de valeur en retour.

Paramètres

WBPort: Adresse de la structure MessagePort qui doit contenir le message.

Intuition

Transmettre une liste à Intuition

Syntaxe

```
InputEvent = Intuition(InputEvent);
    DO      -36      A0
    struct InputEvent *InputEvent;
    struct InputEvent *InputEvent;
```

Description

Cette fonction transmet une liste de structures Input-Event à Intuition. D'après la documentation, on ne doit pas utiliser la fonction.

Paramètres

InputEvent: Adresse de la première structure de la liste des événements Input.

Retour

InputEvent: Adresse de la première structure de la liste des événements Input.

*LockIBase

Mettre une protection sur LockIBase

Syntaxe

```
Lock = LockIBase(LockNumber);
    DO      -414      DO
    ULONG Lock;
    ULONG LockNumber;
```

Description

Cette fonction met en place la protection contre l'accès posée par la fonction LockIBase.

Paramètres

LockNumber: Valeur de protection.

Retour

Lock: Valeur pour annuler les effets de cette fonction (en utilisant la fonction UnLockIBase).

Voir aussi

[UnlockIBase\(\)](#), [LockLayerInfo\(\)](#), [ObtainSemaphore\(\)](#)

*UnlockIBase

Enlever la protection de LockIBase

Syntaxe

```
UnlockIBase(Lock);
      -420      A0
      ULONG Lock;
```

Description

Cette fonction enlève la protection contre l'accès posée par la fonction LockIBase. Pas de valeur en retour.

Paramètres

Lock: Valeur fournie par la fonction LockIBase.

Voir aussi

[LockIBase\(\)](#)

ViewAddress

Donner l'adresse d'une structure View

Syntaxe

```
Address = ViewAddress();
      DO      -294
      struct View *Address;
```

Description

Cette fonction communique l'adresse d'une structure View.

Retour

Address: Adresse de la structure View.

ViewPortAddress**Donner l'adresse d'une structure ViewPort****Syntaxe**

```
Address = ViewPortAddress(Window);
      DO          -300      A0
      struct View *Address;
      struct Window *Window;
```

Description

Cette fonction communique l'adresse de la structure d'une fenêtre ViewPort.

Paramètres

Window: Pointeur sur la structure Window pour laquelle vous voulez l'adresse Viewport.

Retour

Address: Adresse de la structure ViewPort.

Structure

```
Display_Modes
DMODECOUNT    0x0002
HIRESPICK     0x0000
LOWRESPICK    0x0001
```

```
System_Gadgets
RESCOUNT       2
HIRESGADGET   0
LOWRESGADGET  1
GADGETCOUNT   8
UPFRONTGADGET 0
DOWNBACKGADGET 1
SIZEGADGET    2
CLOSEGADGET   3
DRAGGADGET    4
SUPFRONTGADGET 5
SDOWNBACKGADGET 6
SDRAGGADGET   7
```

```
Intuition_Locking
NUMILOCKS      7
ISTATELOCK     0
LAYERINFOLOCK 1
GADGETSLOCK   2
LAYERROMLOCK  3
VIEWLOCK       4
IBASELOCK      5
```

RPLOCK

6

```
Interne Intuition-Structure
struct FatIntuiMessage
{
0x00 00 struct IntuiMessage;
0x34 52 ULONG PrevKeys;
0x38 56
};

struct IBox
{
0x00 00 SHORT Left;
0x02 02 SHORT Top;
0x04 04 SHORT Width;
0x06 06 SHORT Height;
0x08 08
};

struct Point
{
0x00 00 SHORT X;
0x02 02 SHORT Y;
0x04 04
};

struct PenPair
{
0x00 00 UBYTE DetailPen;
0x01 01 UBYTE BlockPen;
0x02 02
};

Gadget_Environments
struct GListEnv
{
0x00 00 struct Screen *ge_Screen;
0x04 04 struct Window *ge_Window;
0x08 08 struct Requester *ge_Requester;
0x0C 12 struct RastPort *ge_RastPort;
0x10 16 struct Layer *ge_Layer;
0x14 20 struct Layer *ge_GZZLayer;
0x18 24 struct PenPair ge_Pens;
0x1A 26 struct IBox ge_Domain;
0x22 34 struct IBox ge_GZZdims;
0x2A 42
};

struct GadgetInfo
{
0x00 00 struct GListEnv *gi_Eviron;
0x04 04 struct Gadget *gi_Gadget;
0x08 08 struct IBox gi_Box;
0x10 16 struct IBox gi_Container;
0x18 24 struct Layer *gi_Layer;
0x1C 28 struct IBox gi_NewKnob;
0x24 36
};
Intuition_Base
```

```

struct IntuitionBase <intuition/intuitionbase.h>
{
0x000 00 struct Library LibNode;
0x022 34 struct View ViewLord;
0x036 54 struct Window *ActiveWindow;
0x03A 58 struct Screen *ActiveScreen;
0x03D 62 struct Screen *FirstScreen;
0x042 66 ULONG Flags;
0x046 70 WORD MouseY;
0x048 72 WORD MouseX;
0x04A 74 ULONG Seconds;
0x04D 78 ULONG Micros;
0x052 82 WORD MinXMouse;
0x054 84 WORD MaxXMouse;
0x056 86 WORD MinYMouse;
0x058 88 WORD MaxYMouse;
0x05A 90 ULONG StartSecs;
0x05D 94 ULONG StartMicros;
0x062 98 APTR SysBase;
0x066 102 struct GfxBase *GfxBase;
0x06A 106 APTR LayersBase;
0x06D 110 APTR ConsoleDevice;
0x072 114 USHORT *APointer;
0x076 118 BYTE APtrHeight;
0x077 119 BYTE APtrWidth;
0x078 120 BYTE AXOffset;
0x079 121 BYTE AYOffset;
0x07A 122 USHORT MenuDrawn;
0x07C 124 USHORT MenuSelected;
0x07E 126 USHORT OptionList;
0x080 128 struct RastPort MenuRPort;
0x0E4 228 struct TmpRas MenuTmpRas;
0x0EC 236 struct ClipRect ItemCRect;
0x110 272 struct ClipRect SubCRect;
0x134 308 struct BitMap IBitMap;
0x144 324 struct BitMap SBitMap;
0x154 340 struct IOStdReq InputRequest;
0x17E 382 struct Interrupt InputInterrupt;
0x194 404 struct Remember *EventKey;
0x198 408 struct InputEvent *IEvents;

#define NUM_IEVENTS 4

0x19C 412 SHORT EventCount;
0x19E 414 struct InputEvent IEBuffer[NUM_IEVENTS];
}

0x000 00 struct Gadget *ActiveGadget;
0x000 00 struct PropInfo *ActivePInfo;
0x000 00 struct Image *ActiveImage;
0x000 00 struct GListEnv GadgetEnv;
0x000 00 struct GadgetInfo GadgetInfo;
0x000 00 struct Point KnobOffset;
0x000 00 struct Window *getOKWindow;
0x000 00 struct IntuiMessage *getOKMessage;
0x000 00 USHORT setWExcept;
0x000 00 USHORT GadgetReturn;
0x000 00 USHORT StateReturn;
0x000 00 struct RastPort *RP;

```

```
0x000 00 struct TmpRas ITmpRas;
0x000 00 struct Region *OldClipRegion;
0x000 00 struct Point OldScroll;
0x000 00 struct IBox IFrame;
0x000 00 SHORT hthick;
0x000 00 SHORT vthick;
0x000 00 VOID (*frameChange)();
0x000 00 VOID (*sizeDrag)();
0x000 00 struct Point FirstPt;
0x000 00 struct Point OldPt;
0x000 00 struct Gadget *SysGadgets[RESCOUNT] [GADGETCOUNT];
0x000 00 struct Image *CheckImage[RESCOUNT];
0x000 00 struct Image *AmigaIcon[RESCOUNT];

/* #ifdef OLDPATTERN */
/*   USHORT apattern[3], bpattern[4]; */
/* #else */
0x000 00 USHORT apattern[8], bpattern[4];
/* #endif */

0x000 00 USHORT *IPointer;
0x000 00 BYTE IPtrHeight;
0x000 00 BYTE IPtrWidth;
0x000 00 BYTE IXOffset;
0x000 00 BYTE IYOffset;
0x000 00 LONG DoubleSeconds;
0x000 00 LONG DoubleMicros;
0x000 00 BYTE WBorLeft[DMODECOUNT];
0x000 00 BYTE WBorTop[DMODECOUNT];
0x000 00 BYTE WBorRight[DMODECOUNT];
0x000 00 BYTE WBorBottom[DMODECOUNT];
0x000 00 BYTE BarVBorder[DMODECOUNT];
0x000 00 BYTE BarHBorder[DMODECOUNT];
0x000 00 BYTE MenuVBorder[DMODECOUNT];
0x000 00 BYTE MenuHBorder[DMODECOUNT];
0x000 00 USHORT color0;
0x000 00 USHORT color1;
0x000 00 USHORT color2;
0x000 00 USHORT color3;
0x000 00 USHORT color17;
0x000 00 USHORT color18;
0x000 00 USHORT color19;
0x000 00 struct TextAttr SysFont;
0x000 00 struct Preferences *Preferences;
0x000 00 struct DistantEcho *Echoes;
0x000 00 WORD ViewInitX;
0x000 00 WORD ViewInitY;
0x000 00 SHORT CursorDX;
0x000 00 SHORT CursorDY;
0x000 00 struct KeyMap *KeyMap;
0x000 00 SHORT MouseYMinimum;
0x000 00 SHORT ErrorX;
0x000 00 SHORT ErrorY;
0x000 00 struct timerequest IOExcess;
0x000 00 SHORT HoldMinYMouse;
0x000 00 struct MsgPort *WBPort;
0x000 00 struct MsgPort *iqd_FNKUHDPort;
0x000 00 struct IntuiMessage WBMessag;
0x000 00 struct Screen *HitScreen;
```

```

0x000 00 struct SimpleSprite *SimpleSprite;
0x000 00 struct SimpleSprite *AttachedSSprite;
0x000 00 BOOL           GotSprite1;
0x000 00 struct List    SemaphoreList;
0x000 00 struct SignalSemaphore ISemaphore[NUMILOCKS];
0x000 00 WORD           MaxDisplayHeight;
0x000 00 WORD           MaxDisplayRow;
0x000 00 WORD           MaxDisplayWidth;
0x000 00 ULONG          Reserved[7];
0x000 00

```

10. Fonctions spéciales

CurrentTime

Donner l'heure système

Syntaxe

```

currentTime(Second, Micros);
      -84      A0      A1
      ULONG *Seconds;
      ULONG *Micros;

```

Description

Cette fonction donne l'heure système actuelle en secondes et millisecondes. Valeurs en retour dans les variables pointées.

Paramètres

Seconds: Adresse de la variable pour les secondes.

Micros: Adresse de la variable pour les millisecondes.

DoubleClick

Tester un double clic

Syntaxe

```

Is = DoubleClick(StartSecs, StartMicros, CurrentSecs, CurrentMicros);
      D0      -102      D0      D1      D2      D3
      BOOL Is;
      LONG StartSecs;
      LONG StartMicros;
      LONG CurrentSecs;
      LONG CurrentMicros;

```

Description

Cette fonction effectue un test, pour voir si un double clic a eu lieu sur le bouton gauche de la souris. La décision est fonction des réglages effectués dans Preferences. Les temps de début et de fin peuvent être communiqués à l'aide de la fonction CurrentTime.

Paramètres

StartSecs Premier clic en secondes.

StartMicros: Premier clic en microsecondes.

CurrentSecs: Second clic en secondes.

CurrentsMicros: Second clic en microsecondes.

Retour

Is: Si la durée du double clic est trop longue, on obtient FALSE, sinon TRUE.

Voir aussi

[CurrentTime\(\)](#)

GetDefPrefs

Lire les réglages par défaut de Preferences

Syntaxe

```
Prefs = GetDefPrefs(PrefBuffer, Size);
      DO      -126      A0      D0
      struct Preferences *Prefs;
      struct Preferences *PrefBuffer;
      SHORT           Size;
```

Description

Cette fonction permet de lire les réglages par défaut de Preferences.

Paramètres

PrefBuffer: Adresse du secteur tampon devant être réservé au préalable.

Size: Nombre d'octets devant être copiés.

Retour

Prefs: Adresse du secteur tampon.

Voir aussi

[GetPrefs\(\)](#)

GetPrefs

Lire les réglages Preferences

Syntaxe

```
Prefs = GetPrefs(PrefBuffer, Size);
      DO      -132      A0      D0
      struct Preferences *Prefs;
      struct Preferences *PrefBuffer;
      SHORT           Size;
```

Description

Cette fonction permet de lire les réglages Preferences.

Paramètres

PrefBuffer: Adresse du secteur tampon devant être réservé au préalable.

Size: Nombre d'octets devant être copiés.

Retour

Prefs: Adresse du secteur tampon.

Voir aussi

GetPrefs()

ReportMouse

Activer ou désactiver la souris

Syntaxe

```
ReportMouse(Boolean, Window);
      -234      A0      D0
      BOOL      Boolean;
      struct Window *Window;
```

Description

Cette fonction permet d'activer ou de désactiver la surveillance des coordonnées de la souris dans une fenêtre. Pas de valeur en retour.

Paramètres

Window: Adresse de la structure Window.

Boolean: Valeur de vérité pour l'activation ou la désactivation.

*SetPrefs

Sauvegarder les réglages Preferences

Syntaxe

```
Prefs = SetPrefs(PrefBuffer, Size, Inform);
      D0      -324      A0      D0      D1
      struct Preferences *Prefs;
      struct Preferences *PrefBuffer;
      int      Size;
      BOOL      Inform;
```

Description

Cette fonction permet de sauvegarder les réglages Preferences sur une disquette, ou bien dans le secteur mémoire prévu à cet effet.

Paramètres

- PrefBuffer: Adresse du secteur mémoire qui contient les nouvelles données.
Size: Taille du secteur mémoire.
Inform: 0=modifier les réglages dans la RAM, 1=sauvegarder sur disquette.

Retour

- Prefs: Adresse du secteur mémoire qui contient les nouvelles données.

Voir aussi

`GetDefPrefs()`, `GetPrefs()`

Structure

```
struct Preferences <intuition/intuition.h>
                    <intuition/preferences.h> /* Version 1.3 */
{
    0x00 00 BYTE FontHeight;
    0x01 01 UBYTE PrinterPort;
    0x02 02 USHORT BaudRate;
    0x04 04 struct timeval KeyRptSpeed;
    0x0C 12 struct timeval KeyRptDelay;
    0x14 20 struct timeval DoubleClick;
    0x1C 28 USHORT PointerMatrix[36L];
    0x64 100 BYTE XOffset;
    0x65 101 BYTE YOffset;
    0x66 102 USHORT color17;
    0x68 104 USHORT color18;
    0x6A 106 USHORT color19;
    0x6C 108 USHORT PointerTicks;
    0x6E 110 USHORT color0;
    0x70 112 USHORT color1;
    0x72 114 USHORT color2;
    0x74 116 USHORT color3;
    0x76 118 BYTE ViewXOffset;
    0x77 119 BYTE ViewYOffset;
    0x78 120 WORD ViewInitX;
    0x7A 122 WORD ViewInitY;
    0x7C 124 BOOL EnableCLI;
    0x7E 126 USHORT PrinterType;
    0x80 128 UBYTE PrinterFilename[30L];
    0x9E 158 USHORT PrintPitch;
    0xA0 160 USHORT PrintQuality;
    0xA2 162 USHORT PrintSpacing;
    0xA4 164 UWORLD PrintLeftMargin;
    0xA6 166 UWORLD PrintRightMargin;
    0xA8 168 USHORT PrintImage;
    0xAA 170 USHORT PrintAspect;
```

```
0xAC 172 USHORT PrintShade;
0xAE 174 WORD PrintThreshold;
0xB0 176 USHORT PaperSize;
0xB2 178 UWORLD PaperLength;
0xB4 180 USHORT PaperType;
0xB6 182 UBYTE SerRWBits;
0xB7 183 UBYTE SerStopBuf;
0xB8 184 UBYTE SerParShk;
0xB9 185 UBYTE LaceWB;
0xBA 186 UBYTE WorkName[30L];
0xD8 216 BYTE RowSizeChange;
0xD9 217 BYTE ColumnSizeChange;
0xDA 218 UWORLD PrintFlags;
0xDC 220 UWORLD PrintMaxWidth;
0xDE 222 UWORLD PrintMaxHeight;
0xE0 224 UBYTE PrintDensity;
0xE1 225 UBYTE PrintXOffset;
0xE2 226 UWORLD wb_Width;
0xE4 228 UWORLD wb_Height;
0xE6 230 UBYTE wb_Depth; /* Version 1.3 */
0xE7 231 UBYTE ext_size;
0xE8 232
};

Preferences_FontHeight
TOPAZ_EIGHTY    8L
TOPAZ_SIXTY     9L

Preferences_LaceWB
LACEWB          0x01L

Preferences_PrinterPort
PARALLEL_PRINTER 0x00L
SERIAL_PRINTER   0x01L

Preferences_BaudRate
BAUD_110        0x00L
BAUD_300        0x01L
BAUD_1200       0x02L
BAUD_2400       0x03L
BAUD_4800       0x04L
BAUD_9600       0x05L
BAUD_19200      0x06L
BAUD_MIDI       0x07L

Preferences_PaperType
FANFOLD         0x00L
SINGLE          0x80L

Preferences_PrintPitch
PICA            0x000L
ELITE           0x400L
FINE            0x800L

Preferences_PrintQuality
DRAFT           0x000L
LETTER          0x100L

Preferences_PrintSpacing
```

```
SIX_LPI      0x000L
EIGHT_LPI    0x200L

Preferences_PrintImage
IMAGE_POSITIVE 0x00L
IMAGE_NEGATIVE 0x01L

Preferences_PrintAspect
ASPECT_HORIZ   0x00L
ASPECT_VERT    0x01L

Preferences_PrintShade
SHADE_BW       0x00L
SHADE_GREYSCALE 0x01L
SHADE_COLOR    0x02L

Preferences_PaperSize
US_LETTER     0x00L

US_LEGAL      0x10L
N_TRACTOR     0x20L
W_TRACTOR     0x30L
CUSTOM        0x40L

Preferences_PrinterType
CUSTOM_NAME    0x00L
ALPHA_P_101    0x01L
BROTHER_15XL   0x02L
CBM_MPS1000   0x03L
DIAB_630      0x04L
DIAB_ADV_D25   0x05L
DIAB_C_150    0x06L
EPSON         0x07L
EPSON_JX_80    0x08L
OKIMATE_20    0x09L
QUME_LP_20    0x0AL
HP_LASERJET   0x0BL
HP_LASERJET_PLUS 0x0CL

Preferences_SerialBuffer
SBUF_512      0x00L
SBUF_1024     0x01L
SBUF_2048     0x02L
SBUF_4096     0x03L
SBUF_8000     0x04L
SBUF_16000    0x05L

Preferences_SerRWBits
SREAD_BITS   0xF0L
SWRITE_BITS  0x0FL

Preferences_SerStopBuf
SSTOP_BITS   0xF0L
SBUFSIZE_BITS 0x0FL

Preferences_SerParShk
SPARITY_BITS  0xF0L
SPARITY_NONE   0L
```

```
SPARITY_EVEN 1L
SPARITY_ODD 2L
SHSHAKE_XON 0L
SHSHAKE_RTS 1L
SHSHAKE_NONE 2L
```

La structure Intuition

```
struct IntuiMessage <intuition/intuition.h>
{
0x00 00 struct Message ExecMessage;
0x14 20 ULONG Class;
0x18 24 USHORT Code;
0x1A 26 USHORT Qualifier;
0x1C 28 APTR IAddress;
0x20 32 SHORT MouseX;
0x22 34 SHORT MouseY;
0x24 36 ULONG Seconds;
0x28 40 ULONG Micros;
0x2C 44 struct Window *IDCMPWindow;
0x30 48 struct IntuiMessage *SpecialLink;
0x34 52
};

IntuiMessage_IDCMPFlags
SIZEVERIFY      0x00000001L
NEWSIZE         0x00000002L
REFRESHWINDOW   0x00000004L
MOUSEBUTTONS    0x00000008L
MOUSEMOVE       0x00000010L
GADGETDOWN     0x00000020L
GADGETUP       0x00000040L
REQSET          0x00000080L
MENUPICK        0x00000100L
CLOSEWINDOW    0x00000200L
RAWKEY          0x00000400L
REQVERIFY       0x00000800L
REQCLEAR        0x00001000L
MENUVERIFY      0x00002000L
NEWPREFS        0x00004000L
DISKINSERTED    0x00008000L
DISKREMOVED     0x00010000L
WBENCHMESSAGE   0x00020000L
ACTIVEWINDOW    0x00040000L
INACTIVEWINDOW  0x00080000L
DELTAMOVE       0x00100000L
VANILLAKEY     0x00200000L
INTUITICKS      0x00400000L
LONELYMESSAGE   0x80000000L

Menu_Flags
MENUHOT         0x0001L
MENUCANCEL      0x0002L
MENUWAITING    0x0003L
OKOK            MENUHOT
OKABORT         0x0004L
OKCANCEL        MENUCANCEL

Workbench_Flags
```

```
WBENCHOPEN    0x0001L
WBENCHCLOSE   0x0002L
Intuition_Macros
Menu_Macros
MENUNUM(n)
ITEMNUM(n)   ((n >> 5) & 0x003F)
SUBNUM(n)    ((n >> 11) & 0x001F)
SHIFTMENU(n) (n & 0x1F)
SHIFTITEM(n) ((n & 0x3F) << 5)
SHIFTSUB(n)  ((n & 0x1F) << 11)

Serial_Macros
SRBNUM(n)   (0x08 (n >> 4))
SWBNUM(n)   (0x08 (n & 0x0F))
SSBNUM(n)   (0x01 + (n >> 4)).
SPARNUM(n)  (n >> 4)
SHAKNUM(n)  (n & 0x0F)

Preferences_Definition
NOMENU     0x001FL
NOITEM     0x003FL
NOSUB      0x001FL
MENUNull   0xFFFFL
FOREVER for(:)
SIGN(x)    ( ((x) > 0) ((x) < 0) )
NOT        !
CHECKWIDTH 19L
COMMWIDTH  27L
LOWCHECKWIDTH 13L
LOWCOMMWIDTH 16L

ALERT_TYPE   0x80000000L
RECOVERY_ALERT 0x00000000L
DEADEND_ALERT 0x80000000L

AUTOFRONTPEN 0L
AUTOBACKPEN  1L
AUTODRAWMODE JAM2
AUTOLEFTEDGE 6L
AUTOTOPEDGE  3L
AUTOITEXTFONT Null
AUTONEXTTEXT Null

SELECTUP      (IECODE_LBUTTON | IEPCODE_UP_PREFIX)
SELECTDOWN    (IECODE_LBUTTON)
MENUUP       (IECODE_RBUTTON | IEPCODE_UP_PREFIX)
MENUDOWN     (IECODE_RBUTTON)

ALTLEFT       (IEQUALIFIER_LALT)
ALTRIGHT      (IEQUALIFIER_RALT)
AmigaLEFT    (IEQUALIFIER_LCOMMAND)
AmigaRIGHT   (IEQUALIFIER_RCOMMAND)
AmigaKEYS    (AmigaLEFT | AmigaRIGHT)

CURSORUP      0x4CL
CURSORLEFT    0x4FL
CURSORRIGHT   0x4EL
CURSORDOWN   0x4DL
KEYCODE_Q    0x10L
```

KEYCODE_X	0x32L
KEYCODE_N	0x36L
KEYCODE_M	0x37L
KEYCODE_V	0x34L
KEYCODE_B	0x35L

6.4. La librairie Layers

Cette fonction permet la gestion des éléments graphiques nommés Layers. L'ouverture de cette fonction se fait de la manière suivante:

```
long *LayersBase;
.
LayersBase = OpenLibrary ("layers.library",0);
```

1. La création des Layers

CreateBehindLayer	878
CreateUpfrontLayer	878
FattenLayerInfo	879
InitLayers	880
NewLayerInfo()	880

2. Le travail des Layers

BeginUpdate	881
BehindLayer	882
EndUpdate	884
InstallClipRegion	884
LockLayer	885
LockLayerInfo	885
LockLayers	885
MoveLayer	886
MoveLayerInFrontOf	886
ScrollLayer	887
SizeLayer	887
SwapBitsRastPortClipRect	888
UpfrontLayer	888
WhichLayer	889

3. Définition des Layers

DeleteLayer	890
DisposeLayerInfo	890
ThinLayerInfo	891

UnlockLayer	891
UnlockLayerInfo	892
UnlockLayers	891

1. La création des Layers

CreateBehindLayer

Créer un nouveau Layer

Syntaxe

```
Layer = CreateBehindLayer (LayerInfo, Bitmap, x1,y1,x2, y2,Flags [.Super-Bitmap])
    D0          -42           A0          A1      D0 D1 D2 D3 D4      [ A2 ]
    struct Layer *Layer;
    struct Layer_Info *LayerInfo;
    struct Bitmap *Bitmap;
    LONG x1,y1,
    x2,y2;
    LONG Flags;
    [struct Bitmap *Super-Bitmap;]
```

Description

Cette fonction permet de créer un nouveau layer et de le placer derrière tous les autres layers.

Paramètres

LayerInfo: Adresse de la structure LayerInfo.
 Bitmap: Adresse de la structure BitMap.
 x1,y1 Coordonnées du coin supérieur gauche.
 x2,y2: Coordonnées du coin inférieur droit.
 Flags: Type du layer.
 Super-Bitmap: Adresse optionnelle d'un superbitmap.

Retour

Layer Adresse de la structure Layer.

Voir aussi

[CreateUpfrontLayer\(\)](#)

CreateUpfrontLayer

Créer un nouveau Layer

Syntaxe

```
Layer = CreateUpfrontLayer (LayerInfo, Bitmap, x1,y1,x2,y2,Flags [.Super-Bitmap])
    D0          -36           A0          A1      D0 D1 D2 D3 D4      [ A2 ]
    struct Layer *Layer;
    struct Layer_Info *LayerInfo;
```

```

struct Bitmap      *Bitmap;
LONG                x1,y1,
                    x2,y2;
LONG                Flags;
struct Bitmap      *Super-Bitmap;]

```

Description

Cette fonction permet de créer un nouveau layer et de le placer devant tous les autres layers.

Paramètres

LayerInfo:	Adresse de la structure LayerInfo.
Bitmap:	Adresse de la structure BitMap.
x1,y1	Coordonnées du coin supérieur gauche.
x2,y2:	Coordonnées du coin inférieur droit.
Flags:	Type du layer.
Super-Bitmap:	Adresse optionnelle d'un superbitmap.

Retour

Layer	Adresse de la structure Layer.
-------	--------------------------------

FattenLayerInfo

Réserver un emplacement mémoire

Syntaxe	FattenLayerInfo (LayerInfo); -156 A0 struct Layer_Info *LayerInfo;
----------------	--

Description

Cette fonction permet de réserver un emplacement supplémentaire en mémoire avec la fonction FattenLayerInfo pour une structure LayerInfo lorsque la fonction InitLayer() a été utilisée. Pas de valeur en retour.

Paramètres

LayerInfo:	Adresse de la structure LayerInfo.
------------	------------------------------------

Voir aussi

InitLayers(), ThinLayerInfo()

InitLayers**Mettre en place une structure LayerInfo****Syntaxe**

```
InitLayers(LayerInfo);
    -30      A0
    struct Layer_Info *LayerInfo;
```

Description

Cette fonction permet de mettre en place une structure LayerInfo. C'est une fonction de l'ancien système d'exploitation V1.0. La place en mémoire pour la structure doit être réservée avec la fonction AllocMem(). Pas de valeur en retour.

Paramètres

LayerInfo: Adresse de la structure LayerInfo.

Voir aussi

FattenLayerInfo(), ThinLayerInfo()

NewLayerInfo()**Réserver et activer une structure LayerInfo****Syntaxe**

```
LayerInfo = NewLayerInfo()
    D0      -144
    struct Layer_Info *LayerInfo;
```

Description

Cette fonction réserve un emplacement en mémoire et initialise une structure LayerInfo.

Paramètres

Aucun.

Retour

LayerInfo: Adresse de la nouvelle structure LayerInfo.

Structure

Offsets	Structure
-----	-----
	struct Layer_Info <graphics/layers.h>
	{
0x00 0	struct Layer *top_layer;
0x04 4	struct Layer *check_lp,
	*obs;
0x08 8	
0x0c 12	struct MinList FreeClipRects;
0x18 24	struct SignalSemaphore Lock;
0x46 70	struct List gs_Head;

```

0x54 84      LONG longreserved;
0x58 88      UWORLD Flags;
0x5a 90      BYTE fatten_count;
0x5b 91      BYTE LockLayersCount;
0x5c 92      UWORLD LayerInfo_extra_size;
0x5e 94      WORD *bltbuff;
0x62 98      struct LayerInfo_extra *LayerInfo_extra;
}
struct Layer <graphics/clip.h>
{
0x00 0      struct Layer *front,
0x04 4      *back;
0x08 8      struct ClipRect *ClipRect;
0x0c 12     struct RastPort *rp;
0x10 16     struct Rectangle bounds;
0x18 24     UBYTE reserved[4];
0x1c 28     UWORLD priority;
0x1e 30     UWORLD Flags;
0x20 32     struct Bitmap *Super-Bitmap;
0x22 34     struct ClipRect *SuperClipRect;
0x26 38     APTR Window;
0x2a 42     SHORT Scroll_X,
0x2c 44     Scroll_Y;
0x30 48     struct ClipRect *cr,
0x34 52     *cr2,
0x38 56     *crnew;
0x3c 60     struct ClipRect *SuperSaveClipRects;
0x40 64     struct ClipRect *_cliprects;
0x44 68     struct Layer_Info *LayerInfo;
0x48 72     struct SignalSemaphore Lock;
0x76 118    UBYTE reserved3[8];
0x7e 126    struct Region *ClipRegion;
0x82 130    struct Region *saveClipRects;
0x86 134    UBYTE reserved2[22];
0x9c 156    struct Region *DamageList;
}

```

2. Le travail des Layers

BeginUpdate

Modifier un Layer

Syntaxe

```
Status = BeginUpdate (Layer)
D0           -78          A0
```

Description

Cette fonction permet de préparer le système pour la modification d'un layer:

```

struct Region *OldDamageList;
..
OldDamage-List = Layer->DamageList;
struct Region *Region;
..
Layer->DamageList = Region;
```

Paramètres

Layer: Adresse de la structure Layer.

Retour

Status: TRUE si OK ou FALSE si pas de modification (manque de mémoire, etc).

Voir aussi

EndUpdate()

BehindLayer**Placer un Layer derrière les autres****Syntaxe**

```
Status = BehindLayer (dummy, Layer)
      DO          -54      A0      A1
      BOOL        Status;
      LONG        dummy;
      struct Layer *Layer;
```

Description

Cette fonction permet de placer le Layer derrière tous les autres.

Paramètres

dummy: Adresse de la structure LayerInfo.

Layer: Adresse de la structure Layer.

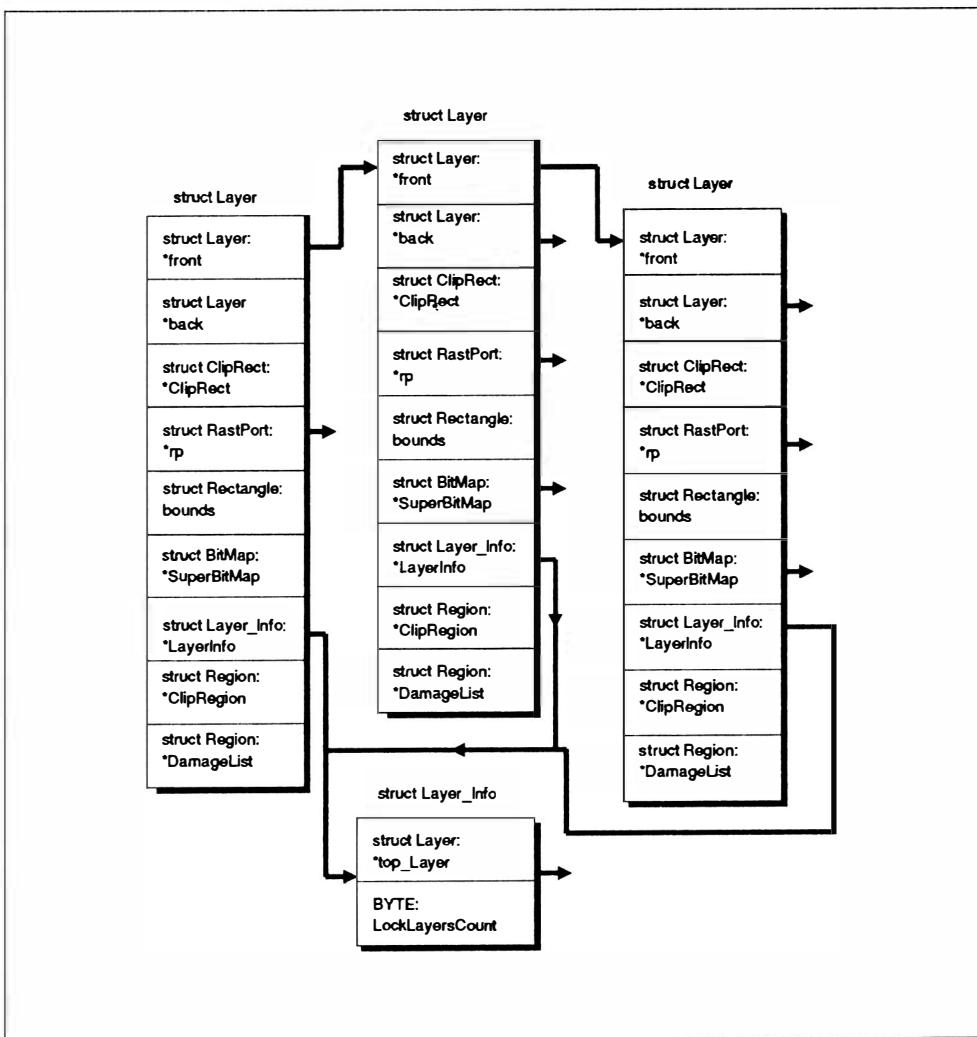


Figure 6 - 59

Retour

Status: TRUE si OK ou FALSE si impossible (manque de mémoire, etc).

Voir aussi

`UpfrontLayer()`

EndUpdate**Mettre fin au tracé d'un Layer****Syntaxe**

```
EndUpdate (Layer, Flag);
    -84      A0      D0
    struct Layer *Layer;
    BOOL      Flag;
```

Description

Cette fonction met fin au retracage d'un layer simple-refresh. Elle est également appelée par EndRefresh(). Pas de valeur en retour.

Paramètres

Layer: Adresse de la structure Layer.

Flag: Flags de layer.

Voir aussi

[BeginUpdate\(\)](#)

InstallClipRegion**Définir un rectangle de clipping****Syntaxe**

```
OldClipRegion = InstallClipRegion (Layer, Region)
    D0      -174      A0      A1
    struct Layer *Layer;
    struct Region *Region;
```

Description

Cette fonction permet de définir un rectangle de clipping à l'intérieur d'un layer. Tous les affichages graphiques sur ce layer ne seront plus visibles qu'à l'intérieur de ce rectangle de clipping.

Paramètres

Layer: Adresse de la structure Layer.

Region: Adresse de la structure Region contenant les coordonnées du rectangle de clipping.

Retour

OldClipRegion: Adresse de la dernière structure Region ou 0.

Voir aussi

[BeginUpdate\(\)](#), [EndUpdate\(\)](#)

LockLayer**Interdir l'accès à un Layer****Syntaxe**

```
LockLayer (dummy, Layer);
-96      A0    A1
LONG      dummy;
struct Layer *Layer;
```

Description

Cette fonction interdit à d'autres programmes l'accès à un layer. Pas de valeur en retour.

Paramètres

dummy: Adresse de la structure LayerInfo.

Layer: Adresse de la structure Layer.

Voir aussi

[LockLayerInfo\(\)](#), [LockLayers\(\)](#)

LockLayerInfo**Interdir l'accès à la structure LayerInfo****Syntaxe**

```
LockLayerInfo (LayerInfo);
-120      A0
struct Layer_Info *LayerInfo;
```

Description

Cette fonction interdit à d'autres programmes l'accès à la structure LayerInfo indiquée en paramètre. Pas de valeur en retour.

Paramètres

LayerInfo: Adresse de la structure LayerInfo.

Voir aussi

[UnlockLayerInfo\(\)](#)

LockLayers**Interdire l'accès à tous les Layers****Syntaxe**

```
LockLayers (LayerInfo);
-108      A0
struct Layer_Info *LayerInfo;
```

Description

Cette fonction interdit à d'autres programmes l'accès à tous les layers. Pas de valeur en retour.

Paramètres

LayerInfo: Adresse de la structure LayerInfo.

Voir aussi

[LockLayer\(\)](#)

MoveLayer

Déplacer des Layers

Syntaxe

```
Status = MoveLayer (dummy, Layer, DeltaX, DeltaY);
      D0      -60      A0      A1      D0      D1
      BOOL      Status;
      LONG      dummy;
      struct Layer *Layer;
      LONG      DeltaX,
      DeltaY;
```

Description

Cette fonction permet de déplacer les layers normaux. Pas de valeur en retour.

Paramètres

dummy: Adresse de la structure LayerInfo.

Layer: Adresse de la structure Layer.

DeltaX,DeltaY: Nouvelles coordonnées du layer, relatives par rapport aux anciennes.

Retour

Status: TRUE si OK ou FALSE si impossible (manque de mémoire, etc...)

MoveLayerInFrontOf

Déplacer des Layers

Syntaxe

```
Status = MoveLayerInFrontOf (Layer1, Layer2);
      D0      -168      A0      A1
      BOOL      Status;
      struct Layer *Layer1,
      *Layer2;
```

Description

Cette fonction permet de déplacer à volonté des layers.

Paramètres

Layer1: Adresse du Layer à déplacer.

Layer2: Adresse du Layer devant lequel le précédent doit être affiché.

Retour

Status: TRUE si OK ou FALSE si impossible (manque de mémoire, etc...)

ScrollLayer

Déplacer le contenu d'un Layer

Syntaxe

```
ScrollLayer (dummy, Layer, DeltaX, DeltaY);
    -72      A0      A1      D0      D0
    LONG      dummy;
    struct Layer *Layer;
    LONG      DeltaX,
    DeltaY;
```

Description

Cette fonction permet de déplacer le contenu d'un layer superbitmap. Pas de valeur en retour.

Paramètres

dummy: Adresse de la structure LayerInfo.

Layer: Adresse de la structure Layer.

DeltaX,DeltaY: Nouvelles coordonnées du contenu, relatives par rapport aux anciennes.

SizeLayer

Modifier la taille d'un Layer

Syntaxe

```
Status = SizeLayer (dummy, Layer, DeltaX, DeltaY);
    DO      -66      a0      A1      D0      D1
    BOOL      Status;
    LONG      dummy;
    struct Layer *Layer;
    LONG      DeltaX,
    DeltaY;
```

Description

Cette fonction permet d'agrandir ou de réduire un layer..

Paramètres

- dummy: Adresse de la structure LayerInfo.
 Layer: Adresse de la structure Layer.
 DeltaX,DeltaY: Nouvelles coordonnées du layer, relatives par rapport aux anciennes.

Retour

- Status: TRUE si OK ou FALSE si impossible (manque de mémoire).

SwapBitsRastPortClipRect**Copier le contenu d'un rasport****Syntaxe**

```
SwapBitsRastPortClipRect (RastPort, ClipRect);
    -126           A0      A1
    struct RastPort *RastPort;
    struct ClipRect *ClipRect;
```

Description

Cette fonction permet de copier le contenu d'un rasport dans le BitPlane d'un rectangle de clipping. Pas de valeur en retour.

Paramètres

- RastPort: Adresse de la structure RastPort.
 ClipRect: Adresse de la structure ClipRect.

UpfrontLayer**Placer un Layer devant tous les autres****Syntaxe**

```
Status = UpfrontLayer (dummy, Layer);
    D0      -48      A0      A1
    BOOL      Status;
    LONG      dummy;
    struct Layer *Layer;
```

Description

Cette fonction permet de placer devant tous les autres layers celui indiqué.

Paramètres

dummy: Adresse de la structure LayerInfo.

Layer: Adresse de la structure Layer.

Retour

Status: TRUE si OK ou FALSE si impossible (manque de mémoire, etc).

Voir aussi

BehindLayer()

WhichLayer**Donner l'adresse d'un Layer****Syntaxe**

```
Layer = WhichLayer (LayerInfo, x, y);
      D0      -132      A0      D0  D1
      struct Layer      *Layer;
      struct Layer_Info *LayerInfo;
      SHORT            x,y;
```

Description

Cette fonction communique l'adresse du layer, auquel les coordonnées pourraient convenir.

Paramètres

LayerInfo: Adresse de la structure LayerInfo.

x, y: Coordonnées à l'intérieur du layer.

Retour

Layer: Adresse de la structure Layer ou 0.

Structure

Offsets	Structure
-----	-----
0x00	struct ClipRect <graphics/clip.h>
0x04	{
0x08	struct ClipRect *Next;
0x0c	struct ClipRect *prev;
0x10	struct Layer *lobs;
0x14	struct Bitmap *Bitmap;
0x18	struct Rectangle bounds;
0x1c	struct ClipRect *_p1,
0x20	*_p2;
0x24	LONG reserved;
	#ifdef NEWCLIPRECTS_1_1

```
0x24    36      LONG Flags;
        #endif
    }
```

3. Définition des Layers

DeleteLayer

Effacer un Layer dans une liste

Syntaxe

```
Status = DeleteLayer (dummy, Layer);
DO          -90      A0      A1
BOOL        Status;
LONG        dummy;
struct Layer *Layer;
```

Description

Cette fonction permet d'effacer un layer dans une liste de layers.

Paramètres

dummy: Adresse de la structure LayerInfo.

Layer: Adresse de la structure Layer.

Retour

Status: TRUE si OK ou FALSE si impossible (manque de mémoire, etc).

Voir aussi

[DisposeLayerInfo\(\)](#)

DisposeLayerInfo

Effacer une structure LayerInfo

Syntaxe

```
DisposeLayerInfo (LayerInfo)
                  -150      A0
struct Layer_Info *LayerInfo;
```

Description

Cette fonction permet de libérer la place en mémoire pour la structure LayerInfo. Pas de valeur en retour.

Paramètres

LayerInfo: Adresse de la structure LayerInfo.

ThinLayerInfo**Effacer une structure LayerInfo****Syntaxe**

```
ThinLayerInfo (LayerInfo);
-162          A0
struct Layer_Info *LayerInfo;
```

Description

Cette fonction efface le secteur mémoire supplémentaire d'une structure LayerInfo, mise en place par la fonction FattenLayerInfo(). Pas de valeur en retour.

Paramètres

LayerInfo: Adresse de la structure LayerInfo.

Voir aussi

FattenLayerInfo(), InitLayers()

UnlockLayer**Libérer l'accès à un Layer****Syntaxe**

```
UnlockLayer (Layer);
-102          A0
struct Layer *Layer;
```

Description

Cette fonction libère à nouveau l'accès à un layer. Pas de valeur en retour.

Paramètres

Layer: Adresse de la structure Layer.

Voir aussi

LockLayer()

UnlockLayers**Libérer l'accès aux Layers****Syntaxe**

```
UnlockLayers (LayerInfo);
-114          A0
struct Layer_Info *LayerInfo;
```

Description

Cette fonction libère à nouveau l'accès aux layers. Pas de valeur en retour.

Paramètres

LayerInfo: Adresse de la structure Layer.

Voir aussi

[LockLayers\(\)](#)

UnlockLayerInfo**Libérer l'accès à la structure LayerInfo**

Syntaxe

```
UnlockLayerInfo (LayerInfo)
    -128          A0
    struct Layer_Info *LayerInfo;
```

Description

Cette fonction libère à nouveau l'accès à une structure LayerInfo. Pas de valeur en retour.

Paramètres

LayerInfo: Adresse de la structure LayerInfo.

Voir aussi

[LockLayerInfo](#)

6.5. La librairie Icon

Les fonctions de la librairie Icon permettent d'élaborer les applications qui affichent des icônes et qui ouvrent des fenêtres sur l'écran du Workbench.

Fonctions de la librairie Icon:**1. Fonctions pour les objets Workbench**

AllocWBObject	893
FreeWBObject	893
GetWBObject	894
PutWBObject	894

2. Fonctions pour les Icônes

GetIcon	896
PutIcon	896

3. Fonctions pour les objets du Disque

FreeDiskObject	897
GetDiskObject	898
PutDiskObject	898

4. Fonctions de gestion de blocs mémoire

AddFreeList	899
FreeFreeList	900

5. Autres fonctions

BumpRevision	901
FindToolType	902
MatchToolValue	903

1. Fonctions pour les objets Workbench

AllocWBObject

Réserver de la mémoire pour le Workbench

Syntaxe

```
object = AllocWBObject()
      D0           -66
      struct WBObject *object;
```

Description

Cette fonction alloue l'espace nécessaire en mémoire pour les fichiers d'information Workbench.

Retour

object: adresse de la structure WBObject ou 0.

Voir aussi

AllocEntry, FreeEntry, FreeWBObject

FreeWBObject

Supprimer les entrées d'une structure WBObject

Syntaxe

```
FreeWBObject(object)
      -60          A0
      struct WBObject *object;
```

Description

Cette fonction supprime toutes les entrées d'une structure WBObject et la structure elle-même. Pas de valeur en retour.

Paramètres

object: Adresse de la structure WBObject.

Voir aussi

AllocEntry, FreeEntry, AllocWBObject.

GetWBObject lire un fichier info d'un objet Workbench du disque

Syntaxe

```
object = GetWBObject(name)
      D0      -30     A0
      struct WBObject *object;
      UBYTE *name;
```

Description

Cette fonction lit un fichier info d'un objet Workbench du disque vers la mémoire.

Paramètres

name: Un pointeur sur le nom du fichier dont il faut lire le fichier info.

Retour

object: En cas de succès, un pointeur est renvoyé sur la structure Workbench. Sinon la valeur NULL est retournée.

Voir aussi

PutWBObject

PutWBObject Ecrire un fichier info d'un objet Workbench

Syntaxe

```
ok = PutWBObject(name, object)
      D0 -36     A0     A1
      BOOL ok;
      UBYTE *name;
      struct WBObject *object;
```

Description

Cette fonction écrit un fichier info d'un objet Workbench de la mémoire sur le disque.

Paramètres

- name: Un pointeur sur le nom du fichier qui sera affecté au fichier info de l'objet Workbench.
- object: Un pointeur sur la structure WBObject devant être incluse dans le fichier lors de son écriture sur disque.

Retour

- ok: En cas d'échec, la fonction renvoie une valeur nulle. Vous pouvez utiliser la routine système IoErr pour déterminer la cause de l'erreur.

Voir aussi

[GetWBObject](#)

Structure

```
struct WBObject
{
Offset:
0x00 0 struct Node      wo_MasterNode;
0x0E 14 struct Node     wo_Siblings;
0x1C 28 struct Node     wo_SelectNode;
0x2A 42 struct Node     wo_UtilityNode;
0x38 56 struct WBObject *wo_Parent;
0x3C 60 UBYTE           wo_Flags;
0x3D 61 UBYTE           wo_Type;
0x3E 62 USHORT          wo_UseCount;
0x40 64 char             *wo_Name;
0x44 68 SHORT            wo_NameXOffset;
0x46 70 SHORT            wo_NameYOffset;
0x48 72 char             *wo_DefaultTool;
0x4C 76 struct DrawerData *wo_DrawerData;
0x50 80 struct Window     *wo_IconWin;
0x54 84 LONG              wo_CurrentX;
0x58 88 LONG              wo_CurrentY;
0x5C 92 char             **wo_ToolTypes;
0x60 96 struct Gadget     *wo_Gadget;
0x64 100 struct FreeList   *wo_FreeList;
0x68 104 char             *wo_ToolWindow;
0x6C 108 LONG              wo_StackSize;
0x70 112 LONG              wo_Lock;
sizeof(struct WBObject) -
0x74 116
}
struct DrawerData <workbench/workbench.h>
{
Offset:
0x00 0 struct NewWindow   dd_NewWindow;
0x30 48 LONG               dd_CurrentX;
0x34 52 LONG               dd_CurrentY;
sizeof(struct DrawerData) -
0x38 56
};
```

2. Fonction pour les icônes

GetIcon

Lire en mémoire le fichier info d'un objet

Syntaxe

```
ok = GetIcon(name,icon,free)
      D0 -42 A0   A1   A2
      BOOL ok;
      UBYTE *name;
      struct DiskObject *icon;
      struct FreeList *free;
```

Description

Cette fonction lit en mémoire le fichier info d'un objet disque.

Paramètres

- name: Un pointeur sur le nom de fichier info de l'objet disque que vous voulez charger.
- icon: Un pointeur sur la structure DiskObject.
- free: Un pointeur sur une structure FreeList.

Retour

- ok: En cas d'échec, la valeur NULL est renournée. Vous pouvez utiliser la routine système IoErr pour déterminer la cause de l'erreur.

Voir aussi

PutIcon

PutIcon

Ecrire un fichier info d'un objet

Syntaxe

```
ok = PutIcon(name,icon)
      D0 -48 A0   A1
      BOOL ok;
      UBYTE *name;
      struct DiskObject *icon;
```

Description

Cette fonction écrit un fichier info d'un objet disque depuis la mémoire vers le disque.

Paramètres

- name: Un pointeur sur le nom de fichier devant être affecté au fichier info de l'objet disque.

icon: Un pointeur sur la structure DiskObject devant être incluse dans le fichier lors de son écriture sur disque.

Retour

ok: En cas d'échec, la valeur renournée est nulle. Vous pouvez utiliser la routine système IoErr pour déterminer la cause de l'erreur.

Voir aussi

GetIcon

Structure

```
struct DiskObject <workbench/workbench.h>
{
    Offset:
    0x00 0  UWORLD           do_Magic;
    WB_DISKMAGIC      0xe310
    Offset:
    0x02 2  UWORLD           do_Version;
    WB_DISKVERSION     1

    Offset:
    0x04 4   struct Gadget      do_Gadget;
    0x30 48  UWORLD           do_Type;
    0x32 50   char            *do_DefaultTool;
    0x36 54   char            **do_ToolTypes;
    Offset:
    0x3A 58   LONG            do_CurrentX;
    0x3E 62   LONG            do_CurrentY;
    NO_ICON_POSITION (0x80000000)
    Offset:
    0x42 66   struct DrawerData *do_DrawerData;
    0x46 70   char            *do_ToolWindow;
    0x4A 74   LONG            do_StackSize;
    sizeof(struct DiskObject) -
    0x4E 78
};
```

3. Fonctions pour les objets du disque

FreeDiskObject Libérer toute la mémoire associée à un fichier info

Syntaxe	FreeDiskObject(object) -90 A0 struct DiskObject *object;
----------------	--

Description

Cette fonction permet de libérer toute la mémoire associée à un fichier info d'un objet disque ainsi que l'objet lui-même.

Paramètres

object: Un pointeur sur une structure DiskObject.

Voir aussi

[GetDiskObject](#)

GetDiskObject**lire un fichier info d'un objet disque****Syntaxe**

```
object = GetDiskObject(name)
          D0 -78      A0
          struct DiskObject *object;
          UBYTE *name;
```

Description

Cette fonction lit un fichier info d'un objet disque depuis le disque vers la mémoire.

Paramètres

name: Un pointeur sur le nom de l'objet

Retour

object: Pointeur de retour sur la structure DiskObject. Vous pouvez utiliser la routine système IoErr pour déterminer la cause de l'erreur.

Voir aussi

[PutDiskObject](#)

PutDiskObject**Ecrire le fichier info d'un objet disque****Syntaxe**

```
ok = PutDiskObject(name,object)
          D0 -84      A0      A1
          BOOL ok;
          UBYTE *name;
          struct DiskObject *object;
```

Description

Cette fonction écrit le fichier info d'un objet disque depuis la mémoire sur le disque.

Paramètres

name: Un pointeur sur le nom du fichier qui sera affecté au fichier info

object: Un pointeur sur une structure DiskObject qui sera incluse dans le fichier info quand il sera écrit sur disque.

Retour

ok: La valeur renvoyée est non nulle en cas de succès, nulle en cas d'échec. Vous pouvez utiliser la routine système IoErr pour déterminer la cause de l'erreur.

Voir aussi

[GetDiskObject](#)

Structure

```
struct DiskObject <workbench/workbench.h>
{
    Offset:           do_Magic;
    0x00 0  UWORLD      WB_DISKMAGIC      0xe310

    Offset:           do_Version;
    0x02 2  UWORLD      WB_DISKVERSION     1
    Offset:
    0x04 4  struct Gadget      do_Gadget;
    0x30 48 UWORLD          do_Type;
    0x32 50 char            *do_DefaultTool;
    0x36 54 char            **do_ToolTypes;
    Offset:
    0x3A 58 LONG            do_CurrentX;
    0x3E 62 LONG            do_CurrentY;
    NO_ICON_POSITION (0x80000000)
    Offset:
    0x42 66 struct DrawerData *do_DrawerData;
    0x46 70 char            *do_ToolWindow;
    0x4A 74 LONG            do_StackSize;
    sizeof(struct DiskObject) -
    0x4E 78
};
```

4. Fonctions de gestion de blocs mémoire

AddFreeList

Ajouter un bloc mémoire spécifié à la liste libre

Syntaxe

```
ok = AddFreeList(free,mem,taille)
      D0 -72   A0   A1   A2
      BOOL ok;
      struct FreeList *free;
      UBYTE *mem;
      ULONG taille;
```

Description

Cette fonction ajoute un bloc mémoire spécifié à la liste libre. AddFreeList n'alloue pas la mémoire demandée. Elle ne fait qu'enregistrer la mémoire dans la liste libre.

Paramètres

- free: Un pointeur sur une structure FreeList.
 mem: Un pointeur sur la base du bloc mémoire à ajouter (premier octet du bloc).
 taille: La taille de la mémoire à enregistrer.

Retour

- ok: En cas d'échec, la valeur renvoyée est nulle. Vous pouvez utiliser la routine système IoErr pour déterminer la cause de l'erreur.

Voir aussi

AllocEntry, FreeEntry, FreeFreeList

FreeFreeList

Libérer toute la mémoire occupée par toutes les structures

Syntaxe	FreeFreeList(free) -54 A0 struct FreeList *free;
----------------	--

Description

Cette fonction libère toute la mémoire occupée par toutes les structures reliées à la structure FreeList ainsi que la structure Freelist elle-même.

Paramètres

- free: Un pointeur sur la structure FreeList.

Voir aussi

AllocEntry, FreeEntry, AddFreeList

Structure

```
struct FreeList <workbench/workbench.h>
{
  Offset:
  0x00 0 WORD           fl_NumFree;
  0x02 2 struct List   fl_MemList;
  sizeof(struct FreeList) =
```

```

0x10 16
};

struct MemList <exec/memory.h>
{
Offset:
0x00 0 struct Node      ml_Node;
0x0E 14 UWORLD          ml_NumEntries;
0x10 16 struct MemEntry ml_ME[1];
sizeof(struct MemList) -
0x18 24
};
struct MemEntry <exec/memory.h>
{
Offset:
0x00 0 union
{
0x00 0 ULONG   meu_Req;
0x00 0 APTR    meu_Addr;
} me_Un;
(meu_Req);
MEMF_PUBLIC (1<<0)
MEMF_CHIP  (1<<1)
MEMF_FAST   (1<<2)
MEMF_CLEAR   (1<<16)
MEMF_LARGEST (1<<17)
Offset:
0x04 4 ULONG   me_Length;
sizeof(struct MemEntry) -
0x08 8
};

```

5. Autres fonctions

BumpRevision

Prendre un "nom" et le transformer en 'copy of "nom"'

Syntaxe

```

result = BumpRevision(newbuf,oldname)
      DO           -108      A0      A1
UBYTE *result;
UBYTE *newbuf,oldname;
```

Description

Cette fonction prend un "nom" et le transforme en 'copy of "nom"'". Elle gère directement les copies des copies. La routine va tronquer le nom pour le rendre conforme au DOS (30 caractères actuellement).

Paramètres

newbuf: Le nouveau buffer destiné à recevoir le nouveau nom. Sa taille doit être supérieure à 31 caractères

oldname: Le nom original.

Retour

result: Un pointeur sur newbuf.

Exemple

```
oldname          newbuf
"Test"           "copy of Test"
"copy of Test"   "copy 2 of Test"
"copy 2 of Test" "copy 3 of Test"
"copy 199 of Test" "copy 200 of Test"
"copy Test"      "copy of copy Test"
"copy 0 of Test" "copy 1 of Test"
```

FindToolType

Rechercher un tableau ToolTypes

Syntaxe

```
valeur = FindToolType(toolTypeArray,name)
        D0 -96      A0      A1
UBYTE *valeur;
UBYTE **toolTypeArray,*name;
```

Description

Cette fonction recherche un tableau ToolTypes d'une certaine entrée indiquée par name.

Paramètres

toolTypeArray: Un tableau de chaînes.

name: Le nom de l'entrée ToolTypes.

Retour

valeur: Contient un pointeur sur l'entrée. Plus exactement, la valeur pointe sur le premier caractère après le signe Egal qui apparaît dans la chaîne name.

Exemple

```
UBYTE *tTA[] =
{
    "FILETYPE=text",
    "TEMPDIR=:t"
}
FindToolType(tTA,"FILETYPE") -> Renvoie "text"
FindToolType(tTA,"TEMPDIR")  -> Renvoie ":t"
FindToolType(tTA,"MAXSIZE") -> Null
@gras1 - Voir aussi
MatchToolValue
```

MatchToolValue**Rechercher une sous-chaîne****Syntaxe**

```
ok = MatchToolValue(typeString,valeur)
      D0 -102          A0 A BOOL ok;
      UBYTE *typeString,*valeur;
```

Description

Cette fonction permet de rechercher une sous-chaîne à l'intérieur d'un tableau ToolTypes.

Paramètres

typeString: Un pointeur sur un tableau ToolTypes du même type que celui renvoyé par la fonction ToolType

valeur: Un pointeur sur la sous-chaîne que vous recherchez.

Retour

ok: Si la sous-chaîne est trouvée, la fonction renvoie 1 sinon elle renvoie 0.

Exemples

Supposons que les deux variables suivantes soient déclarées :

```
type1 = "text";
type2 = "a|b|c";
```

Voici comment la fonction agira :

```
MatchToolValue(type1,"text") => TRUE
MatchToolValue(type1,"data") => FALSE
MatchToolValue(type2,"a" ) => TRUE
MatchToolValue(type2,"b" ) => TRUE
MatchToolValue(type2,"d" ) => FALSE
MatchToolValue(type2,"a|b" ) => FALSE
```

Voir aussi

[FindToolType](#)

6.6. La librairie Graphics

La librairie Graphics fournit les fonctions graphiques et d'animation de l'Amiga. Elles permettent de gérer des objets variés tels que les sprites hardware, les sprites virtuels, les bobs (objets Blitter), les composants de l'animation, etc.

Avant d'utiliser les routines graphiques, il est bien entendu nécessaire d'ouvrir la librairie avec `OpenLibrary()` :

```
"GfxBase = (struct GfxBase) OpenLibrary("graphics.library",0)"
```

La structure `GfxBase` peut être décrite ainsi :

Offset	

0x00	0
0x22	34
0x26	38
0x2a	42
0x2e	46
0x32	50
0x36	54
0x3a	58
0x3e	62
0x42	66
0x46	70
0x4a	74
0x60	96
0x76	118
0x8c	140
0x9a	154
0x9e	158
0xa0	160
0xa1	161
0xa2	162
0xa4	164
0xa6	166
0xa7	167
0xa8	168
0xaa	170
0xac	172
0xae	174
0xbc	188
0xc0	192
0xce	206
0xd0	208
0xd4	212
0xd6	214
0xd8	216
0xda	218
0xdc	220
0xde	222
0xe0	224
0xe4	228
0xe8	232
0xea	234
0xf2	242

```

-----
```

```

        struct GfxBase <graphics/gfxbase.h>
        {
            struct Library LibNode;
            struct View *ActivView;
            struct copinit *copinit;
            long *cia;
            long *blitter;
            UWORLD *LOFlist;
            UWORLD *SHFlist;
            struct bltnode *blthd,
                            *blttl;
            struct bltnode *bsblthd,
                            *bsblttl;
            struct Interrupt vbsrv,
                            timsrv,
                            bltsrv;
            struct List TextFonts;
            struct TextFont *DefaultFont;
            UWORLD Modes;
            BYTE VBlank;
            BYTE Debug;
            SHORT Beamsync;
            SHORT system_bplcon0;
            UBYTE SpriteReserved;
            UBYTE bytereserved;
            USHORT Flags;
            SHORT BlitLock;
            short BlitNest;
            struct List BlitWaitQ;
            struct Task *BltOwner;
            struct List TOF_WaitQ;
            UWORLD DisplayFlags;
            struct SimpleSprite **SimpleSprites;
            UWORLD MaxDisplayRow;
            UWORLD MaxDisplayColumn;
            UWORLD NormalDisplayRows;
            UWORLD NormalDisplayColumns;
            UWORLD NormalDPMX;
            UWORLD NormalDPMY;
            struct SignalSemaphore *LastChanceMemory;
            UWORLD *LCMPtr;
            UWORLD MicrosPerLine;
            ULONG reserved[2];
        }
```

Les fonctions de la librairie Graphics

1. Initialisation et édition des Rasters

AllocRaster	908
FreeRaster	908
InitBitMap	909
InitRastPort	909
Move	910
SetAPen	910
SetBPen	911
SetDrMd	911
SetDrPt	912
SetRast	912
SetWrMsk	913

2. Dessin en Rastport

ClearEOL	914
ClearScreen	915
Draw	916
DrawCircle	917
Move()DrawEllipse	916
PolyDraw	917
ReadPixel	918
ScrollRaster	919
Text	919
TextLength	920
WritePixel	920

3. Remplissage de surface en Rastport

AreaCircle	921
AreaDraw	922
AreaEllipse	922
AreaEnd	923
AreaMove	923
BNDRYOFF	924
Flood	924
InitArea	925
InitTmpRas	925
RectFill	926
SetAfPt	928
SetOPen	928

4. Les commandes ColorMap

FreeColorMap	929
GetColorMapA	929
GetRGB4	930
LoadRGB4	931
SetRGB4	931
SetRGB4CM	932

5. Les commandes Blitter

BltBitmap	933
BltBitmapRastPort	934
BltClear	934
BltMaskBitmapRastPort	935
BltPattern	936
BltTemplate	936
ClipBlit	937
DisownBlitter	938
OwnBlitter	938
QBlit	939
QBSBlit	940
WaitBlit	940

6. Les commandes Copper

CBump	941
CEND	941
CMove	943
CWait	943
FreeCopList	944
FreeCprList	944
FreeVPortCopLists	944
InitView	945
InitVPort	945
LoadView	946
MakeVPort	946
MrgCop	947
ScrollVPort	947
UCopListInit	947
VBeamPos	948
WaitBOVP	948
WaitTOF	949

7. Les fonctions Layer

AndRectRegion	950
AndRegionRegion	950
AttemptLockLayerRom	951
ClearRectRegion	953
ClearRegion	953
CopySBitMap	954
DisposeRegion	954
LockLayerRom	954
NewRegion	955
OrRectRegion	955
OrRegionRegion	956
SyncSBitMap	956
UnlockLayerRom	957
XorRectRegion	957
XorRegionRegion	958

8. Manipulation des caractères

AddFont	959
AskFont	960
AskSoftStyle	960
CloseFont	961
OpenFont	961
RemFont	962
SetFont	962
SetSoftStyle	963

9. GELs et Sprites

AddAnimOb	965
AddBob	965
AddVSprite	966
Animate	966
ChangeSprite	966
DoCollision	967
DrawGList	967
FreeGBuffers	968
FreeSprite	969
GetGBuffers	969
GetSprite	970
InitAnimate	970
InitGels	971
InitGMasks	971
InitMasks	972

MoveSprite	972
RemBob	972
RemLBob	973
RemVSprite	973
SetCollision	974
SortGList	974

1. Initialisation et édition des Rasters

AllocRaster Appeler les routines système d'allocation de mémoire

Syntaxe

```
Plan = AllocRaster (Largeur,Hauteur)
      DO          -492        D0      D1
      PLANEPTR Plan;
      SHORT      Largeur,Hauteur;
```

Description

Cette fonction appelle les routines système d'allocation de mémoire pour allouer une place mémoire pour un plan de bits (bitplane).

Paramètres

Largeur: Nombre de bits dans la direction x selon le système de coordonnées d'un raster.

Hauteur: Nombre de bits dans la direction y selon le système de coordonnées d'un raster.

Retour

Plan: Contient l'adresse du premier mot du plan de bits.

Voir aussi

FreeRaster()

FreeRaster Restituer la place mémoire réservée avec AllocRaster()

Syntaxe

```
FreeRaster (Plan, Largeur, Hauteur)
           ~498       A0       D0      D1
           PLANEPTR BitPlane;
           SHORT      Largeur, Hauteur;
```

Description

Cette fonction restitue la place mémoire réservée avec AllocRaster().

Paramètres

- Plan: Un pointeur sur une zone mémoire renvoyée en résultat lors de l'appel de AllocRaster.
- Largeur: Largeur en bits du plan de bits.
- Hauteur: Hauteur en bits du plan de bits.

Voir aussi

[AllocRaster\(\)](#)

InitBitMap

Initialiser la structure Bitmap

Syntaxe

```
InitBitMap (BitMap, Profondeur, Largeur, Hauteur)
           -390      A0      D0      D1      D2
           struct BitMap *BitMap;
           BYTE      Profondeur,
           SHORT     Largeur,
           Hauteur;
```

Description

Cette routine initialise la structure Bitmap spécifiée en indiquant les paramètres de taille.

Paramètres

- BitMap: Pointeur sur une structure Bitmap.
- Profondeur: Nombre de plans de bits composant le bitmap.
- Largeur: Nombre de bits en largeur (colonnes).
- Hauteur: Nombre de bits en hauteur (lignes).

Voir aussi

[AllocRaster\(\)](#)

InitRastPort

Initialiser la structure d'un Raster port

Syntaxe

```
InitRastPort (RastPort)
           -198      A1
           struct RastPort *RastPort;
```

Description

Cette fonction initialise la structure d'un Raster port. Cette structure représente l'interface entre l'utilisateur et le Bitmap. La RastPort contient la couleur de crayon de premier plan, un pointeur sur le Bitmap ainsi que d'autres variables.

Paramètres

RastPort: Pointeur sur une structure RastPort.

Move

Déplacer le curseur graphique

Syntaxe

```
Move (RastPort, x, y)
-240      A1      D0 D1
struct RastPort *RastPort;
SHORT        x,y;
```

Description

Cette fonction déplace le curseur graphique du RastPort.

Paramètres

RastPort: Pointeur sur une structure RastPort.

x,y: Coordonnées de la nouvelle position.

Voir aussi

Draw(), Text()

SetAPen

Redéfinir la couleur de crayon primaire

Syntaxe

```
SetAPen (RastPort, RegistreCouleur)
-342      A1      D0
struct RastPort *RastPort;
SHORT        RegistreCouleur;
```

Description

Cette fonction redéfinit la couleur de crayon primaire pour le dessin des lignes, le remplissage et le texte.

Paramètres

RastPort: Pointeur sur une structure RastPort.

RegistreCouleur: Indique le numéro de registre de couleur.

Voir aussi

`SetBPen()`, `SetOPen()`

SetBPen**Définir la couleur du crayon secondaire****Syntaxe**

```
SetBPen (RastPort, RegistreCouleur)
        -348      A1      D0
        struct RastPort *RastPort;
        SHORT RegistreCouleur;
```

Description

Cette fonction définit la couleur du crayon secondaire.

Paramètres

`RastPort:` Pointeur sur une structure `RastPort`.

`RegistreCouleur:` Indique le numéro de registre de couleur.

Voir aussi

`SetDrMd()`, `SetAPen()`

SetDrMd**Définir le mode de dessin****Syntaxe**

```
SetDrMd (RastPort, Mode)
        -354      A1      D0
        struct RastPort *RastPort;
        SHORT      Mode;
```

Description

Cette routine définit le mode de dessin pour les lignes, remplissages et textes. Les définitions de bits sont lues dans `<rastport.h>`.

Paramètres

`RastPort:` Pointeur sur une structure `Rastport`.

`Mode:` Un mode de dessin (0-255).

Modes de dessin possibles :

`JAM1 (0)` On dessine uniquement avec `APen`.

`JAM2 (0)` Le dessin apparaît sur un fond de la couleur du `BPen`.

COMPLEMENT (2)

L'opération OU est effectuée entre les points mis et les nouveaux points.

INVERSVID (4) Les points sont inversés avant sortie : les pixels qui devraient être mis ne le sont pas et inversement (plan de bits par plan de bits).

Les modes COMPLEMENT et INVERSVID fonctionnent uniquement en liaison avec JAM1 et JAM2.

SetDrPt**Définir le motif de remplissage des lignes****Syntaxe**

```
SetDrPt (RastPort, Motif)
(Macro)
struct RastPort *RastPort;
UWORD           Motif;
```

Description

Cette fonction définit le motif de remplissage des lignes pour les lignes dessinées avec les fonctions Draw, Move et Polydraw. Le réglage du motif devient alors partie intégrante de la structure du RastPort actuel.

Paramètres

RastPort: Un pointeur sur la structure RastPort.

Motif: Un motif de remplissage sur 1 mot.

SetRast**Régler une surface de dessin****Syntaxe**

```
SetRast (RastPort, RegistreCouleur)
-234      A1      D0
struct RastPort *RastPort;
UBYTE        RegistreCouleur;
```

Description

Cette fonction règle une surface de dessin sur une couleur spécifiée.

Paramètres

RastPort: Pointeur sur une structure RastPort.

RegistreCouleur: Numéro de couleur (0-255).

Voir aussi

[SetAPen\(\)](#), [SetBPen\(\)](#)

SetWrMsk**Définir le masque d'écriture des plans****Syntaxe**

```
SetWrMsk (RastPort, Masque)
(Macro)
struct RastPort *RastPort;
UBYTE           Masque;
```

Description

Cette fonction définit le masque d'écriture des plans.

Paramètres

RastPort: Un pointeur sur une structure Rast.

Masque: Une valeur sur 16 bits dont les six bits inférieurs déterminent quels plans de bits du bitmap peuvent être écrits.

Structure

Offset	Structure
0x00	struct BitMap <graphics/gfx.h>
0x02	UWORD BytesPerRow;
0x04	UWORD Rows;
0x05	UBYTE Flags;
0x06	UBYTE Depth;
0x08	UWORD pad;
0x08	PLANEPTR Planes[8];
0x10	}
0x00	struct RastPort <graphics/rastport.h>
0x04	struct Layer *Layer;
0x04	struct BitMap *BitMap;
0x08	USHORT *AreaPtrn;
0x0c	struct TmpRas *TmpRas;
0x10	struct AreaInfo *AreaInfo;
0x14	struct GelsInfo *GelsInfo;
0x18	UBYTE Mask;
0x19	BYTE FgPen;
0x1a	BYTE BgPen;
0x1b	BYTE A01Pen;
0x1c	BYTE DrawMode;
0x1d	BYTE AreaPtSz;
0x1e	BYTE linpatcnt;
0x1f	BYTE dummy;
0x20	USHORT Flags;

```

0x22 34      USHORT LinePtn;
0x24 36      SHORT cp_x,
0x26 38          cp_y;
0x28 40      UBYTE minterms[8];
0x30 48      SHORT PenWidth;
0x32 50      SHORT PenHeight;
0x34 52      struct TextFont *Font;
0x38 56      UBYTE AlgoStyle;
0x39 57      UBYTE TxFlags;
0x3a 58      WORD TxHeight;
0x3c 60      WORD TxHeight;
0x3e 62      WORD TxBaseline;
0x40 64      WORD TxSpacing;
0x42 66      APTR *RP_User;
0x46 70      ULONG longreserved[2];
#ifndef GFX_RASTPORT_1_2
0x4e 78      WORD wordreserved[7];
#endif
0x5c 92      UBYTE reserved[8];
0x64 100 }

```

2. Dessin en RastPort

ClearEOL

Effacer le texte depuis la position en cours

Syntaxe

```

ClearEOL (RastPort)
    -42      A1
    struct RastPort *RastPort;

```

Description

Cette fonction efface le texte depuis la position en cours jusqu'à la fin de la ligne. La hauteur de la zone effacée est lue à partir de la fonte texte en cours, la base verticale est ajustée sur la base du texte.

Paramètres

RastPort: Pointeur sur une structure Rastport.

Voir aussi

[ClearScreen\(\)](#), [Move\(\)](#)

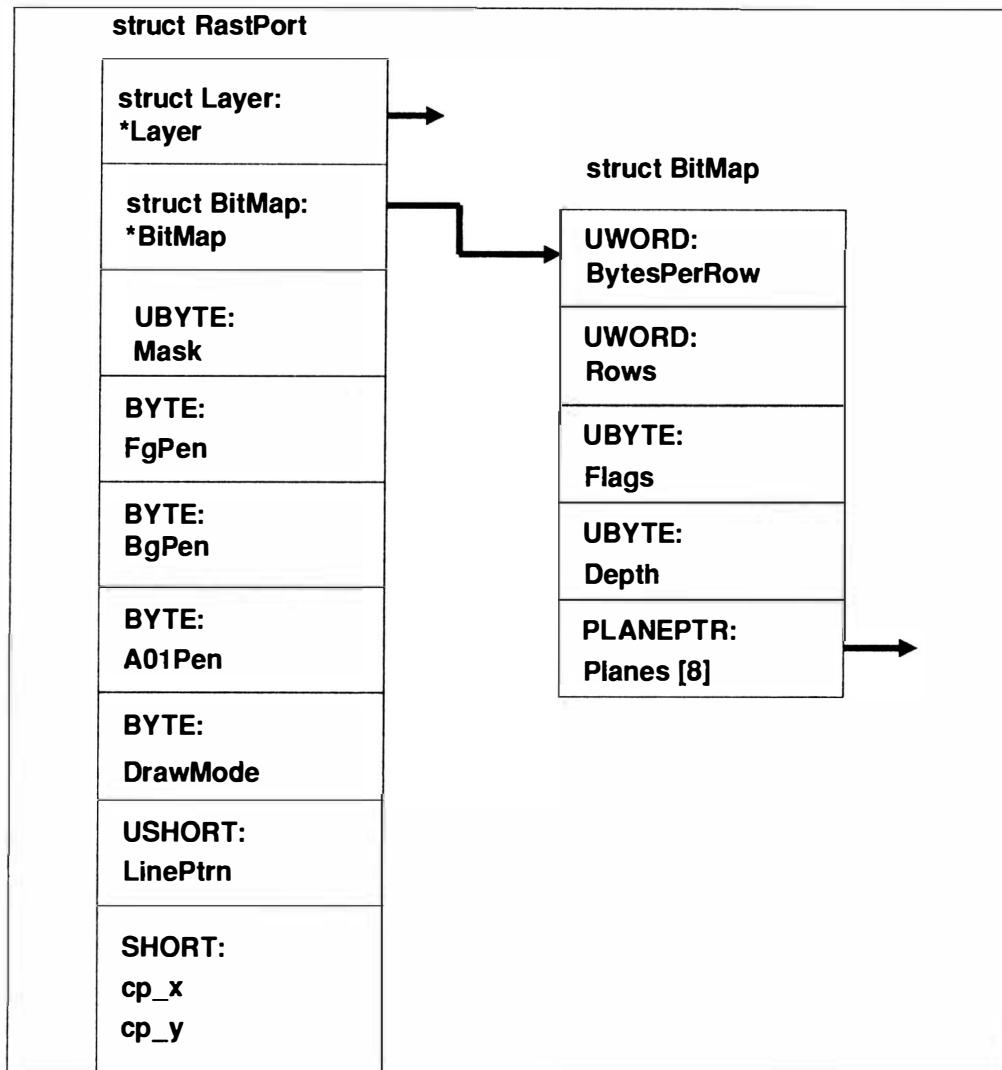


Figure 6 - 60

ClearScreen**Effacer le texte****Syntaxe**

```

ClearScreen (RastPort)
    -48          A1
    struct RastPort *RastPort;
  
```

Description

Cette fonction efface le texte depuis la position en cours jusqu'à la fin du RastPort. D'abord la routine ClearEOL est exécutée, puis le reste de l'écran depuis la zone effacée jusqu'à la fin du RastPort est supprimé.

Paramètres

RastPort: Pointeur sur une structure RastPort.

Voir aussi

Draw	Tracer un trait
------	-----------------

Syntaxe

```
Draw (RastPort, x, y)
-246      A1      D0  D1
struct RastPort *RastPort;
SHORT        x,y;
```

Description

Cette fonction trace un trait depuis la position actuelle du curseur (crayon) jusqu'au point de coordonnées raster x,y.

Paramètres

RastPort: Pointeur sur une structure RastPort.

x,y: Coordonnées horizontale et verticale d'un pixel en système de coordonnées raster.

Voir aussi

Move()DrawEllipse	Dessiner une ellipse
-------------------	----------------------

Syntaxe

```
DrawEllipse (RastPort, XM, YM, Xr, Yr)
-180      A1      D0  D1  D2  D3
struct RastPort *RastPort;
SHORT        XM,YM;
SHORT        Xr,Yr;
```

Description

Cette fonction permet de dessiner une ellipse avec la couleur de crayon en cours. Seul le pourtour de l'ellipse est dessiné. Elle n'est pas remplie.

Paramètres

- RastPort: Pointeur sur une structure RastPort dans laquelle l'ellipse sera dessinée.
- XM, YM: Coordonnées du centre de l'ellipse.
- Xr,Yr: Rayons horizontal et vertical de l'ellipse.

Voir aussi

[AreaEllipse\(\)](#)

DrawCircle

Dessiner un cercle

Syntaxe

```
DrawCircle (RastPort, XM, YM, Radius)
(Macro)
struct RastPort *RastPort;
SHORT           XM, YM;
SHORT           Radius;
```

Description

Cette fonction permet de dessiner un cercle avec la couleur de crayon en cours. Seul le pourtour du cercle est dessiné. Il n'est pas rempli.

Paramètres

- RastPort: Pointeur sur une structure RastPort.
- XM, YM: Coordonnées du centre.
- Radius: Rayon

Voir aussi

[AreaEllipse\(\)](#)

PolyDraw

Dessiner des lignes en joignant des points successifs

Syntaxe

```
PolyDraw (RastPort, Count, Array)
-336      A1      D0      A0
struct RastPort RastPort;
SHORT       Count;
struct tPoint Array[MaxCount];
```

Description

Cette fonction dessine des lignes en joignant des points successifs définis dans une table.

Paramètres

- RastPort: Pointeur sur une structure RastPort.
 Count: Nombre de points de la table de définition du polygone.
 Array: Pointeur sur le premier couple (x,y) de la table.

Voir aussi

[Move\(\)](#), [Draw\(\)](#)

ReadPixel

Combiner les bits de chaque plan de bit à une position spécifiée

Syntaxe

```
ReadPixel (RastPort, x, y)
DO -318      A1      DO D1
      LONG      RegistreCouleur;
struct RastPort *RastPort;
      SHORT      x,y;
```

Description

Cette routine combine les bits de chaque plan de bit à une position spécifiée. Elle calcule ensuite la couleur de crayon correspondant à la combinaison.

Paramètres

- RastPort: Pointeur sur une structure RastPort.
 x,y: Coordonnées pixels du point étudié.

Retour

RegistreCouleur: Récupère le numéro de couleur du crayon (0-255). En cas de problème, la valeur -1 est renvoyée.

Voir aussi

[WritePixel\(\)](#)

ScrollRaster**Effectuer un scrolling****Syntaxe**

```
ScrollRaster (RastPort, deltaX, deltaY, x1, y1, x2, y2)
              -396      A1      D0      D1      D2      D3      D4      D5
    struct RastPort *RastPort;
    SHORT          deltaX, deltaY;
    SHORT          x1,y1,y2,x2;
```

Description

Cette fonction permet d'effectuer un scrolling d'un certain nombre de pixels, en limitant le mouvement à l'intérieur d'un rectangle.

Paramètres

- RastPort: Pointeur sur une structure RastPort.
- deltaX,deltaY: Nombre de pixels à scroller.
- x1,y1: Coordonnées du coin supérieur gauche du rectangle.
- x2,y2: Coordonnées du coin inférieur droit du rectangle.

Text**Ecrire des caractères texte à l'intérieur d'un raster bitmap****Syntaxe**

```
Text (RastPort, String, Count)
      -54      A1      A0      D0
    struct RastPort *RastPort;
    char          *String;
    SHORT         Count;
```

Description

Cette fonction écrit des caractères texte à l'intérieur d'un raster bitmap en commençant à la position pixel en cours.

Paramètres

- RastPort: Pointeur sur une structure RastPort.
- String: Pointeur sur une chaîne de caractères à dessiner.
- Count: Nombre de caractères de la chaîne texte. Si cette valeur est nulle, il n'y a pas de caractères à traiter.

Voir aussi

[Move\(\)](#), [SetDrMd\(\)](#)

TextLength**Déterminer la taille des données texte****Syntaxe**

```
Length = TextLength (RastPort, String, Count)
DO          -54      A1      A0      D0
SHORT        Length;
struct RastPort *RastPort;
char         *String;
SHORT        Count;
```

Description

Cette fonction détermine la taille en pixels des données texte à sortir.

Paramètres

RastPort: Pointeur sur une structure Rastport.

String: Pointeur sur la chaîne String.

Count: Nombre de caractères de la chaîne.

Retour

Length: Récupère la taille en pixels de la chaîne de caractères.

Voir aussi

[Text\(\)](#), [SetFont\(\)](#)

WritePixel**Changer la couleur et les autres attributs d'un pixel****Syntaxe**

```
Status = WritePixel (RastPort, x, y)
DO          -324      A1      D0      D1
LONG        Status;
struct RastPort *RastPort;
SHORT        x,y;
```

Description

Cette fonction change la couleur et les autres attributs d'un pixel donné en prenant ceux qui sont définis par la fonction [SetAPen](#) et par les paramètres actuels de la structure de contrôle RastPort.

Paramètres

RastPort: Pointeur sur une structure Rastport.

x,y: Coordonnées du pixel en question.

Retour

Status: Vaut 0 si l'opération a réussi, -1 si le point est hors du RastPort.

Voir aussi

[ReadPixel\(\)](#)

Structure

```
Offset   Structure
-----  -----
          struct tPoint <graphics/gfx.h>
{
  0x00  0      WORD x,
  0x02  2      y;
  0x04  4
}
```

3. Remplissage de surface en Rastport**AreaCircle****Ajouter un cercle****Syntaxe**

```
Status = AreaCircle (RastPort,Xm,Ym,Radius)
(Macro)
  LONG           Status;
  struct RastPort *RastPort;
  SHORT          Xm,Ym;
  SHORT          Radius;
```

Description

Cette fonction ajoute un cercle dans la structure `AreaInfo` pour un remplissage de surface

Paramètres

RastPort: Pointeur sur une structure Rastport.

Xm, Ym: "Point central" du raster.

Radius: Rayon du cercle.

Retour

Status: 0 s'il n'y a pas d'erreur, -1 si la structure est pleine.

Voir aussi

[AreaEnd\(\)](#), [InitArea\(\)](#)

AreaDraw**Ajouter un point à un buffer****Syntaxe**

```
AreaDraw (RastPort, x, y)
DO -258      A1    DO  D1
LONG          Status;
struct RastPort *RastPort;
SHORT         x,y;
```

Description

Cette fonction ajoute un point à un buffer défini par une série de valeurs x,y. Celles-ci sont utilisées pour définir des surfaces devant être remplies par la fonction AreaEnd.

Paramètres

RastPort: Pointeur sur une structure Rastport.

x,y: Les coordonnées du pixel.

Retour

Status: 0 s'il n'y a pas d'erreur, -1 si la structure est pleine.

Voir aussi

[AreaEnd\(\)](#), [AreaEllipse\(\)](#), [AreaMove\(\)](#), [InitArea\(\)](#)

AreaEllipse**Ajouter une ellipse à la structure AreaInfo****Syntaxe**

```
Status = AreaEllipse (RastPort, Xm, Ym, Xr, Yr)
DO      -186      A1    DO  D1  D2  D3
LONG          Status;
struct RastPort *RastPort;
SHORT         Xm,Ym;
SHORT         Xr,Yr;
```

Description

Cette fonction ajoute une ellipse à la structure AreaInfo.

Paramètres

RastPort: Pointeur sur une structure Rastport.

Xm, Ym: Coordonnées du centre relativement au RastPort.

Xr, Yr: Rayons horizontal et vertical de l'ellipse.

Retour

Status: 0 s'il n'y a pas d'erreur, -1 si la structure est pleine.

Voir aussi

[AreaEnd\(\)](#), [InitArea\(\)](#)

AreaEnd**Traiter la table des vecteurs**

Syntaxe

```
Status = AreaEnd (RastPort)
    D0      -264     A1
    struct RastPort *RastPort;
```

Description

Cette fonction traite la table des vecteurs et produit le remplissage de la surface.

Paramètres

RastPort: Pointeur sur une structure Rastport.

Retour

Status: 0 s'il n'y a pas d'erreur, -1 si une erreur survient.

Voir aussi

[AreaDraw\(\)](#), [AreaMove\(\)](#), [AreaEllipse\(\)](#), [InitArea\(\)](#), [InitTmpRas\(\)](#)

AreaMove**Définir un nouveau point de départ**

Syntaxe

```
Status = AreaMove (RastPort, x, y)
    D0      -252     A1      D0  D1
    struct RastPort *RastPort;
    SHORT           x,y;
```

Description

Cette fonction définit un nouveau point de départ pour une seule forme unique.

Paramètres

RastPort: Pointeur sur une structure Rastport.

x,y: Coordonnées du pixel.

Retour

Status: 0 s'il n'y a pas d'erreur, -1 si une erreur survient.

Voir aussi

`AreaDraw()`, `AreaEllipse()`, `AreaEnd()`, `InitArea()`

BNDRYOFF**Annuler le dessin d'un contour**

Syntaxe

```
BNDRYOFF (RastPort)
(Macro)
struct *RastPort;
```

Description

Cette macro annule le dessin du contour lors de l'exécution de `AreaEnd` et `RectFill`. Le flag `AreaOutline` de la structure `RastPort` est aussi réinitialisé.

Paramètres

`RastPort`: Pointeur sur une structure `Rastport`

Flood**Remplir un raster comme une opération de remplissage de surface**

Syntaxe

```
Flood (RastPort, Modus, x, y)
-330      A1      D2  D0 D1
struct RastPort *RastPort;
ULONG      Mode;
SHORT      x,y;
```

Description

Cette fonction remplit un raster comme une opération de remplissage de surface. `Flood` recherche le bitmap commençant au point `x,y`. Les pixels adjacents sont ensuite remplis selon le mode de remplissage défini par `Modus`.

Paramètres

`RastPort`: Pointeur sur une structure `Rastport`.

`Modus`: Mode de remplissage. S'il vaut 0, la fonction `Flood` remplit les pixels s'ils ne correspondent pas aux réglages en cours de `AOLPen`. S'il vaut 1, `Flood` remplit les pixels s'ils ont la même couleur que `x,y`.

`x, y`: Les coordonnées du pixel.

Voir aussi

InitTmpRas(), SetOPen()

InitArea**Allouer de la mémoire pour un matrice de points et l'initialiser****Syntaxe**

```
InitArea (AreaInfo, Buffer, Nombre-points)
        -282      A0      A1      D0
        struct AreaInfo *AreaInfo;
        APTR          Buffer;
        SHORT         Nombre-points;
```

Description

Cette fonction alloue de la mémoire et initialise les spécifications de la matrice des points qui définissent une surface.

Paramètres

- AreaInfo: Un pointeur sur une structure AreaInfo.
 Buffer: Un pointeur sur le bloc mémoire destiné à stocker les points x,y.
 Nombre-points: Nombre maximal de points que le buffer devra contenir.

Voir aussi

AreaDraw(), AreaEnd(), AreaEllipse(), AreaMove()

InitTmpRas Initialiser une zone mémoire dans les premiers 512 Ko**Syntaxe**

```
InitTmpRas (TmpRas, Buffer, Buffersize)
        D0      -468      A0      A1      D0
        struct TmpRas *ITmpRas;
        struct TmpRas *TmpRas;
        APTR          Buffer;
        LONG         Buffersize;
```

Description

Cette fonction initialise une zone mémoire dans les premiers 512 Ko de la RAM pour être utilisée par des opérations flood-fill, area-fill et des fonctions texte.

Paramètres

- TmpRas: Un pointeur sur une structure TmpRas.

Buffer: Un pointeur sur un bloc mémoire dans les premiers 512 Ko de RAM.

Buffersize: La taille du buffer en octets.

RectFill

Remplir une surface rectangulaire

Syntaxe

```
RectFill (RastPort, x1, y1, x2, y2)
          -306      A1      D0      D1      D2      D3
          struct RastPort *RastPort;
          SHORT           x1,y1,x2,y2;
```

Description

Cette fonction remplit la surface rectangulaire spécifiée avec les couleurs de crayon et de motif indiquées.

Paramètres

RastPort: Pointeur sur une structure Rastport.

x1,y1,x2,y2: Coordonnées du coin supérieur gauche et du coin inférieur droit.

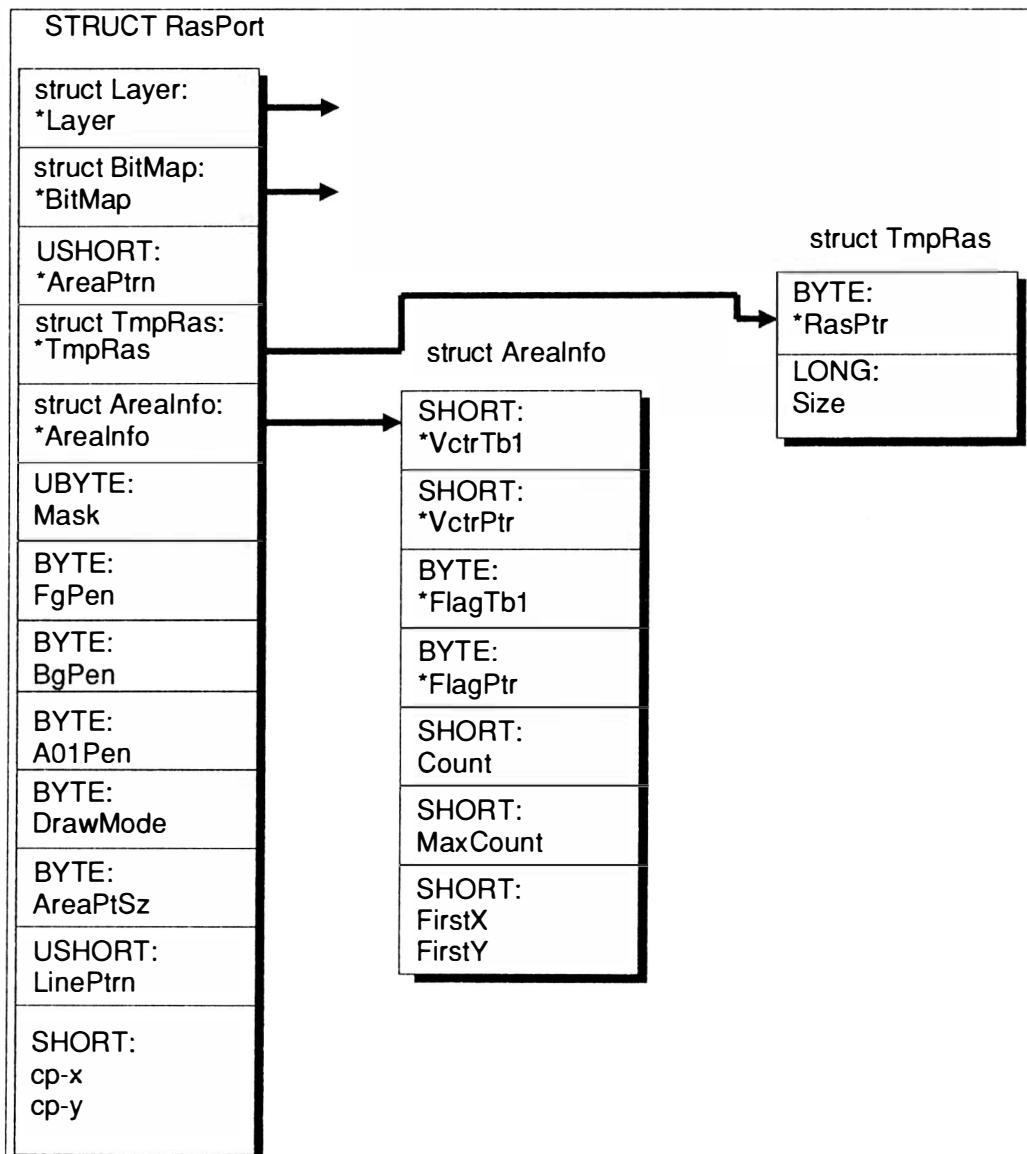


Figure 6 - 61

Voir aussi

BNDRYOFF()

SetAfPt**Définir le motif de remplissage des surfaces****Syntaxe**

```
SetAfPt (RastPort, Motif, Power)
(Macro)
struct RastPort *RastPort;
UWORD           Motif[];
SHORT            Power;
```

Description

Cette fonction définit le motif de remplissage des surfaces.

Paramètres

- RastPort: Pointeur sur une structure Rastport.
 Motif: Pointeur sur le premier mot du motif dans la RAM.
 Power: Nombre de mots du motif (puissance de 2).

Exemple

```
WORD Motif[NbrBitPlanes][16];
..
SetAfPt (&RastPort, Motif, -4);
```

SetOPen**Définir la couleur de contour du crayon****Syntaxe**

```
SetOPen (RastPort, RegistreCouleur)
(Macro)
struct RastPort *RastPort;
SHORT           RegistreCouleur;
```

Description

Cette fonction définit la couleur de contour du crayon (AOLPen) pour le texte et les bordures de surfaces.

Paramètres

- RastPort: Pointeur sur une structure Rastport.
 RegistreCouleur: Une couleur de crayon entre 0 et 255.

Voir aussi

BNDRYOFF()

Structure

```

Offset   Structure
-----
struct AreaInfo <graphics/gfx.h>
{
0x00 0      SHORT *VctrTbl;
0x04 4      SHORT *VctrPtr;
0x08 8      BYTE *FlagTbl;
0x0c 12     BYTE *FlagPtr;
0x10 16     SHORT Count;
0x12 18     SHORT MaxCount;
0x14 20     SHORT FirstX,
0x16 22     FirstY;
0x18 24 }
struct TmpRas <graphics/rastport.h>
{
0x00 0      BYTE *RasPtr;
0x04 4      LONG Size;
0x08 8 }

```

4. Les commandes ColorMap

FreeColorMap

Libérer la mémoire

Syntaxe FreeColorMap (ColorMap)
 -576 A0
 struct ColorMap *ColorMap;

Description

Cette fonction libère la mémoire précédemment allouée à une structure ColorMap.

Paramètres

ColorMap: Un pointeur sur une structure ColorMap.

Voir aussi

GetColorMap()

GetColorMapA

Allouer de la mémoire pour ColorMap

Syntaxe ColorMap = GetColorMap (Entries)
 D0 - -570 D0
 struct ColorMap *ColorMap;
 LONG Entries;

Description

Cette fonction alloue la mémoire pour une structure ColorMap et initialise la structure.

Paramètres

Entries: Le nombre d'entrées de registres de couleur.

Retour

ColorMap: Une valeur nulle sera retournée si la routine ne peut pas allouer la mémoire nécessaire.

GetRGB4**Renvoyer la valeur d'une entrée de ColorMap****Syntaxe**

```
Couleur = GetRGB4 (ColorMap, RegistreCouleur)
DO      -582      A0      DO
ULONG      Couleur;
struct ColorMap *ColorMap;
LONG      RegistreCouleur;
```

Description

Cette fonction renvoie la valeur d'une entrée dans une table de couleur d'une structure ColorMap.

Paramètres

ColorMap: Pointeur sur une structure ColorMap.

RegistreCouleur: Un indice sur une table de couleurs.

Retour

Couleur: Il s'agit d'une valeur RGB en UWORLD (deux octets), 4 bits par couleur (red, green, blue), justifiée à droite. Les bits 0 à 3 sont affectés au bleu, les bits 4 à 7 au vert, les bits 8 à 11 au rouge. Cela permet de représenter une couleur parmi 4096.

Rouge = (Couleur>>8) & 0xf

Vert = (Couleur>>4) & 0xf

Bleu = (Couleur>>0) & 0xf

Voir aussi

GetColorMap(), LoadRGB4(), SetRGB4(), SetRGB4CM(), MrgCop(), LoadView()

LoadRGB4**Modifier les réglages RGB****Syntaxe**

```
LoadRGB4 (ViewPort, Colortable, Count)
    -192      A0      A1          D0
    struct ViewPort *ViewPort;
    UWORD      Colortable[Count];
    SHORT       Count;
```

Description

Cette fonction affecte des réglages de valeurs de registre de couleur RGB à la structure ViewPort.

Paramètres

- ViewPort: Un pointeur sur une structure ViewPort.
- Colortable: Un pointeur sur le tableau de définition des couleurs.
- Count: Le nombre sur deux octets à charger depuis le tableau des couleurs.

Voir aussi

[GetRGB4\(\)](#), [SetRGB4\(\)](#)

SetRGB4**Sauver les couleurs****Syntaxe**

```
SetRGB4 (ViewPort, Entry, Rouge, Vert, Bleu)
    -288      A0          D0      D1      D2      D3
    struct ViewPort *ViewPort;
    SHORT      Entry;
    UBYTE     Rouge, Vert, Bleu;
```

Description

Cette fonction sauve les valeurs de définition des couleurs dans le tableau ColorTable.

Paramètres

- ViewPort: Pointeur sur la structure ViewPort.
- Entry: Numéro de crayon (0-31).
- Rouge, Vert, Bleu: Niveaux des différentes couleurs.

Voir aussi

[LoadRGB4\(\)](#), [GetRGB4\(\)](#), [SetRGB4CM\(\)](#)

SetRGB4CM**Définir un registre de couleur****Syntaxe**

```
SetRGB4CM (ColorMap, Entry, Rouge, Vert, Bleu)
          -630      A0      D0      D1      D2      D3
struct ColorMap *ColorMap;
SHORT        Entry;
UBYTE       Rouge, Vert, Bleu;
```

Description

Cette fonction définit l'un des registres de couleur pour le ColorMap.

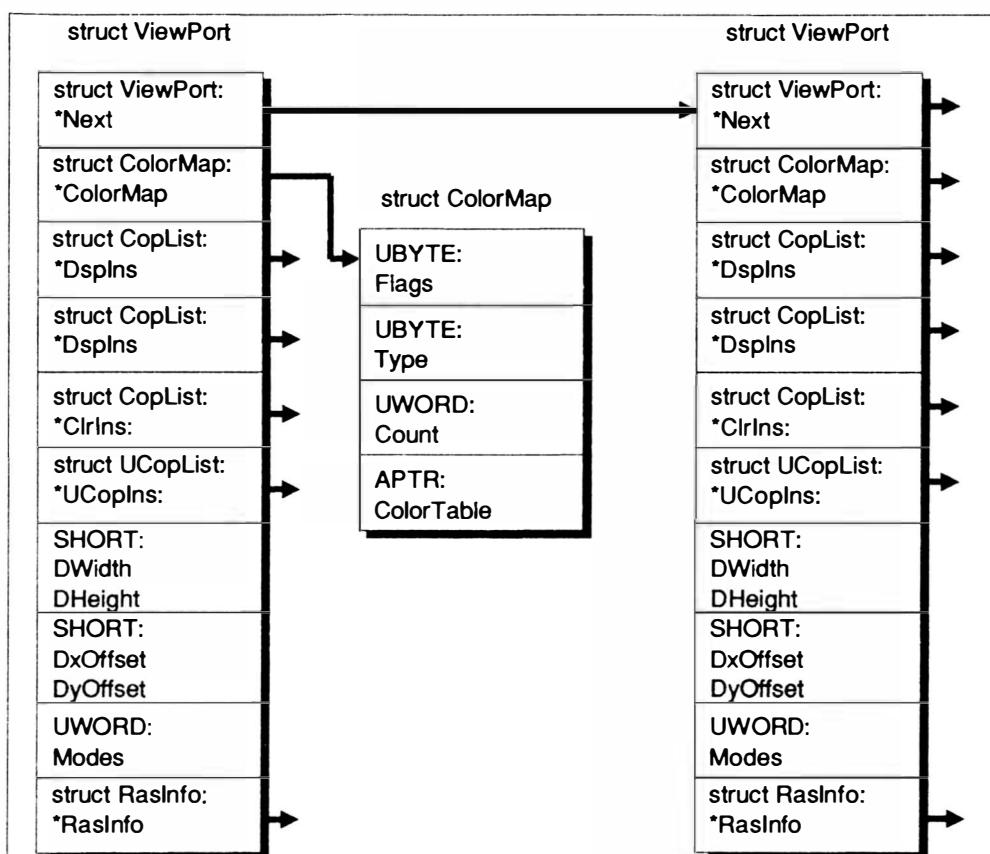


Figure 6 - 62

Paramètres

ColorMap: Pointeur sur la structure ColorMap.

Entry: Numéro de registre de couleur (0-31).

Rouge, Vert, Bleu: Valeurs des intensités des couleurs (valeur entière comprise entre 0 et 15).

Voir aussi

SetRGB40

Structure

```
Offset   Structure
-----
struct ColorMap <graphics/view.h>
{
0x00 0      UBYTE Flags;
0x01 1      UBYTE Type;
0x02 2      UWORLD Count;
0x04 4      APTR ColorTable;
0x08 8      }
```

5. Les commandes Blitter

BltBitMap

Exécuter un déplacement

Syntaxe

```
BitPlanes = BltBitMap (SourceBitMap, x1, y1, CibleBitMap,
                      A0          D0          D1          A1
                      x2, y2, Largeur, Hauteur, Minterm, Masque, Buffer)
D2  D3  D4  D5  D6  D7  A2
ULONG           BitPlanes;
struct BitMap *SourceBitMap;
SHORT           x1,y1;
struct BitMap *CibleBitMap;
SHORT           x2,y2;
SHORT           Largeur, Hauteur;
UBYTE          Minterm, Masque;
PLANEPTR       Buffer;
```

Description

Cette fonction exécute un déplacement d'une zone de bits entre un source Bitmap et une cible Bitmap.

Paramètres

SourceBitMap: Un pointeur sur une structure Bitmap.

x1,y1: Les coordonnées du point supérieur gauche dans le source Bitmap.

CibleBitMap: Un pointeur sur la structure bitmap cible via la structure RastPort.

x2,y2: Les coordonnées du point supérieur gauche dans la cible Bitmap.

Largeur, Hauteur: Taille du rectangle à déplacer.

Minterm: La fonction logique à utiliser pour l'opération Blitter.

Masque: Le masque d'écriture à appliquer à l'opération Blitter.

Buffer: Pointeur sur une zone buffer temporaire.

Retour

BitPlanes: Contient le nombre de plans de bits. Ce nombre sera inférieur à celui qui est attendu si le buffer n'est pas alloué proprement.

Voir aussi

ClipBlit()

BltBitmapRastPort

Effectuer une opération Blitter

Syntaxe

```
BltBitmapRastPort (SourceBitMap, x1, y1,
                    CibleRastPort,x2,y2, Largeur, Hauteur, Minterm)
                    -6D6           A0      D0  D1           A1      D2  D3
                    D4      D5      D6
struct BitMap  *SourceBitMap;
SHORT          x1,y1;
struct RastPort *CibleRastPort;
SHORT          x2,y2;
SHORT          Largeur, Hauteur;
UBYTE         Minterm;
```

Description

Cette fonction effectue une opération Blitter entre un source Bitmap et une cible RastPort.

Paramètres

Voir BltBitmap

BltClear

Effacer un bloc de mémoire

Syntaxe

```
BltClear (Bloc, Bytecount, Flags)
        -3DD           A1      D0      D1
APTR  Bloc;
ULONG Bytecount;
ULONG Flags;
```

Description

Cette fonction utilise le Blitter pour effacer un bloc mémoire dans les premiers 512 Ko de RAM. Il existe deux méthodes d'effacement. La première, standard, permet d'appeler le blitter plusieurs fois pour effacer la mémoire. Dans la seconde, Lignes / octets par ligne, Le nombre de lignes doit être ≤ 1024 , le nombre d'octets par ligne doit être ≤ 128 .

Paramètres

- Bloc: Pointeur sur le bloc mémoire à effacer.
- Bytecount: Si le second bit (bit 1) de la variable Flags est à 0, il s'agit d'un nombre pair de bits à effacer. Sinon, les 16 bits faibles de la variable indiquent le nombre d'octets / ligne, les 16 bits forts donnent le nombre de lignes.
- Flags: C'est une variable d'un octet. Le bit 0 permet de forcer la fonction BltClear à attendre que l'opération Blitter soit terminée, le bit 1 définit le mode utilisé.

BltMaskBitMapRastPort

Copier avec le Blitter

Syntaxe

```
BltMaskBitMapRastPort (SourceBitMap, X1, Y1,
                      -636           A0      D0  D1
                      CibleRastPort, X2, Y2, Largeur, Hauteur, Minterm, BltMask)
                      A1      D2  D3   D4      D5      D6       A2
struct BitMap *CibleBitMap;
SHORT          X1,Y1;
struct RastPort *RastPort;
SHORT          X2,Y2;
SHORT          Largeur, Hauteur;
UBYTE         Minterm;
APTR          BltMask;
```

Paramètres

- SourceBitMap: voir BltBitMap().
- x1,y1: voir BltBitMap().
- CibleRastPort: voir BltBitMap().
- x2,y2: voir BltBitMap().
- Largeur, Hauteur: voir BltBitMap().
- Minterm: Permet de préciser le mode de copie (telle quelle ou sens inversé) : (ABC | ABNC | ANBC) si le source soit être copié et blitté à travers un masque, (ANBC) si le source doit d'abord être inversé.

BltMask: Pointeur sur un masque one-bitplane.

Voir aussi

[BltBitMap\(\)](#)

BltPattern

Dessiner avec le Blitter

Syntaxe

```
    BltPattern (RastPort, Mask, x1, y1, x2, y2, Bytecount)
    -312          A1          A0          D0          D1          D2          D3          D4
    struct RastPort *RastPort;
    APTR           Mask;
    SHORT          x1, y1,
    SHORT          x2, y2;
    SHORT          Bytecount;
```

Description

Cette fonction effectue une opération Blitter en utilisant le mode de dessin, le remplissage de surface, etc du RastPort en cours.

Paramètres

RastPort: Pointeur sur une structure Rastport.

Mask: Pointeur sur un masque à deux dimensions.

x1,y1,x2, y2: Coordonnées du coin supérieur gauche et du coin inférieur droit d'une zone rectangulaire.

Bytecount: Nombre d'octets par ligne (mode octets par ligne).

Voir aussi

[BltMaskBitMapRastPort\(\)](#)

BltTemplate

Extraire une zone avec le Blitter

Syntaxe

```
    BltTemplate (Source, BitPosition, Modulo, RastPort, x, y,
    Largeur, Hauteur)
    -36          A0          D0          D1          A1          D2          D3
    D4          D5
    APTR           *Source;
    SHORT          BitPosition;
    SHORT          Modulo;
    struct RastPort *RastPort;
    SHORT          x,y;
    SHORT          Largeur,Hauteur;
```

Description

Cette fonction utilise une opération Blitter pour extraire une zone rectangulaire d'un tableau source pour le placer dans un Rastport.

Paramètres

- Source: Pointeur sur le tableau source des bits de données.
- BitPosition: Coordonnée x du coin supérieur gauche dans le tableau source.
- Modulo: Le modulo source qui permet de déterminer le prochain indice du tableau correspondant à la prochaine ligne du rectangle source.
- RastPort: Pointeur sur une structure Rastport.
- x,y: Coordonnées du coin supérieur gauche du rectangle dans le système de coordonnées du bitmap cible.
- Largeur, Hauteur: la taille du rectangle dans le système de coordonnées du bitmap source.

ClipBlit

Extraire une zone avec le Blitter

Syntaxe

```
ClipBlit (SourceRastPort, x1, y1, CibleRastPort, x2,
          y2, Largeur, Hauteur, Minterm)
          -552           A0      D0  D1       A1       D2  D3  D4
D5      D6
struct RastPort *SourceRastPort;
SHORT      x1,y1;
struct RastPort *CibleRastPort;
SHORT      x2,y2;
SHORT      Largeur, Hauteur;
UBYTE     Minterm;
```

Description

Cette instruction a la même fonction que BltBitMap(). la différence est que le transfert s'opère d'un RastPort à un autre. Il est donc plus facile d'utiliser cette instruction puisqu'il n'est pas nécessaire de réserver un buffer et qu'il n'y a pas de problèmes de chevauchement. Le fait de blitter au-delà des limites d'un RastPort n'entraîne pas de conséquence grave comme avec BltBitMap().

Paramètres

Voir BltBitMap()

DisownBlitter**Libérer le Blitter****Syntaxe**`DisownBlitter()`**Description**

Cette fonction libère le Blitter que vous aviez occupé .

Voir aussi

`OwnBlitter()`

OwnBlitter**Monopoliser le Blitter****Syntaxe**`OwnBlitter()`
-456**Description**

Cette fonction vous permet de monopoliser le Blitter. Les autres tâches n'y pourront plus alors accéder. Après `OwnBlitter()`, il est conseillé d'attendre avec `WaitBlit()` que le Blitter termine l'opération.

Voir aussi

`DisownBlitter()`, `WaitBlit()`

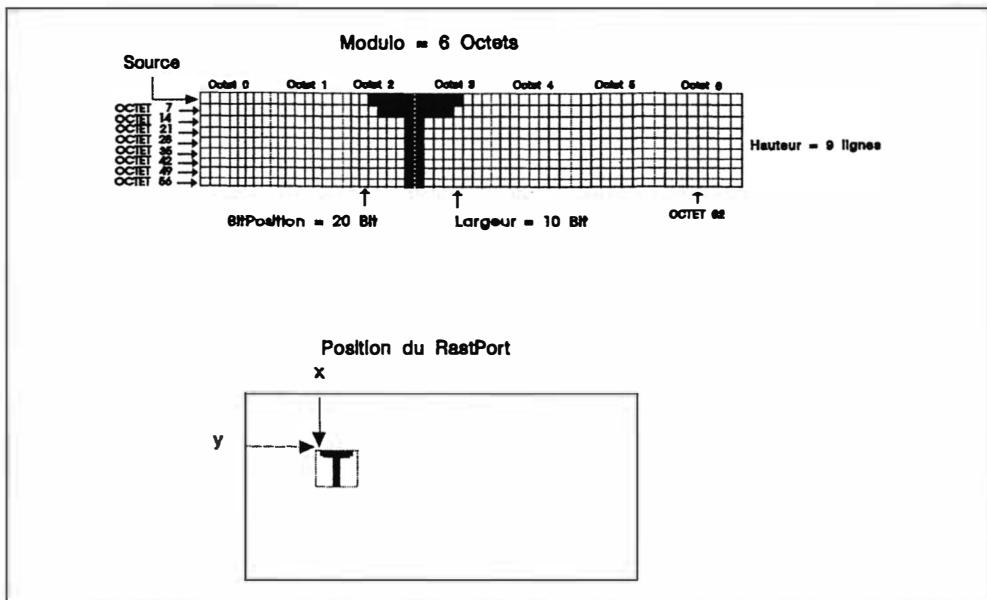


Figure 6 - 63

QBlit

Lier un Requester

Syntaxe

```
QBlit (BlitNode)
-276
struct blitnode *BlitNode;
```

Description

Cette fonction permet de lier un requester pour l'utilisation du Blitter à la fin de la queue actuelle du Blitter.

Paramètres

BlitNode: Pointeur sur une structure BlitNode.

Voir aussi

QBSBlit()

QBSBlit**Synchroniser le Blitter****Syntaxe**

```
QBSBlit (BltNode)
-294
struct bltnode *BltNode;
```

Description

Cette fonction synchronise le request Blitter avec le faisceau vidéo.

Paramètres

BltNode: Pointeur sur une structure BltNode.

Voir aussi

QBlit()

WaitBlit**Attendre la fin du travail du Blitter****Syntaxe**

```
WaitBlit()
```

Description

Cette fonction suspend l'exécution du programme jusqu'à ce que le Blitter ait achevé son travail. Mais attention : il peut arriver que le programme se relance alors que le Blitter n'a même pas commencé son travail. Cela est dû à un défaut du processeur qui se manifeste surtout en Hi-Res avec 4 plans de bits.

Structure

Offset	Structure
-----	-----
0x00 0	struct bltnode <hardware/blit.h>
0x04 4	{
0x08 8	struct bltnode *n;
0x0a 10	int (*function)();
0x0c 12	char stat;
0x0e 14	short bltsize;
0x12 18	short beamsync;
	int (*cleanup)();
	}

6. Les commandes Copper

CBump

Incrémenter le pointeur de la Copper-List Utilisateur

Syntaxe

```
CBump (UCopList)
-366      A1
struct UCopList *UCopList;
```

Description

Cette fonction incrémente le pointeur de la liste Copper utilisateur.

Paramètres

UCopList: Pointeur sur une structure UCopList.

Voir aussi

CMove(), CWait, UCopListInit

CEND

Déterminer une Copper-List Utilisateur

Syntaxe

```
CEND (UCopList)
(Macro)
struct UCopList *UCopList;
```

Description

Cette macro termine une liste d'instructions Copper utilisateur.

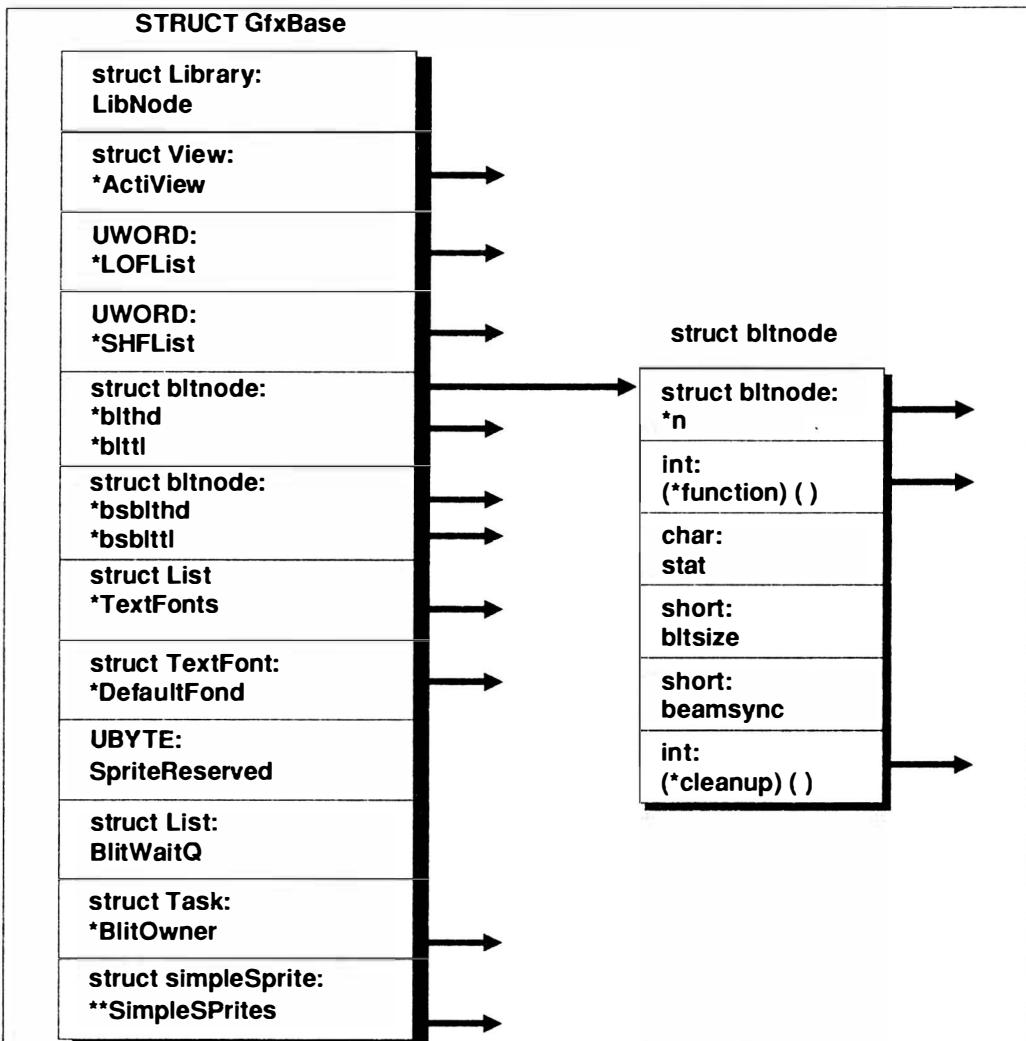


Figure 6 - 64

Paramètres

UCopList: Pointeur sur une structure UCopList.

Voir aussi

CMove(), CWait(), UCopListInit()

CMove**Charger un registre dans une Copper-List Utilisateur****Syntaxe**

```
CMove (UCopList, Register, Valeur)
-372      A1      D0      D1
struct UCopList *UCopList;
APTR      Register;
SHORT     Valeur;
```

Description

Cette fonction ajoute une instruction de déplacement Copper à la liste utilisateur.

Paramètres

UCopList: Pointeur sur une structure UCopList.

Register: Un numéro de registre hardware.

Valeur: Une valeur 16 bits devant être écrite dans ce registre.

Voir aussi

UCopListInit(), CWait

CWait**Attendre le faisceau****Syntaxe**

```
CWait (UCopList, Y, X)
-378      A1      D0      D1
struct UCopList *UCopList;
SHORT     Y,X;
```

Description

Cette fonction ajoute une instruction d'attente Copper à la liste utilisateur

Paramètres

UCopList: Pointeur sur une structure UCopList.

Voir aussi

CMove(), CWait(), UCopListInit()

FreeCopList**Libérer la mémoire d'une Copper-List****Syntaxe**

```
FreeCopList (CopList)
    -546      A0
struct coplist *CopList;
```

Description

Cette fonction libère la mémoire affectée aux instructions intermédiaires de la liste Copper.

Paramètres

CopList: Pointeur sur une structure CopList.

Voir aussi

[FreeVPortCopLists\(\)](#)

FreeCprList**Libérer la mémoire d'une Copper-List****Syntaxe**

```
FreeCprList (CprList)
    -564      A0
struct cprlist *CprList;
```

Description

Cette fonction libère la mémoire affectée aux instructions Copper hardware.

Paramètres

CprList: Pointeur sur une structure CprList.

Voir aussi

[MrgCop\(\)](#)

FreeVPortCopLists**Libérer la mémoire des Copper-List****Syntaxe**

```
FreeVPortCopLists (ViewPort)
    -540      A0
struct ViewPort *ViewPort;
```

Description

Cette fonction libère la mémoire affectée à toutes les listes Copper intermédiaires et leurs en-têtes.

Paramètres

ViewPort: Pointeur sur une structure ViewPort.

Voir aussi

MakeVPort(), FreeCopList()

InitView

Initialiser une structure View

Syntaxe

```
InitView (View)
-360      A1
struct View *View;
```

Description

Cette fonction initialise une structure View. Toutes les variables et pointeurs de la structure sont mis à 0.

Paramètres

View: Pointeur sur une structure View.

Voir aussi

MrgCop()

InitVPort

Annuler les variables de la structure ViewPort

Syntaxe

```
InitVPort (ViewPort)
-204      A0
struct ViewPort *ViewPort;
```

Description

Cette fonction annule toutes les variables et pointeurs de la structure ViewPort.

Paramètres

ViewPort: Pointeur sur une structure ViewPort

Voir aussi

[MakeVPort\(\)](#), [MrgCop\(\)](#), [LoadView\(\)](#)

LoadView**Afficher les ViewPort****Syntaxe**

```
LoadView (View)
-222      A1
struct View *View;
```

Description

Le View indiqué et tous les ViewPort internes sont affichés à l'écran.

Paramètres

View: Pointeur sur une structure View.

Voir aussi

[InitView\(\)](#), [InitVPort\(\)](#), [MakeVPort\(\)](#), [MrgCop\(\)](#)

MakeVPort**Générer une Copper-List****Syntaxe**

```
MakeVPort (View, ViewPort)
-216      A0      A1
struct View      *View;
struct ViewPort *ViewPort;
```

Description

Cette fonction génère les listes Copper du ViewPort en tenant compte du View dont dépend le ViewPort.

Paramètres

View: Pointeur sur une structure View.

ViewPort: Pointeur sur une structure ViewPort.

Voir aussi

[MrgCop\(\)](#), [LoadView\(\)](#)

MrgCop**Calculer une Copper-List****Syntaxe**

```
MrgCop (View)
-210      A1
struct View *View;
```

Description

Cette fonction calcule et regroupe les instructions Copper utilisateur d'affichage, de couleur et de sprite dans un seul groupe d'instructions Copper.

Paramètres

View: Pointeur sur une structure View.

Voir aussi

MakeVPort(), LoadView()

ScrollVPort**Scroller un ViewPort****Syntaxe**

```
ScrollVPort (ViewPort)
-588      A0
struct ViewPort *ViewPort;
```

Description

Une fois que l'utilisateur a fixé les valeurs de scrolling dans la structure RasInfo du ViewPort, la fonction ViewPort recalcule la Copper-List pour la nouvelle position du Bitmap.

Paramètres

ViewPort: Pointeur sur la structure ViewPort.

UCopListInit**Initialiser une Copper-List utilisateur****Syntaxe**

```
(UCopList *) UCopListInit (CopperList, CountCommands)
-594      A0      D0
struct UCopList *UCopList;
struct UCopList *CopperList;
SHORT      CountCommands;
```

Description

Cette fonction permet d'initialiser une Copper-List utilisateur (CopperList != 0) ou une nouvelle liste (CopperList =0).

Paramètres

CopperList: La liste utilisateur à réinitialiser

CountCommands: Le nombre de commandes de la Copper-List

Voir aussi

CBump(), CMove(), CWait()

VBeamPos

Ligne actuelle du faisceau

Syntaxe

```
Position = VBeamPos()
-384
ULONG Position;
```

Description

Cette fonction fournit la position verticale du faisceau électronique à cet instant

Retour

Position: Contient la valeur numérique de la position (0-255). Comme le faisceau peut en réalité balayer 262 lignes, les positions entre 256 et 262 sont signalées par les valeurs 0-6.

WaitBOVP

Attendre le faisceau à une position du ViewPort

Syntaxe

```
WaitBOVP (ViewPort)
-402      A0
struct ViewPort *ViewPort;
```

Description

Cette fonction permet d'interrompre l'exécution du programme jusqu'à ce que le faisceau ait atteint la dernière ligne du ViewPort spécifié.

Paramètres

ViewPort: Pointeur sur une structure ViewPort

WaitTOF**Attendre le début du Refresh écran****Syntaxe**

```
WaitTOF()
270
```

Description

Cette fonction attend que le faisceau ait atteint le haut du prochain frame vidéo.

Structure

Offset	Structure
-----	-----
	struct UCopList <graphics/copper.h>
	{
0x00 0	struct UCopList *Next;
0x04 4	struct CopList *FirstCopList;
0x08 8	struct CopList *CopList;
0x0c 12	}
	struct CopList <graphics/copper.h>
	{
0x00 0	struct CopList *Next;
0x04 4	struct CopList *_CopList;
0x08 8	struct ViewPort *_ViewPort;
0x0c 12	struct CopIns *CopIns;
0x10 16	struct CopIns *CopPtr;
0x14 20	UWORD *CopLStart;
0x18 24	UWORD *CopSStart;
0x1c 28	SHORT Count;
0x1e 30	SHORT MaxCount;
0x20 32	SHORT DyOffset;
0x22 34	}
	struct cpplist <graphics/copper.h>
	{
0x00 0	struct cpplist *Next;
0x04 4	UWORD *start;
0x08 8	SHORT MaxCount;
0x0a 10	}
	struct ViewPort <graphics/view.h>
	{
0x00 0	struct ViewPort *Next;
0x04 4	struct ColorMap *ColorMap;
0x08 8	struct CopList *DspIns;
0x0c 12	struct CopList *SprIns;
0x10 16	struct CopList *ClrIns;
0x14 20	struct UCopList *UCopIns;
0x18 24	SHORT DWidth,
0x1a 26	DHeight;
0x1c 28	SHORT DxOffset,
0x1e 30	DyOffset;
0x20 32	UWORD Modes;
0x22 34	UBYTE SpritePriorities;
0x23 35	UBYTE reserved;
0x24 36	struct RasInfo *RasInfo;
0x28 40	}

```

        struct RasInfo <graphics/view.h>
    {
0x00 0      struct RasInfo *Next;
0x04 4      struct BitMap *BitMap;
0x08 8      SHORT RxOffset,
0x0a 10     RyOffset;
0x0c 12     }

        struct View
    {
0x00 0      struct ViewPort *ViewPort;
0x04 4      struct cprlist *LOFCprList;
0x08 8      struct cprlist *SHFCprList;
0x0c 12     short DxOffset,
0x0e 14     DyOffset;
0x10 16     UWORLD Modes;
0x14 20     }

```

7. Les fonctions Layer

AndRectRegion

ET logique sur une région

Syntaxe

```

AndRectRegion (Region, Rectangle)
    -504          A0          A1
    struct Region    *Region;
    struct Rectangle *Rectangle;

```

Description

Cette fonction exécute une opération And à deux dimensions d'un rectangle avec une zone, en laissant le résultat dans la zone.

Paramètres

Region: Pointeur sur une structure Region

Rectangle: Pointeur sur une structure Rectangle

Voir aussi

OrRectRegion(), XorRectRegion(), OrRegionRegion()

AndRegionRegion

ET logique sur deux régions

Syntaxe

```

Status = AndRegionRegion (Region1, Region2)
    D0          -624          A0          A1

```

Description

Cette fonction exécute une opération And à deux dimensions entre deux régions, en laissant le résultat dans la seconde région.

Paramètres

Region1: Pointeur sur une structure Region

Region2: Pointeur sur la structure Region destinée à contenir le résultat

Retour

Status: TRUE si l'opération réussit, FALSE si la mémoire est saturée

AttemptLockLayerRom	Etat actuel d'un Layer
----------------------------	-------------------------------

Syntaxe

Status = AttemptLockLayerRom (Layer)
d0 -654 a5

Description

Cette fonction prend note de l'état d'activation du layer. Si le layer est déjà verrouillé, la fonction renvoie FALSE. Sinon, la fonction verrouille le layer et renvoie TRUE.

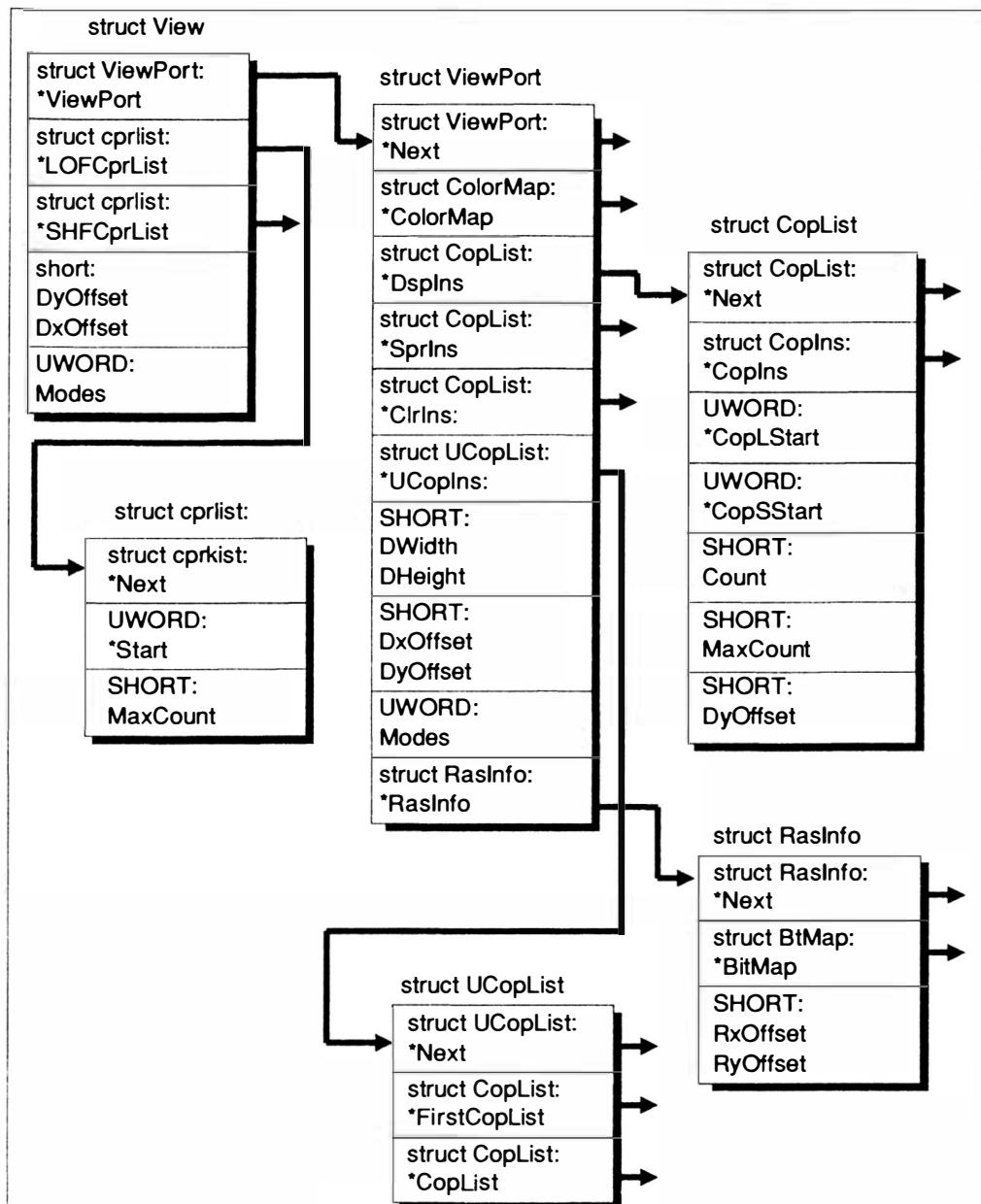


Figure 6 - 65

Paramètres

Layer: Pointeur sur une structure Layer

Retour

Status: FALSE ou TRUE. Voir la description.

ClearRectRegion**Effacer une région d'un rectangle****Syntaxe**

```
Status = ClearRectRegion (Region, Rectangle)
      DO          -522           A0          A1
      BOOL    Status;
      struct Region   *Region;
      struct Rectangle *Rectangle;
```

Description

Cette fonction effectue une opération CLEAR en 2D entre un rectangle et une région, en laissant le résultat dans la région. Toutes les zones de la région qui sont aussi dans le rectangle sont effacées dans la région.

Paramètres

Region: Pointeur sur une structure Region

Rectangle: Pointeur sur une structure Rectangle

Voir aussi

ClearRegion()

ClearRegion**Effacer une région****Syntaxe**

```
ClearRegion (Region)
      -528           A0
      struct Region *Region;
```

Description

Cette fonction supprime tous les rectangles de la région.

Paramètres

Region: Pointeur sur une structure Region

Voir aussi

AndRectRegion(), OrRectRegion()

CopySBitMap**Copier une partie d'un SuperBitmap****Syntaxe**

```
CopySBitMap (Layer)
    -450      A0
    struct Layer *Layer;
```

Description

C'est l'inverse de la fonction SyncSBitMap. Cette fonction copie une partie d'un SuperBitmap affecté à un layer dans un layer bitmap.

Paramètres

Layer: Pointeur sur un layer SuperBitmap. Le layer doit auparavant être verrouillé par l'appelant.

Voir aussi

SyncSBitMap()

DisposeRegion**Libérer une région****Syntaxe**

```
DisposeRegion (Region)
    -534      A0
    struct Region *Region;
```

Description

Cette fonction libère tous les rectangles de la région puis la région elle-même.

Paramètres

Region: Pointeur sur une structure Region

Voir aussi

NewRegion()

LockLayerRom**Verrouiller un Layer****Syntaxe**

```
LockLayerRom (Layer)
    -432      A5
    struct Layer *Layer;
```

Description

Cette fonction verrouille une structure Layer en utilisant le code ROM. En retour, le layer est verrouillé pour un usage exclusif par la tâche appelante.

Paramètres

Layer: Pointeur sur une structure Layer

Voir aussi

UnLockLayerRom()

NewRegion

Créer une région

Syntaxe

```
Region = NewRegion ()
      D0      -516
      struct Region *Region;
```

Description

Cette fonction crée une structure de région, l'initialise et renvoie un pointeur correspondant.

Retour

Region: Pointeur sur la région initialisée. Si la fonction ne peut pas allouer la mémoire nécessaire, Region=NULL.

Voir aussi

OrRectRegion(), DisposeRegion()

OrRectRegion

OR logique entre une région et un rectangle

Syntaxe

```
Status = OrRectRegion (Region, Rectangle)
      -510      A0      A1
      BOOL      Status;
      struct Region *Region;
      struct Rectangle *Rectangle;
```

Description

Cette fonction exécute une opération OR entre un rectangle et une région, en laissant le résultat dans la région. Lorsqu'une portion du rectangle n'est pas dans la région, elle est alors ajoutée dans cette région.

Paramètres

Region: Pointeur sur une structure Region

Rectangle: Pointeur sur une structure Rectangle

Retour

Status:TRUE si l'opération réussit, FALSE en cas de problème de mémoire.

Voir aussi

AndRectRegion(), XorRectRegion()

OrRegionRegion**OU logique entre deux région****Syntaxe**

```
Status = OrRegionRegion (Region1, Region2)
DO           -612
BOOL Status;
struct Region *Region1, *Region2;
```

Description

Cette fonction exécute une opération OR en 2D entre 2 régions, en laissant le résultat dans la seconde région.

Paramètres

Region1,Region2: Pointeurs sur une structure Region

Retour

Status: TRUE ou FALSE selon le résultat

Voir aussi

NewRegion(), DisposeRegion()

SyncSBitMap**Sauver des Layers****Syntaxe**

```
SyncSBitMap (Layer)
-444      A0
struct Layer *Layer;
```

Description

Cette fonction copie tous les bits de ClipRects dans Layer vers SuperBitmap.

Paramètres

Layer: Pointeur sur un Layer qui a un SuperBitmap

Voir aussi

[CopySBitMap\(\)](#)

UnlockLayerRom**Déverrouiller un Layer**

Syntaxe

```
UnlockLayerRom (Layer)
    -438          A5
    struct Layer *Layer;
```

Description

Cette fonction annule le verrouillage du Layer par un code Rom.

Paramètres

Layer: Pointeur sur une structure Layer

Voir aussi

[LockLayerRom\(\)](#)

XorRectRegion**OU exclusif entre un rectangle et une région**

Syntaxe

```
Status = XorRectRegion (Region, Rectangle)
    0           -558          A0          A1
    BOOL        Status;
    struct Region *Region;
    struct Rectangle *Rectangle;
```

Description

Cette fonction exécute une opération XOR en 2D d'un rectangle avec une région, en laissant le résultat dans la région.

Paramètres

Region: Pointeur sur une structure Region

Rectangle: Pointeur sur une structure Rectangle

Retour

Status: TRUE ou FALSE selon le résultat

Voir aussi

OrRectRegion(), AndRectRegion()

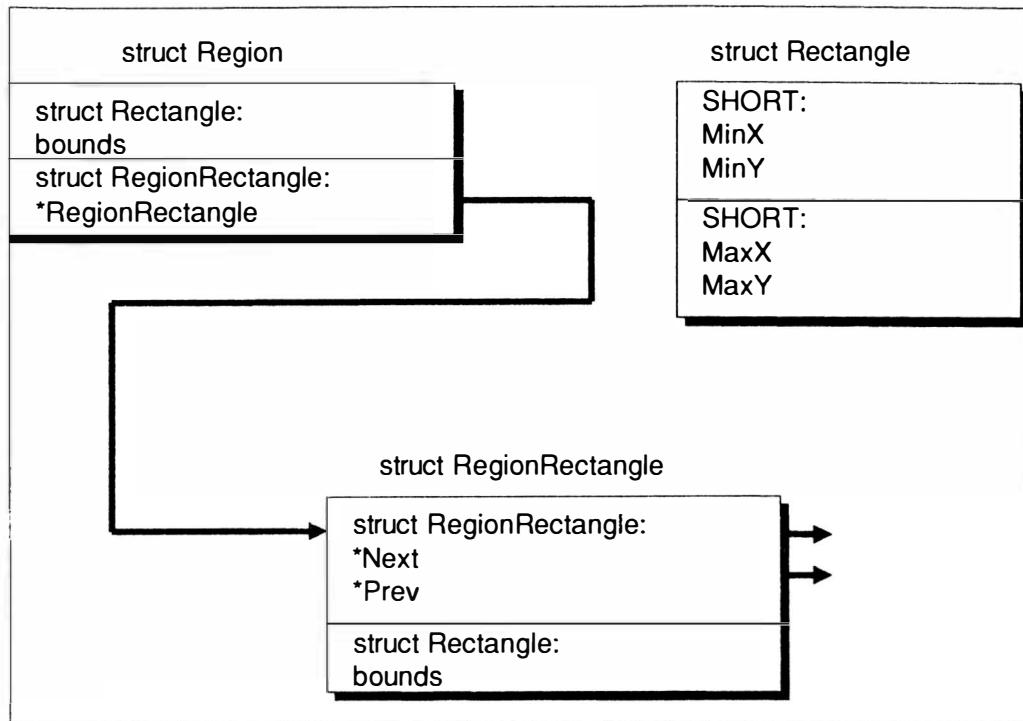


Figure 6 - 66

XorRegionRegion**OU exclusif entre deux régions****Syntaxe**

XorRegionRegion	(Region1,	Region2)
	-558	A0
		A1

Description

Cette fonction exécute une opération 2D d'une région avec une seconde région, en laissant le résultat dans la seconde région.

Paramètres

Region1,Region2: Pointeurs sur une structure Region

Retour

Status: TRUE ou FALSE selon le résultat

Voir aussi

OrRegionRegion()

Structure

```
Offset   Structure
-----
          struct Rectangle <graphics/gfx.h>
{
 0x00    0      SHORT MinX,
 0x02    2      MinY;
 0x04    4      SHORT MaxX,
 0x06    6      MaxY;
 0x08    8      }
          struct RegionRectangle <graphics/regions.h>
{
 0x00    0      struct RegionRectangle *Next,
 0x04    4      *Prev;
 0x08    8      struct Rectangle bounds;
 0x10    16     }
          struct Region <graphics/region.h>
{
 0x00    0      struct Rectangle bounds;
 0x08    8      struct RegionRectangle *RegionRectangle;
 0x0c    12     }
```

8. Manipulation des caractères**AddFont****Ajouter une nouvelle fonte****Syntaxe**

```
AddFont (TextFont)
-480      A1
struct TextFont *TextFont;
```

Description

Cette fonction ajoute la fonte à la liste système. La fonte est ensuite accessible aux applications.

Paramètres

TextFont: Une structure TextFont dans la RAM Public

Voir aussi

RemFont(), OpenFont(), OpenDiskFont()

AskFont**Attribut de la fonte courante****Syntaxe**

```
AskFont (RastPort, TextAttr)
    -474      A1      A0
    struct RastPort *RastPort;
    struct TextAttr *TextAttr;
```

Description

Cette fonction lit les attributs texte de la fonte en cours et les répercute dans la structure des attributs texte du RastPort.

Paramètres

RastPort: Le RastPort d'où les attributs texte sont extraits

TextAttr: La structure TextAttr à remplir.

Voir aussi

[SetFont\(\)](#)

AskSoftStyle**Attribut de texte de la fonte courante****Syntaxe**

```
Ok = AskSoftStyle (RastPort)
    D0      -84      A1
    ULONG      Ok;
    struct RastPort *RastPort;
```

Description

Cette fonction renvoie les bits de style de la fonte en cours qui ne font pas partie de la fonte intrinsèquement, mais qui sont générés par algorithme. On peut alors utiliser ces bits de style en tant que masque valide pour SetSoftStyle.

Chaque fonte système possède une série de bits de style. Les bits de style par défaut d'une fonte sont donnés dans le paramètre Tf_Style (1 octet) de la structure TextFont. Voici les bits correspondants :

Style	= Octet de style
FSF_UNDERLINED	= 00000001 (Souligné = 1)
FSF_BOLD	= 00000010 (Gras = 2)
FSF_ITALIC	= 00000100 (Italique = 4)
FSF_NORMAL	= 00000000 (Normal = 0)

Paramètres

RastPort: Le RastPort d'où la fonte et le style sont extraits

Retour

Ok: Les bits de style. Cette variable peut être ensuite utilisée en tant que masque.

Voir aussi

[SetSoftStyle\(\)](#)

CloseFont**Fermer un fichier de fonte****Syntaxe**

```
CloseFont (TextFont)
    -78      A1
struct TextFont *TextFont;
```

Description

Cette fonction indique que la fonte mentionnée n'est plus utilisée. Les ressources système sont ainsi libérées.

Paramètres

TextFont: Pointeur Font

Voir aussi

[OpenFont\(\)](#), [AddFont\(\)](#)

OpenFont**Ouvrir un fichier de fonte****Syntaxe**

```
TextFont = OpenFont (TextAttr)
    D0      -72      A0
struct TextFont *TextFont;
struct TextAttr *TextAttr;
```

Description

Cette fonction autorise l'accès à une fonte en recherchant l'espace mémoire adéquat pour la fonte et en retournant un pointeur sur la fonte système.

Paramètres

TextAttr: Une structure TextAttr qui décrit les attributs de la fonte texte

Retour

TextFont: Pointeur sur la structure de la fonte ou sur celle qui en est la plus proche lorsque les attributs ne sont pas trouvés.

Voir aussi

`CloseFont()`, `AddFont()`, `RemFont()`

RemFont**Supprimer une fonte de la liste système****Syntaxe**

```
RemFont (TextFont)
-486      A1
struct TextFont *TextFont;
```

Description

Cette fonction supprime une fonte de la liste des fontes système.

Paramètres

TextFont: Pointeur sur la structure de la fonte

Voir aussi

`AddFont()`

SetFont**Définir la fonte du RastPort****Syntaxe**

```
SetFont (RastPort, TextFont)
-66      A1      A0
struct RastPort *RastPort;
struct TextFont *TextFont;
```

Description

Définit la fonte texte et les attributs dans un RastPort.

Paramètres

RastPort: Pointeur sur une structure Rastport

TextFont: Pointeur sur une structure TextFont

Voir aussi

`OpenFont()`, `CloseFont()`

SetSoftStyle**Définir les attributs de la fonte****Syntaxe**

```
Newstyle = SetSoftStyle (RastPort,Style,Enable)
      D0          -90        A1          D0
      D1
      ULONG      Newstyle;
      struct RastPort *RastPort;
      ULONG      Style;
      ULONG      Ok;
```

Description

Cette fonction définit le style de la fonte en cours.

Paramètres

- RastPort: Le Rastport d'où la fonte et le style sont extraits
 Style: le style à définir
 Enable: Les bits de style à changer

Retour

Newstyle: Le style résultant

Voir aussi

AskSoftStyle()

Structure

Offsets	Structure
-----	-----
0x00	struct TextAttr <graphics/text.h>
0x04	{
0x00	STRPTR ta_Name;
0x04	UWORD ta_YSize;
0x06	UBYTE ta_Style;
0x07	UBYTE ta_Flags;
0x08	}
0x00	struct TextFont <graphics/text.h>
0x04	{
0x00	struct Message tf_Message;
0x14	UWORD tf_YSize;
0x16	UBYTE tf_Style;
0x17	UBYTE tf_Flags;
0x18	UWORD tf_XSize;
0x1a	UWORD tf_Baseline;
0x1c	UWORD tf_BoldSmear;
0x1e	UWORD tf_Accessors;
0x20	UBYTE tf_LoChar;
0x21	UBYTE tf_HiChar;

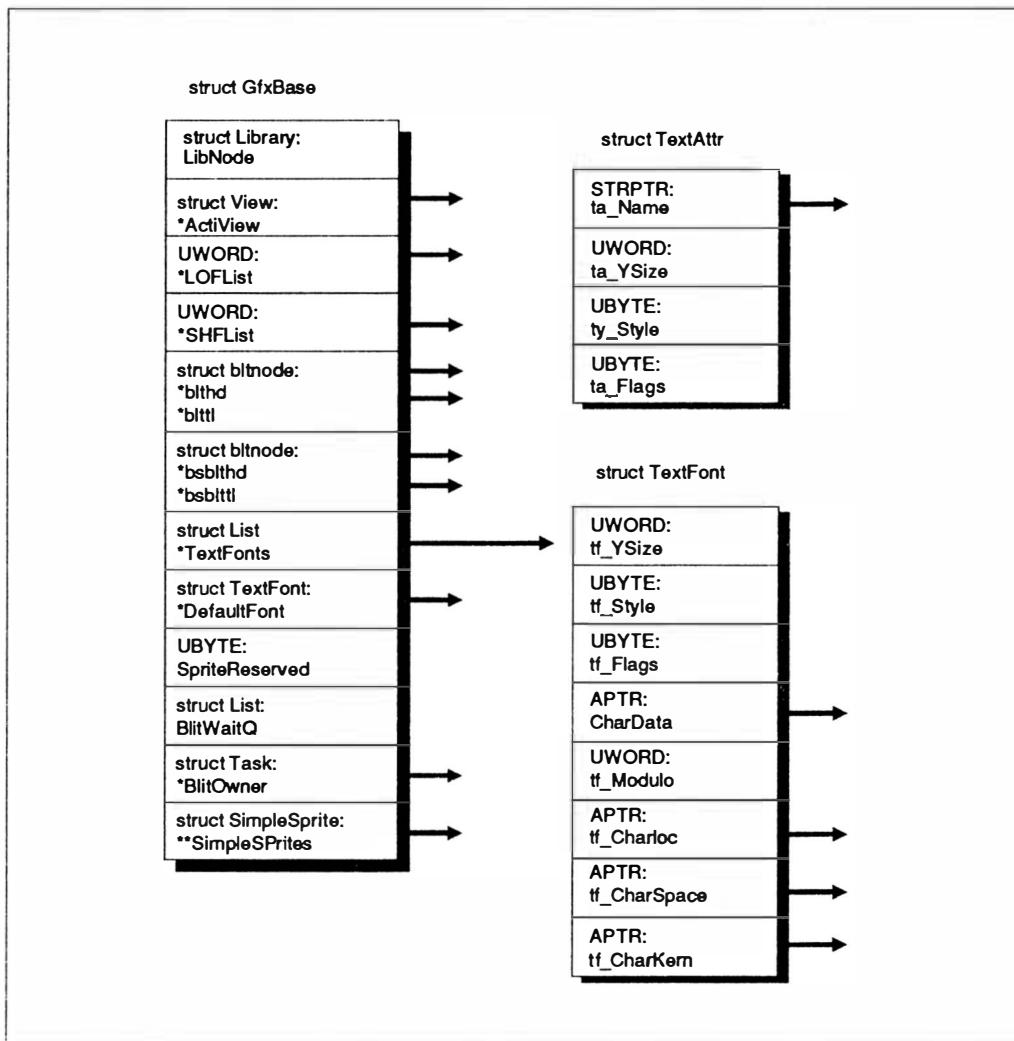


Figure 6 - 67

0x22	34	APTR tf_CharData;
0x26	38	UWORD tf_Modulo;
0x28	40	APTR tf_CharLoc;
0x2c	44	APTR tf_CharSpace;
0x30	48	APTR tf_Charcern;
0x34	52	}

9. GELs et Sprites

AddAnimOb

Classer des BOB pour une animation

Syntaxe

```
AddAnimOb (AnimOb, AnimKey, RastPort)
    -156      A0      A1      A2
    struct AnimOb *AnimOb;
    struct AnimOb *AnimKey;
    struct RastPort *RastPort;
```

Description

Cette routine classe tous les bobs d'un objet d'animation, ou plus précisement tous les bobs de ses composantes d'animation, dans la liste GEL du RastPort.

Paramètres

- AnimOb: Pointeur sur une structure AnimOb à ajouter à la liste
- AnimKey: Adresse du pointeur sur le premier AnimOb de la liste (NULL s'il n'y a plus d'AnimOb).
- RastPort: Pointeur sur un RastPort valide

Voir aussi

Animate(), DrawGLList()

AddBob

Ajouter un BOB

Syntaxe

```
AddBob (Bob, RastPort)
    -96      A0      A1
    struct Bob *Bob;
    struct RastPort *RastPort;
```

Description

Cette fonction ajoute un bob à la liste GEL en cours.

Paramètres

- Bob: Pointeur sur la structure Bob à ajouter à la liste Gel
- RastPort: Pointeur sur une structure Rastport

Voir aussi

AddVSprite(), DrawGLList()

AddVSprite**Ajouter un Sprite****Syntaxe**

```
AddVSprite (VSprite, RastPort)
    -102      A0      A1
    struct VSprite *VSprite;
    struct RastPort *RastPort;
```

Description

Cette fonction ajoute un VSprite à la liste GEL en cours.

Paramètres

VSprite: Pointeur sur la structure VSprite à ajouter à la liste

RastPort: Pointeur sur une structure Rastport

Voir aussi

[DrawGLList\(\)](#)

Animate**Animer des BOB****Syntaxe**

```
Animate (AnimKey, RastPort)
    -162      A0      A1
    struct AnimOb *AnimKey;
    struct RastPort *RastPort;
```

Description

Cette fonction traite tous les AnimOb de la liste d'animation en cours.

Paramètres

AnimKey: Adresse de la variable qui pointe sur le premier AnimOb

RastPort: Pointeur sur une structure Rastport

Voir aussi

[AddAnimOb\(\)](#)

ChangeSprite**Changer de Sprite****Syntaxe**

```
ChangeSprite (ViewPort, Sprite, SpriteData)
    -420      A0      A1      A2
    struct ViewPort *ViewPort;
    struct Sprite *Sprite;
    struct SpriteData *SpriteData;
```

```
struct ViewPort      *ViewPort;
struct SimpleSprite *Sprite;
struct SpriteData   *SpriteData;
```

Description

L'image sprite est changée pour utiliser les données commençant à SpriteData.

```
struct SpriteData
{
    UWORLD poscl[2];
    UWORLD Data[Hauteur][2];
    UWORLD Reserved[2]; /* = 0,0 */
}
```

Paramètres

- | | |
|-------------|--|
| Viewport: | Pointeur sur une structure ViewPort |
| Sprite: | Pointeur sur une structure SimpleSprite |
| SpriteData: | Pointeur sur une structure de données de la forme décrite plus haut. |

Voir aussi

[GetSprite\(\)](#), [FreeSprite\(\)](#), [MoveSprite\(\)](#)

DoCollision

Tester une collision

Syntaxe	DoCollision (RastPort) -108 A1 struct RastPort *RastPort;
---------	---

Description

Cette fonction teste chaque GEL dans la liste des GEL pour les collisions.

Paramètres

- | | |
|-----------|-------------------------------------|
| RastPort: | Pointeur sur une structure Rastport |
|-----------|-------------------------------------|

Voir aussi

[SortGList\(\)](#), [SetCollision\(\)](#)

DrawGList

Créer la liste des objets graphiques

Syntaxe	DrawGList (RastPort, ViewPort) -114 A1 A0
---------	--

```
struct RastPort *RastPort;
struct ViewPort *ViewPort;
```

Description

Cette fonction traite la liste Gel, construit les VSprites dans la liste Copper, dessine les Bobs.

Paramètres

- | | |
|-----------|--|
| RastPort: | Pointeur sur la structure Rastport où les Bobs seront dessinés |
| ViewPort: | Pointeur sur la structure ViewPort pour laquelle les VSprites seront créés |

Voir aussi

[MakeVPort](#), [MrgCop\(\)](#), [LoadView\(\)](#)

FreeGBuffers

Libérer la mémoire des objets graphique

Syntaxe

```
FreeGBuffers (AnimObject, RastPort, DoubleBuffer)
              -600          A0          A1          D0
              struct AnimOb  *AnimObject;
              struct RastPort *RastPort;
              BOOL           DoubleBuffer;
```

Description

Cette fonction désalloue la mémoire obtenue par GetBuffers

Paramètres

- | | |
|---------------|--|
| AnimObject: | Pointeur sur une structure AnimOb |
| RastPort: | Pointeur sur une structure Rastport |
| DoubleBuffer: | Indicateur de double buffer (mettre à TRUE pour produire un double buffer) |

Voir aussi

[GetGBuffers\(\)](#), [InitGBuffers\(\)](#)

FreeSprite**Libérer un sprite****Syntaxe**

```
FreeSprite (Nombre)
  -414          D0
  SHORT Nombre;
```

Description

Cette fonction restitue au système le sprite précédemment réservé pour un usage exclusif avec GetSprite().

Paramètres

Nombre: Numéro du sprite

Voir aussi

GetSprite(), ChangeSprite(), MoveSprite()

GetGBuffers**Allouer de la mémoire à un objet graphique****Syntaxe**

```
GetGBuffers (AnimObject, RastPort,DoubleBuffer)
  D0   -168      A0      A1      D0
  BOOL           Status;
  struct AnimOb *AnimObject;
  struct RastPort *RastPort;
  BOOL           DoubleBuffer;
```

Description

Cette fonction tente d'allouer tous les buffers d'un AnimObject complet.

Paramètres

AnimObject: Pointeur sur une structure AnimOb

RastPort: Pointeur sur une structure Rastport

DoubleBuffer: Indicateur de double buffer (mettre à TRUE pour produire un double buffer)

Retour

Status: TRUE si toutes les allocations mémoire ont réussi, FALSE sinon.

Voir aussi

FreeGBuffer(), InitGBuffers()

GetSprite**Déclarer un sprite****Syntaxe**

```

Nombre = GetSprite (Sprite, NumeroSprite)
      D0          -408      A0          D0
      SHORT        Nombre;
      struct SimpleSprite *Sprite;
      SHORT        NumeroSprite;

```

Description

Cette fonction permet de déclarer un sprite électronique personnel.

Paramètres

Sprite: Pointeur sur une structure SimpleSprite utilisateur

NumeroSprite: Numéro du sprite à réserver (0-7) ou -1 si peut vous importe quel sprite sera attribué

Retour

Nombre: Numéro du sprite reçu

Voir aussi

FreeSprite(), *ChangeSprite()*

InitAnimate**Initialiser une séquence d'animation****Syntaxe**

```

InitAnimate (AnimKey)
(Macro)
struct AnimOb *AnimKey;

```

Description

Cette macro permet d'initialiser une séquence d'animation.

Paramètres

AnimKey: Pointeur sur une structure AnimOb

Voir aussi

AddAnimOb()

InitGels**Initialiser la structure GelsInfo****Syntaxe**

```
InitGels (DebutListe, FinListe, GelsInfo)
    -120          A0          A1          A2
    struct VSprite *DebutListe,
                *FinListe;
    struct GelsInfo *GelsInfo;
```

Description

Cette fonction initialise la structure GelsInfo spécifiée. Cette structure est nécessaire pour que tous les VSprites et Bobs puissent être correctement dessinés.

Paramètres

DebutListe,FinListe:

Structures VSprite d'une liste

GelsInfo:

Pointeur sur la structure GelsInfo devant être initialisée

Voir aussi

SortGList()

InitGMasks**Initialiser les masques d'une AnimObject****Syntaxe**

```
InitGMasks (AnimOb)
    -174          A0
    struct AnimOb *AnimOb;
```

Description

Cette fonction initialise tous les masques d'une AnimObject. InitMasks est appelé pour chaque séquence de chaque composante.

Paramètres

AnimOb: Pointeur sur la structure AnimOb

Voir aussi

InitMasks(), GetGBuffers(), FreeGBuffers()

InitMasks**Initialiser des variables graphique****Syntaxe**

```
InitMasks (Vsprite)
    -126      A0
    struct Vsprite *Vsprite;
```

Description

Cette fonction initialise BorderLine et CollMask de la structure VSprite spécifiée.

Voir aussi

[InitGels\(\)](#)

MoveSprite**Déplacer un sprite****Syntaxe**

```
MoveSprite(ViewPort, Sprite, x, y)
    -426      A0      A1      D0 D1
    struct ViewPort *ViewPort;
    struct SimpleSprite *Sprite;
    SHORT      x,y;
```

Description

Cette fonction permet de déplacer un Sprite vers un point relatif à un ViewPort.

Paramètres

- ViewPort: Pointeur sur une structure ViewPort
- Sprite: Pointeur sur une structure SimpleSprite
- x,y: Nouvelle position relativement au ViewPort

Voir aussi

[ChangeSprite\(\)](#), [GetSprite\(\)](#)

RemBob**Supprimer un bob****Syntaxe**

```
RemBob (Bob)
(Macro)
struct Bob *Bob;
```

Description

Cette fonction permet de supprimer un Bob d'une liste Gel.

Paramètres

Bob: Pointeur sur le Bob à supprimer

Voir aussi

AddBob()

RemIBob

Supprimer un bob

Syntaxe

```
RemIBob (Bob, RastPort, ViewPort)
        -132      A0      A1      A2
        struct Bob      *Bob;
        struct RastPort *RastPort;
        struct ViewPort *ViewPort;
```

Description

Cette fonction permet de supprimer immédiatement un Bob de la liste Gel et du RastPort.

Paramètres

Bob: Pointeur sur le Bob à supprimer

RastPort: Pointeur sur la RastPort si le Bob doit être effacé

ViewPort: Pointeur sur le ViewPort pour une synchro du faisceau

Voir aussi

RemBob()

RemVSprite

Supprimer un VSprite

Syntaxe

```
RemVSprite(VSprite)
        -138      A0
        struct VSprite *VSprite;
```

Description

Cette fonction permet de supprimer un VSprite de la liste Gel

Paramètres

VSprite: Pointeur sur la structure VSprite à supprimer

Voir aussi

AddVSprite()

SetCollision**Définir une collision****Syntaxe**

```
SetCollision (Num, Routine, GelsInfo)
    -144      D0      A0      A1
    ULONG      Num;
    VOID       (*Routine())
    struct GelsInfo *GelsInfo;
```

Description

Cette fonction permet de définir un pointeur sur une routine de collision utilisateur

Paramètres

Num: Numéro du vecteur de collision

Routine: Pointeur sur la routine de collision utilisateur

GelsInfo: Pointeur sur une structure GelsInfo

SortGLList**Trier la liste Gel****Syntaxe**

```
SortGLList (RastPort)
    -150      A1
    struct RastPort *RastPort;
```

Description

Cette fonction permet de trier la liste Gel en cours en fonction des coordonnées y,x.
Ce tri est essentiel avant l'appel de DrawGLList et DoCollision.

Paramètres

RastPort: Pointeur sur la structure RastPort contenant GelsInfo

Voir aussi

DrawGLList()

Structure

Offset	Structure

	struct VSprite <graphics/gels.h>
	{
0x00	0 struct VSprite *NextVSprite;
0x04	4 struct VSprite *PrevVsprite;
0x08	8 struct VSprite *DrawPath;
0x1c	12 struct VSprite *ClearPath;
0x10	16 WORD OldY.
0x12	18 OldX;
0x14	20 WORD Flags;
0x16	22 WORD Y.
0x18	24 X;
0x1a	26 WORD Height;
0x1c	28 WORD Width;
0x1e	30 WORD Depth;
0x20	32 WORD MeMask;
0x22	34 WORD HitMask;
0x26	38 WORD *ImageData;
0x2a	42 WORD *BorderLine;
0x2e	46 WORD *CollMask;
0x32	50 WORD *SprColors;
0x36	54 struct Bob *VSBob;
0x37	56 BYTE BlocePick;
0x38	57 BYTE BloceOnOff;
0x3a	58 VUserStuff VUserExt;
	}
	struct Bob <graphics/gels.h>
	{
0x00	0 WORD Flags;
0x02	2 WORD *SaveBuffer;
0x06	6 WORD *ImageShadow;
0xa	10 struct Bob *Before;
0xe	14 struct Bob *After;
0x12	18 struct VSprite *BobVSprite;
0x16	22 struct AnimComp *BobComp;
0x1a	26 struct DBuffPacket *DBuffer;
0x1e	30 BUserStuff BUserExt;
	}
	struct DBufPacket <graphics/gels.h>
	{
0x00	0 WORD BufY,
0x02	2 BufX;
0x04	4 struct VSprite *BufPath;
0x08	8 WORD *BufBuffer;
0x0c	12 }
	struct AnimComp <graphics/gels.h>
	{
0x00	0 WORD Flags;
0x02	2 WORD Timer;
0x04	4 WORD TimeSet;
0x06	6 struct AnimComp *NextComp;
0xa	10 struct AnimComp *PrevComp;
0xe	14 struct AnimComp *NextSeq;
0x12	18 struct AnimComp *PrevSeq;
0x14	22 WORD (*AnimCRoutine)();

```

0x1a 26      WORD XTrans;
0x1c 28      WORD YTrans;
0x1e 30      struct AnimOb *HeadOb;
0x22 34      struct Bob *AnimBob;
0x26 38 }
            struct AnimOb <graphics/gels.h>
{
0x00 0       struct AnimOb *NextOb,
0x04 4           *PrevOb;
0x08 8       LONG Clock;
0x0c 12      WORD AnOldY,
0x0e 14          AnOldX;
0x10 16      WORD AnY,
0x12 18          AnX;
0x14 20      WORD YVel,
0x16 22          XVel;
0x18 24      WORD YAccel,
0x1a 26          XAccel;
0x1c 28      WORD RingYTrans,
0x1e 30          RingXTrans;
0x20 32      WORD (*AnimORoutine)();
0x24 36      struct AnimComp *HeadComp;
0x28 40 }

            struct SimpleSprite <graphics/sprite.h>
{
0x00 0       UWORLD *posCtlData;
0x04 4       UWORLD height;
0x06 6       UWORLD x,
0x08 8           y;
0x0a 10      UWORLD num;
0x0c 12 }

            struct GeslsInfo <graphics/rastport.h>
{
0x00 0       UBYTE SprRsrvd;
0x01 1       UBYTE Flags;
0x02 2       struct VSprite *gelHead,
0x06 6           *gelTail;
0x0a 10      WORD *nextLine;
0x0e 14      WORD **lastColor;
0x12 18      struct collTable *collHandler;
0x16 22      short leftmost,
0x18 24          rightmost,
0x1a 26          topmost,
0x1c 28          bottommost;
0x1c 30      APTR firstBlissObj,
0x22 34          lastBlissObj;
0x26 38 }

            struct CollTable <graphics/gels.h>
{
0x00 0       int (*collPtrs[16])();
0x40 64 }
0x02 2       struct VSprite *gelHead,
0x06 6           *gelTail;
0x0a 10      WORD *nextLine;
0x0e 14      WORD **lastColor;
0x12 18      struct collTable *collHandler;
0x16 22      short leftmost,

```

```

0x18 24          rightmost,
0x1a 26          topmost,
0x1c 28          bottommost;
0x1c 30          APTR firstBlissObj,
0x22 34          lastBlissObj;
0x26 38      }
                  struct CollTable <graphics/gels.h>
{
0x00 0           int (*collPtrs[16])();
0x40 64      }

```

6.7. La librairie Diskfont

Cette librairie permet d'avoir accès aux fontes, quelles soient en mémoire ou bien sur disque.

Fonctions de la librairie Diskfont

AvailFonts	977
DisposeFontContents	978
NewFontContents	978
OpenDiskFont	979

AvailFonts

Connaître les fontes disponibles

Syntaxe

```

error = AvailFonts(buffer,bufBytes,types)
      DO          -36        A0      D0    D1
      LONG error;
      UBYTE *buffer;
      LONG bufBytes;
      LONG types;

```

Description

Cette fonction permet de connaître les fontes disponibles soit en mémoire, soit sur disque. Pour les fontes sur disque, il est nécessaire de les charger en mémoire et de les ouvrir avec la fonction OpenDiskFont(). Celles qui sont en mémoire peuvent être ouvertes avec la fonction OpenFont.

Paramètres

buffer: Emplacement mémoire contenant la liste des fontes disponibles, ainsi que leur nom.

bufTypes: Taille du buffer.

types: AFF_MEMORY pour chercher les fontes en mémoire et AFF_DISK pour celles sur disque.

Retour

error: Si non 0, taille nécessaire pour le buffer.

Voir aussi

La structure FontContents

DisposeFontContents

Libérer les entrées FontContents

Syntaxe

```
DisposeFontContents(fontContentsHeader)
        -48          A1
struct FontContentsHeader *fontContentsHeader;
```

Description

Cette fonction libère le tableau des entrées de FontContents retourné par la fonction NewFontContents.

Paramètres

fontContentsHeader:

Pointeur sur la structure FontContentsHeader retournées par la fonction NewFontContents.

Avertissement

Un appel de cette fonction sans un appel préalable de la fonction NewFontContents provoquera un effondrement du système.

Voir aussi

NewFontContents

NewFontContents

Créer les entrées FontContents

Syntaxe

```
NewFontContents(fontsLock,fontName)
D0      -42          A0          A1
struct FileLock *fontsLock;
UBYTE *fontName;
```

Description

Cette fonction crée un nouveau tableau pour toutes les entrées FontContents qui décrivent toutes les fontes associées à FontName.

Paramètres

fondsLock: Chemin d'accès au répertoire des fontes.

fontName: Nom de la fonte avec l'extension ".font".

Retour

fontContentsHeader:

Pointeur sur la structure FontContentsHeader.

Voir aussi

DisposeFontContents

OpenDiskFont

Rendre disponible une fonte du disque

Syntaxe

```
exception = OpenDiskFont(textAttr)
          DO      -30      A0
struct Font *exception;
struct TextAttr *textAttr;
```

Description

Cette fonction charge la fonte spécifiée en mémoire, et retourne un pointeur sur elle pour pouvoir l'utiliser avec les fonctions SetFont() et CloseFont(). Si la fonte est déjà en mémoire, elle ne sera pas rechargée. Attention, cette fonction doit toujours être associée avec la fonction CloseFont().

Paramètres

textAttr: structure TextAttr décrivant les attributs de la fonte.

Retour

exception: DO est à 0 si la fonte n'a pas été trouvée.

Voir aussi

CloseFont(), SetFont()

Structure

```
struct FontContentsHeader <libraries/diskfont.h>
{
Offset:
```

```
0x00 0  UWORD fch_FileID;
FCH_ID      0x0f00
Offset:
0x02 2  UWORD fch_NumEntries;
sizeof(struct FontContentsHeader) -
0x04 4                                     /* struct FontContents */
};
struct FontContents <libraries/diskfont.h>
{
Offset:
0x000 0  char  fc_FileName[MAXFONTPATH];
MAXFONTPATH 256
Offset:
0x100 256 UWORD fc_YSize;
Offset:
0x102 258 UBYTE fc_Style;
Offset:
0x103 259 UBYTE fc_Flags;
sizeof(struct FontContents) -
0x104 260
};
struct AvailFontsHeader <libraries/diskfont.h>
{
Offset:
0x00 0  UWORD afh_NumEntries;
sizeof(struct AvailFontsHeader) -
0x02 2                                     /* struct AvailFonts */
};
struct AvailFonts <libraries/diskfont.h>
{
Offset:
0x00 0  UWORD af_Type;
AFF_MEMORY 1
AFF_DISK   2
Offset:
0x02 2  struct  TextAttr af_Attr;
sizeof(struct AvailFonts) -
0x0A 10
};
struct TextAttr <graphics/text.h>
{
Offset:
0x00 0  STRPTR   ta_Name;
Offset:
0x04 4  UWORD    ta_YSize;
Offset:
0x06 6  UBYTE    ta_Style;
FS_NORMAL      0
FSF_EXTENDED   (1<<3)
FSF_ITALIC     (1<<2)
FSF_BOLD       (1<<1)
FSF_UNDERLINED (1<<0)
Offset:
0x07 7  UBYTE    ta_Flags;
FPF_ROMFONT   (1<<0)
FPF_DISKFONT  (1<<1)
FPF_REVPATH   (1<<2)
FPF_TALLDOT   (1<<3)
```

```

FPF_WIDEDOT (1<<4)
FPF_PROPORIONAL (1<<5)
FPF_DESIGNED (1<<6)
FPF_REMOVED (1<<7)
sizeof(struct TextAttr) -
0x08 8
};
struct TextFont <graphics/text.h>
{
Offset:
0x00 0 struct Message tf_Message;
Offset:
0x14 20 UWORD tf_YSize;
Offset:
0x16 22 UBYTE tf_Style;
0x17 23 UBYTE tf_Flags;
Offset:
0x18 24 UWORD tf_XSize;
Offset:
0x1A 26 UWORD tf_Baseline;
0x1C 28 UWORD tf_BoldSmear;
0x1E 30 UWORD tf_Accessors;
Offset:
0x20 32 UBYTE tf_LoChar;
0x21 33 UBYTE tf_HiChar;
Offset:
0x22 34 APTR tf_CharData;
0x26 38 UWORD tf_Modulo;
0x28 40 APTR tf_CharLoc;
0x2C 44 APTR tf_CharSpace;
sizeof(struct TextFont) -
0x34 52
};

```

6.8. Les librairies Mathématiques

Cette librairie permet de manipuler des nombres au format FFP (Fast Flotting Point, nombre en virgule flottante) et au format IEEE (standard international).

6.8.1. La librairie Mathffp

Fonctions de la Mathffp-Library

FFP	986
SPAbs	982
SPAdd	982
SPCmp	983
SPDiv	983
SPFix	983

SPFlt	984
SPMul	984
SPNeg	985
SPSub	985
SPTst	985

SPAbs**Calculer une valeur absolue****Syntaxe**

```
retour = SPAbs(valeur)
      D0      -54      D0
      FLOAT retour;
      FLOAT valeur;
```

Description

Cette fonction permet de calculer la valeur absolue d'un nombre flottant (au format FFP).

Code de condition Assembleur

```
N = 0
Z = 1, si retour = 0
V = 0
C = non défini
X = non défini
```

SPAdd**Additionner deux nombres flottants****Syntaxe**

```
retour = SPAdd(valeur1,valeur2)
      D0      -66      D1      D0
      FLOAT retour;
      FLOAT valeur1,valeur2;
```

Description

Cette fonction permet d'additionner deux nombres flottants (valeur1 + valeur2).

Code de condition Assembleur

```
N = 1, si retour < 0
Z = 1, si retour = 0
V = 1, si Overflow
C = Non défini
X = Non défini
```

SPCmp**Comparer deux nombres flottants****Syntaxe**

```
retour = SPCmp(valeur1,valeur2)
      DO      -42      D1      DO
      LONG retour;
      FLOAT valeur1,valeur2;
```

Description

Cette fonction permet de comparer deux nombres flottants.

Valeur de retour

```
+1  si valeur1 < valeur2,
 0  si valeur1 = valeur2,
-1  si valeur1 > valeur2.
```

Code de condition Assembleur

```
GT, si valeur2 > valeur1
GE, si valeur2 >= valeur1
EQ, si valeur2 = valeur1
NE, si valeur2 <> valeur1
LT, si valeur2 < valeur1
LE, si valeur2 <= valeur1
```

SPDiv**Diviser deux nombres flottants****Syntaxe**

```
retour = SPDiv(valeur1,valeur2)
      DO      -84      D1      DO
      FLOAT retour;
      FLOAT valeur1,valeur2;
```

Description

Cette fonction permet de diviser deux nombres flottants (valeur1 / valeur2).

Code de condition Assembleur

```
N = 1, si retour < 0
Z = 1, si retour = 0
V = 1, si Overflow
C = Non défini
X = Non défini
```

SPFix**Convertir un flottant en entier****Syntaxe**

```
retour = SPFix(valeur)
      DO      -30      D0
      LONG retour;
      FLOAT valeur;
```

Description

Cette fonction permet de convertir un nombre flottant en un nombre entier (en complément à deux).

Code de condition Assembleur

```
N = 1, si retour < 0
Z = 1, si retour = 0
V = 1, si Overflow
C = Non défini
X = Non défini
```

SPFlt

Convertir un entier en flottant

Syntaxe

```
retour = SPFlt(valeur)
      DO      -36      D0
FLOAT retour;
LONG valeur;
```

Description

Cette fonction permet de convertir un nombre entier (en complément à deux) en un nombre flottant.

Code de condition Assembleur

```
N = 1, si retour < 0
Z = 1, si retour = 0
V = 0
C = Non défini
X = Non défini
```

SPMul

Multiplier deux nombres flottants

Syntaxe

```
retour = SPMul(valeur1,valeur2)
      DO      -78      D1      D0
FLOAT retour;
FLOAT valeur1,valeur2;
```

Description

Cette fonction permet de multiplier deux nombres flottants (`valeur1 * valeur2`).

Code de condition Assembleur

```
N = 1, si retour < 0
Z = 1, si retour = 0
V = 1, si Overflow
C = Non défini
X = Non défini
```

SPNeg**Donner l'opposé d'un nombre****Syntaxe**

```
retour = SPNeg(valeur)
      DO      -60      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer l'opposé d'un nombre (valeur). valeur => retour = -valeur.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 0
 C = Non défini
 X = Non défini

SPSub**Soustraire deux nombres****Syntaxe**

```
retour = SPSub(valeur1,valeur2)
      DO      -72      D1      DO
FLOAT retour;
FLOAT valeur1,valeur2;
```

Description

Cette fonction permet de soustraire deux nombres flottants (valeur2 - valeur1). retour = valeur2 - valeur1.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 1, si Overflow
 C = Non défini
 X = Non défini

SPTst**Comparer un nombre avec 0****Syntaxe**

```
retour = SPTst(valeur)
      DO      -48      D1
LONG retour;
FLOAT valeur;
```

Description

Cette fonction permet de comparer un nombre flottant avec la valeur 0. Le retour est:

```
+1, si valeur > 0
 0, si valeur = 0
-1, si valeur < 0
```

Code de condition Assembleur

```
N = 1, si valeur < 0
Z = 1, si valeur = 0
V = 0
C = Non défini
X = Non défini
```

FFP	Format
-----	--------

Syntaxe	MMMMMM 31	MMMMMM 23	MMMMMM 15	SEEEEE 7
----------------	--------------	--------------	--------------	-------------

Définition

M = Mantisso 24-Bits
 S = Signe
 E = Exposant 7-Bits

Domaine de calcul (Decimal)

```
9.22337177 * 10^18 > +valeur > 5.42101070 * 10^-20
-9.22337177 * 10^18 < -valeur < -2.71050535 * 10^-20
```

Domaine de calcul (Hexadécimal):

```
0.FFFFFFF * 2^3F > +valeur > 0.800000 * 2^-3F
-0.FFFFFFF * 2^3F < -valeur < -0.800000 * 2^-40
```

6.8.2. La librairie MathTrans

La librairie MathTrans permet d'effectuer des calculs trigonométriques et logarithmiques.

Fonctions de la librairie MathTrans

FFP	994
SPACos	987
SPASin	987
SPAtan	988
SPCos	988
SPCosh	988

SPExp	989
SPFieee	989
SPLog	990
SPLog10	990
SPPow	990
SPSin	991
SPSincos	991
SPSinh	992
SPSqrt	992
SPTan	992
SPTanh	993
SPTieee	993

SPACos**Calculer l'Arc-Cosinus d'un nombre****Syntaxe**

```

retour = SPACos(valeur)
      DO      -120      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer l'Arc-Cosinus d'un nombre flottant.

Code de condition Assembleur

```

N = 0
Z = 1, si retour = 0
V = 0
C = Non défini
X = Non défini
```

SPASin**Calculer l'Arc-Sinus d'un nombre****Syntaxe**

```

retour = SPASin(valeur)
      DO      -114      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer l'Arc-Sinus d'un nombre flottant.

Code de condition Assembleur

```

N = 0
Z = 1, si retour = 0
V = 0
```

C = Non défini
 X = Non défini

SPAtan**Calculer l'Arc-Tangente d'un nombre****Syntaxe**

```
retour = SPAtan(valeur)
      DO      -30      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer l'Arc-Tangente d'un nombre flottant.

Code de condition Assembleur

N = 0
 Z = 1, si retour = 0
 V = 0
 C = Non défini
 X = Non défini

SPCos**Calculer le Cosinus d'un nombre****Syntaxe**

```
retour = SPCos(valeur)
      DO      -42      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer le Cosinus d'un nombre flottant.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 1, si valeur trop grande
 C = Non défini
 X = Non défini

SPCosh**Calculer le cosinus hyperbolique d'un nombre****Syntaxe**

```
retour = SPCosh(valeur)
      DO      -66      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer le Cosinus hyperbolique d'un nombre flottant.

Code de condition Assembleur

```
N - 1, si retour < 0
Z - 1, si retour = 0
V - 1, si Overflow
C - Non défini
X - Non défini
```

SPExp

Calculer l'exposant d'un nombre

Syntaxe

```
retour = SPExp(valeur)
      D0      -78      D0
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer l'exposant ($e^{**}X$) d'un nombre.

Code de condition Assembleur

```
N - 0
Z - 1, si retour = 0
V - 1, si Overflow
C - Non défini
X - Non défini
```

SPFieee

Convertir le format standard IEEE en FFP

Syntaxe

```
retour = SPFieee(valeur)
      D0      -108      D0
FLOAT retour;
FLOAT valeur; /* Format standard IEEE !!! */
```

Description

Cette fonction permet de convertir le format standard IEEE au format FFP.

Code de condition Assembleur

```
N - Non défini
Z - 1, si retour = 0
V - 1, si Overflow
C - Non défini
X - Non défini
```

SPLog**Calculer le logarithme naturel d'un nombre****Syntaxe**

```
retour = SPLog(valeur)
      DO      -84      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer le logarithme népérien d'un nombre flottant.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 1, si valeur <= 0
 C = Non défini
 X = Non défini

SPLog10**Calculer le logarithme d'un nombre****Syntaxe**

```
retour = SPLog10(valeur)
      DO      -126      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer le logarithme naturel (base 10) d'un nombre flottant.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 1, si valeur <= 0
 C = Non défini
 X = Non défini

SPPow**Elever un nombre à une puissance****Syntaxe**

```
retour = SPPow(valeur1,valeur2)
      DO      -90      DO      D1
FLOAT retour;
FLOAT valeur1,valeur2;
```

Description

Cette fonction permet d'elever un nombre à une puissance : valeur1^{valeur2}.

Code de condition Assembleur

N = 0
 Z = 1, si retour = 0
 V = 1, si Overflow ou valeur1 < 0
 C = Non défini
 X = Non défini

SPSin**Calculer le Sinus d'un nombre****Syntaxe**

```
retour = SPSin(valeur)
      D0      -36    D0
      FLOAT retour;
      FLOAT valeur;
```

Description

Cette fonction permet de calculer le sinus d'un nombre flottant.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 1, si valeur trop grande
 C = Non défini
 X = Non défini

SPSincos**Calculer le Sinus et le Cosinus d'un nombre****Syntaxe**

```
retour = SPSincos(valeur,adr_c)
      D0      -54    D0    D1
      FLOAT retour;
      FLOAT valeur,*adr_c; /* résultat Cosinus */
```

Description

Cette fonction permet de calculer le Sinus et le Cosinus d'un nombre flottant.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 1, si valeur trop grande
 C = Non défini
 X = Non défini

SPSinh**Calculer le Sinus hyperbolique d'un nombre****Syntaxe**

```
retour = SPSinh(valeur)
      DO      -60      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer le Sinus hyperbolique d'un nombre flottant.

Code de condition Assembleur

```
N = 1, si retour < 0
Z = 1, si retour = 0
V = 1, si Overflow
C = Non défini
X = Non défini
```

SPSqrt**Calculer la racine carrée d'un nombre****Syntaxe**

```
retour = SPSqrt(valeur)
      DO      -96      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer la racine carrée d'un nombre flottant.

Code de condition Assembleur

```
N = 0
Z = 1, si retour = 0
V = 1, si valeur < 0
C = Non défini
X = Non défini
```

SPTan**Calculer la tangente d'un nombre****Syntaxe**

```
retour = SPTan(valeur)
      DO      -48      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer la tangente d'un nombre flottant.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 1, si cos(valeur) = 0
 C = Non défini
 X = Non défini

SPTanh**Calculer la tangente hyperbolique d'un nombre****Syntaxe**

```
retour = SPTanh(valeur)
      DO      -72      DO
FLOAT retour;
FLOAT valeur;
```

Description

Cette fonction permet de calculer la tangente hyperbolique d'un nombre flottant.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 1, si Overflow
 C = Non défini
 X = Non défini

SPTieee**Convertir le format FFP au format standard-IEEE****Syntaxe**

```
retour = SPTieee(valeur)
      DO      -102      DO
FLOAT retour; /* Format standard-IEEE !!! */
FLOAT valeur;
```

Description

Cette fonction permet de convertir le format FFP au format standard IEEE.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = Non défini
 C = Non défini
 X = Non défini

FFP	Format
-----	--------

Syntaxe MMMMMMMMM MMMMMMMMM MMMMMMMMM SEEEEEEE
 31 23 15 7

Définition

M = Mantisso 24-Bits
 S = Signe
 E = Exposant 7-Bits

9.22337177 * 10^18 > +valeur > 5.42101070 * 10^-20
 -9.22337177 * 10^18 < -valeur < -2.71050535 * 10^-20

0.FFFFFF * 2^3F > +valeur > 0.800000 * 2^-3F
 -0.FFFFFF * 2^3F < -valeur < -0.800000 * 2^-40

6.8.3. La MathIeeeDoubBas

Fonctions de la librairie MathIeeeDoubBas

IEEEEDPAbs	994
IEEEEDPAdd	995
IEEEEDPCeil	995
IEEEEDPCmp	995
IEEEEDPDiv	996
IEEEEDPFix	996
IEEEEDPFloor	997
IEEEEDPFlt	997
IEEEEDPMul	997
IEEEEDPNeg	998
IEEEEDPSub	998
IEEEEDPTst	999

IEEEEDPAbs	Calculer une valeur absolue
------------	-----------------------------

Syntaxe retour = IEEEEDPAbs(valeur)
 D0/D1 -54 D0/D1
 DOUBLE retour;
 DOUBLE valeur;

Description

Cette fonction permet de calculer la valeur absolue d'un nombre au format IEEE.

Code de condition Assembleur

N = 0
 Z = 1, si retour = 0
 V = 0
 C = Non défini
 X = Non défini

IEEEEDPAdd**Additionnner deux nombres****Syntaxe**

```
retour = IEEEEDPAdd(valeur1,valeur2)
      D0/D1      -66      D0/D1    D2/D3
```

Description

Cette fonction permet d'additionner deux nombres au format IEEE (valeur1 + valeur2)

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 1, si Overflow
 C = Non défini
 X = Non défini

IEEEEDPCeil**Syntaxe**

```
retour = IEEEEDPCeil(valeur)
      D0/D1      -96      D0/D1
DOUBLE retour;
DOUBLE valeur;
```

Description

Cette fonction donne le nombre entier supérieur ou égal à la valeur transmise.

Code de condition Assembleur

Non connu

IEEEEDPCmp**Comparer deux nombres****Syntaxe**

```
retour = IEEEEDPCmp(valeur1,valeur2)
      D0      -42      D0/D1    D2/D3
LONG retour;
DOUBLE valeur1,valeur2;
```

Description

Cette fonction permet de comparer deux nombres au format IEEE.

Valeur de retour

```
+1, si valeur1 > valeur2
0, si valeur1 = valeur2
-1, si valeur1 < valeur2
Code de condition Assembleur
GT, si valeur1 > valeur2
GE, si valeur1 >= valeur2
EQ, si valeur1 = valeur2
NE, si valeur1 <> valeur2
LT, si valeur1 < valeur2
LE, si valeur1 <= valeur2
```

IEEEEDPDiv

Diviser deux nombres

Syntaxe

```
retour = IEEEEDPDiv(valeur1,valeur2)
      D0/D1      -84      D0/D1    D2/D3
DOUBLE retour;
DOUBLE valeur1,valeur2;
```

Description

Cette fonction permet de diviser deux nombres au format IEEE (valeur1 / valeur2).

Code de condition Assembleur

```
N = 1, si retour < 0
Z = 1, si retour = 0
V = 1, si Overflow
C = Non défini
X = Non défini
```

IEEEEDPFix

convertir un IEEE en un entier

Syntaxe

```
retour = IEEEEDPFix(valeur)
      D0      -30      D0/D1
LONG retour;
DOUBLE valeur;
```

Description

Cette fonction permet de convertir un nombre au format IEEE en un entier.

Code de condition Assembleur

```
N = 1, si retour < 0
Z = 1, si retour = 0
V = 1, si Overflow
```

C = Non défini
X = Non défini

IEEEDPFloor**Calculer l'entier inférieur ou égal****Syntaxe**

```
retour = IEEEFPFloor(valeur)
      DO/D1      -90      DO/D1
      DOUBLE retour;
      DOUBLE valeur;
```

Description

Cette fonction permet de calculer le plus grand entier inférieur ou égal à la valeur transmise.

Code de condition Assembleur

Non connu

IEEEDPFlt**Convertir un entier en IEEE****Syntaxe**

```
retour = IEEEFPFlt(valeur)
      DO/D1      -36      DO
      DOUBLE retour;
      LONG valeur;
```

Description

Cette fonction permet de convertir un nombre entier au format IEEE.

Code de condition Assembleur

N = 1, si retour < 0
Z = 1, si retour = 0
V = 0
C = Non défini
X = Non défini

IEEEDPMul**Multiplier deux nombres****Syntaxe**

```
retour = IEEEFPMul(valeur1,valeur2)
      DO/D1      -78      DO/D1    D2/D3
      DOUBLE retour;
      DOUBLE valeur1,valeur2;
```

Description

Cette fonction permet de multiplier deux nombres au format IEEE ($valeur1 * valeur2$).

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 1, si Overflow
 C = Non défini
 X = Non défini

IEEEEDPNeg**Donner l'opposé d'un nombre****Syntaxe**

```
retour = IEEEEDPNeg(valeur)
      DO/D1      -60      DO/D1
      DOUBLE retour;
      DOUBLE valeur;
```

Description

Cette fonction permet de calculer l'opposé d'un nombre au format IEEE.

retour = -valeur.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 0
 C = Non défini
 X = Non défini

IEEEEDPSub**Soustraire deux nombres****Syntaxe**

```
retour = IEEEEDPSub(valeur1,valeur2)
      DO/D1      -72      DO/D1    D2/D3
      DOUBLE retour;
      DOUBLE valeur1,valeur2;
```

Description

Cette fonction permet de soustraire deux nombres au format IEEE.

retour = valeur1 - valeur2.

Code de condition Assembleur

N = 1, si retour < 0
 Z = 1, si retour = 0
 V = 1, si Overflow
 C = Non défini
 X = Non défini

IEEEEDPTst**Comparer un nombre avec 0****Syntaxe**

```
retour = IEEEEDPTst(valeur)
      DO          -48      D0/D1
      LONG retour;
      DOUBLE valeur;
```

Description

Cette fonction permet de comparer un nombre au format IEEE avec la valeur 0. Le retour est:

```
+1, si valeur > 0.
0, si valeur = 0.
-1, si valeur < 0.
```

Code de condition Assembleur

```
N = 1, si valeur < 0
Z = 1, si valeur = 0
V = 0
C = Non défini
X = Non défini
```

Structure

```
MathIEEE.resource:
struct MathIEEE =
{
Offset:
0x00 0 struct Node MathIEEE_node;
0x0E 14 WORD MathIEEE_Flags
Offset:
0x10 16 ULONG MathIEEE_BaseAddr           Pour Coprocesseur 68881.
Offset:
0x14 20 ULONG MathIEEE_Db1BasInit
0x18 24 ULONG MathIEEE_Db1TransInit

0x1C 28 ULONG MathIEEE_SnglBasInit
0x20 32 ULONG MathIEEE_SnglTransInit
sizeof(struct MathIEEE) =
0x24 36
}
```

6.8.4. La librairie MathIeeeDoubTrans*Fonctions de la MathTrans.Library*

IEEEEDPACos	1000
IEEEEDPASin	1000
IEEEEDPAtan	1000
IEEEEDPCos	1001

IEEEEDPCosh	1001
IEEEEDPExp	1001
IEEEEDPFieee	1001
IEEEEDPLog	1002
IEEEEDPLog10	1002
IEEEEDPPow	1002
IEEEEDPSin	1003
IEEEEDPSincos	1003
IEEEEDPSinh	1003
IEEEEDPSqrt	1003
IEEEEDPTan	1004
IEEEEDPTanh	1004
IEEEEDPTieee	1004

IEEEEDPAcos**Calculer l'Arc-Cosinus d'un nombre****Syntaxe**

```
retour = IEEEEDPAcos(valeur)
      DO/D1      -120      DO/D1
      DOUBLE retour;
      DOUBLE valeur;
```

Description

Cette fonction permet de calculer l'Arc-Cosinus d'un nombre au format IEEE.

IEEEEDPAsin**Calculer l'Arc-Sinus d'un nombre****Syntaxe**

```
retour = IEEEEDPAsin(valeur)
      DO/D1      -114      DO/D1
      DOUBLE retour;
      DOUBLE valeur;
```

Description

Cette fonction permet de calculer l'Arc-Sinus d'un nombre au format IEEE.

IEEEEDPAtan**Calculer l'Arc-Tangente d'un nombre****Syntaxe**

```
retour = IEEEEDPAtan(valeur)
      DO/D1      -30      DO/D1
      DOUBLE retour;
      DOUBLE valeur;
```

Description

Cette fonction permet de calculer l'Arc-Tangente d'un nombre au format IEEE.

IEEEEDPCos**Calculer le Cosinus d'un nombre****Syntaxe**

```
retour = IEEEEDPCos(valeur)
      D0/D1      -42    D0/D1
DOUBLE retour;
DOUBLE valeur;
```

Description

Cette fonction permet de calculer le Cosinus d'un nombre au format IEEE.

IEEEEDPCosh**Calculer le cosinus hyperbolique d'un nombre****Syntaxe**

```
retour = IEEEEDPCosh(valeur)
      D0/D1      -66    D0/D1
DOUBLE retour;
DOUBLE valeur;
```

Description

Cette fonction permet de calculer le Cosinus hyperbolique d'un nombre au format IEEE.

IEEEEDPExp**Calculer l'exposant d'un nombre****Syntaxe**

```
retour = IEEEEDPExp(valeur)
      D0/D1      -78    D0/D1
DOUBLE retour;
DOUBLE valeur;
```

Description

Cette fonction permet de calculer l'exposant ($e^{**}X$) d'un nombre.

IEEEEDPFieee**Convertir un IEEE en Double IEEE****Syntaxe**

```
retour = IEEEEDPFieee(valeur)
      D0/D1      -108   D0
```

```
DOUBLE retour;
FLOAT valeur; /* Format IEEE !!! */
```

Description

Cette fonction permet de convertir un nombre au format IEEE en un nombre au format Double IEEE.

IEEEDPLog

Calculer le logarithme naturel d'un nombre

Syntaxe

```
retour = IEEEDPLog(valeur)
      D0/D1      -84      D0/D1
DOUBLE retour;
DOUBLE valeur;
```

Description

Cette fonction permet de calculer le logarithme népérien d'un nombre au format IEEE.

IEEEDPLog10

Calculer le logarithme d'un nombre

Syntaxe

```
retour = IEEEDPLog10(valeur)
      D0/D1      -126     D0/D1
DOUBLE retour;
DOUBLE valeur;
```

Description

Cette fonction permet de calculer le logarithme naturel (base 10) d'un nombre au format IEEE.

IEEEDPPow

Elever un nombre à une puissance

Syntaxe

```
retour = IEEEDPPow(valeur2,valeur1)
      D0/D1      -90      D2/D3      D0/D1
DOUBLE retour;
DOUBLE valeur1,valeur2;
```

Description

Cette fonction permet d'élever un nombre à une puissance : valeur1^valeur2.

IEEEEDPSin**Calculer le Sinus d'un nombre****Syntaxe**

```
retour = IEEEEDPSin(valeur)
      D0/D1      -36      D0/D1
DOUBLE retour;
DOUBLE valeur;
```

Description

Cette fonction permet de calculer le sinus d'un nombre au format IEEE.

IEEEEDPSincos**Calculer le Sinus et le Cosinus d'un nombre****Syntaxe**

```
retour = IEEEEDPSincos(adr_c,valeur)
      D0/D1      -54      A0      D0/D1
DOUBLE retour;
DOUBLE valeur,*adr_c; /* résultat Cosinus */
```

Description

Cette fonction permet de calculer le Sinus et le Cosinus d'un nombre au format IEEE.

IEEEEDPSinh**Calculer le Sinus hyperbolique d'un nombre****Syntaxe**

```
retour = IEEEEDPSinh(valeur)
      D0/D1      -60      D0/D1
DOUBLE retour;
DOUBLE valeur;
```

Description

Cette fonction permet de calculer le Sinus hyperbolique d'un nombre au format IEEE.

IEEEEDPSqrt**Calculer la racine carrée d'un nombre****Syntaxe**

```
retour = IEEEEDPSqrt(valeur)
      D0/D1      -96      D0/D1
DOUBLE retour;
DOUBLE valeur;
```

Description

Cette fonction permet de calculer la racine carrée d'un nombre au format IEEE.

IEEEDPTan**Calculer la tangente d'un nombre****Syntaxe**

```
retour = IEEEDPTan(valeur)
      D0/D1      -48      D0/D1
      DOUBLE retour;
      DOUBLE valeur;
```

Description

Cette fonction permet de calculer la tangente d'un nombre au format IEEE.

IEEEDPTanh**Calculer la tangente hyperbolique d'un nombre****Syntaxe**

```
retour = IEEEDPTanh(valeur)
      D0/D1      -72      D0/D1
      DOUBLE retour;
      DOUBLE valeur;
```

Description

Cette fonction permet de calculer la tangente hyperbolique d'un nombre au format IEEE.

IEEEDPTieee**Convertir un IEEEDP en IEEESP****Syntaxe**

```
retour = IEEEDPTieee(valeur)
      D0      -102      D0/D1
      FLOAT retour; /* Format IEEE !!! */
      DOUBLE valeur;
```

Description

Cette fonction permet de convertir un nombre au format Double IEEE en un nombre au format IEEE.

Structure

```
MathIEEE.resource:
struct MathIEEE =
{
Offset:
0x00 0 struct Node MathIEEE_node;
0x0E 14 WORD MathIEEE_Flags
Offset:
0x10 16 ULONG MathIEEE_BaseAddr           Pour Coprocesseur 68881.
Offset:
0x14 20 ULONG MathIEEE_Db1BasInit
0x18 24 ULONG MathIEEE_Db1TransInit
```

```

0x1C 28 ULONG MathIEEE_SnglBasInit
0x20 32 ULONG MathIEEE_SnglTransInit
sizeof(struct MathIEEE) =
0x24 36
}

```

6.9. La librairie Expansion

Les fonctions de la librairie Expansion sont très particulières et nécessitent de la part du programmeur une bonne connaissance de la machine, surtout au niveau du Hardware. Elles permettent par exemple d'ajouter un disque au système et de le déclarer, reconnaître la mémoire d'expansion et la gérer, etc. Voici les fichiers d'en-tête habituellement utilisés avec cette librairie :

```

<libraries/expansion.h>
<libraries/configregs.h>
<libraries/configvars.h>
<libraries/filehandler.h>

```

Fonctions de la librairie Expansion

AddConfigDev	1006
AddDosNode	1006
AllocBoardMem	1007
AllocConfigDev	1007
AllocExpansionMem	1008
ConfigBoard	1008
ConfigChain	1009
FindConfigDev	1009
FreeBoardMem	1010
FreeConfigDev	1011
FreeExpansionMem	1011
GetCurrentBinding	1012
MakeDosNode	1012
ObtainConfigBinding	1013
ReadExpansionByte	1014
ReadExpansionRom	1014
ReleaseConfigBinding	1015
RemConfigDev	1015
SetCurrentBinding	1016
WriteExpansionByte	1016

AddConfigDev**Ajouter un nouveau ConfigDev****Syntaxe**

```
AddConfigDev(configDev)
    -30          A0
    struct ConfigDev *configDev;
```

Description

Cette fonction ajoute la structure ConfigDev indiquée dans la liste des Devices de configuration du système.

Paramètres

configDev: Une structure ConfigDev

Voir aussi

[RemConfigDev](#)

AddDosNode**Ajouter un disque au système****Syntaxe**

```
Ok = AddDosNode(bootPri,flags,deviceNode)
    D0      -150     D0     D1      A0
    BOOL ok;
    BYTE bootPri;
    LONG flags;
    struct DeviceNode *deviceNode;
```

Description

Cette routine s'assure que votre disk device (ou un device qui doit être traité comme un disque) soit reconnu par le système.

Paramètres

bootPri: Un octet qui indique la priorité de boot du disque.

Voici les priorités conseillées :

- +5 Lecteur DFO. Cette priorité doit toujours être la plus grande pour permettre à l'utilisateur d'annuler un boot à partir du disque dur.
- 0 Disque dur
- 5 Un lecteur sur réseau.
- 128 Ne s'embarasse pas de booter à partir de ce device.

flags: Bits supplémentaires pour l'appel : ADN_STARTPROC (bit 0) lance le traitement handler immédiatement.

deviceNode: Un DeviceNode DOS correct

Retour

Ok: Nul si un problème apparaît

Voir aussi

[MakeDosNode](#)

AllocBoardMem**Allouer de la mémoire d'expansion****Syntaxe**

```
Slot = AllocBoardMem(slotSpec)
      DO          -42      00
      LONG startSlot;
      LONG slotSpec;
```

Description

Cette fonction alloue un nombre de slots de l'espace d'expansion (chaque slot prend E_SLOTSIZE octets).

Paramètres

slotSpec: Le champ de taille mémoire.

Retour

startSlot: Numéro du slot du début de la mémoire d'expansion. En cas d'erreur, StartSlot prend la valeur -1.

Exemple

```
struct ExpansionRom *er;
slot = AllocBoardMem(er->er_Type & ERT_MEMMASK);
```

Voir aussi

[AllocExpansionMem](#), [FreeExpansionMem](#), [FreeBoardMem](#)

AllocConfigDev**Allouer une structure ConfigDev****Syntaxe**

```
ConfigDev = AllocConfigDev()
      DO          -48
      struct ConfigDev *configDev;
```

Description

Cette fonction alloue une structure ConfigDev, en permettant ainsi d'ajouter de nouveaux champs à la structure sans altérer le code existant.

Retour

`configDev:` Une structure `ConfigDev` ou `NULL`.

Voir aussi

`FreeConfigDev`

AllocExpansionMem**Allouer la mémoire d'expansion****Syntaxe**

```
startSlot = AllocExpansionMem(numSlots,slotOffset)
          D0           -54        D0      D1
LONG startSlot;
LONG numSlots,slotOffset;
```

Description

Cette fonction alloue un nombre de slots d'expansion (chaque slot fait `E_SLOT_SIZE` octets).

Paramètres

`numSlots:` Nombre de slots demandés.

`slotOffset:` Offset du StartSlot.

Retour

`startSlot:` Le numéro du slot alloué ou `-1` en cas d'erreur.

Voir aussi

`AllocBoardMem`, `FreeBoardMem`

ConfigBoard**Configurer la carte d'expansion****Syntaxe**

```
error = ConfigBoard(board,configDev)
       D0      -60        A0      A1
LONG error;
LONG board;
struct ConfigDev *configDev;
```

Description

Cette routine configure la carte d'expansion. Elle allouera la mémoire d'expansion et placera la carte à la nouvelle adresse. `ConfigDev` sera mis à jour en conséquence.

Paramètres

board: Adresse en cours de la carte d'expansion
 configDev: Structure ConfigDev de la carte

Retour

error: Non nul si un problème apparaît

Voir aussi

[FreeConfigDev](#)

ConfigChain**Configurer le système entier**

Syntaxe error = ConfigChain(baseAddr)
 DO -66 A0
 LONG error;
 LONG baseAddr;

Description

Cette routine prend une adresse de base (en général E_EXPANSIONBASE) et configure tous les devices existant à cet endroit. Toutes les cartes trouvées seront linkées à la liste de configuration.

Paramètres

baseAddr: Adresse de base pour commencer la recherche des cartes

Retour

error: Non nul si une erreur apparaît.

Voir aussi

[FreeConfigDev](#)

FindConfigDev**Rechercher une entrée ConfigDev**

Syntaxe ConfigDev = FindConfigDev(oldConfigDev,manufacturer,product)
 -72 A0 D0 D1
 struct ConfigDev *configDev;
 struct ConfigDev *oldConfigDev;
 LONG manufacturer,product;

Description

Cette fonction traite la liste des structures ConfigDev existantes dans le système et y recherche celle qui possède les codes product et manufacturer adéquats.

Paramètres

oldConfigDev: Une structure ConfigDev valide ou NULL pour commencer au début de la liste

manufacturer: Le code du fabricant (ID) ou -1 sinon

product: Le code du produit recherché ou -1 sinon

Retour

configDev: L'entrée ConfigDev qui correspond aux demandes ou NULL s'il n'y a aucune correspondance

Exemple

```
struct ConfigDev *cd = Null;
while (cd = FindConfigDev(cd,MANUFACTURER,PRODUCT) )
{
    /* Boucle de traitement */
    /* Utiliser la valeur de retour */
}
```

FreeBoardMem

Libérer la mémoire d'expansion

Syntaxe

```
FreeBoardMem(startSlot,slotSpec)
            -78      D0      D1
LONG startSlot,slotSpec;
```

Description

Cette fonction libère des slots d'espace mémoire d'expansion (chaque slot fait E_SLOTSIZE octets). C'est la fonction inverse de AllocBoardMem().

Paramètres

startSlot: Numéro de slot

slotSpec: Champ de la taille mémoire

Exemple

```
struct ExpansionRom *er;
LONG startSlot,slotSpec;
slotSpec = er->er_Type & ERT_MEMMASK;
startSlot = AllocBoardMem(slotSpec);
if (startSlot != -1)
{
```

```

        FreeBoardMem(startSlot,slotSpec);
    }
}

```

Voir aussi

[AllocExpansionMem](#), [FreeExpansionMem](#), [AllocBoardMem](#)

FreeConfigDev

Libérer une structure ConfigDev

Syntaxe

```

FreeConfigDev(configDev)
-84          A0
struct ConfigDev *configDev;

```

Description

Cette fonction libère une structure ConfigDev telle qu'elle est retournée par AllocConfigDev.

Paramètres

configDev: Une structure ConfigDev valide.

Voir aussi

[AllocConfigDev](#)

FreeExpansionMem

Libérer la mémoire d'expansion du device

Syntaxe

```

FreeExpansionMem(startSlot,numSlots)
-90          D0      D1
LONG startSlot,numSlots;

```

Description

Cette fonction libère la mémoire d'expansion du device standard. C'est l'inverse de AllocExpansionMem().

Paramètres

startSlot: Numéro du slot ou -1 pour une erreur

numSlots: Le nombre de slots à libérer

Voir aussi

[AllocExpansionMem](#), [AllocExpansionBoard](#), [FreeExpansionBoard](#)

GetCurrentBinding**Définit la zone de configuration de la carte****Syntaxe**

```
Count = GetCurrentBinding(currentBinding,taille)
      DO           -138          A0          DO
      UWORLD taille;
      struct CurrentBinding *currentBinding;
      UWORLD taille;
```

Description

Cette fonction écrit les contenus de la structure "currentbinding" hors d'une place privée. Elle peut être réglée via SetCurrentBinding().

Paramètres

currentBinding:

Structure CurrentBinding

taille:

Taille de la structure binddriver utilisateur.

Retour

Count:

Taille réelle de la structure CurrentBinding

Voir aussi

SetCurrentBinding

MakeDosNode**Construit des structures de données****Syntaxe**

```
MakeDosNode(parameterPkt)
      -144          A0
      struct DeviceNode *deviceNode;
      LONG *parameterPkt;
```

Description

Cette routine fabrique les structures de données nécessaires pour ajouter un device dos disk au système. On y trouve un DeviceNode, un FileSysStartupMsg, un vecteur d'environnement disque et jusqu'à deux chaînes bcpl. Vous pouvez consulter les fichiers include des libraries/dosextens et libraries/filehandler.

Paramètres

parameterPkt:

C'est un tableau longword contenant toutes les informations nécessaires à l'initialisation des structures de données.

```

0  Chaîne avec nom du dos handler
4  Chaîne avec nom du device exec.
8  unit number (pour OpenDevice)
12 Flags pour OpenDevice.
16 Nombre de longword
20 environnement du file handler.

```

Retour

deviceNode: Structure DeviceNode

Exemple

Définit un lecteur 3,5" Amiga pour le lecteur 1

```

char execName[] = "trackdisk.device";
char dosName[] = "df1";
ULONG parmPkt[] =
{
    (ULONG) dosName,
    (ULONG) execName,
    1,                      /* Numéro des Lecteurs */
    0,                      /* Flags pour OpenDevice */
    /* Bloc d'informations */

    11,
    512 >> 2,              /* longword par Bloc */
    0,                      /* Origine du sector */
    2,                      /* Nombre de surfaces*/
    1,
    1,                      /* Secteurs par bloc logique */
    11,
    2,                      /* Secteurs par piste */
    2,                      /* (Boot-) Blocs réservés */
    0,
    /* ??? */
    0,                      /* Interleave */
    0,                      /* Cylindre bas */
    79,
    5                      /* Nombre de buffers */
};
struct DeviceNode *dnode,*MakeDosNode();
dnode = MakeDosNode(parmPkt);

```

Voir aussi

AddDosNode

ObtainConfigBinding	Autoriser les bind-drivers
----------------------------	-----------------------------------

Syntaxe

ObtainConfigBinding()
-120

Description

Cette fonction donne la permission aux bind-drivers

Voir aussi

[ReleaseConfigBinding](#)

ReadExpansionByte**Lire un octet sur la carte d'expansion****Syntaxe**

```
byte = ReadExpansionByte(board,offset)
      DO          -96      A0      D0
      BYTE byte;
      LONG board,offset;
```

Description

Cette fonction lit un octet sur une carte d'expansion de nouveau style. Ce type de carte organise les données à lire en une série de nybbles en mémoire.

Paramètres

board: Pointeur sur la base de la carte d'expansion

offset: Un offset logique de la base de la carte

Retour

byte: un octet de données de la carte d'expansion, ou -1 s'il y a une erreur de lecture.

Exemple

```
type = ReadExpansionByte(cd->BoardAddr, EROFFSET(er_Type));
ints = ReadExpansionByte(cd->BoardAddr, ECOFFSET(ec_Interrupt));
```

Voir aussi

[ReadExpansionRom](#), [WriteExpansionByte](#)

ReadExpansionRom**Lire les données de la configuration****Syntaxe**

```
error = ReadExpansionRom(board,configDev)
      DO          -102      A0      A1
      LONG error;
      LONG board;
      struct ConfigDev *configDev;
```

Description

Cette fonction lit une zone Rom d'un device d'expansion vers une zone cd_Rom d'une structure ConfigDev. La routine sait détecter la présence de la carte.

Paramètres

board: Pointeur sur la base de la carte d'expansion
 configDev: Structure ConfigDev

Retour

error: Non Nul si une erreur survient.

Exemple

```
configDev = AllocConfigDev();
if (!configDev) error();
error = ReadExpansionRom(board, configDev);
if (!error)
{
    configDev->cd_BoardAddr = board;
    ConfigBoard(configDev);
}
```

Voir aussi

[ReadExpansionByte](#), [WriteExpansionByte](#)

ReleaseConfigBinding**Libérer un séaphore**

Syntaxe `ReleaseConfigBinding()`
 -126

Description

Cette fonction libère le séaphore pour permettre aux autres de lier leurs drivers aux structures ConfigDev.

Voir aussi

[ObtainConfigBinding](#)

RemConfigDev**Supprimer une structure ConfigDev**

Syntaxe `RemConfigDev(configDev)`
 -108 A0
 struct ConfigDev *configDev;

Description

Cette fonction supprime la structure ConfigDev de la liste des devices de configuration.

Paramètres

configDev: Structure ConfigDev

Voir aussi

AddConfigDev

SetCurrentBinding**Définir la zone de configuration d'une carte****Syntaxe**

```
SetCurrentBinding(currentBinding,Count)
      -132          A0      D0
      struct CurrentBinding *currentBinding;
      ULONG Count;
```

Description

Cette fonction définit la zone de configuration d'une carte. Elle enregistre les contenus de la structure "currentbinding" dans une place privée.

Paramètres

currentBinding: Pointeur sur une structure currentbinding

Count: Taille de la structure binddriver utilisateur

Voir aussi

GetCurrentBinding

WriteExpansionByte**Ecrit un octet nibble par nibble****Syntaxe**

```
error = WriteExpansionByte(board,offset,byte)
      D0          -114          A0      D0      D1
      LONG error;
      LONG board,offset;
      BYTE byte;
```

Description

Cette fonction écrit un octet sur une carte d'expansion non standard. Ce type de carte organise les données pouvant être écrites en une série de nybbles en mémoire.

Paramètres

board: Pointeur sur la base de la carte d'expansion

offset: Un offset logique de la base ConfigDev

byte: L'octet de données à écrire sur la carte

Exemple

```
error = WriteExpansionByte
        (cd->BoardAddr,ECOFFSET(ec_Shutup),(LONG)0);
error = WriteExpansionByte
        (cd->BoardAddr,ECOFFSET(ec_Interrupt),1L);
```

Voir aussi

```
ReadExpansionByte, ReadExpansionRom
Structure
Global
E_SLOTSIZE          0x10000
E_EXPANSIONBASE    0xe80000
E_EXPANSIONSIZE    0x080000
E_EXPANSIONLOTS     8
E_MEMORYBASE        0x200000
E_MEMORYSIZE        0x800000
E_MEMORYLOTS        128
struct ExpansionRom <libraries/configregs.h>
{
Offset:
0x00 0 UBYTE   er_Type;
ERT_TYPEMASK        0xc0
ERT_NEWBOARD        0xc0
ERT_MEMMASK         0x07
ERTF_CHAINEDCONFIG (1<<3)
ERTF_DIAGVALID      (1<<4)
ERTF_MEMLIST         (1<<5)
Offset:
0x01 1 UBYTE   er_Product;
Offset:
0x02 2 UBYTE   er_Flags;
ERFF_MEMSPACE       (1<<7)
ERFF_NOSHUTUP       (1<<6)
Offset:
0x03 3 UBYTE   er_Reserve03;
0x04 4 WORD    er_Manufacturer;
0x06 6 ULONG   er_SerialNumber;
0x0A 10 WORD   er_InitDiagVec;
0x0C 12 UBYTE   er_Reserve0c;
0x0D 13 UBYTE   er_Reserve0d;
0x0E 14 UBYTE   er_Reserve0e;
0x0F 15 UBYTE   er_Reserve0f;
sizeof(struct ExpansionRom) =
0x10 16
};

struct ExpansionControl <libraries/configregs.h>
{
Offset:
0x00 0 UBYTE   ec_Interrupt;
Interrupt-Controlregister:
ECIF_INTENA        (1<<1)
ECIF_RESET         (1<<3)
ECIF_INT2PEND      (1<<4)
ECIF_INT6PEND      (1<<5)
ECIF_INT7PEND      (1<<6)
```

```
ECIF_INTERRUPTING      (1<<7)
Offset:
0x01 1 UBYTE          ec_Reserved11;
0x02 2 UBYTE          ec_BaseAddress;
0x03 3 UBYTE          ec_Shutup;
0x04 4 UBYTE          ec_Reserved14;
0x05 5 UBYTE          ec_Reserved15;
0x06 6 UBYTE          ec_Reserved16;
0x07 7 UBYTE          ec_Reserved17;
0x08 8 UBYTE          ec_Reserved18;
0x09 9 UBYTE          ec_Reserved19;
0x0A 10 UBYTE         ec_Reserved1a;
0x0B 11 UBYTE         ec_Reserved1b;
0x0C 12 UBYTE         ec_Reserved1c;
0x0D 13 UBYTE         ec_Reserved1d;
0x0E 14 UBYTE         ec_Reserved1e;
0x0F 15 UBYTE         ec_Reserved1f;
sizeof(struct ExpansionControl) -
0x10 16
};
struct ConfigDev <libraries/configvars.h>
{
Offset:
0x00 0 struct Node    cd_Node;
Offset:
0x0E 14 UBYTE          cd_Flags;
CDF_SHUTUP      0x01
CDF_CONFIGME   0x02
Offset:
0x0F 15 UBYTE          cd_Pad;
0x10 16 struct ExpansionRom cd_Rom;
0x20 32 APTR           cd_BoardAddr;
0x24 36 APTR           cd_BoardSize;
0x28 40 WORD            cd_SlotAddr;
0x2A 42 WORD            cd_SlotSize;
0x2C 44 APTR           cd_Driver;
0x30 48 struct ConfigDev * cd_NextCD;
0x34 52 ULONG           cd_Unused[4];
sizeof(struct ConfigDev) -
0x44 68
};
struct CurrentBinding <libraries/configvars.h>
{
Offset:
0x00 0 struct ConfigDev *cb_ConfigDev;
0x04 4 UBYTE            *cb_FileName;
0x08 8 UBYTE            *cb_ProductString;
0x0C 12 UBYTE           **cb_ToolTypes;
sizeof(struct CurrentBinding) -
0x10 16
};
struct FileSysStartupMsg <libraries/filehandler.h>
{
Offset:
0x00 0 ULONG            fssm_Unit;
0x04 4 BSTR             fssm_Device;
0x08 8 BPTR              fssm_Environ;
0x0C 12 ULONG            fssm_Flags;
sizeof(struct FileSysStartupMsg) -
```

```
0x10 16
}:
struct DeviceNode <libraries/filehandler.h>
{
Offset:
0x00 0 BPTR dn_Next;
0x04 4 ULONG dn_Type;
0x08 8 struct MsgPort *dn_Task;
0x0C 12 BPTR dn_Lock;
0x10 16 BSTR dn_Handler;
0x14 20 ULONG dn_StackSize;
0x18 24 LONG dn_Priority;
0x1C 28 BPTR dn_Startup;
0x20 32 BPTR dn_SegList;
0x24 36 BPTR dn_GlobalVec;
0x28 40 BSTR dn_Name;
sizeof(struct DeviceNode) =
0x2C 44
};
```


7. L'Amiga 3000

Découvert en 1990, l'Amiga 3000 se veut une machine Multimédia, mais en conservant une compatibilité avec les anciennes versions de l'Amiga.

Pour obtenir une telle machine, il faut bien sûr mettre à la disposition des utilisateurs des moyens et une puissance presque inégalée. Mais voyez plutôt.

7.1. Caractéristiques techniques

Processeur

Il existe deux configurations possibles :

- ✓ 68030 + 68881 à 16 Mhz
- ✓ 68030 + 68882 à 25 Mhz

Co-Processeurs

- ✓ Super Fat Agnus (capable de gérer 2 Mo de Chip Ram)
- ✓ Fat Gary
- ✓ Fat Denise
- ✓ Contrôleur SCSI
- ✓ Contrôleur de mémoire Fast
- ✓ Désentrelaceur

Mémoire

- ✓ 2 Mo de Chip RAM (actuellement).
- ✓ 1 Mo de Fast RAM.

Modes de résolutions

- ✓ Résolution graphique allant de 320 à 1448 points en overscan)
- ✓ 640 x 512 en 16 couleurs non entrelacé

- ✓ Overscan soutenu

Lecteurs de disquettes

- ✓ 1 ou 2 lecteurs de 880 Ko.

Mémoire de masse

- ✓ 1 disque dur Quantum SCSI de 40 Mo avec un temps d'accès de 19 ms.

Possibilités d'extensions

- ✓ 4 connecteurs d'extension (ZORRO III) 100 broches dont
 - * 2 connecteurs 100 broches
 - * 2 connecteurs 100 avec extension AT

Connecteurs externes

- ✓ 1 connecteur série
- ✓ 1 connecteur parallèle
- ✓ 1 connecteur de disque externe (880 Ko)
- ✓ 1 connecteur SCSI
- ✓ 2 connecteurs vidéo dont 1 Amiga et 1 pour moniteur Multisynchronie
- ✓ 2 connecteurs audio (Cinch)
et un interrupteur pour activer ou pas le Flicker fixer (désentrelateur vidéo, nécessite un moniteur Multisynchronie).

7.2. La nouvelle interface utilisateur

Pour pouvoir exploiter toute la puissance de l'Amiga 3000, il fallait une nouvelle interface utilisateur. Celle-ci reste toujours aussi ergonomique et intuitive que celle du 1.3, mais avec une nouvelle présentation et surtout une puissance beaucoup plus grande. Nous allons maintenant passer à sa description détaillée.

7.2.1. La fenêtre Workbench

Oui, il n'y a pas d'erreur, le Workbench tourne maintenant en fenêtre. Cela signifie qu'il est multi-tâches. Comme dans les anciennes versions, c'est dans cette fenêtre que se trouvent les icônes des disquettes, du Système et du Ram-Disk.

De même que l'interface, les icônes ont été changées. Elles apparaissent maintenant en pseudo-profondeur (style bouton poussoir). Quand une icône n'est pas activée, elle apparaît en relief. Et au contraire, quand elle est activée, elle semble s'enfoncer dans l'écran. Même si ce rendu n'est dû qu'à un changement de couleurs, cette impression est du plus bel effet.

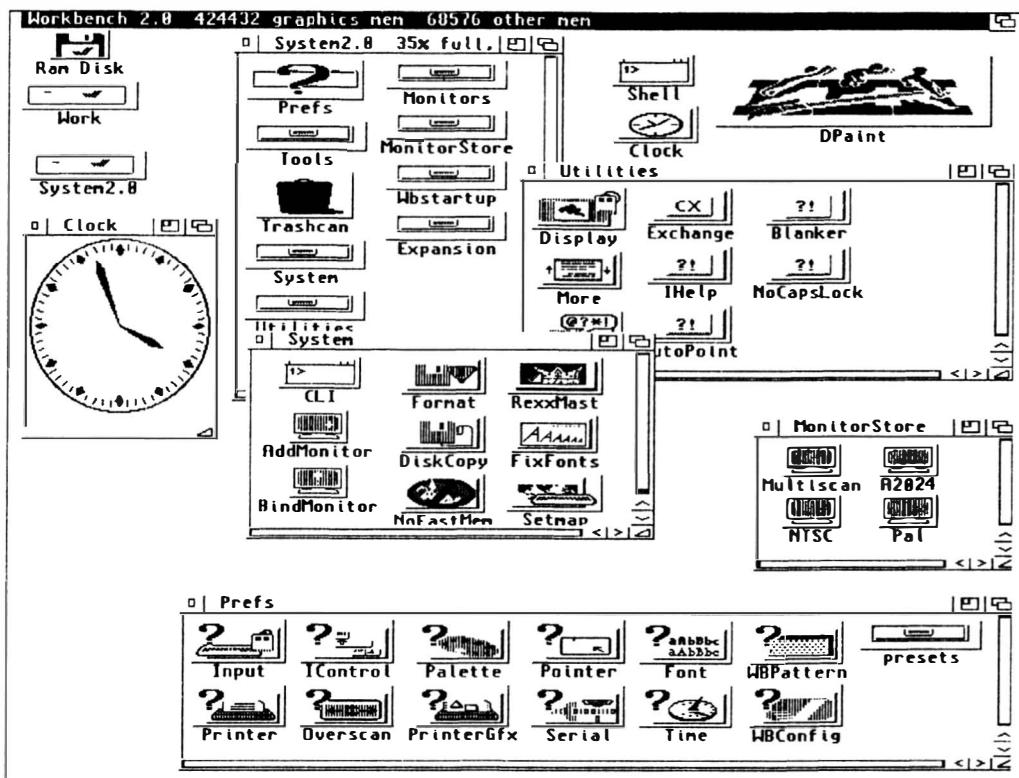


Figure 7 - 68

Maintenant, et c'est une nouveauté, il est possible de fermer toutes les fenêtres (y compris celles du Shell et du CLI) à l'aide de la cellule de fermeture en haut à gauche des fenêtres.

De même, en double-cliquant dans une fenêtre, celle-ci passe directement au premier-plan dans la fenêtre Workbench.

Enfin, une nouvelle possibilité, pour sélectionner plusieurs icônes. Ce n'est plus la peine de maintenir la touche Shift enfoncée, il est possible de directement le faire à la souris. Pour cela, il suffit de maintenir le bouton gauche enfoncé et de déplacer la souris. Il se crée alors un cadre élastique qui permet, une fois le bouton de la souris recherché, de sélectionner toutes les icônes qui se trouvent sous ce cadre.

7.2.2. La barre des menus

En activant la barre des menus, on constate que certaines options du 1.3 ont disparu (Disk et Special), mais en contrepartie, il y a maintenant quatre options au lieu de trois (Workbench, Window, Icons et Tools).

De même que pour la version 1.3, les options qui ne sont pas disponibles (temporairement) apparaissent en grisé.

Sans plus attendre, nous allons passer à la description de ces nouvelles options.

7.2.2.1. Le menu Workbench

Ce menu comporte maintenant 7 options. C'est lui qui gère les opérations générales de l'Amiga, les fenêtres ainsi que l'exécution directe des commandes AmigaDOS.

L'option "Backdrop"

Cette option permet de placer l'écran Workbench devant tous les autres. Cependant, une fois cette option activée, il ne vous sera plus possible de modifier sa taille. Pour réactiver de nouveau la fenêtre, resélectionnez la commande.

L'option "Execute Command..."

Cette option permet d'exécuter une commande AmigaDOS sans avoir à activer pour cela le Shell ou le CLI. En activant cette option, une fenêtre apparaît dans laquelle vous pouvez entrer votre commande. Une fois celle-ci exécutée (une commande comme dir ou list ouvre une fenêtre de sortie "Output Window") au Workbench.

L'option "Redraw All" (inchangée)

Cette option permet de redessiner tout l'écran suite à un problème de gestion de celui-ci. Celle-ci peut être nécessaire avec certains programmes qui ne réorganisent pas entièrement l'écran Workbench à la fin de leur travail.

L'option "Update all"

Cette option permet de remettre à jour l'écran Workbench. Si par exemple, sous Shell, vous supprimez certains répertoires ou fichiers, ceux-ci ne le seront pas sous Workbench. Cette option permet donc de pallier à ce problème.

L'option "Last Error" (inchangée)

Cette option permet de réafficher le dernier message d'erreur généré par le système. Cette option est très pratique par le fait qu'elle permet de revoir le message, même s'il a été effacé par un autre texte.

Voici quelques messages d'erreur affichés par le système :

- ✓ Error while opening ...
- ✓ Disk write protected ...
- ✓ Object not found ...
- ✓ Etc

L'option "Version" (inchangée)

Cette option permet d'afficher le numéro de version du Workbench et du Kickstart. Cette option n'a d'intérêt que les programmeurs.

L'option "Quit..."

Cette option permet de terminer le travail sous le Workbench. Attention, cette option peut être dangereuse pour une personne qui ne connaît pas l'utilisation du Shell ou du CLI. En effet, une fois l'écran Workbench fermé, il n'y a plus de possibilité de le relancer, sauf par l'intermédiaire du Shell (loadWB). S'il n'est pas actif, il est nécessaire de réamorcer le système !

De même, si cette option est appelée quand des applications tournent encore, le système sera dans l'impossibilité de fermer l'interface Workbench.

7.2.2.2. Le menu Window

L'option "New Drawers"

Cette option permet de créer "un nouveau tiroir" dans la fenêtre et le répertoire correspondant sur le disque dur. Le tiroir ainsi créé est automatiquement nommé "Unnamed1".

Pour obtenir la même chose avec l'ancien système, il est obligatoire de dupliquer le tiroir "Empty", de le mettre en place là où il devait être et de le renommer. Cette option remplace donc l'ancien tiroir. En effet, si celui-ci est effacé par mégarde, il devient alors impossible d'en créer de nouveau.

L'option "Open Parent"

Cette option permet d'afficher la "fenêtre mère" des fenêtres devant toutes les autres.

L'option "Close"

Cette option permet de fermer une fenêtre. Cependant, il est plus simple d'utiliser la case de fermeture en haut à gauche de cette même fenêtre.

L'option "UpDate"

Cette option permet de remettre à jour une fenêtre. Si par exemple, sous Shell, vous supprimez certains répertoires ou fichiers, ceux-ci ne le seront pas dans la fenêtre. Cette option est identique à "UpDate All", sauf qu'elle n'agit que sur la fenêtre sélectionnée au lieu d'agir sur tout l'écran Workbench.

L'option "Select Contents"

Cette option permet de sélectionner toutes les icônes présentes dans la fenêtre active.

L'option "Clean Up" (Inchangée)

Cette option permet de réorganiser les icônes de la fenêtre active. Il peut arriver que lors d'une copie ou d'une suppression certaines icônes soient déplacées, supprimées ou modifiées. Cette option permet donc une réorganisation optimum en fonction de la place dans la fenêtre et de la taille des icônes.

L'option "Snapshot >>"

Cette option permet de mémoriser l'emplacement des icônes et de la fenêtre active, de même que sa taille.

Snapshot Window

Cette option permet de mémoriser la taille et l'emplacement de la fenêtre active..

Snapshot Contents

Cette option est identique à "Snapshot Window", mais elle permet en plus de mémoriser l'emplacement des icônes.

L'option "Show"

Cette option permet de visualiser tous les fichiers présents sur le disque dur, même ceux qui ne sont pas pourvus d'une icône. Dans ce cas, le système leur en attribuera une automatiquement. Une icône de tiroir représentera un répertoire, tandis qu'une normale représentera un simple fichier.

Show Only Icons

Cette option permet la visualisation des fichiers pourvus d'une icône.

Show All Files

Cette option permet de visualiser tous les fichiers.

L'option "View By"

Cette option permet de changer l'affichage des fichiers. Ceux-ci peuvent maintenant être affichés par leur icône (View by icons), cette option est celle normalement sélectionnée par le système. De même, les fichiers peuvent être triés et affichés par ordre alphabétique (View by name), par leur date de création (View by date) ou bien encore par leur taille.

7.2.2.3. Le menu Icons

L'option "Open"

Cette option permet de visualiser tout ce qui se rapporte à l'icône active.

Si l'icône se rapporte à un Tiroir, son contenu sera affiché à l'écran. S'il se rapporte à une application, celle-ci sera exécutée.

L'option "Copy"

Cette option permet de copier tous les objets de l'Amiga (tiroir, programmes ou objets). De même, il est possible de copier une disquette avec cette option. Pour cela, il ne sera fait objet que d'un seul lecteur de disquette.

L'option "Rename"

Cette option permet de renommer une icône ou bien une disquette.

L'option "Information..."

Cette option permet de fournir des informations au sujet de l'objet se rapportant à l'icône active. Selon que cette option porte sur une disquette ou sur un objet, les informations retournées ne sont pas les mêmes.

Pour un objet, on obtient les informations suivantes :

- ✓ Le nom du disque
- ✓ Son status (lecture seule ou pas)

- ✓ Le nombre de blocs total
- ✓ Le nombre de blocs utilisés
- ✓ Le nombre de blocs libres
- ✓ La taille d'un bloc en octets
- ✓ La date de création
- ✓ Les options par défaut (Default Tools)
- ✓ Une représentation de son icône

Pour un objet, on obtient les informations suivantes :

- ✓ Le nom de l'objet
- ✓ Le nombre de blocs occupés
- ✓ La taille en octets
- ✓ La taille de la pile attribuée à cette objet
- ✓ La date des derniers changements
- ✓ Les types des objets (Tool Types)
- ✓ Une représentation de son icône
- ✓ L'état des bits de protection
 - * Archived
 - * Readable
 - * Writable
 - * Executable
 - * Deletable

L'option "Snapshot"

Cette option permet de mémoriser la position des icônes actives au moment de l'appel de la fonction. Attention, cette option est effacée par la fonction "Clean Up".

L'option "UnSnapshot"

Cette fonction permet d'annuler les effets de la fonction précédente.

L'option "Leave Out"

Cette option permet de placer une icône dans la fenêtre Workbench (vous pouvez bien sûr mémoriser sa position avec l'option Snapshot). De cette manière, vous pouvez placer dans un coin de l'écran toutes les applications que vous utilisez le plus souvent (Dpaint, Superbase, etc), et il ne vous reste plus qu'à les activer de là.

Il faut quand même savoir que seules les icônes sont déplacées, et que le système mémorise les nouvelles positions.

L'option "Put Away"

Cette option est l'inverse de la précédente. Elle permet donc de remettre l'icône dans son tiroir d'origine.

L'option "Delete..."

Cette option permet d'effacer des disques (disquettes et disque dur) des fichiers et les icônes qui s'y rapportent.

Attention, cette option peut être dangereuse. Une fois que vous avez effacé une icône, il est impossible de récupérer son contenu, ainsi que les informations qu'elle contenait.

L'option "Format Disk..."

Cette option permet de formater (initialiser) une disquette en vue de son utilisation par le système. Attention, toutes les données que la disquette pouvait contenir seront complètement effacées.

L'option "Empty Trash"

Cette option permet de "vider la poubelle". Comme pour la version 1.3, la poubelle représente une zone permettant de stocker des fichiers dont vous ne voulez plus. Cette option permet donc de supprimer les fichiers de cette zone.

7.2.2.4. Le menu Tools

Ce menu permet d'installer des applications pour pouvoir les lancer depuis ce menu plutôt que d'avoir à double-cliquer dessus.

7.3. Le Shell et l'Amiga OS 2.0

Le shell de l'amiga est une interface de commande "en ligne". Il a pour principale particularité de pouvoir être exécuté dans une fenêtre du WorkBench.

7.3.1. Le Shell

L'AmigaOS 2.0 comporte une partie DOS (pour Disk Operating System) gérant les accès disque et l'organisation des fichiers : le système de "gestion de fichiers".

Ce système est accessible par une interface gérée par des commandes clavier : le Shell (et le CLI = Command Line Interface).

7.3.1.1. Fonctionnement du Shell et du CLI

Le Shell, comme le CLI, utilise des commandes dont certaines sont dites "internes" et d'autres "externes". Une commande "externe" appartient au répertoire C: du disque système (en général) et est chargée pour être exécutée au contraire d'une commande interne qui est incluse dans le CLI ou le Shell et qui réside en mémoire.

Les commandes internes du Shell 2.0 sont les suivantes :

```
Alias
Ask
CD
Echo
Else
EndCLI
EndIf
EndShell
EndSkip
Failat
Fault
Get
GetEnv
If
Lab
NewCLI
NewShell
Path
Prompt
Quit
Resident
Run
Set
SetEnv
Skip
Stack
UnAlias
UnSet
Why
```

7.3.1.2. Gestionnaire du Shell

Le Shell est défini par un gestionnaire : SYS:System/CLI ! Les utilisateurs chevronnés de l'Amiga seront surpris de voir ce gestionnaire ici : le CLI de l'OS 2.0 est aussi un Shell ! Un CLI sous AmigaDOS 1.x n'était pas vraiment équivalent à un Shell : pas de commande interne, pas de possibilité d'édition- modification de la ligne courante ou de rappel des commandes précédentes... Dans le 2.0, le CLI n'est présent que pour conserver une certaine compatibilité avec des fichiers Script des versions 1.x.

7.3.1.3. Accès au commandes

Le Shell n'inclut, à priori, aucune commande de façon interne.

Il utilise seulement des commandes externes et accède à ces commandes par ordre de priorité dans :

- ✓ les commandes définies par Alias (commandes "internes"),
- ✓ les commandes résidentes (commandes "internes" aussi),
- ✓ le répertoire courant,
- ✓ le répertoire SYS:C:/,
- ✓ les chemins d'accès définis par la commande Path.

Chaque commande pouvant être suivie de paramètres.

7.3.2. Les nouvelles commandes Amiga OS 2.0

Certaines des commandes Amiga OS 1.3 et 1.3.2 ont été modifiées et d'autres commandes ont été rajoutées. D'une manière générale, toutes les commandes ont été optimisées en taille et en vitesse et peuvent afficher leur ligne d'aide correctement à l'aide de '?'.

Options :

- | | |
|------|--|
| /A : | nécessite un Argument. |
| /K : | mot-clé nécessaire (Key word). |
| /S : | non strictement nécessaire (Switch). |
| /N : | nécessite un argument Numérique. |
| /M : | plusieurs arguments (Multiple) possibles, séparés par des espaces (nombre illimité). |
| /F : | marque de Fin de commande. |
| , | pas d'argument. |

7.3.2.1. MAKEDIR

Syntaxe : MAKEDIR NAME /M

Plusieurs répertoires peuvent être créés simultanément en séparant les noms par des espaces.

7.3.2.2. DELETE

Syntaxe : DELETE FILE/M/A, ALL/S, QUIET/S, FORCE/S

FILE/M : plusieurs fichiers, séparés par des espaces, peuvent être spécifiés.

FORCE/S : permet de détruire des fichiers actifs.

7.3.2.3. COPY

Syntaxe : COPY FROM/A/M, TO/A, ALL/S, QUIET/S, BUF=BUFFER/K/N,
CLONE/S, DATE/S, NOPRO/S, COM/S

FROM/M : plusieurs sources peuvent être définies simultanément en les
séparant par des espaces.

7.3.2.4. LIST

ALL/S : liste la totalité des sous-répertoires du répertoire courant.

7.3.2.5. RENAME

Syntaxe : RENAME FROM/A/M, TO=AS/A, QUIET/S

FROM/A/M : plusieurs sources possible.

QUIET/S : n'affiche pas les fichiers renommés à l'écran.

7.3.2.6. INSTALL

Syntaxe : INSTALL DRIVE/A, NOBOOT/S, CHECK/S, FFS/S

FFS/S : permet de préparer la disquette avec le FastFileSystem (FFS).

7.3.2.7. JOIN

Syntaxe : JOIN FILE/M, AS=TO/K/A

FILE/M : plusieurs fichiers séparés par des espaces peuvent être spécifiés.

7.3.2.8. SEARCH

Syntaxe : SEARCH FROM/M/A, SEARCH/A, ALL/S, NONUM/S, QUIET/S,
QUICK/S, FILE/M, PATTERN/S

FILE/M : plusieurs fichiers possibles, séparés par des espaces.

PATTERN/S : permet la recherche par attribut.

7.3.2.9. SORT

Syntaxe : SORT FROM/A, TO/A, COLSTART/K, CASE/S, NUMERIC/S

CASE/S : valide les majuscules/minuscules.

NUMERIC/S : valide le tri numérique.

7.3.2.10. PROTECT

Syntaxe : PROTECT FILE, FLAGS, ADD/S, SUB/S, ALL/S, QUIET/S

ALL/S : place le bit de protection sur tous les fichiers et sous-répertoires du répertoire courant.

QUIET/S : n'affiche pas les fichiers modifiés.

7.3.2.11. FILENOTE

Syntaxe : FILENOTE FILE/A, COMMENT/A, ALL/S ,QUIET/S

ALL/S et QUIET/S
ont la même fonction que pour PROTECT.

7.3.2.12. SETDATE

Syntaxe : SETDATE FILE/A, DATE, TIME, ALL/S

ALL/S : affecte tous les fichiers.

7.3.2.13. DISKDOCTOR

Syntaxe : DISKDOCTOR DRIVE/A

Cette version de DiskDoctor permet de réparer une disquette FFS. ATTENTION : la valeur 0x444F5301 DOIT être placée dans l'entrée DOSTYPE de la MountList !

7.3.2.14. ENDCLI

La commande EndCLI ou EndShell n'est plus nécessaire : la fenêtre CLI (ou Shell) contient un gadget de fermeture.

7.3.2.15. STATUS

Syntaxe : STATUS PROCESS/N, FULL/S, TCB/S, CLI=ALL/S, COM=COMMAND/K

Identique au 1.3 hormis PROCESS qui est donné sous forme numérique.

7.3.2.16. CHANGETASKPRI

Syntaxe : CHANGETASKPRI PRI=Priorité/N/A, PROCESS/K/N

Identique au 1.3 hormis Priorité et PROCESS qui sont sous forme numérique.

7.3.2.17. PATH

Syntaxe : PATH Chemin/M, ADD/S, SHOW/S, RESET/S, REMOVE/S

Chemin/M : plusieurs définitions possibles, séparées par des virgules.

REMOVE/S : retire un chemin de la liste active.

7.3.2.18. ASSIGN

Syntaxe : ASSIGN NAME,TARGET/M, LIST/S, EXISTS/S, DISMOUNT/S,
DEFER/S, PATH/S, ADD/S, REMOVE/S, VOL/S, DIRS/S, DEVICES/S

TARGET/M : permet de définir plusieurs répertoires comme unité d'accès. Ainsi
ASSIGN FONTS: SYS:FONTS RAM:Custom_Fonts définit
SYS:FONTS et RAM:Custom_Fonts comme contenant des fontes
utilisables.

DISMOUNT/S : enlève la source de la liste d'assignation.

DEFER/S : n'accède au répertoire que lorsque cela est nécessaire (demande
explicite).

PATH/S : permet la recherche dans d'autres répertoires du disque courant.

VOL/S : affiche les ASSIGN correspondant au volume courant.

DIRS/S : idem pour le répertoire courant.

DEVICES/S : idem pour le device courant.

7.3.2.19. DATE

DATE accepte maintenant l'abréviation du nom du mois ou son numéro (1-12-90 = 1
dec-90).

7.3.2.20. EXECUTE

Le numéro du CLI ou du Shell actuel est disponible dans la variable <\$\$>. Les lignes de commentaires peuvent être introduites dans le script avec ./

7.3.2.21. ECHO

Syntaxe : ECHO ,NOLINE/S, FIRST/K/N, LEN/K/N

FIRST et LEN sont maintenant de type numérique.

Exemple : ECHO "Bonjour" NOLINE FIRST-2 LEN-5

7.4. Les nouvelles "Preferences"

De même que l'interface utilisateur a été entièrement pensée, il en est de même avec le programme Preferences qui comporte maintenant 13 réglages possibles.

Voici une description de ces possibilités :

Input

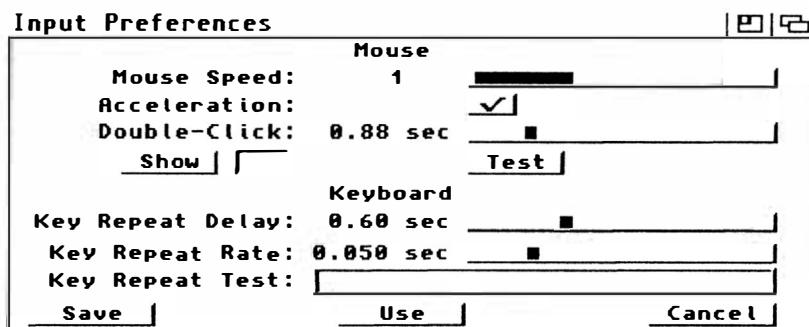


Figure 7 - 69

Cette fonction permet de modifier la vitesse de la souris (accélération, vitesse du double clic,...) de même que les réglages prédéfinis du clavier (vitesse de répétition des touches...). Pour ces deux réglages, il existe une possibilité de tester avant de valider.

Printer

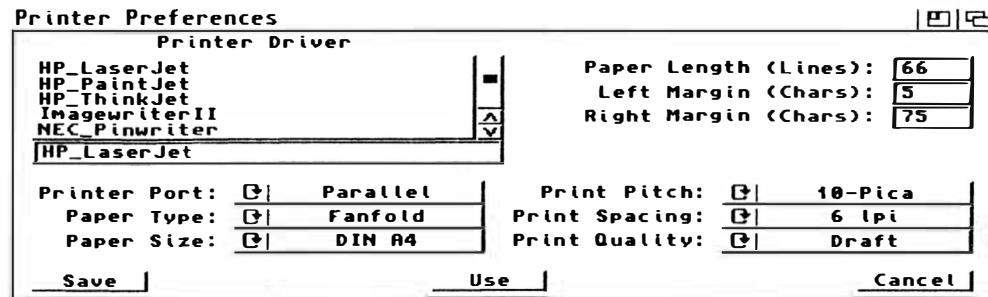


Figure 7 - 70

Cette fonction permet d'effectuer les réglages et la définition de l'imprimante utilisée avec l'Amiga. C'est à partir de là que vous sélectionnez le gestionnaire d'imprimante, de même que la taille du papier (format, nombre de caractères par lignes, nombre de lignes par page, etc), la taille des caractères, etc.

IControl

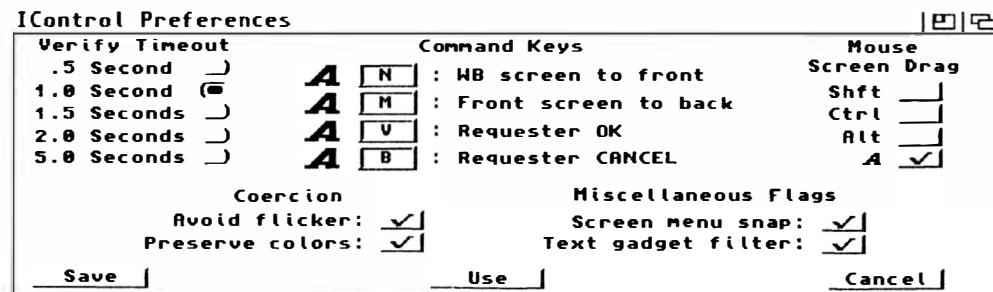


Figure 7 - 71

Cette fonction permet de sélectionner les touches associées à l'écran, les raccourcis clavier.

Overscan

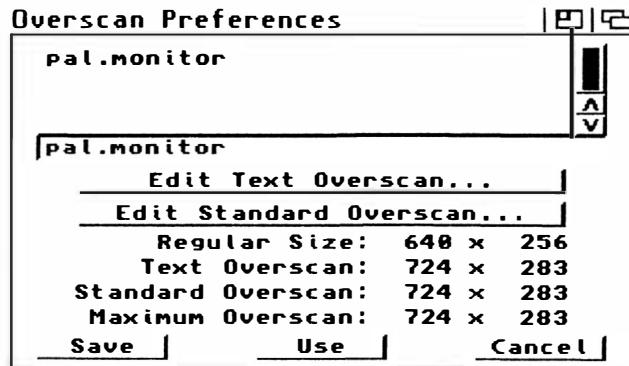


Figure 7 - 72

Cette fonction permet de régler la taille de votre écran (texte, mais aussi graphique) en fonction de votre moniteur. Ainsi, avec un moniteur multisynchronie, vous avez la possibilité d'agrandir votre écran d'environ 20%.

Palette

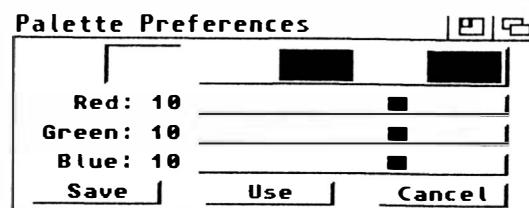


Figure 7 - 73

Cette fonction permet de redéfinir les couleurs de base de l'écran Workbench (jusqu'à 16 couleurs).

PrinterGfx

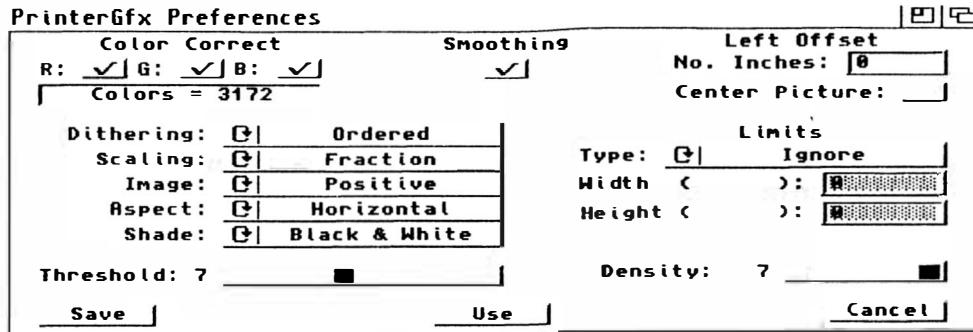


Figure 7 - 74

Cette fonction permet de définir les paramètres de l'imprimante pour une impression graphique (ancienne fonction "Graphic 2").

Pointer

Cette fonction permet de modifier le curseur (ancienne fonction "Edit Pointer").

Serial

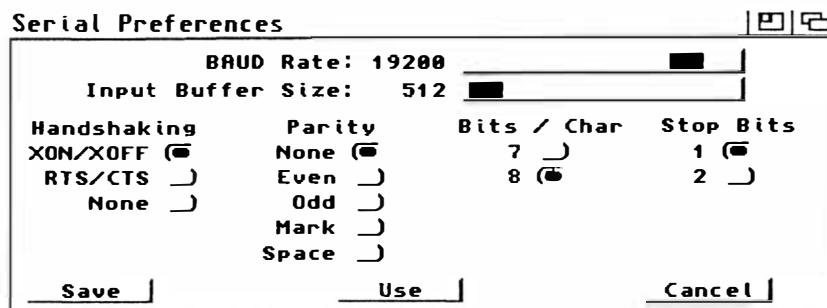


Figure 7 - 75

Cette fonction permet de définir les paramètres de l'interface série en vue de l'utilisation d'un modem. Vous pouvez par exemple définir la vitesse de transfert, la taille du buffer, la parité, le nombre de bits de transmission, etc.

Font

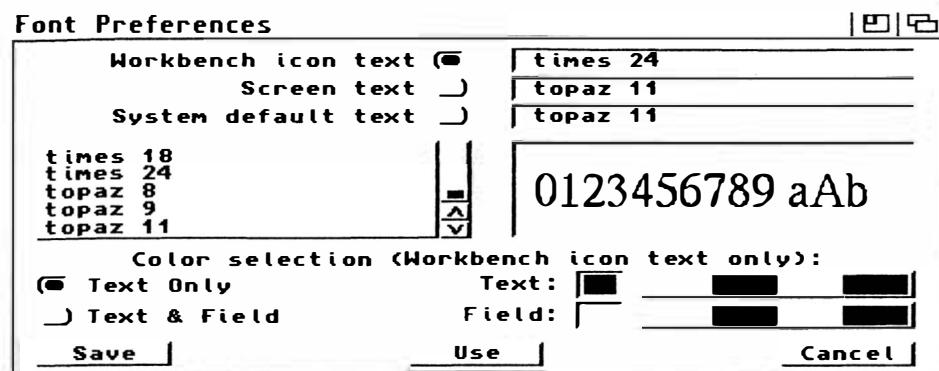


Figure 7 - 76

Cette fonction permet de définir les fontes du système (icônes, fenêtres et textes), de même que les couleurs associées.

Time

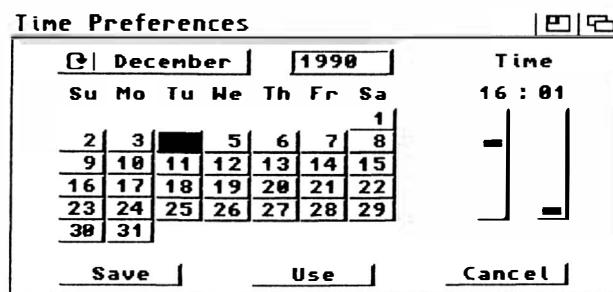


Figure 7 - 77

Cette fonction permet de régler le jour et l'heure du système sous la forme d'un calendrier.

WBPattern

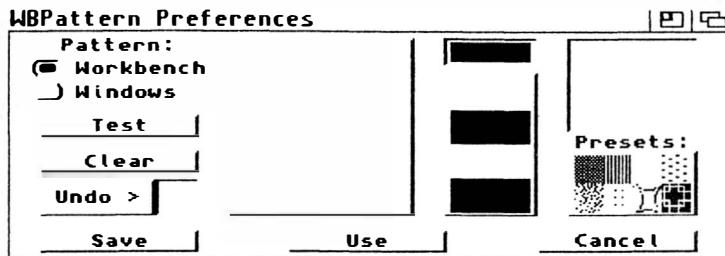


Figure 7 - 78

Cette fonction permet de définir un motif de fond pour l'écran Workbench. Cette fonction permet donc de personnaliser un peu l'environnement de travail.

WBconfig

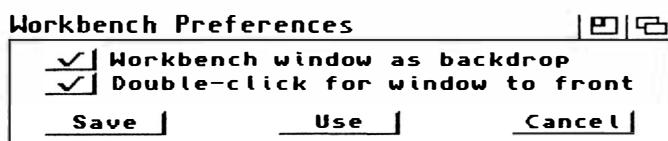


Figure 7 - 79

Cette fonction permet de définir deux fonctions. La première permet de mettre l'écran Workbench en arrière plan, la seconde de ramener une fenêtre au premier plan en double-cliquant dedans.

ScreenMode

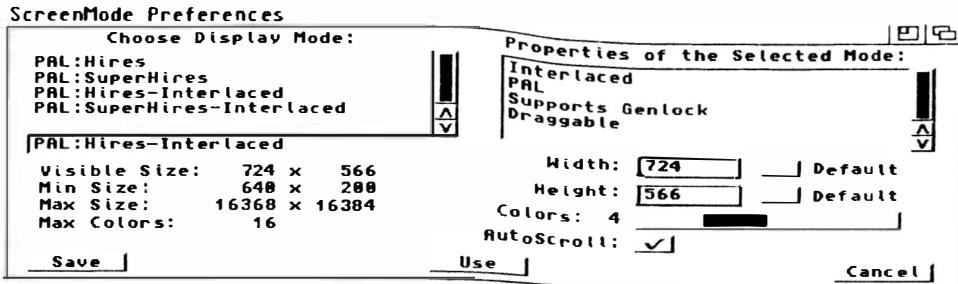


Figure 7 · 80

Cette fonction permet de régler la taille de l'écran de travail. C'est cette option qui doit être utilisée si vous travaillez avec un moniteur Multisynchrone.

Avec un tel écran, il est possible d'obtenir une résolution de travail de 724 x 566 en 16 couleurs sur tout l'écran (le maximum étant de 16368 x 16384 visible partiellement).

Annexe A : Détail des registres du 8520

Registre	Nom	Description
0 0	PRA	Registre données port A
1 1	PRB	Registre données port B
2 2	DDRA	Registre direction des données port A
3 3	DDR B	Registre direction des données port B
4 4	TALO	Minuterie A (octet bas)
5 5	TAHI	Minuterie A (octet haut)
6 6	TBLO	Minuterie B (octet bas)
7 7	TBHI	Minuterie B (octet haut)
8 8	EVENT LO	Compteur (valeur événement octet bas) bits 0-7
9 9	E. 8-15	Compteur bits 8-15 (octet moyen)
10 A	EVENT HI	Compteur bits 16-23 (octet haut)
11 B	—	inutilisé
12 C	SP	Données série
13 D	ICR	Contrôle Interruption
14 E	CRA	Registre contrôle port A
15 F	CRB	Registre contrôle port B

► *Le port parallèle*

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
0	PRA	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
1	PRB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
2	DDRA	DPA7	DPA6	DPA5	DPA4	DPA3	DPA2	DPA1	DPA0
3	DDR B	DPB7	DPB6	DPB5	DPB4	DPB3	DPB2	DPB1	DPB0

Les minuteries (Timer en anglais)

Accès lecture

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
4	TALO	TAL7	TAL6	TAL5	TAL4	TAL3	TAL2	TAL1	TAL0
5	TAHI	TAH7	TAH6	TAH5	TAH4	TAH3	TAH2	TAH1	TAH0
6	TBLO	TBL7	TBL6	TBL5	TBL4	TBL3	TBL2	TBL1	TBL0
7	TBHI	TBH7	TBH6	TBH5	TBH4	TBH3	TBH2	TBH1	TBH0

Accès écriture

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
4	PALO	PAL7	PAL6	PAL5	PAL4	PAL3	PAL2	PAL1	PAL0
5	PAHI	PAH7	PAH6	PAH5	PAH4	PAH3	PAH2	PAH1	PAH0
6	PBLO	PBL7	PBL6	PBL5	PBL4	PBL3	PBL2	PBL1	PBL0
7	PBHI	PBH7	PBH6	PBH5	PBH4	PBH3	PBH2	PBH1	PBH0

Détail des bits du registre de contrôle A

Registre N 14 / \$E Nom : CRA

D7	D6	D5	D4	D3	D2	D1	D0
TOD IN	SPMODE	INMODE	LOAD	RUNMODE	OUTMODE	PBON	START
0-60Hz	0=entrée	0-Hor1	1-FORCE	0-cont.	0-pulse	0-PB6OFF	0-stop
1-50Hz	1-sortie	1-CNT	LOAD (strobe)	1-one-shot	1-toggle	1-PB6ON	1-start

Détail des bits du registre de contrôle B

Registre N 15 / \$F Nom : CRB

D7	D6+D5	D4	D3	D2	D1	D0
ALARM	INMODE	LOAD	RUNMODE	OUTMODE	PBON	START
0-TOD	00-Hor1.	1-FORCE	0-cont.	0-pulse	0-PB70FF	0-stop
1-Alarm	01-CNT 10-Timer A 11-CNT+ Timer A	LOAD (strobe)	1-one-shot	1-toggle	1-PB70N	1-start

Le compteur (Event-counter)

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
8 \$8	LSB Event	E7	E6	E5	E4	E3	E2	E1	E0
9 \$9	Event 8-15	E15	E14	E13	E12	E11	E10	E9	E8
10 \$A	MSB Event	E23	E22	E21	E20	E19	E18	E17	E16

Le Port Série

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
12 \$C	SDR	S7	S6	S5	S4	S3	S2	S1	S0

Le registre de contrôle d'interruption (ICR)

Accès lecture (READ) = registre de données

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
13 \$D	ICR	IR	0	0	FLAG	SP	Alarm	TB	TA

Accès écriture (WRITE) = registre masque

Registre	Nom	D7	D6	D5	D4	D3	D2	D1	D0
13 \$D	ICR	S/C	x	x	FLAG	SP	Alarm	TB	TA

- ① passage à zéro de la minuterie A (TA, Bit 0).
- ② passage à zéro de la minuterie B (TB, Bit 1).
- ③ concordance de la valeur de l'Event-Counter avec celle de l'alarme (alarm, bit 2).
- ④ registre à décalage du port série plein (entrée) ou vide (sortie) (SP, bit 3).
- ⑤ niveau négatif de l'entrée FLAG (FLAG, bit 4).

Annexe B

CIA - A : adresses des registres

Adresse	Nom	D7	D6	D5	D4	D3	D2	D1	D0
\$BFE001	PRA	/FIR1	/FIRO	/RDY	/TK0	/WPRO	/CHNG	/LED	/OVL
\$BFE101	PRB	port parallèle							
\$BFE201	DDRA	0	0	0	0	0	0	1	1
\$BFE301	DDRB	utilisés pour marquer la direction des ports (entrées/sorties)							
\$BFE401	TAL0	minuterie A utilisée en permanence pour le test du clavier							
\$BFE501	TAHI								
\$BFE601	TBLO	minuterie B: nécessaire pour différentes tâches							
\$BFE701	TBHI								
\$BFE801	E.LSB	compteur événement. Il compte les impulsions à							
\$BFE901	E.8.15	50HZ du compteur d'alimentation, qui dérive							
\$BFEA01	E.MSB	de la fréquence du réseau							
\$BFEBO1		inutilisé							
\$BFEC01	SP	Registre de données séries (clavier)							
\$BFED01	ICR	Registre de contrôle d'interruption							
\$BFEE01	CRA	Registre de contrôle A							
\$BFEF01	CRB	Registre de contrôle B							

Annexe C : CIA - B : adresses des registres

Adresse	Nom	Adresses des registres							
\$BFD000	PRA	DTR	/RST	/CD	/CTS	/DSR	/SEL	/POUT	/BUSY
\$BFD100	PRB	MTR	/SEL3	/SEL2	/SEL1	/SELO	/SIDE	/DIR	/STEP
\$BFD200	DDRA	1	1	0	0	0	0	0	0
\$BFD300	DDR B	1	1	1	1	1	1	1	1
\$BFD400	TAL0	La minuterie A n'est utilisée que pour le transfert de données séries							
\$BFD500	TAHI								
\$BFD600	TBLO	La minuterie B est utilisée par le blitter en mode synchrone pour le transfert d'images							
\$BFD700	TBHI								
\$BFD800	E.LSB	L'évent counter du CIA-B compte les impulsions synchrones horizontales ; en temps normal elles							
\$BFD900	E.8.15								
\$BFDA00	E.MSB	ont une fréquence de 15625 par seconde							
\$BFDB00		inutilisé							
\$BFDC00	SP	registre de données séries							
\$BFDD00	ICR	registre de contrôle d'interruption							
\$BFDE00	CRA	registre de contrôle A							
\$BFDF00	CRB	registre de contrôle B							

CIA-A

IRQ	INT2 entrée de PAULA
RES	broche de reset system
D0-D7	bus de données du processeur bits 0-7
A0-A3	bus d'adresse du processeur bits 8-11
Phi 2	horloge E du processeur
R/W	processeur R/W
PA 7	port manette 1/brocbe 6 (bouton de tir)
PA 6	port manette 0/brocbe 6 (bouton de tir)
PA 5	/RDY "disk ready", disque prêt
PA 4	/TK0 "disk track 0", disque piste 0
PA 3	/WPRO "write protect", protection en écriture
PA 2	/CHNG "disk change", disque changé
PA 1	/LED état de la diode LED (0= allumé)
PA 0	/OVL "memory overlay bit" recouvrement mémoire
SP	KDAT données série du clavier
CNT	KCLK horloge clavier
PB0-PB7	signaux de données pour port parallèle (centronic)

PC /DRDY données prêtes pour port parallèle
 FLAG /ACK acquittement pour port parallèle

CIA-B

/IRQ	/INT	6-entrée de PAULA
/RES		signaux de reset system
D0-D7		bus de données du processeur bits 8-15
A0-A3		bus d'adresse du processeur bits 8-11
Phi 2		horloge E du processeur
R/W		processeur R/W
PA 7	/DTR	connecteur série sortie DTR
PA 6	/RTS	connecteur série sortie RTS
PA 5	/CD	connecteur série sortie CD (carrier detect)
PA 4	/CTS	connecteur série sortie CTS
PA 3	/DSR	connecteur série sortie DSR
PA 2	SEL	"SELECT" contrôle port parallèle
PA 1	POUT	"paper out", papier absent (imprimante sur port centronic)
PA 0	BUSY	"busy", imprimante occupée sur port parallèle
SP	BUSY	imprimante occupée port A bit 0
CNT	POUT	imprimante occupée port A bit 1
PB 7	/MTR	"motor" moteur lecteur de disquette
PB 6	/SEL 3	"drive select" sélection lecteur disquette n°3 (DF3:)
PB 5	/SEL 2	"drive select" sélection lecteur disquette n°2 (DF2:)
PB 4	/SEL 1	"drive select" sélection lecteur disquette n°1 (DF1:)
PB 3	/SEL 0	"drive select" sélection lecteur disquette interne (DF0:)
PB 2	/SIDE	"side select" sélection de la face de la disquette
PB1	DIR	"direction" direction d'avancement du moteur de lecteur de disquette
PB 0	STEP	"step", avance d'un pas du moteur du lecteur de disquette
FLAG	/INDEX	"index", début du cylindre (disquette)
PC		non utilisé.

Annexe D : Le connecteur Centronics

sortie	1	donnée valide
entrée/sortie	2	bit de donnée 0
entrée/sortie	3	bit de donnée 1
entrée/sortie	4	bit de donnée 2
entrée/sortie	5	bit de donnée 3
entrée/sortie	6	bit de donnée 4
entrée/sortie	7	bit de donnée 5
entrée/sortie	8	bit de donnée 6
entrée/sortie	9	bit de donnée 7
entrée	10	/acknowledge - acquittement
entrée/sortie	11	busy : imprimante occupée
entrée/sortie	12	paper out : papier manquant
entrée/sortie	13	on line : imprimante sélectionnée
	14	+ 5 volts
	15	inutilisé
sortie	16	reset
	17- 25	GND masse de référence

Sur l'AMIGA 1000, certaines broches sont disposées d'une autre façon :

14- 22	GND
23	+ 5 volts
24	inutilisé
25	reset

Centronics Nº de broche	Fonction	CIA	Broche	Descriptif
1	Donnée valide	A	18	PC
2	bit de donnée 0	A	10	PB0
3	bit de donnée 1	A	11	PB1
4	bit de donnée 2	A	12	PB2
5	bit de donnée 3	A	13	PB3
6	bit de donnée 4	A	14	PB4
7	bit de donnée 5	A	15	PB5
8	bit de donnée 6	A	16	PB6
9	bit de donnée 7	A	17	PB7
10	Acknowledge	A	24	Flag
11	busy	B	2	PA0
			et 39	SP

Centronics N° de broche	Fonction	CIA	Broche	Descriptif
12	paper out	B	3	PA1
			et 40	CNT
13	select	B	4	PA2

Annexe E : Le connecteur série

	1	GND	(FRAME GROUND) masse de protection
sortie	2	TXD	(TRANSMIT DATA) donnée transmise
entrée	3	RXD	(RECEIVE DATA) donnée reçue
sortie	4	RTS	(REQUEST TO SEND) demande d'émission
entrée	5	CTS	(CLEAR TO SEND) prêt à émettre
entrée	6	DSR	(DATA SET READY) donnée prête à envoyer
	7	GND	masse de référence
entrée	8	CD	(CARRIER DETECT) modem détecté
	9	+ 12 volts	
	10	- 12 volts	
sortie	11	AUDOUT	sortie audio à recevoir
entrée	22	RI	(RING INDICATOR)
	23	inutilisé	
	24	inutilisé	
	25	inutilisé	

Certains signaux sont disposés d'une autre façon sur l'AMIGA 1000 :

	9	inutilisé	
	10	inutilisé	
	11	inutilisé	
	12	inutilisé	
	13	inutilisé	
	14	- 5 volts	
sortie	15	AUDOUT	sortie audio
entrée	16	AUDIN	entrée audio
sortie	17	EB	horloge tampon (716 KHZ)
entrée	18	/INT2	entrée d'interruption de niveau 2
	19	inutilisé	
sortie	20	DTR	(Data Terminal Ready) terminal prêt à recevoir
	21	+ 5 volts	
	22	inutilisé	
	23	+ 12 volts	
sortie	24	MCLK	horloge de transmission à 3,58 MHZ
sortie	25	/MERS	RESET

Annexe F : Connecteur disquette externe

Entrée	1	/RDY	disque prêt
entrée	2	/DKRD	lecture données disquette
	3	GND	
	4	GND	
	5	GND	
	6	GND	
	7	GND	
sortie	8	/MTRX	moteur on/off
sortie	9	/SEL	sélection disque2 off
sortie	10	/DRES	floppyreset (arrêt moteur)
entrée	11	/CHNG	disque changé
	12	+5 volts	
sortie	13	/SIDE	sélection de la face de la disquette
entrée	14	/WPRO	protection en écriture
entrée	15	/TK0	piste 0
sortie	16	/DKWE	autorisation écriture
sortie	17	/DKWD	écriture donnée
sortie	18	/STEP	déplacement de la tête pas à pas
sortie	19	/DIR	direction déplacement de la tête (0 = intérieur)
sortie	20	/SEL3	sélection disque DF3
sortie	21	/SEL1	sélection disque DF1
entrée	22	/INDEX	index début de cylindre
	23	+ 12 volts	

Toutes les broches impaires sont de type GND.

2	/CHNG	4	/INUSE
6	inutilisé	8	INDEX
10	SEL0	12	inutilisé
14	inutilisé	16	/MTR0
18	DIR	20	/STEP
22	/DKWD	24	/DKWE
26	/TK0	28	/WPRD
30	/DKRD	32	/SIDE
34	/RDY		

Boîtier d'alimentation pour lecteur de disquette interne :

1	+ 5 volts
2	GND
3	GND
4	+ 12 volts

Annexe G : Le connecteur souris-joystick

	Souris	Joystick 1	Joystick 2	Light Pen
entrée 1	impulsion V	avant	inutilisé	inutilisé
entrée 2	impulsion H	arrière	inutilisé	inutilisé
entrée 3	impulsion VQ	gauche	bouton gauche	inutilisé
entrée 4	impulsion HQ	droite	bouton droit	inutilisé
E/S 5	bouton 2	inutilisé	potentiomètre droit	crayon touche l'écran
E/S 6	bouton 1	bouton feu	inutilisé	signal LP
7	+ 5 volts	+ 5 volts	+ 5 volts	+ 5 volts
8	GND	GND	GND	GND
E/S 9	bouton 3	inutilisé	potentiomètre vertical	inutilisé

GAMEPORT 0

Broche N°	Circuit	Broche
1	Denise	M0V
2	Denise	M0H
3	Denise	M1V
4	Denise	M1H
5	Paula	POY
6	CIA-A	PA-6
9	Paula	POX

GAMEPORT 1

Broche N°	Circuit	Broche
1	Denise	M0V
2	Denise	M0H
3	Denise	M1V
4	Denise	M1H
5	Paula	P1Y
6	CIA-A	PA7
9	Paula	P1X

Annexe H : Connecteur d'extension

1	GND	2	GND
3	GND	4	GND
5	+5 Volts	6	+5 Volts
7	extension	8	-5 Volts
9	extension (28 MHz)	10	+12 Volts
11	extension	12	Config
13	GND	14	/C3
15	CDAC	16	/C1
17	/OVR	18	XRDY
19	/INT2	20	/PALOPE
21	A5	22	/INT6
23	A6	24	A4
25	GND	26	A3
27	A2	28	A7
29	A1	30	A8
31	FC0	32	A9
33	FC1	34	A10
35	FC2	36	A11
37	GND	38	A12
39	A13	40	/IPL0
41	A14	42	/IPL1
43	A15	44	/IPL2
45	A16	46	/BERR
47	A17	48	/VPA
49	GND	50	E
51	/VMA	52	A18
53	/RES	54	A19
55	/HLT	56	A20
57	A22	58	A21
59	A23	60	/BR
61	GND	62	/BGACK
63	PD15	64	/BG
65	PD14	66	/DTACK
67	PD13	68	/PRW
69	PD12	70	/LDS
71	PD11	72	/UDS
73	GND	74	/AS
75	PD0	76	PD10
77	PD1	78	PD9
79	PD2	80	PD8
81	PD3	82	PD7
83	PD4	84	PD6
85	GND	86	PD5

Annexe I : Adresses des registres spécialisés

L'adresse de base est : \$DFF000

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
BLTDDAT	000	A	er	d	destination BLITTER
DMACONR	002	AP	r	p	registre de contrôle DMA
VPOSR	004	A	r	p	position verticale ; bit de poids fort
VHPOSR	006	A	r	p	position horizontale et verticale
DSDATR	008	P	er	d	lecture données disque
JOYODAT	00A	D	r	p	position souris/joystick gameport 0
JOY1DAT	00C	D	r	p	position souris/joystick gameport 1
CLXDAT	00E	D	r	p	registre des collisions
ADKCONR	010	P	r	p	contrôle AUDIO/DISQUE
POTODAT	012	P	r	p	potentiomètre gameport 0
POT1DAT	014	P	r	p	potentiomètre gameport 1
POTGOR	016	P	r	p	lecture données sur POT
SERDATR	018	P	r	p	status du port série
DSKBYTR	01A	P	r	p	status du port disque
INTENAR	01C	P	r	p	autorisation interrup.
INTREQR	01E	P	r	p	demande interruption
DSKPTH	020	A	w	p	adresse DMA disque bits 16-18
DSKPTL	022	A	w	p	adresse DMA disque bits 1-15
DSKLEN	024	P	w	p	longueur données disque
DSKDAT	026	P	w	d	données disque
REFPTR	028	A	w	d	compteur rafraîchis.
VPOSW	02A	A	w	p	MSB de la position vert.
VHPOSW	02C	A	w	p	position verticale et horizontale
COPCON	02E	A	w	p	registre de contrôle COPPER
SERDAT	030	P	w	p	données série et bit de stop
SERPER	032	P	w	p	période et contrôle du port série

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
POTGOT	034	P	w	p	démarrage compteur PDT
JOYTEST	036	D	w	p	écriture dans 2 compteurs souris/joystick
STREQU	038	D	s	d	synchronisation vert.
STRVBL	03A	D	s	d	synchronisation horiz.
STRHOR	03C	DP	s	d	signal synchronisation horizontale
STRLONG	03E	D	s	d	identification longueur horizontale

Seuls les registres suivants peuvent être accessibles au COPPER lorsque, COPCON = 1.

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
BLTCONO	040	A	w	p	contrôle Blitter
BLTCON1	042	A	w	p	contrôle Blitter
BLTAFWM	044	A	w	p	masque premier mot
BLTALWM	046	A	w	p	masque dernier mot
BLTCPPTH	048	A	w	p	source C (bits 16-18)
BLTCPTL	04A	A	w	p	source C (bits 1-15)
BLTBPTH	04C	A	w	p	source B (bits 16-18)
BLTBPTL	04E	A	w	p	source B (bits 1-15)
BLTAPTH	050	A	w	p	source A (bits 16-18)
BLTAPTL	052	A	w	p	source A (bits 1-15)
BLTDPTH	054	A	w	p	destination D (bits 16-18)
BLTDPTL	056	A	w	p	destination D (bits 1-15)
BLTSIZE	058	A	w	p	démarrage Blitter et dimensionnement taille fenêtre
	05A				inutilisé
	05C				inutilisé
	05E				inutilisé
BLTCMOD	060	A	w	p	modulo source C
BLTBMOD	062	A	w	p	modulo source B

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
BLTAMOD	064	A	w	p	modulo source A
BLTDMOD	066	A	w	p	modulo destination D
	068				inutilisé
	06A				inutilisé
	06C				inutilisé
	06E				inutilisé
BLTCDAT	070	A	w	d	données source C
BLTBDAT	072	A	w	d	données source B
BLTADAT	074	A	w	d	données source A
	076				inutilisé
	078				inutilisé
	07A				inutilisé
	07C				inutilisé
DSKSYNC	07E	P	w	p	remplissage sync. disque

Seuls les registres suivants peuvent être, dans tous les cas, accessibles en mode écriture par le COPPER.

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
COP1LCH	080	A	w	p	1ère adresse COPPER (bits 16-18)
COP1LCL	082	A	w	p	1ère adresse COPPER (bits 1-15)
COP2LCH	084	A	w	p	2ème adresse COPPER (bits 16-18)
COP2LCL	086	A	w	p	2ème adresse COPPER (bits 1-15)
COPJMP1	088	A	s	p	redémarrage COPPER 1ère adresse
COPJMP2	08A	A	s	p	redémarrage COPPER 2ème adresse
COPINS	08C	A	w	d	identification instruc.
DIWSTRT	08E	A	w	p	coin sup. gauche fenêtre
DIWSTOP	090	A	w	p	coin inf. droit fenêtre
DDFSTRT	092	A	w	p	début bitplane (pos.hor)
DDFSTOP	094	A	w	p	fin bitplane (pos.hor)
DMACON	096	ADP	w	p	registre contrôle DMA
CLXCON	098	D	w	p	contrôle collision

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
INTENA	09A	P	W	p	autorisation interrup.
INTREQ	09C	P	W	p	demande d'interruption
ADKCON	09E	P	W	p	contrôle audio,disk,UART
AUDOLCH	0A0	A	W	p	canal audio 0 (bits 16-18)
AUDOLCL	0A2	A	W	p	canal audio 0 (bits 1-15)
AUDOLEN	0A4	P	W	p	longueur données audio
AUDOPER	0A6	P	W	p	période canal audio 0
AUDOVOL	0A8	P	W	p	volume canal audio 0
AUDODAT	0AA	P	W	d	canal audio 0 données
	OAC				inutilisé
	OAE				inutilisé
AUD1LCH	0B0	A	W	p	canal audio (1 bits 16-18)
AUD1LCL	0B2	A	W	p	canal audio 1 (bits 1-15)
AUD1LEN	0B4	P	W	p	longueur données audio
AUD1PER	0B6	P	W	p	période canal audio 1
AUD1VOL	0B8	P	W	p	volume canal audio 1
AUD1DAT	0BA	P	W	d	canal audio 1 données
	OBC				inutilisé
	OBE				inutilisé
AUD2LCH	0C0	A	W	p	canal audio 2 (bits 16-18)
AUD2LCL	0C2	A	W	p	canal audio 2 (bits 1-15)
AUD2LEN	0C4	P	W	p	longueur données audio
AUD2PER	0C6	P	W	p	période canal audio 2
AUD2VOL	0C8	P	W	p	volume canal audio 2
AUD2DAT	0CA	P	W	d	canal audio 2 données
	OCC				inutilisé
	OCE				inutilisé
AUD3LCH	0D0	A	W	p	canal audio 3 (bits 16-18)
AUD3LCL	0D2	A	W	p	canal audio 3 (bits 1-15)
AUD3LEN	0D4	P	W	p	longueur données audio
AUD3PER	0D6	P	W	p	période canal audio 3
AUD3VOL	0D8	P	W	p	volume canal audio 3

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
AUD3DAT	0DA	P	w	d	canal audio 0 données
	0DC				inutilisé
	0DE				inutilisé
BPL1PTH	0E0	A	w	p	bitplane 1 (bits 16-18)
BPL1PTL	0E2	A	w	p	bitplane 1 (bits 1-15)
BPL2PTH	0E4	A	w	p	bitplane 2 (bits 16-18)
BPL2PTL	0E6	A	w	p	bitplane 2 (bits 1-15)
BPL3PTH	0E8	A	w	p	bitplane 3 (bits 16-18)
BPL3PTL	0EA	A	w	p	bitplane 3 (bits 1-15)
BPL4PTH	0EC	A	w	p	bitplane 4 (bits 16-18)
BPL4PTL	0EE	A	w	p	bitplane 4 (bits 1-15)
BPL5PTH	0F0	A	w	p	bitplane 5 (bits 16-18)
BPL5PTL	0F2	A	w	p	bitplane 5 (bits 1-15)
BPL6PTH	0F4	A	w	p	bitplane 6 (bits 16-18)
BPL6PTL	0F6	A	w	p	bitplane 6 (bits 1-15)
	0F8				inutilisé
	0FA				inutilisé
	0FC				inutilisé
	0FE				inutilisé
BPLCON0	100	AD	w	p	reg. contrôle 0 bitplane
BPLCON1	102	D	w	p	reg. contrôle 1 bitplane
BPLCON2	104	D	w	p	reg. contrôle 2 bitplane
	106				inutilisé
BPL1MOD	108	A	w	p	contr. bitplane modulo plan impair
BPL2MOD	10A	A	w	p	contr. bitplane modulo plan pair
	10C				inutilisé
	10E				inutilisé
BPL1DAT	110	D	w	d	données bitplane 1 (RGB)
BPL2DAT	112	D	w	d	données bitplane 2 (RGB)
BPL3DAT	114	D	w	d	données bitplane 3 (RGB)
BPL4DAT	116	D	w	d	données bitplane 4 (RGB)
BPL5DAT	118	D	w	d	données bitplane 5 (RGB)

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
BPL6DAT	11A	D	w	d	données bitplane 6 (RGB)
	11C				inutilisé
	11E				inutilisé
SPROPTH	120	A	w	p	données sprite 0 (bits 16-18)
SPROPTL	122	A	w	p	données sprite 0 (bits 1-15)
SPR1PTH	124	A	w	p	données sprite 1 (bits 16-18)
SPR1PTL	126	A	w	p	données sprite 1 (bits 1-15)
SPR2PTH	128	A	w	p	données sprite 2 (bits 16-18)
SPR2PTL	12A	A	w	p	données sprite 2 (bits 1-15)
SPR3PTH	12C	A	w	p	données sprite 3 (bits 16-18)
SPR3PTL	12E	A	w	p	données sprite 3 (bits 1-15)
SPR4PTH	130	A	w	p	données sprite 4 (bits 16-18)
SPR4PTL	132	A	w	p	données sprite 4 (bits 1-15)
SPR5PTH	134	A	w	p	données sprite 5 (bits 16-18)
SPR5PTL	136	A	w	p	données sprite 5 (bits 1-15)
SPR6PTH	138	A	w	p	données sprite 6 (bits 16-18)
SPR6PTL	13A	A	w	p	données sprite 6 (bits 1-15)
SPR7PTH	13C	A	w	p	données sprite 7 (bits 16-18)
SPR7PTL	13E	A	w	p	données sprite 7 (bits 1-15)
SPROPOS	140	AD	w	dp	position départ sprite 0
SPROCTL	142	AD	w	dp	contrôle sprite 0
SPRODATA	144	D	w	dp	données A sprite 0 (RGB)
SPRODATB	146	D	w	dp	données B sprite 0 (RGB)
SPR1POS	148	AD	w	dp	position départ sprite 1
SPR1CTL	14A	AD	w	dp	contrôle sprite 1
SPR1DATA	14C	D	w	dp	données A sprite 1 (RGB)
SPR1DATB	14E	D	w	dp	données B sprite 1 (RGB)
SPR2POS	150	AD	w	dp	position départ sprite 2
SPR2CTL	152	AD	w	dp	contrôle sprite 2
SPR2DATA	154	D	w	dp	données A sprite 2 (RGB)
SPR2DATB	156	D	w	dp	données B sprite 2 (RGB)
SPR3POS	158	AD	w	dp	position départ sprite 3

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
SPR3CTL	15A	AD	w	dp	contrôle sprite 3
SPR3DATA	15C	D	w	dp	données A sprite 3 (RGB)
SPR3DATB	15E	D	w	dp	données B sprite 3 (RGB)
SPR4POS	160	AD	w	dp	position départ sprite 4
SPR4CTL	162	AD	w	dp	contrôle sprite 4
SPR4DATA	164	D	w	dp	données A sprite 4 (RGB)
SPR4DATB	166	D	w	dp	données B sprite 4 (RGB)
SPR5POS	168	AD	w	dp	position départ sprite 5
SPR5CTL	16A	AD	w	dp	contrôle sprite 5
SPR5DATA	16C	D	w	dp	données A sprite 5 (RGB)
SPR5DATB	16E	D	w	dp	données B sprite 5 (RGB)
SPR6POS	170	AD	w	dp	position départ sprite 6
SPR6CTL	172	AD	w	dp	contrôle sprite 6
SPR6DATA	174	D	w	dp	données A sprite 6 (RGB)
SPR6DATB	176	D	w	dp	données B sprite 6 (RGB)
SPR7POS	178	AD	w	dp	position départ sprite 7
SPR7CTL	17A	AD	w	dp	contrôle sprite 7
SPR7DATA	17C	D	w	dp	données A sprite 7 (RGB)
SPR7DATB	17E	D	w	dp	données B sprite 7 (RGB)
COLOR00	180	D	w	p	valeur de la couleur 0
COLOR01	182	D	w	p	valeur de la couleur 1
COLOR02	184	D	w	p	valeur de la couleur 2
COLOR03	186	D	w	p	valeur de la couleur 3
COLOR04	188	D	w	p	valeur de la couleur 4
COLOR05	18A	D	w	p	valeur de la couleur 5
COLOR06	18C	D	w	p	valeur de la couleur 6
COLOR07	18E	D	w	p	valeur de la couleur 7
COLOR08	190	D	w	p	valeur de la couleur 8
COLOR09	192	D	w	p	valeur de la couleur 9
COLOR10	194	D	w	p	valeur de la couleur 10
COLOR11	196	D	w	p	valeur de la couleur 11
COLOR12	198	D	w	p	valeur de la couleur 12

NOM	Adresse registre	CHIP	R/W	p/d	Fonction
COLOR13	19A	D	w	p	valeur de la couleur 13
COLOR14	19C	D	w	p	valeur de la couleur 14
COLOR15	19E	D	w	p	valeur de la couleur 15
COLOR16	1A0	D	w	p	valeur de la couleur 16
COLOR17	1A2	D	w	p	valeur de la couleur 17
COLOR18	1A4	D	w	p	valeur de la couleur 18
COLOR19	1A6	D	w	p	valeur de la couleur 19
COLOR20	1A8	D	w	p	valeur de la couleur 20
COLOR21	1AA	D	w	p	valeur de la couleur 21
COLOR22	1AC	D	w	p	valeur de la couleur 22
COLOR23	1AE	D	w	p	valeur de la couleur 23
COLOR24	1B0	D	w	p	valeur de la couleur 24
COLOR25	1B2	D	w	p	valeur de la couleur 25
COLOR26	1B4	D	w	p	valeur de la couleur 26
COLOR27	1B6	D	w	p	valeur de la couleur 27
COLOR28	1B8	D	w	p	valeur de la couleur 28
COLOR29	1BA	D	w	p	valeur de la couleur 29
COLOR30	1BC	D	w	p	valeur de la couleur 30
COLOR31	1BE	D	w	p	valeur de la couleur 31

Les registres de 1C0 à 1FC sont inutilisés.

Annexe J - Fonctions des bibliothèques

diskfont.library

```

-$001E  -30  OpenDiskFont (textAttr)(A0)
-$0024  -36  AvailFonts (buffer,bufBytes,flags)(A0,D0,D1)
-$002A  -42  NewFontContents (fontsLock, fontName) (A0,A1)
-$0030  -48  DisposeFontContents (fontContentsHeader) (A1)

```

dos.library

```

-$001E  -30  Open (name,accessMode)(D1,D2)
-$0024  -36  Close (file)(D1)
-$002A  -42  Read (file,buffer,length)(D1,D2,D3)
-$0030  -48  Write (file,buffer,length)(D1,D2,D3)
-$0036  -54  Input ()
-$003C  -60  Output ()
-$0042  -66  Seek (file,position,offset)(D1,D2,D3)
-$0048  -72  DeleteFile (name)(D1)
-$004E  -78  Rename (oldName,newName)(D1,D2)
-$0054  -84  Lock (name,type)(D1,D2)
-$005A  -90  UnLock (lock)(D1)
-$0060  -96  DupLock (lock)(D1)
-$0066  -102 Examine (lock,fileInfoBlock)(D1,D2)
-$006C  -108 ExNext (lock,fileInfoBlock)(D1,D2)
-$0072  -114 Info (lock,parameterBlock)(D1,D2)
-$0078  -120 CreateDir (name)(D1)
-$007E  -126 CurrentDir (lock)(D1)
-$0084  -132 IoErr ()
-$008A  -138 CreateProc (name,pri,segList,stackSize)(D1,D2,D3,D4)
-$0090  -144 Exit (returnCode)(D1)
-$0096  -150 LoadSeg (fileName)(D1)
-$009C  -156 UnLoadSeg (segment)(D1)
-$00A2  -162 GetPacket (wait)(D1)
-$00A8  -168 QueuePacket (packet)(D1)
-$00AE  -174 DeviceProc (name)(D1)
-$00B4  -180 SetComment (name,comment)(D1,D2)
-$00BA  -186 SetProtection (name,mask)(D1,D2)
-$00C0  -192 DateStamp (date)(D1)
-$00C6  -198 Delay (timeout)(D1)
-$00CC  -204 WaitForChar (file,timeout)(D1,D2)
-$00D2  -210 ParentDir (lock)(D1)
-$00D8  -216 IsInteractive (file)(D1)
-$00DE  -222 Execute (string,file,file)(D1,D2,D3)

```

exec.library

```

-$001E  -30  Supervisor ()
-$0024  -36  ExitIntr ()
-$002A  -42  Schedule ()
-$0030  -48  Reschedule ()
-$0036  -54  Switch ()
-$003C  -60  Dispatch ()

```

```

-$0042 -66 Exception ()
-$0048 -72 InitCode (startClass,version)(D0,D1)
-$004E -78 InitStruct (initTable,memory,size)(A1,A2,D0)
-$0054 -84 MakeLibrary
(funcInit,structInit,libInit,dataSize,codeSize)
(A0,A1,A2,D0,D1)
-$005A -90 MakeFunctions
(target,functionArray,funcDispBase)(A0,A1,A2)
-$0060 -96 FindResident (name)(A1)
-$0066 -102 InitResident (resident,segList)(A1,D1)
-$006C -108 Alert (alertNum,parameters)(D7,A5)
-$0072 -114 Debug ()
-$0078 -120 Disable ()
-$007E -126 Enable ()
-$0084 -132 Forbid ()
-$008A -138 Permit ()
-$0090 -144 SetSR (newSR,mask)(D0,D1)
-$0096 -150 SuperState ()
-$009C -156 UserState (sysStack)(D0)
-$00A2 -162 SetIntVector (intNumber,interrupt)(D0,A1)
-$00A8 -168 AddIntServer (intNumber,interrupt)(D0,A1)
-$00AE -174 RemIntServer (intNumber,interrupt)(D0,A1)
-$00B4 -180 Cause (interrupt)(A1)
-$00BA -186 Allocate (freeList,byteSize)(A0,D0)
-$00C0 -192 Deallocate (freeList,memoryBlock,byteSize)(A0,A1,D0)
-$00C6 -198 AllocMem (byteSize,requirements)(D0,D1)
-$00CC -204 AllocAbs (byteSize,location)(D0,A1)
-$00D2 -210 FreeMem (memoryBlock,byteSize)(A1,D0)
-$00D8 -216 AvailMem (requirements)(D1)
-$00DE -222 AllocEntry (entry)(A0)
-$00E4 -228 FreeEntry (entry)(A0)
-$00EA -234 Insert (list,node,pred)(A0,A1,A2)
-$00FO -240 AddHead (list,node)(A0,A1)
-$00F6 -246 AddTail (list,node)(A0,A1)
-$00FC -252 Remove (node)(A1)
-$0102 -258 RemHead (list)(A0)
-$0108 -264 RemTail (list)(A0)
-$010E -270 Enqueue (list,node)(A0,A1)
-$0114 -276 FindName (list,name)(A0,A1)
-$011A -282 AddTask (task,initPC,finalPC)(A1,A2,A3)
-$0120 -288 RemTask (task)(A1)
-$0126 -294 FindTask (name)(A1)
-$012C -300 SetTaskPri (task,priority)(A1,D0)
-$0132 -306 SetSignal (newSignals,signalSet)(D0,D1)
-$0138 -312 SetExcept (newSignals,signalSet)(D0,D1)
-$013E -318 Wait (signalSet)(D0)
-$0144 -324 Signal (task,signalSet)(A1,D0)
-$014A -330 AllocSignal (signalNum)(D0)
-$0150 -336 FreeSignal (signalNum)(D0)
-$0156 -342 AllocTrap (trapNum)(D0)
-$015C -348 FreeTrap (trapNum)(D0)
-$0162 -354 AddPort (port)(A1)
-$0168 -360 RemPort (port)(A1)
-$016E -366 PutMsg (port,message)(A0,A1)
-$0174 -372 GetMsg (port)(A0)
-$017A -378 ReplyMsg (message)(A1)
-$0180 -384 WaitPort (port)(A0)
-$0186 -390 FindPort (name)(A1)
-$018C -396 AddLibrary (library)(A1)

```

```

-$0192 -402 RemLibrary (library)(A1)
-$0198 -408 OldOpenLibrary (libName)(A1)
-$019E -414 CloseLibrary (library)(A1)
-$01A4 -420 SetFunction (library,funcOffset,funcEntry)(A1,A0,D0)
-$01AA -426 SumLibrary (library)(A1)
-$01B0 -432 AddDevice (device)(A1)
-$01B6 -438 RemDevice (device)(A1)
-$01BC -444 OpenDevice (devName,unit,ioRequest,flags)(A0,D0,A1,D1)
-$01C2 -450 CloseDevice (ioRequest)(A1)
-$01C8 -456 DoIO (ioRequest)(A1)
-$01CE -462 SendIO (ioRequest)(A1)
-$01D4 -468 CheckIO (ioRequest)(A1)
-$01DA -474 WaitIO (ioRequest)(A1)
-$01E0 -480 AbortIO (ioRequest)(A1)
-$01E6 -486 AddResource (resource)(A1)
-$01EC -492 RemResource (resource)(A1)
-$01F2 -498 OpenResouce (resName,version)(A1,D0)
-$01F8 -504 RawIOInit ()
-$01FE -510 RawMayGetChar ()
-$0204 -516 RawPutChar (char)(D0)
-$020A -522 RawDoFmt ()(A0,A1,A2,A3)
-$0210 -528 GetCC ()
-$0216 -534 TypeOfMem (address)(A1)
-$021C -540 Procure (semaport,bidMsg)(A0,A1)
-$0222 -546 Vacate (semaport)(A0)
-$0228 -552 OpenLibrary (libName,version)(A1,D0)
-$022E -558 InitSemaphore (SignalSemaphore)(A0)
-$0234 -564 ObtainSemaphore (SignalSemaphore)(A0)
-$023A -570 ReleaseSemaphore (SignalSemaphore)(A0)
-$0240 -576 AttemptSemaphore (SignalSemaphore)A0)
-$0246 -582 ObtainSemaphoreList (List)(A0)
-$024C -588 ReleaseSemaphoreList (List)(A0)
-$0252 -594 FindSemaphore (SignalSemaphore)(A0)
-$0258 -600 AddSemaphore (SignalSemaphore) (A0)
-$025E -606 RemSemaphore (SignalSemaphore)(A0)
-$0264 -612 SumKickData (SignalSemaphore)(A0)
-$026A -618 AddMemList (Size,Attr,Pri,BasePtr,Name)(D0,D1,D2,A0,A1)
-$0270 -624 CopyMem (SourcePtr,DestPtr,Size)(A0,A1,D0)
-$276 -630 CopyMemQuick (SourcePtr,DestPtr,Size)(A0,A1,D0)

```

graphics.library

```

-$001E -30 BltBitMap
(srcBitMap,srcX,srcY,destBitMap,destX,destY,sizeX,sizeY,
 minterm.mask,tempa) (A0,D0,D1,A1,D2,D3,D4,D5,D6,D7,A2)
-$0024 -36 BltTemplate
(source,srcX,srcMod,destRastPort,destX,destY,sizeX,sizeY)
 (A0,D0,D1,A1,D2,D3,D4,D5)
-$002A -42 ClearEOL (rastPort)(A1)
-$0030 -48 ClearScreen (rastPort)(A1)
-$0036 -54 TextLength (RastPort,string,count)(A1,A0,D0)
-$003C -60 Text (RastPort,String,count)(A1,A0,D0)
-$0042 -66SetFont (RastPortID,textField)(A1,A0)
-$0048 -72 OpenFont (textAttr)(A0)
-$004E -78 CloseFont (textField)(A1)
-$0054 -84 AskSoftStyle (rastPort)(A1)
-$005A -90 SetSoftStyle (rastPort,style,enable)(A1,D0,D1)
-$0060 -96 AddBob (bob,rastPort)(A0,A1)

```

```

-$0066 -102 AddVSprite (vSprite,rastPort)(A0,A1)
-$006C -108 DoCollision (rastPort)(A1)
-$0072 -114 DrawGLList (rastPort,viewPort)(A1,A0)
-$0078 -120 InitGels (dummyHead,dummyTail,GelsInfo)(A0,A1,A2)
-$007E -126 InitMasks (vSprite)(A0)
-$0084 -132 RemIBob (bob,rastPort,viewPort)(A0,A1,A2)
-$008A -138 RemVSprite (vSprite)(A0)
-$0090 -144 SetCollision (type,routine,gelsInfo)(D0,A0,A1)
-$0096 -150 SortGLList (rastPort)(A1)
-$009C -156 AddAnimObj (obj,animationKey,rastPort)(A0,A1,A2)
-$00A2 -162 Animate (animationKey,rastPort)(A0,A1)
-$00A8 -168 GetGBuffers (animationObj,rastPort,doubleBuffer)(A0,A1,D0)
-$00AE -174 InitGMasks (animationObj)(A0)
-$00B4 -180 GelsFuncE ()
-$00BA -186 GelsFuncF ()
-$00C0 -192 LoadRGB4 (viewPort,colors,count)(A0,A1,D0)
-$00C6 -198 InitRastPort (rastPort)(A1)
-$00CC -204 InitVPort (viewPort)(A0)
-$00D2 -210 MrgCop (view)(A1)
-$00D8 -216 MakeVPort (view,viewPort)(A0,A1)
-$00DE -222 LoadView (view)(A1)
-$00E4 -228 WaitBlit ()
-$00EA -234 SetRast (rastPort,color)(A1,D0)
-$00F0 -240 Move (rastPort,x,y)(A1,D0,D1)
-$00F6 -246 Draw (rastPort,x,y)(A1,D0,D1)
-$00FC -252 AreaMove (rastPort,x,y)(A1,D0,D1)
-$0102 -258 AreaDraw (rastPort,x,y)(A1,D0,D1)
-$0108 -264 AreaEnd (rastPort)(A1)
-$010E -270 WaitTOF ()
-$0114 -276 OBlit (blit)(A1)
-$011A -282 InitArea (areaInfo,vectorTable,vectorTableSize)(A0,A1,D0)
-$0120 -288 SetRGB4 (viewPort,index,r,g,b)(A0,D0,D1,D2,D3)
-$0126 -294 OBSBlit (blit)(A1)
-$012C -300 BltClear (memory,size,flags)(A1,D0,D1)
-$0132 -306 RectFill (rastPort,x1,y1,xu,yu)(A1,D0,D1,D2,D3)
-$0138 -312 BltPattern
(rastPort,ras,x1,y1,maxX,maxY,fillBytes)(A1,A0,D0,D1,D2,D3,D4)
-$013E -318 ReadPixel (rastPort,x,y)(A1,D0,D1)
-$0144 -324 WritePixel (rastPort,x,y)(A1,D0,D1)
-$014A -330 Flood (rastPort,mode,x,y)(A1,D2,D0,D1)
-$0150 -336 PolyDraw (rastPort,count,polyTable)(A1,D0,A0)
-$0156 -342 SetAPen (rastPort,pen)(A1,D0)
-$015C -348 SetBPen (rastPort,pen)(A1,D0)
-$0162 -354 SetDrMd (rastPort,drawMode)(A1,D0)
-$0168 -360 initView (view)(A1)
-$016E -366 CBump (copperList)(A1)
-$0174 -372 CMove (copperList,destination,data)(A1,D0,D1)
-$017A -378 CWait (copperList,x,y)(A1,D0,D1)
-$0180 -384 VBeamPos ()
-$0186 -390 InitBitMap (bitMap,depth,width,heighth)(A0,D0,D1,D2)
-$018C -396 ScrollRaster
(rastPort,dx,dY,minx,miny,maxx,maxy)(A1,D0,D1,D2,D3,D4,D5)
-$0192 -402 WaitBOVP (viewPort)(A0)
-$0198 -408 GetSprite (simpleSprite,num)(A0,D0)
-$019E -414 FreeSprite (num)(D0)
-$01A4 -420 ChangeSprite (vp.simpleSprite,data)(A0,A1,A2)
-$01AA -426 MoveSprite (viewPort,simpleSprite,x,y)(A0,A1,D0,D1)
-$01B0 -432 LockLayerRom (layer)(A5)
-$01B6 -438 UnlockLayerRom (layer)(A5)

```

```

-$01BC -444 SyncSBitMap (1)(A0)
-$01C2 -450 CopySBitMap (11,12)(A0,A1)
-$01C8 -456 OwnBlitter ()
-$01CE -462 DisownBlitter ()
-$01D4 -468 InitTmpRas (tmpras,buff,size)(A0,A1,D0)
-$01DA -474 AskFont (rastPort,textAttr)(A1,A0)
-$01E0 -480 AddFont (textFont)(A1)
-$01E6 -486 RemFont (textFont)(A1)
-$01EC -492 AllocRaster (width,height)(D0,D1)
-$01F2 -498 FreeRaster (planepr,width,height)(A0,D0,D1)
-$01F8 -504 AndRectRegion (rgn,rect)(A0,A1)
-$01FE -510 OrRectRegion (rgn,rect)(A0,A1)
-$0204 -516 NewRegion ()
-$020A -522 ** réservée **
-$0210 -528 ClearRegion (rgn)(A0)
-$0216 -534 DisposeRegion (rgn)(A0)
-$021C -540 FreeVPortCopLists (viewPort)(A0)
-$0222 -546 FreeCopList (coplist)(A0)
-$0228 -552 ClipBlit
(srcrp,srcX,srcY,destrp,destX,destY,sizeX,sizeY,minterm)
(A0,D0,D1,A1,D2,D3,D4,D5,D6)
-$022E -558 XorRectRegion (rgn,rect)(A0,A1)
-$0234 -564 FreeCprList (cprlist)(A0)
-$023A -570 GetColorMap (entries)(D0)
-$0240 -576 FreeColorMap (colormap)(A0)
-$0246 -582 GetRGB4 (colormap,entry)(A0,D0)
-$024C -588 ScrollVPort (vp)(A0)
-$0252 -594 UCopperListInit (copperlist,num)(A0,D0)
-$0258 -600 FreeGBuffers
(animationObj,rastPort,doubleBuffer)(A0,A1,D0)
-$025E -606 BltBitMapRastPort
(srcbm,srcx,srcy,destrp,destX,destY,sizeX,sizeY,minter)
(A0,D0,D1,A1,D2,D3,D4,D5,D6)

```

icon.library

```

-$001E -30 GetWBObject (name)(A0)
-$0024 -36 PutWBObject (name,object)(A0,A1)
-$002A -42 GetIcon (name,icon,freelist)(A0,A1,A2)
-$0030 -48 PutIcon (name,icon)(A0,A1)
-$0036 -54 FreeFreeList (freelist)(A0)
-$003C -60 FreeWBObject (WBObject)(A0)
-$0042 -66 AllocWBObject ()
-$0048 -72 AddFreeList (freelist,mem,size)(A0,A1,A2)
-$004E -78 GetDiskObject (name)(A0)
-$0054 -84 PutDiskObject (name,diskobj)(A0,A1)
-$005A -90 FreeDiskObj (diskobj)(A0)
-$0060 -96 FindToolType (toolTypeArray,typeName)(A0,A1)
-$0066 -102 MatchToolValue (typeString,value)(A0,A1)
-$006C -108 BumbRevision (newname,oldname)(A0,A1)

```

intuition.library

```

-$001E -30 OpenIntuition ()
-$0024 -36 Intuition (ievent)(A0)
-$002A -42 AddGadget (AddPtr,Gadget,Position)(A0,A1,D0)

```

-\$0030 -48 ClearDMRequest (Window)(A0)
 -\$0036 -54 ClearMenuStrip (Window)(A0)
 -\$003C -60 ClearPointer (Window)(A0)
 -\$0042 -66 CloseScreen (Screen)(A0)
 -\$0048 -72 CloseWindow (Window)(A0)
 -\$004E -78 CloseWorkBench ()
 -\$0054 -84 CurrentTime (Seconds,Micros)(A0,A1)
 -\$005A -90 DisplayAlert (AlertNumber,String,Height)(D0,A0,D1)
 -\$0060 -96 DisplayBeep (Screen)(A0)
 -\$0066 -102 DoubleClick
 (sseconds,smicros,cseconds,cmicros)(D0,D1,D2,D3)
 -\$006C -108 DrawBorder
 (Rport,Border,LeftOffset,TopOffset)(A0,A1,D0,D1)
 -\$0072 -114 DrawImage (RPort,Image,LeftOffset,TopOffset)(A0,A1,D0,D1)
 -\$0078 -120 EndRequest (requester,window)(A0,A1)
 -\$007E -126 GetDefPrefs (preferences,size)(A0,D0)
 -\$0084 -132 GetPrefs (preferences,size)(A0,D0)
 -\$008A -138 InitRequester (req)(A0)
 -\$0090 -144 ItemAddress (MenuStrip,MenuNumber)(A0,D0)
 -\$0096 -150 ModifyIDCMP (Window,Flags)(A0,D0)
 -\$009C -156 ModifyProp (Gadget,Ptr,Reg,Flags,HPos,VPos,HBody,VBody)
 (A0,A1,A2,D0,D1,D2,D3,D4)
 -\$00A2 -162 MoveScreen (Screen,dx,dy)(A0,D0,D1)
 -\$00A8 -168 MoveWindow (Window,dx,dy)(A0,D0,D1)
 -\$00AE -174 OffGadget (Gadget,Ptr,Req)(A0,A1,A2)
 -\$00B4 -180 OffMenu (Window,MenuNumber)(A0,D0)
 -\$00BA -186 OnGadget (Gadget,Ptr,Req)(A0,A1,A2)
 -\$00C0 -192 OnMenu (Window,MenuNumber)(A0,D0)
 -\$00C6 -198 OpenScreen (OSArgs)(A0)
 -\$00CC -204 OpenWindow (OWArgs)(A0)
 -\$00D2 -210 OpenWorkBench ()
 -\$00D8 -216 PrintIText (rp,itext,left,top)(A0,A1,D0,D1)
 -\$00DE -222 RefreshGadgets (Gadgets,Ptr,Req)(A0,A1,A2)
 -\$00E4 -228 RemoveGadgets (RemPtr,Gadget)(A0,A1)
 -\$00EA -234 ReportMouse (Window,Boolean)(A0,D0)
 -\$00FO -240 Request (Requester,Window)(A0,A1)
 -\$00F6 -246 ScreenToBack (Screen)(A0)
 -\$00FC -252 ScreenToFront (Screen)(A0)
 -\$0102 -258 SetDMRequest (Window,req)(A0,A1)
 -\$0108 -264 SetMenuStrip (Window,Menu)(A0,A1)
 -\$010E -270 SetPointer (Window,Pointer,Height,Width,XOffset,YOffset)
 (A0,A1,D0,D1,D2,D3)
 -\$0114 -276 SetWindowTitle (Window,windowTitle,screenTitle)(A0,A1,A2)
 -\$011A -282 ShowTitle (Screen,ShowIt)(A0,D0)
 -\$0120 -288 SizeWindow (Window,dx,dy)(A0,D0,D1)
 -\$0126 -294 ViewAddress ()
 -\$012C -300 ViewPortAddress (Window)(A0)
 -\$0132 -306 WindowToBack (Window)(A0)
 -\$0138 -312 WindowToFront (Window)(A0)
 -\$013E -318 WindowLimits
 (Window,minwidth,minheight,maxwidth,maxheight) (A0,D0,D1,D2,D3)
 -\$0144 -324 SetPrefs (preferences,size,flag)(A0,D0,D1)
 -\$014A -330 IntuitextLength (itext)(A0)
 -\$0150 -336 WBenchToBack ()
 -\$0156 -342 WBenchToFront ()
 -\$015C -348 AutoRequest (Window,Body,PText,NText,PFlag,NFlag,W,H)
 (A0,A1,A2,A3,D0,D1,D2,D3)
 -\$0162 -354 BeginRefresh (Window)(A0)
 -\$0168 -360 BuildSysRequest (Window,Body,PostText,NegText,Flags,W,H)

```
(A0,A1,A2,A3,D0,D1,D2)
-$016E -366 EndRefresh (Window,Complete)(A0,D0)
-$0174 -372 FreeSysRequest (Window)(A0)
-$017A -378 MakeScreen (Screen)(A0)
-$0180 -384 RemakeDisplay ()
-$0186 -390 RethinkDisplay ()
-$018C -396 AllocRemember (RememberKey,Size,Flags)(A0,D0,D1)
-$0192 -402 AlohaWorkbench (wbport)(A0)
-$0198 -408 FreeRemember (RememberKey,ReallyForget)(A0,D0)
-$019E -414 LockIBase (dontknow)(D0)
-$01A4 -420 UnlockIBase (IBLock)(A0)
```

layers.library

```
-$001E -30 InitLayers (li)(A0)
-$0024 -36 CreateUpfrontLayer
(li,bm,x0,y0,x1,y1,flags,bm2)(A0,A1,D0,D1,D2,D3,D4,A2)
-$002A -42 CreateBehindLayer
(li,bm,x0,y0,x1,y1,flags,bm2)(A0,A1,D0,D1,D2,D3,D4,A2)
-$0030 -48 UpfrontLayer (li,layer)(A0,A1)
-$0036 -54 BehindLayer (li,layer)(A0,A1)
-$003C -60 MoveLayer (li,layer,dx,dy)(A0,A1,D0,D1)
-$0042 -66 SizeLayer (li,layer,dx,dy)(A0,A1,D0,D1)
-$0048 -72 ScrollLayer (li,layer,dx,dy)(A0,A1,D0,01)
-$004E -78 BeginUpdate (layer)(A0)
-$0054 -84 EndUpdate (layer)(A0)
-$005A -90 DeleteLayer (li,layer)(A0,A1)
-$0060 -96 LockLayer (li,layer)(A0,A1)
-$0066 -102 UnlockLayer (li,layer)(A0,A1)
-$006C -108 LockLayers (li)(A0)
-$0072 -114 UnlockLayers (li)(A0)
-$0078 -120 LockLayerInfo (li)(A0)
-$007E -126 SwapBitsRastPortClipRect (rp,cr)(A0,A1)
-$0084 -132 WhichLayer (li,x,y)(A0,D0,D1)
-$008A -138 UnlockLayerInfo (li)(A0)
-$0090 -144 NewLayerInfo ()
-$0096 -150 DisposeLayerInfo (li)(A0)
-$009C -156 FattenLayerInfo (li)(A0)
-$00A2 -162 ThinLayerInfo (li)(A0)
-$00A8 -168 MoveLayerInFrontOf
(layer_to_move,layer_to_be_in_front_of)(A0,A1)
```

mathffp.library

```
-$001E -30 SPFix (float)(D0)
-$0024 -36 SPFlt (integer)(D0)
-$002A -42 SPCmp (leftFloat,rightFloat)(D1,D0)
-$0030 -48 SPTst (float)(D1)
-$0036 -54 SPAbs (float)(D0)
-$003C -60 SPNeg (float)(D0)
-$0042 -66 SPAdd (leftFloat,rightFloat)(D1,D0)
-$0048 -72 SPSub (leftFloat,rightFloat)(D1,D0)
-$004E -78 SPMul (leftFloat,rightFloat)(D1,D0)
-$0054 -84 SPDIV (leftFloat,rightFloat)(D1,D0)
```

mathieeedoubbas.library

```

-$001E -30 IEEEDPFix (integer,integer)(D0,D1)
-$0024 -36 IEEEDPFlt (integer)(D0)
-$002A -42 IEEEDPCmp (integer,integer,integer,integer)(D0,D1,D2,D3)
-$0030 -48 IEEEDPTst (integer,integer)(D0,D1)
-$0036 -54 IEEEDPAbs (integer,integer)(D0,D1)
-$003C -60 IEEEDPNeg (integer,integer)(D0,D1)
-$0042 -66 IEEEDPAdd (integer,integer,integer,integer)(D0,D1,D2,D3)
-$0048 -72 IEEEDPSub (integer,integer,integer,integer)(D0,D1,D2,D3)
-$004E -78 IEEEDPMul (integer,integer,integer,integer)(D0,D1,D2,D3)
-$0054 -84 IEEEDPDiv (integer,integer,integer,integer)(D0,D1,D2,D3)

```

mathtrans.library

```

-$001E -30 SPAtan (float)(D0)
-$0024 -36 SPSin (float)(D0)
-$002A -42 SPCos (float)(D0)
-$0030 -48 SPTan (float)(D0)
-$0036 -54 SPSincos (leftFloat,rightFloat)(D1,D0)
-$003C -60 SPSinh (float)(D0)
-$0042 -66 SPCosh (float)(D0)
-$0048 -72 SPTanh (float)(D0)
-$004E -78 SPExp (float)(D0)
-$0054 -84 SPLog (float)(D0)
-$005A -90 SPPow (leftFloat,rightFloat)(D1,D0)
-$0060 -96 SPSqrt (float)(D0)
-$0066 -102 SPTieee (float)(D0)
-$006C -108 SPFieee (float)(D0)
-$0072 -114 SPAasin (float)(D0)
-$0078 -120 SPAacos (float)(D0)
-$007E -126 SPLog10 (float)(D0)

```

Annexe K - Fonctions des bibliothèques classées par ordre alphabétique

*AbortIO	789
*ActivateGadget	830
*ActivateWindow	823
*AddGList	831
*AddMemList	797
*AddSemaphore	795
*Alert	751
*AttemptSemaphore	794
*CopyMem	797
*CopyMemQuick	798
*Debug	751
*FindResident	750
*FindSemaphore	795
*GetScreenData	850
*InitCode	749
*InitResident	750
*InitSemaphore	792
*LockIBase	863
*MakeFonctions	780
*NewModifyProp	833
*ObtainSemaphore	793
*ObtainSemaphoreList	794
*OldOpenLibrary	782
*RefreshGList	835
*RefreshWindowFrame	825
*ReleaseSemaphore	793
*ReleaseSemaphoreList	795
*RemoveGList	836
*RemSemaphore	796
*SetPrefs	871
*UnlockIBase	864
AddAnimOb	965
AddBob	965
AddConfigDev	1006
AddDevice	785
AddDosNode	1006
AddFont	959
AddFreeList	899
AddGadget	831
AddHead	764
AddIntServer	755
AddLibrary	779
AddPort	775
AddResource	791
AddTail	764

AddTask	768
AddVSprite	966
AllocAbs	759
Allocate	757
AllocBoardMem	1007
AllocConfigDev	1007
AllocEntry	761
AllocExpansionMem	1008
AllocMem	758
AllocRaster	908
AllocRemember	860
AllocSignal	772
AllocTrap	773
AllocWBObject	893
AlohaWorkbench	863
AndRectRegion	950
AndRegionRegion	950
Animate	966
AreaCircle	921
AreaDraw	922
AreaEllipse	922
AreaEnd	923
AreaMove	923
AskFont	960
AskSoftStyle	960
AttemptLockLayerRom	951
AutoRequest	843
AvailFonts	977
AvailMem	760
BeginRefresh	861
BeginUpdate	881
BehindLayer	882
BltBitMap	933
BltBitMapRastPort	934
BltClear	934
BltMaskBitMapRastPort	935
BltPattern	936
BltTemplate	936
BNDRYOFF	924
BuildSysRequest	844
BumpRevision	901
Cause	756
CBump	941
CEND	941
ChangeSprite	966
CheckIO	788
ClearDMRequest	844

ClearEOL	914
ClearMenuStrip	839
ClearPointer	855
ClearRectRegion	953
ClearRegion	953
ClearScreen	915
ClipBlit	937
Close	800
CloseDevice	787
CloseFont	961
CloseLibrary	780
CloseScreen	849
CloseWindow	823
CloseWorkBench	849
CMove	943
ConfigBoard	1008
ConfigChain	1009
CopySBitMap	954
CreateBehindLayer	878
CreateDir	803
CreateProc	814
CreateUpfrontLayer	878
CurrentDir	804
CurrentTime	869
CWait	943
DateStamp	815
Deallocate	758
Delay	815
DeleteFile	804
DeleteLayer	890
DeviceProc	816
Disable	752
DisownBlitter	938
DisplayAlert	845
DisplayBeep	850
DisposeFontContents	978
DisposeLayerInfo	890
DisposeRegion	954
DoCollision	967
DoIO	787
DoubleClick	869
Draw	916
DrawBorder	856
DrawCircle	917
DrawGList	967
DrawImage	856
DupLock	810

Enable	752
EndRefresh	861
EndRequest	846
EndUpdate	884
Enqueue	766
Examine	805
Execute	818
Exit	816
ExNext	806
FattenLayerInfo	879
FFP	986
FFP	994
FindConfigDev	1009
FindName	766
FindPort	778
FindTask	769
FindToolType	902
Flood	924
Forbid	753
FreeBoardMem	1010
FreeColorMap	929
FreeConfigDev	1011
FreeCopList	944
FreeCprList	944
FreeDiskObject	897
FreeEntry	762
FreeExpansionMem	1011
FreeFreeList	900
FreeGBuffers	968
FreeMem	760
FreeRaster	908
FreeRemember	860
FreeSignal	773
FreeSprite	969
FreeSysRequest	846
FreeTrap	774
FreeVPortCopLists	944
FreeWBOBJECT	893
GetColorMapA	929
GetCurrentBinding	1012
GetDefPrefs	870
GetDiskObject	898
GetGBuffers	969
GetIcon	896
GetMsg	777
GetPrefs	870
GetRGB4	930

GetSprite	970
GetWBOObject	894
IEEEEDPAbs	994
IEEEEDPACos	1000
IEEEEDPAdd	995
IEEEEDPAsin	1000
IEEEEDPAtan	1000
IEEEEDPCeil	995
IEEEEDPCmp	995
IEEEEDPCos	1001
IEEEEDPCosh	1001
IEEEEDPDiv	996
IEEEEDPExp	1001
IEEEEDPFieee	1001
IEEEEDPFix	996
IEEEEDPFloor	997
IEEEEDPFlt	997
IEEEEDPLog	1002
IEEEEDPLog10	1002
IEEEEDPMul	997
IEEEEDPNeg	998
IEEEEDPPow	1002
IEEEEDPSin	1003
IEEEEDPSincos	1003
IEEEEDPSinh	1003
IEEEEDPSqrt	1003
IEEEEDPSub	998
IEEEEDPTan	1004
IEEEEDPTanh	1004
IEEEEDPTieee	1004
IEEEEDPTst	999
Info	806
InitAnimate	970
InitArea	925
InitBitMap	909
InitGels	971
InitGMasks	971
InitLayers	880
InitMasks	972
InitRastPort	909
InitRequester	846
InitStruct	750
InitTmpRas	925
InitView	945
InitVPort	945
Input	810
Insert	763

InstallClipRegion	884
IntuiTextLength	857
Intuition	863
IoErr	811
IsInteractive	811
ItemAddress	840
LoadRGB4	931
LoadSeg	819
LoadView	946
Lock	812
LockLayer	885
LockLayerInfo	885
LockLayerRom	954
LockLayers	885
MakeDosNode	1012
MakeLibrary	781
MakeScreen	851
MakeVPort	946
MatchToolValue	903
ModifyIDCMP	824
ModifyProp	832
Move	910
Move()DrawEllipse	916
MoveLayer	886
MoveLayerInFrontOf	886
MoveScreen	851
MoveSprite	972
MoveWindow	824
MrgCop	947
NewFontContents	978
NewLayerInfo()	880
NewRegion	955
ObtainConfigBinding	1013
OffGadget	834
OffMenu	840
OnGadget	834
OnMenu	841
Open	800
OpenDevice	786
OpenDiskFont	979
OpenFont	961
OpenLibrary	783
OpenResource	792
OpenScreen	851
OpenWindow	825
OpenWorkBench	852
OrRectRegion	955

OrRegionRegion	956
Output	812
OwnBlitter	938
ParentDir	807
Permit	753
PolyDraw	917
PrintText	857
PutDiskObject	898
PutIcon	896
PutMsg	776
PutWBObject	894
QBlit	939
QBSBlit	940
Read	801
ReadExpansionByte	1014
ReadExpansionRom	1014
ReadPixel	918
RectFill	926
RefreshGadgets	835
ReleaseConfigBinding	1015
RemakeDisplay	862
RemBob	972
RemConfigDev	1015
RemDevice	786
RemFont	962
RemHead	765
RemIBob	973
RemIntServer	756
RemLibrary	782
Remove	765
RemoveGadget	836
RemPort	776
RemResource	791
RemTail	765
RemTask	768
RemVSprite	973
Rename	807
ReplyMsg	777
ReportMouse	871
Request	847
RethinkDisplay	862
ScreenToBack	852
ScreenToFront	853
ScrollLayer	887
ScrollRaster	919
ScrollVPort	947
Seek	802

SendIO	788
SetAfPt	928
SetAPen	910
SetBPen	911
SetCollision	974
SetComment	808
SetCurrentBinding	1016
SetDMRequest	847
SetDrMd	911
SetDrPt	912
SetExcept	771
SetFont	962
SetFunction	784
SetIntVector	755
SetMenuStrip	841
SetOPen	928
SetPointer	858
SetProtection	808
SetRast	912
SetRGB4	931
SetRGB4CM	932
SetSignal	770
SetSoftStyle	963
SetSR	753
SetTaskPri	769
SetWindowTitles	826
SetWrMsk	913
ShowTitle	853
Signal	772
SizeLayer	887
SizeWindow	826
SortGLList	974
SPAbs	982
SPAcos	987
SPAdd	982
SPAsin	987
SPAtan	988
SPCmp	983
SPCos	988
SPCosh	988
SPDiv	983
SPExp	989
SPFieee	989
SPFix	983
SPFlt	984
SPLLog	990
SPLLog10	990

SPMul	984
SPNeg	985
SPPow	990
SPSin	991
SPSincos	991
SPSinh	992
SPSqrt	992
SPSub	985
SPTan	992
SPTanh	993
SPTieee	993
SPTst	985
SumLibrary	784
SuperState	754
SwapBitsRastPortClipRect	888
SyncSBitMap	956
Text	919
TextLength	920
ThinLayerInfo	891
UCopListInit	947
UnLoadSeg	819
UnLock	813
UnlockLayer	891
UnlockLayerInfo	892
UnlockLayerRom	957
UnlockLayers	891
UpfrontLayer	888
UserState	754
Vacate	793
VBeamPos	948
ViewAddress	864
ViewPortAddress	865
Wait	771
WaitBlit	940
WaitBOVP	948
WaitForChar	816
WaitIO	789
WaitPort	778
WaitTOF	949
WBenchToBack	853
WBenchToFront	854
WhichLayer	889
WindowLimits	827
WindowToBack	827
WindowToFront	828
Write	802
WriteExpansionByte	1016

WritePixel	920
XorRectRegion	957
XorRegionRegion	958

Annexe L - Description des Guru-Meditation

Même sans programmer des alertes, je suppose que vous avez déjà vu un Guru-Meditation. Généralement, vous cliquez sur la souris pour rebooter et vous restez perplexe devant cette série de chiffres.

Et pourtant, cette même série de chiffres est très importante car elle permet de déterminer ce qui a provoqué l'effondrement du système.

Un Guru-Meditation a toujours le même format et est constitué de plusieurs parties que nous allons maintenant expliquer:

Guru Meditation : TTSSFFGGGG.AAAAAAAA

Les lettres ont la signification suivante:

TT	ALERT_TYPE
SS	Classe système
FF	Classe d'erreur
GGGG	Description exacte.
AAAAAAA	Adresse de l'erreur.

Il n'y a que deux possibilités pour TT. Vous lisez ces valeurs sous le label ALERT_TYPE. Cela peut être soit une DEAD_ALERT (80), soit une RECOVERY_ALERT (00).

Alert_Types

00000000 RECOVERY_ALERT
80000000 DEADEND_ALERT

Table 1: Alert-Types

Le label suivant (SUBSYSTEM_CODE) précise la cause exacte de l'erreur (SYSTEM_CLASS = classe système).

SUBSYSTEM_CODE

00000000 68000

LIBRARYS

01000000 exec.library
02000000 graphics.library
03000000 layers.library
04000000 intuition.library
05000000 math.library
06000000 clist.library
07000000 dos.library
08000000 ram.library
09000000 icon.library
0A000000 expansion.library

DEVICES

10000000 audio.device
11000000 console.device

```
12000000 gameport.device
13000000 keyboard.device
14000000 trackdisk.device
15000000 timer.device
```

RESOURCES

```
20000000 CIA
21000000 Disk
22000000 Misc
```

MISC

```
30000000 Bootstrap
31000000 Workbench
32000000 DiskCopy
```

Table 2: Subsystem-Codes

Soyez sûr que le message d'erreur se trouve au début de la SUBSYSTEM.LIST. Cela signifie que l'on a affaire à un Processor-Trap. Le reste du code a la signification suivante :

TRAP_CODES

```
00000002 Erreur de donnée ou d'adresse.
00000003 Erreur d'adressage (adresse impaire).
00000004 Instruction illégale.
00000005 Division par zéro.
00000006 Instruction CHR.
00000007 Instruction TRAPV.
00000008 Violation de privilège.
00000009 Mode Single-Step.
0000000A Line A Emulator (OpCode 1010).
0000000B Line F Emulator (OpCode 1111).
```

Table 3: Trap-Codes

Si une erreur est placée après la SubSystem-List, alors la classe d'erreur (ERROR_CLASS) nous donne la cause de l'erreur :

ERROR_CLASS

```
00010000 Pas assez de mémoire.
00020000 La librairie ne peut pas être construite.
00030000 La librairie ne peut pas être ouverte.
00040000 Le Device ne peut pas être ouvert.
00050000 Pas de réaction de la part du Hardware.
00060000 Erreur I/O.
00070000 I/O non présentes.
```

Table 4: Error-Classes

Les 4 derniers chiffres nous donnent plus d'information.

exec.library

81000000 Erreur de checksum.
81000002 Erreur de checksum à l'adresse de départ.
81000003 Checksum d'une librairie.
81000004 Pas assez de mémoire pour la librairie.
81000005 Erreur dans les entrées de la liste mémoire.
81000006 Manque de mémoire pour les interruptions.
81000007 Erreur de Sémaphore.
81000008 Erreur de mémoire.
81000009 Erreur de pointeur.
8100000A Erreur de pointeur avec exception.

graphics.library

82010001 Pas assez de mémoire pour une Copper-List.
82010002 Pas assez de mémoire pour une instruction Copper-List.
82010003 Copper-List pleine.
82010004 Erreur de division dans la Copper-List.
82010005 Pas assez de mémoire pour la tête de Copper-List.
82010006 Pas assez de mémoire pour un "long frame".
82010007 Pas assez de mémoire pour un "short frame".
82010008 Pas assez de mémoire pour une routine de remplissage.
82010009 Pas assez de mémoire pour une routine de texte.
8201000A Pas assez de mémoire pour un bitmap Blitter.
8201000B Zone mémoire incorrecte.
82010030 Erreur pendant la création d'un ViewPort.
82011234 GfxNoLCM (pas de mémoire intermédiaire).

layers.library

03000001 Pas assez de mémoire pour les Layers.

intuition.library

84000000 Type de gadget inconnu.
04000001 Erreur type avec un gadget.
84010002 Pas assez de mémoire pour construire un port.
04010003 Pas assez de mémoire pour construire un menu.
04010004 Pas assez de mémoire pour construire un sous-menu.
84010005 Pas assez de mémoire pour construire une liste de menu.
84000006 Position incorrecte pour une liste de menu.
84010007 Pas assez de mémoire pour OpenScreen().
84010008 Pas assez de mémoire pour construire un RastPort.
84000009 SCREEN_TYPE inconnu ou mauvais.
8401000A Pas assez de mémoire pour un gadget.
8401000B Pas assez de mémoire pour une fenêtre.
8400000C Mauvais status de registre pendant l'ouverture de
Intuition.library.
8400000D Message IDCMP incorrect.
8400000E Pas assez de mémoire pour la pile des messages.
8400000F Pas assez de mémoire pour le Console.device.

dos.library

07000001 Pas assez de mémoire au départ.
07000002 Tâche non déterminée.
07000003 Erreur QPKT rencontrée.
07000004 Paquet de données non attendu.

07000005 Pointeur libre inaccessible.
07000006 Données fausses sur un bloc du disque.
07000007 Bitmap détruit.
07000008 Clé déjà libérée.
07000009 Trop d'erreurs dans le Checksum.
0700000A Erreur disque.
0700000B Clé hors d'une zone allouée.
0700000C Recouvrement impossible.

ram.library

08000001 Entrées fausses dans la liste.

expansion.library

0A000001 Erreur dans le hardware d'expansion.

trackdisk.device

14000001 Erreur pendant une recherche.
14000002 Erreur pendant le timing.

timer.device

15000001 Erreur avec un accès du device.
15000002 Erreur avec une "time coordination network fluctuation".

disk.resource

21000001 Disque inséré inconnu.
21000002 Aucun drive n'est connecté.

bootstrap

30000001 Erreur pendant l'analyse des données du boot.

Index

!

6500/1	106
6526	49
68000	41, 42
7407	95
8087	283
8088	282, 283
8361	63, 67
8362	63, 71
8364	63, 75
8367	63
8370	78
8520	41, 48
A7	19
.info	30

A

A.D.P	114
A1010	96
AbleICR0	425
AbortIO0	398
Accès	
aux listes de tâches	338
direct aux données	480
DMA	125
exclusif	432
protégé	66
ACTION_FIND_OUTPUT	479
Added	336
AddHead()	296
AddICRVector()	425
AddIntServer()	421
AddLibrary()	392
AddMEmList()	381
AddPort()	366
Address Bus Translator	284 - 286
AddSemaphore()	439

AddTail()	297
AddTask()	346
ADKCON	242, 266, 272
ADKCONR	272
Adressage	200
multiplexé	65
par décrémentation	200
Adresses	
codage	277
décodeur	64
des registres	59
des zones sources de données	194
détermination	156
PC	284
AGNUS	41, 63, 67
Alarme	56
Algèbre booléenne	201
Aliasing distortion	245
AllocAbs()	382
Allocate()	380
AllocEntry()	375
AllocMem()	372
AllocSignal()	352
AllocTrap()	366
Amplitude	
d'une oscillation digitalisée	248
d'une vibration	231
ASCII	105
ASSEM	308
AttemptSemaphore()	437
AUD0LCH	237
AUD0LCL	237
AUD0LEN	238
AUD0VOL	238
AUD1LCH	237
AUD1LCL	237
AUD1LEN	238
AUD1VOL	238
AUD2LCH	237
AUD2LCL	237
AUD2LEN	238
AUD2VOL	238
AUD3LCH	237
AUD3LCL	237
AUD3LEN	238
AUD3VOL	238
Auto-configuration	273 - 274

AUX	493
AvailMem()	382
Axe	257

B

Bandes de fréquences	245
BAS	82
Bauds	57, 268
BeginIO()	398, 400, 403
BERR	45
Bibliothèques	
DOS	478
de liens	320
résidentes	518
Binddrivers	279
BIOS-ROM	284
bit Quick	402
BitMap	72
Bitmap block	539
Bitplane	128, 129
combinaison	146
recopie d'un graphique	197
Blitter	69, 194
algorithme	209
déroulement d'une opération	198
établir une fenêtre	195
hauteur de la fenêtre	195
tracé de droite	222
Bloc racine	531, 533
BLTCON0	205
BLTCON1	206
BLTSIZE	195
BOOLE	201
BPL1MOD	159
BPL2MOD	159
BPLCON0	163
BPLCON1	159
BPLCON2	165, 183
Bridgeboard	281
Broche	
CFGIN	274
CFGOUT	274
d'horloge	44

d'interruption	47
Bruit	234
parasite	248
Bss	514
Buffer	65
Bus	
d'adresse	44, 64
de contrôle asynchrone	44, 45
de données	44, 64
SHUGART	92
Zorro	101

C

C	21
Canaux audio	236
Cartes d'extension	103, 273
Cause()	423
CFGIN	277
CFGOUT	277
Champs de jeu	146
Changer	
de disquette	94
une forme d'onde	248
le nom d'un fichier	502
CHIP-RAM	65, 112
CIA	49, 59, 112
CIA-A	59, 61
CIA-B	60, 62
Clavier	104
Close()	398
CloseDevice()	550
CloseLibrary()	332
CLXCON	185
CLXDAT	184
Collision	186
Commandes	
d'état	694
système	45
Complex Interface Adapters	49
Compteur	55
de la souris	73
CON	492
Connecteur	

Audio/Video	82
Centronics	86
CINCH	82
d'extension	100
d'extension mémoire	42
d'extension système	42
de disquettes externe	91
MMU	101
modulateur TV	42
parallèle	88
vidéo RGB	42, 84
série RS-232	42, 88, 264
souris-joystick	97
Console-Device	500
Contrôleur	
de disquettes	268
DMA	65
Copie	
de données	194
par incrémentation d'adresse	199
par décrémentation d'adresse	199
de blocs de données	194, 206
Copper	69, 137
COPPER-LIST	138, 157
Couleur du fond	175
Courbe enveloppe	246
Court-circuits	83, 99
CRA	52, 55
Craquements	248
CRB	52, 55
CreateExtIO()	544
CreateProc()	484
Créer	
une librairie	386
un nouveau Message-Port	360
une structure	301
une structure de librairie	391
CSI (Control Sequence Introducer)	497
Cycles	
Blitter DMA	220
de bus	220
de rafraîchissement	134
Cylindre	530

D

Data Bus Request	69
Data Bus Translator	284 - 286
Data-block	538
DBR	69
DDFSTOP	155
DDFSTRT	155
DDRA	51
DDRB	51
Dead Lock()	436
Deallocate()	381
Db	231
Décalage	
de données	194
de la fenêtre d'écran	156
en cylindre	203
Décaler	
des blocs de données	194
les mots de données	203
Décibel	231
Décodeur d'adresse	64
DeleteExtIO()	550
DeletePort()	550
Delta X	216
Delta Y	216
DENISE	41, 63, 71
Désactiver un sprite	188
Déterminer	
l'octant	214
des adresses Bitmap	156
Device-task	399
Devices	397, 480, 491, 500
communiquer avec	552
initialisation	402
lecture	584
ouverture	555, 562
Device	
Audio	645
Clipboard	641
Console	612
Gameport	587
Input	594
Keyboard	583
Parallel	555

Printer	571
Serial	561
Timer	677
Trackdisk	687
DEVS	21
Digitaliser	236
une forme d'onde	235
des sons	88
DIN	83
Disable()	342, 349
Disk Operating System	477
DISKDOCTOR	532
Disquette	
division	530
DOS	531
état	694
étrangère	531
formatage	693
Kickstart	531
Workbench	532
DIWSTOP	154
DIWSTRT	154
DMA	68
audio	126
bitplane	125, 131
blitter	126, 131, 206
copper	126, 131, 142
disque	126
sprite	126, 176
CONTROLER	65
DMACON	132
DMAL	76
DoIO()	400, 402
Données	
échange	491
écrire	556
DOS	477
DOS.library	478
Drivers	273
DSKBYTR	270
DSKDAT	272
DSKDATTR	272
DSKLEN	269
DSKPTH	269
DSKPTL	269
DSKSYNC	271

DTACK	44
Dual-Playfield	151, 161

E

ECE	210
Echantillon	236
Editeur de liens	515
Effondrement du système	287
EHB	72
Emission	267
Emulation IBM	103
En-tête	11
des fonctions	14
Enable()	342, 349
Enqueue()	298
Entrées analogiques	261
Envoi d'un message	357
Equations booléennes	200
Event-counter	55
Exception	337
Exec	287
Expansion	279
Expunge()	398
Extension	
architecture	273
mémoire	66
Extra Half Bright	72, 149

F

FAILAT	488
Faisceau lumineux	257
Fast-Filing-System	539
FAST	495, 541
RAM	66, 112
Filing-System (FFS)	532, 539
FAT-AGNUS	64, 78
FBAS	82
Fenêtre	
Blitter	197

d'écran	177
ouvrir	482
FFS	532, 539
Fichier	
.info	30
supprimer	502
système	479
FIFO	359
File	
header-block	535
List-Block	536
Systems	479
FILETYPE	35
Filtre passe-bas	244, 250
FindName()	299
FindPort()	367
FindSemaphore()	440
FindTask()	347
Flip-Flop	93
Fonction	
_CreateIO	325
_CreatePort	321
_CreateStdIO	325
_DeleteIO / _DeleteStdIO	326
_DeletePort	323
de sémaphores-signaux	435
FONTS	21
Forbid()	145, 342, 349
Forme	
d'une vibration	233
d'ondes typiques	232
FreeEntry()	376
FreeMem()	373
FreeSignal()	352
FreeTrap()	367
Fréquence	230, 234, 235
d'un demi-ton	249
d'un son	230, 239
parasites	245
réelle des périodes d'échantillonnage	249

G

Gameport	260, 261
GARY	80
Générateur	
d'adresse mémoire	68
d'adresse registre	68
d'horloge	79
Générer	
de nouvelles tâches	344
un processus	484
GENLOCK	68, 70, 82
Gestion	
de la Chip-Ram	80
des interruptions	76
des listes de tâches	338
dynamique de la mémoire	370
GetDiskObject()	509
GetMsg()	358, 367
GURU MEDITATION	287

H

HALT	45
HAM	72
Handlers	477, 479
-task	487
Hardware	41
Harmoniques	245
Hashtable	533, 534
Haute	
résolution	129
fréquences	250
Hauteur	
d'un son	231
du playfield	159
HERTZ	230
Hold and Modify	72, 149
horloge temps réel	55
Hunk	
_break	519
_bss	514, 516
_code	516

_data	516
_debug	518
_end	518
_ext	517
_header	518
_name	516
_overlay	519
_reloc8	517
_reloc16	517
_reloc32	516
_symbol	518
_unit	515
 I	
I/O	491
IBM-Emulator	103
ICE	209
Includes	
utilisation des fichiers	313
Icon-library	509
ICR	57
Impulsions compteur	259
Initialiser	
une structure Node	290
des listes	294
un sprite	180
une opération du Blitter	194
initSemaphore()	435
Intégration	
dans un reset	461
d'un device dans le système	397
INTENA	135
INTENAR	135
Intensité	231, 235, 238
Interface	
parallèle	504
sérielle	503
Interlace	127
Interpréteur	
de commandes	477
Interruption	135, 416
Audio	240
CIA	424

COPPER	142
libérer	51
non masquable	135
processeur de niveau 4	241
traiter	136
INTREQ	135
INTREQR	135
Intuition	500
IOReplyPort	545

J

joysticks	42, 260
---------------------	---------

K

Keymapping	625
Kickstart	85, 122, 531
KickSumData()	469

L

L	21
Largeur	
(Width) de la fenêtre	195
d'un playfield	159
LDS	44
Libérer une interruption	51
Librairie	
personnelle	392
RAM	449
modifier	385
LIBS	22
Light-Pen	97, 134
Ligne	
dessiner	206
du RASTER	239
frontière	215
Tracer	194, 211

Limite	
du filtre	245
fréquentielle	243
Linker	515
Liste	
des ToolTypes	33
de segments	489
chaînage	288
Load file	515
LoadSeg()	484
LONG FRAME	127, 134
Longueur de la pile	486
LSB	148, 265
Lutins	72, 146

M

Macro	300, 308
ADDHEAD	303
ADDTAIL	303
BITDEF	312
IFEMPTY	302
IFNOTEMPTY	302
NEWLIST	301
PRED	302
REMHEAD	304
REMOVE	303
REMTAIL	304
SUCC	301
TSTLIST	301
TSTNODE	302
en assembleur	300
MakeLibrary()	391
manette	42
Masques	204, 215
MEMF	
_CHIP	371
_CLEAR	372
_FAST	371
_LARGEST	372
_PUBLIC	372
Mémoire	
dynamique	68
extension	66

morte	64
organisation	111
rafrâchie	66
static	360
vidéo	66
vive	64
Menus	22
Messages	
d'erreur du DOS	489
d'erreur du device d'imprimante	582
d'erreur du device parallèle	560
réception	358
réponse	359
Minterms	201
Minuteries	52
mn	
_Lenght	357
_Node	357
_ReplyPort	360
Mode	
ascending	200, 206
descending	200, 206
Hires	155
Interlace	161
Lores	155
Superviseur	339
Waiting	340
Modem	89
Modifier une librairie	385
Modulateur TV	83
Modulation	
d'intensité	246
de fréquence	241
Modules résidents	464
Modulo	161
Moniteurs RGB digitaux	85
MOUNT	477
MountList	477, 541
MOVE	137, 140
mp_Node	355
MSB	148, 265
Multi-tâches	336
Multi-utilisateurs	494
Multiplex	283
Musique	
électronique	230
polyphonique	250

N

NEWCON	493
Nibbles	274
NIL	492
Noeud	288
Nombre	
d'échantillons par seconde	236
de vibrations par seconde	230
NODE	288
Ajouter une structure	303
NTSC	82
Numéro	
de tête	530
de version	13

O

Object file	515
Obtain Semaphore()	436
ObtainSemaphoreList()	438
Octant	212
détermination	214
Octave	231, 249
Off line	88
Offsets	482
OldOpenLibrary()	335, 481
On line	88
Onde	
carrée	233, 245
sinusoïdale	233, 234, 245
sonores	230
Open()	398
OPEN-COLLECTOR	95
OpenDevice()	543
OpenLibrary()	331
OU logique	201
Outmode	54
Overlay	519

P

Paddles	261
PAL	65, 83
Palette couleur	147, 175
PAR	492
Paramètres	504
d'exploitation	271
PAULA	41, 63, 75
Pente	216
Période	
calcul	651
d'échantillon	249
Permit()	342, 349
Phase	
affaiblissement	246
attaque	246
maintien	246
relâchement	246
Phonèmes	667
Pile	
de tâche	340
longueur	486
PIPE	494
Pitch	670
Pixel	129
Playfields	146
interlace	162
Poids	
faible	65
fort	65
Pointes de tension	99
Pointeur	481
BPTR	486
Port	
de message	485
parallèle (centronics)	41
Positions de phase	234
PRA	51
PRB	51
Pression acoustique	231
Printer.device	477
Priorité	207
sprites/playfield(s)	182
Processeur	63 - 64

Processus	484
asynchrone	422
générer	484
état de fonctionnement	46
Procure()	441
PRODUCT	279
Program unit	515
Programmation du DMA disquette	269
Programmes résistants au reset	460
PROMPT	488
PRT	492
PutMsg()	357, 368

Q

Qualité d'un son	248
----------------------------	-----

R

RAD	494
Rafraîchissement du signal	68
RAM	41, 64, 112, 492
Dual-Port	284
Raster	127, 129, 140
Rate	670
RAW	493
codes clés	105
RAW-Input-Events	499
Ready	336
Registre	
de contrôle d'interruption	57
de données du contrôleur de disquette	272
de données séries	57
de contrôle Blitter	205
des circuits spécialisés	113
du COPPER	138
UART	265
ReleaseSemaphore()	437
ReleaseSemaphoreList()	438
Relocation table	514
RemHead()	297

RemICRVector()	426
RemIntServer()	422
RemLibrary()	335
Removed	336
Remplir des surfaces	194, 206, 208, 228
RemPort()	368
RemSemaphore()	440
RemTail()	298
RemTask()	347
Répertoire	
racine	534
système	20
ReplyMsg()	369
Requester	23
RESET	45, 110
avec le device de clavier	585
programmes résistants	460
Résolution	
basse	129
haute	129
RGB	72, 84
ROM	64, 122
Routine Switch	339
RS232	89, 264
RUN	489
Running	336

S

S	21
Sample	236
Sampling-rate	236, 651
Sauts d'intensité	248
Scintillements	127
Scrolling	158
à droite	161
à gauche	161
horizontal	159
libre	146
vertical	159
Secteur	530
Segment	
des données	514
du code	514

Semaphore-Port	432
Semaphore-Signal	432
Sémaphores	431
SendIO()	400
Séquences de contrôle	497
SER	492
SERDAT	265
SERDATR	265
Serial.device	477
Série de Fourier	234
SERPER	265
SetExcept()	369
SetFunction()	385
SetICR()	427
SetIntVector()	420
SetSignals()	353
SetTaskPri()	348
SHORT FRAME	127, 134
SHUGART	92
Signal	354
adresses	65
adresses de la mémoire dynamique	69
attribution du bus	47
busy	88
direction	70
données	268
HANDSHAKE	108
horloge	69
Handshake	264
interruption	76
paper out	88
READ/WRITE	64
RGB numériques	84
synchronisation	68
vidéo	70
SKIP	138, 141
Slot DMA	239
Smooth-scrolling	146
sortie	
audio stéréo	42
composite Sync	84
de sons numériques	236
sprite de DENISE	187
vidéo composite	42
Sources d'interruptions	58
souris	42, 257
SPEAK	494

Sprites	72, 146, 174
combiner deux sprites	186
désactiver	188
entrant en collision	184
mots de contrôle	187
mouvement	182
reproduction sans DMA	188
Start-block	537
STARTUP-COPPERLIST	145
Structure	287, 484
CommandLineInterface	487
d'un message	357
d'un playfield	153
d'une COPPER-LIST	141
d'une liste	293
d'un fichier programme	513
de base	288
de fichier	534
de l'auto-configuration	274
de port de message	485
de processus	484, 485
de tâche	337, 484, 485
des sprites	174
Device	397
DiskObject	512
ExecBase	451
FileLock	488
interne des librairies	383
IORequest	399
List	291
Memory-List	374
Node	288
Stylo lumineux	98
Supprimer	
des sous-répertoires	502
la structure Node	304
un fichier	502
une structure Node	303
Synchronisation	108
externe	85
SYS	22, 492
Système	
de communication entre tâches	355
de fichiers	479
des messages	355
Fast-Filing	532

T

Table de redistribution	514
Tâche	336, 484
commutation entre les tâches	339, 350
d'unité	400
générer de nouvelles tâches	344
Tampon	65
bus d'adresse	79
Task	
-Exceptions	362
-Scheduling	339
-Switching	145, 339
States	336
autorisation et interdiction	341
Taux d'échantillonnage	236, 239
Temps mort vertical	85, 182
Tête de liste	338
Texte	
imprimer	572
Timbre	232, 235
Timer	
CIA	677
de retour vertical	677
TimerBase	685
Touches	
traitement	104
TOOL TYPES	509, 512
Transfert	
de données	51, 76
de données série RS232	264
Transformateur digital/ analogique	235, 237
Transmission	
CLI	25
des données	24
des paramètres	504
Transparent	175
Traps	363
du processeur	363
Trémolo	231

U

UART	76
UDS	44
Unit-Task	400
Universal Asynchronous Receive Transmit	76
User directory block	537

V

Vacate()	442
Valeur modulo	197 - 198
Vecteur	
ColCapture	461
CoolCapture	462
d'interruption	135
d'interruption	136
RESET	122
WarmCapture	464
Vibration	
carrée	234
de l'air	230
électrique	230
en dents de scie	234
sonore	230
Vibrato	231

W

WAIT	137, 140, 354
Wait()	358
Waiting	336
WaitPort()	
358, 370	
WOM	122

Z

Zone	
audible	231
mémoire bidimensionnelle	208
mémoire linéaire	195

Dans la même collection...

ML197	Bien débuter avec l'AMIGA	149.00
ML553	Bien débuter en C sur AMIGA	149.00
ML767	Bien débuter Langage Machine AMIGA.....	129.00
ML716	L'Histoire de LARRY - Leisure Suit I,II et III	79.00
ML733	La Bible de l'AMIGA (2ème édition)	340.00
ML741	La Saga des KING'S QUEST.....	78.00
ML504	Le Grand livre de l'AMIGABASIC	249.00
ML604	Le Grand livre de l'AMIGABASIC..... (avec disq.)	349.00
ML713	Le livre de DELUXE PAINT III	145.00
ML780	Le livre de la vidéo sur Amiga	195.00
ML562	Le livre du GFA BASIC AMIGA	149.00
ML662	Le livre du GFA BASIC AMIGA	(avec disq.) 249.00
ML805	Le livre des Imprimantes sur AMIGA	299.00
ML632	Le livre de la Musique..... (avec disq.)	199.00
ML198	Le livre du Langage Machine	199.00
ML634	Le livre du Lecteur de disquette	(avec disq.) 299.00
ML573	Le livre de SUPERBASE (versions PRO).	169.00
ML747	Les meilleurs Domaine public AMIGA	149.00
ML724	Les meilleurs jeux sur AMIGA	125.00
ML771	SPACE QUEST STORY.....	78.00
ML570	Trucs et Astuces AMIGA - version II -	129.00
ML670	Trucs et Astuces AMIGA - version II -	(avec disq.) 229.00

Achevé d'imprimer
sur les presses de l'imprimerie IBP
à Rungis (Val-de-Marne 94) (1) 46.86.73.54
Dépôt légal - Décembre 1990
N° d'impression: 5439

AMIGA

BLEEK/DITTRICH/GELFAND/JENNICH/SCHEMMEL/SCHULZ

Voici le livre de référence pour la programmation des Amiga 500, 1000 et 2000. Toutes les informations techniques sur le hardware sont détaillées : les processeurs, les librairies, les circuits spécialisés, les Devices, le DOS... Avec LA BIBLE, pénétrez au coeur de votre machine et apprenez à concevoir des programmes en Assembleur, créer des applications résidentes ou piloter des périphériques. Découvrez l'ensemble des librairies de fonctions avec toutes les explications sur leur mise en oeuvre, les adresses ainsi qu'une multitude de conseils pour programmer en multitâche, détourner les interruptions ou gérer les entrées/sorties. De nombreux exemples en langages C et Assembleur vous aideront à mettre en pratique vos connaissances. Enfin, des informations inédites sur le format IFF ou les nouvelles caractéristiques de l'Amiga 3000 vous sont proposées.

Principaux sujets traités:

- Le hardware des Amiga 500, 1000 et 2000 (Workbench 1.3): description et programmation du matériel - processeur 68000, coprocesseurs, port vidéo, interfaces série/centronics, Blitter, Copper, clavier, interruptions...
- La bibliothèque EXEC: listes, gestion de la mémoire, des interruptions, accès aux librairies, le multitâche: gestion et communication entre les tâches...
- Les Devices (description, utilisation, structure): entrées/sorties, écran, souris, timer, lecteur de disquettes...
- Le format IFF : construction et utilisation de formats d'échange standard - formats ILBM, ANIM, FTXT, SMUS, 8SVX - compression de données...
- Les librairies: description des fonctions et de leurs caractéristiques : EXEC, DOS, INTUITION, ICON, DISKFONT, Mathématiques...
- L'Amiga 3000: caractéristiques techniques, nouvelle interface utilisateur, les menus du Workbench 2.0, système d'exploitation...
- Annexes: tableaux des registres spécialisés, liste alphabétique des fonctions...

Réf. ML 733. Prix : 340F.

ISBN: 2-86899-345-1/ISBN : 0980-1928



9 782868 993458

EDITIONS MICRO APPLICATION
58 RUE DU FAUBOURG POISSONNIERE
75010 PARIS TEL (1) 47 70 32 44