

le **IIIGS**
“épluché”

Tome 1

par :

D. BÄR
D. DELAY
Y. DURANT
J.L. SCHMITT
Eric WEYLAND

édité par *Toolbox*

Ce livre est tout spécialement dédié à nos amis:

Monsieur Jean-François Carette,
Monsieur Olivier Goguel,
Monsieur Paul Lafonta,
Monsieur Gérard Perret,
Monsieur Pierre Raynaud-Richard,

ainsi qu'à Jean-Pierre Lagrange, qui sait reconnaître les informaticiens particulièrement doués, dans son livre :

"Systèmes d'Exploitation et Systèmes de Protection sur Apple II"

Par ordre alphabétique:

Mr. Daniel Bär,

Étudiant

Il a notamment participé à la programmation de Nucleus, et de Photonix. Il développe actuellement un jeu de course de voitures, sur l'Apple IIGS. Il est l'auteur de la partie son.

Mr. Dominique Delay

Étudiant en médecine

Il a 8 ans d'expérience en programmation sur Apple. Il est l'auteur de la partie graphisme et animation.

Mr. Y. Durant

Titulaire d'un DEA de système informatique. Ingénieur informatique.

Il est l'auteur de la partie ram et drive.

Mr. J-L. Schmitt

Diplômé de l'Institut Commercial Supérieur

D.E.C.F. - Diplôme d'étude de comptabilité et de finance.

Actuellement étudiant en 3ème cycle de génie logiciel.

Il est l'auteur de la partie sur le microprocesseur.

Mr. Éric Weyland

Maitrise en Econométrie

Possède un Apple II depuis 1980

Il est l'auteur de la partie sur le GS/OS.

Mr. Huibert Aalbers

Économiste, MBA

Développeur de SoundSmith et de Laserforce pour l'Apple IIGS

Il est l'auteur de la deuxième édition de l'Apple IIGS épluché, publié en 2016

Bibliographie:

- Apple II GS Firmware reference
- Apple II GS Hardware reference
- Exploring Apple GS/OS & PRODOS 8
- Programming the 65816

Apple, le logo Apple, Appleshare,
AppleTalk, Imagewriter,
Laserwriter, Macintosh et Mac Terminal,
Finder, Multifinder, Hypercard,
Prodos, DOS 3.3, GS/OS, Apple IIGS,
Sane, Apple Desktop bus,
Appleworks, APW

sont des marques déposées d'Apple Computer, inc.

High Sierra,
Amiga,
Atari,

sont des marques déposées

ISBN 2-9504508-0-6

Mars 1990, Edité par Toolbox Dépôt légal, 3/90

Imprimé en France

Conception, mise en page, maquette, impression

Sauveur Cacoub, **Forgest**

La loi du 11 mars 1957 interdit les copies ou reproduction destinées à une utilisation collective.
Toute représentation ou reproduction intégrale ou partielle faite par quelque procédé que ce soit,
sans le consentement de l'auteur ou de ses ayants cause, est illicite et constitue une contrefaçon
sanctionnée par les articles 425 et suivants du code pénal.

L'Apple II, encore et toujours

Éplucher un ordinateur: quelle idée bizarre. Aujourd'hui, le monde de l'ordinateur individuel ne se répartit-il pas en deux catégories bien distinctes, les producteurs d'une part (de matériels et de logiciels), les utilisateurs, ou plutôt les consommateurs, de l'autre? Ne sommes-nous pas dans l'ère industrielle?

L'informatique personnelle n'a pourtant pas - pas du tout - commencé comme cela. Au début des années 80, tout a commencé par un ingénieur inventif, bricoleur, plaisantin un peu anarchiste, qui a fabriqué dans son garage une machine qui s'est vendue ensuite à des millions d'exemplaires: ce bricoleur s'appelait Steve Wozniak, et la machine s'est très vite appelée l'Apple II.

Une cohorte d'étudiants, de professionnels de l'informatique sur gros systèmes, d'intellectuels divers, s'est ensuite emparée de la machine. On a appelé ces gens des hackers. Ils étaient mus par la curiosité (je veux savoir ce que cette machine a dans le ventre), et par la volonté de maîtriser les possibilités apportées, aux individus eux-mêmes, par leur ordinateur (je sais que cette machine peut le faire, et je n'arrêterai que quand j'aurai réussi à le lui faire faire). Ils étaient encouragés par Wozniak et par Apple: Wozniak n'avait-il pas publié, et mis dans le domaine public, le code originel du moniteur (la partie de programme installée dans la machine, et qui la dirige au plus près) de l'Apple?

Beaucoup d'eau a coulé sous les ponts depuis cette époque : le progrès technique, d'une part (il va très vite en informatique), les exigences de la concurrence et du succès économique d'autre part, ont professionnalisé l'informatique individuelle. Apple est devenu une multinationale cotée à Wall Street, l'informatique individuelle a gagné l'entreprise, IBM s'est mis de la partie.

Apple a fait énormément d'efforts, a dépensé beaucoup d'argent gagné avec l'aîné (l'Apple II) pour développer le cadet (le Macintosh) et le faire entrer dans l'entreprise. Beaucoup de hackers sont devenus des développeurs appointés, et généralement pas sur l'Apple II.

L'Apple II lui-même a changé: il s'appelle désormais Apple IIGS.

Il ressemble beaucoup à son cadet, le Macintosh: par exemple, il est muni d'un système d'exploitation (GS/OS), et d'une boîte à outils logicielle (Toolbox) qui font qu'il se programme largement comme un Macintosh.

Il n'y a apparemment plus de place pour les Wozniak et les hackers. N'entendons-nous pas périodiquement de bonnes âmes annoncer la mort de l'Apple II?

Oui, mais voilà : la curiosité, le goût de la liberté individuelle, sont des choses qui ne se périment pas. Tout individu qui touche un clavier d'Apple II est exposé à ce virus (l'auteur de ces lignes, comme tous ceux qui ont contribué à ce livre, a été contaminé de cette façon). D'autant qu'il existe

une communauté d'utilisateurs, largement informelle mais très vivace, qui se charge de partager le savoir entre tous ceux qui le désirent.

D'autant aussi que si l'Apple IIGS est un ordinateur d'aujourd'hui (16 bits, graphisme, son, etc...), c'est toujours un Apple II. On peut donc toujours, si on le veut, si on en prend le temps, le maîtriser soi-même, en faire ce que nous voulons, nous. Il a par exemple un super- moniteur, qui nous permet de savoir à tout instant ce qui se passe dans notre machine. Il a, dans sa ROM, un accessoire "Visit Monitor", qu'on ne peut même pas enlever en rebootant. Wozniak n'est pas mort, et l'Apple II est bien vivant.

Ceux qui ont écrit cet ouvrage ne sont pas des développeurs professionnels sur IIGS. Ce sont des utilisateurs, mais d'un genre spécial, ce sont des gens qui veulent maîtriser leur machine, et qui y parviennent.

Car l'évolution de l'informatique individuelle pose une question importante, une question de civilisation: si je ne suis pas le maître de ma machine, que je me contente de l'utiliser, alors quelqu'un d'autre en est le maître. Et ce quelqu'un, du coup, est aussi maître de moi.

Si en revanche, je comprends exactement ce qui se passe dans mon ordinateur, je peux lui faire faire ce que je veux, et l'utiliser comme un outil de ma liberté. Au lieu de dépendre des bureaucraties qui ont produit la machine et ses logiciels, et de faire exactement ce qu'elles avaient prévu que nous fassions, nous montrons que l'intelligence des individus est toujours supérieure à celle d'une machine, et nous pouvons en particulier, ce qui émerveillait toujours Wozniak, lui faire faire des choses auxquelles les concepteurs n'auraient jamais pensé, et qu'ils croyaient impossibles.

Pour prendre un exemple un peu technique (que vous ne comprendrez peut-être qu'après avoir lu ce livre): j'ai découvert en lisant la partie sur le graphisme un truc auquel je n'aurais pas pensé.

Une des limites du graphisme sur Apple IIGS, c'est qu'on ne dispose, en Super Haute Résolution, que d'une seule page graphique, la différence du graphisme HGR Apple II. Ce qui est gênant pour les animations. Par ailleurs, pour être compatible avec l'Apple II, l'Apple IIGS dispose d'un système d'écho-mémoire appelé shadowing. Cela n'a rien à voir avec les pages graphiques. Seulement, s'est dit l'auteur, supposons que je suspende temporairement le shadowing, n'aurais-je pas l'équivalent d'une pseudo-page 2?

Toujours cette mentalité Apple II:

Cela n'est pas prévu dans la machine, encore moins dans sa boîte à outils. Mais, moi, je veux le lui faire faire: eh bien, j'y arriverai! Et quand j'y suis arrivé, je fais savoir à tous, y compris à Apple qui n'y avait pas pensé, comment on peut le faire. Sur d'autres machines, celui qui aurait osé violer les interdits, et aurait trouvé quelque chose de ce genre, l'aurait gardé pour lui ou pour le vendre, cher. Sur l'Apple II, c'est déjà comme cela qu'on avait inventé la Double Haute Résolution.

C'est cela l'esprit Apple II: les raisons qui font que nous avons choisi un Apple IIGS comme ordinateur personnel ne sont pas seulement techniques (bien que le IIGS soit une excellente machine. Elles ne sont pas non plus économiques (ce n'est pas le moins cher, ce n'est pas la machine du patron, ce n'est pas non plus le meilleur des gagne-pains). C'est simplement la machine la plus polyvalente, celle dont nous pouvons faire ce que nous voulons: c'est un instrument de notre liberté. S'il nous arrive de critiquer Apple dans ce livre, que nul ne s'y trompe: c'est parce que nous aimons nos IIGS.

On le voit, cet ouvrage est destiné à tous ceux, quel que soit leur niveau de départ, du débutant à l'expert, qui veulent comprendre et maîtriser leur ordinateur. Les débutants pourront éplucher, couche par couche, leur machine. Les experts apprendront forcément certaines informations qu'ils ne connaissent pas encore. Sur l'Apple IIGS, mais pas seulement: en mettant du Macintosh dans le GS, Apple a introduit le loup (les bidouilleurs sur Apple II) dans la bergerie du Mac. L'Apple IIGS épluché, c'est peut-être bien aussi le commencement du Macintosh épluché.

—Oui, mais moi, me direz-vous, je veux seulement apprendre à m'en servir. Je ne veux pas devenir un expert, et l'état du bit 7 de \$C035 m'indiffère: je veux seulement pouvoir faire mon courrier sans problème.

—Je vous répondrai que c'est un désir parfaitement légitime : moi aussi, je me sers du téléphone sans chercher à comprendre ce qu'il y a dedans. Je veux juste que ça marche. On ne peut pas être un bricoleur dans tous les domaines, il faut aussi pouvoir se servir des choses telles qu'elles sont, bêtement.

Seulement, il faut être franc: ce qui va suivre ici, vous ne le lirez pas ailleurs, parce que c'est ce que tout le monde cache. Ce désir, parfaitement légitime, ne peut pas aujourd'hui être vraiment satisfait par aucun ordinateur personnel, et par aucun logiciel. Tous ceux qui vous parlent d'ordinateur facile, où il n'y a rien à apprendre, qui vous vantent l'ordinateur comme la solution sans effort à tous vos problèmes -eh bien ceux-là vous mentent, tout simplement. C'est vrai, un ordinateur Apple est plus facile d'accès que ces machines sur lesquelles galèrent ceux qui n'ont pas eu le choix: mais il va vous falloir, de toute façon, beaucoup d'efforts pour pouvoir vous en servir vraiment.

La première expérience de l'utilisateur naïf (celui qui croit les publicités) d'un ordinateur personnel, c'est celle du bug, du plantage, du Fatal System Error, de la bombe sur l'écran, etc. Il ne faut accuser personne de cette fatalité: l'industrie de l'informatique personnelle est si jeune, et le progrès technique y est si rapide, que c'est largement inévitable. Il faut au moins cinq ans d'utilisation pour déboguer entièrement une machine, un système d'exploitation, ou un logiciel. Mais alors, ils sont périmés.

Je ne nie pas tout l'apport qu'a représenté, sur le Macintosh et sur l'Apple IIGS, le travail d'Apple pour promouvoir son interface utilisateur graphique-souris. Effectivement, Apple a appris l'homme

à la machine. Mais la machine est bête, suprêmement bête - et les hommes qui ont conçu cette machine ne sont que des hommes.

Il y a même, dans nos ordinateurs, des pépins introduits exprès pour bloquer l'utilisateur et l'empêcher de faire certaines choses: il y a par exemple, dans la Rom de Apple IIe, une destruction systématique de deux octets par page mémoire en cas de Pomme Contrôle Reset. Il y a dans l'Applesoft en Rom, depuis l'Apple II+, un truc pour empêcher l'utilisateur de lister un programme Basic. Plus un truc pour vérifier qu'on n'a pas enlevé ces trucs!

Je n'accuse pas particulièrement Apple : la même chose est vraie de tous les constructeurs et de tous les éditeurs de logiciels. Simplement, cela implique que si vous voulez vraiment vous servir (simplement vous servir) de votre ordinateur, alors il faut pouvoir le comprendre. Il n'est pas possible de s'en servir bêtement.

Un mot encore sur la devise No Tools que vous trouverez ça et là dans cet ouvrage: elle n'est pas à interpréter de façon négative, comme le refus pur et simple des outils de l'Apple IIGS. Nul d'entre nous ne songe à se priver des excellentes applications orthodoxes qui existent sur le GS, ni de GS/OS, ni des accessoires (NDA et CDA). D'ailleurs la présente préface est écrite avec le traitement de textes d'une excellente application GS orthodoxe, Appleworks GS de Claris. Et nous savons tous qu'il y a au moins un outil à respecter si l'on veut rester compatible, à savoir le Memory Manager. La programmation avec la Boîte à Outils est un des modes de programmation de l'Apple IIGS: elle fait de lui un petit Mac II, et un excellent outil de travail.

Simplement, là où nous divergeons avec certaines proclamations d'Apple, c'est que nous pensons que ce mode de programmation n'est pas le seul possible. D'une part, les utilisateurs d'Apple II sont capables de s'habituer à plus d'un interface utilisateur, car l'interface Macintosh ne peut pas tout. Essayez donc de gérer les sous-sous-catalogues d'un disque dur avec le Finder, vous m'en direz des nouvelles.

D'autre part, l'Apple IIGS est aussi un Apple II, et un Apple II se programme sur le métal: programmer, c'est maîtriser la machine.

La programmation avec la Boîte à Outils, si elle facilite en un sens le travail du programmeur (et rend ses applications plus facilement portables sur d'autres machines), intercale une couche logicielle aveugle entre le programmeur et l'ordinateur. Du coup, au moins trois strates logicielles séparent l'utilisateur de sa machine: la Boîte à outils, le Système d'exploitation, et l'application. Et si jamais, comme c'est souvent le cas, cette application est programmée dans un langage dit évolué, il faut rajouter une quatrième couche (le compilateur et ses bibliothèques). Allez donc ensuite savoir d'où vient le bug!

Il existe de très bons ouvrages, en français et surtout en anglais (dont d'excellents publiés par Apple lui-même) qui permettent de remonter de l'utilisateur vers la boîte à outils. Il en est bien peu,

et aucun en français, qui permettent de descendre directement de la machine vers l'utilisateur. Cet ouvrage remplit donc un vide, dont il aurait été dommage qu'il reste béant trop longtemps.

Comment se gère exactement le Smartport? Comment faire des animations directement sur l'écran graphique? Comment utiliser au mieux le DOC pour le son? Si vous lisez les documentations et notes techniques Apple, vous aurez quelques fragments de réponses, mais vous aurez aussi souvent l'impression que vous n'avez pas besoin de savoir tout ça.

Certains, pourtant, en savent un peu plus: ils se taisent et exploitent le filon. D'autres ont cherché, et, comme c'est un Apple II, ils ont trouvé beaucoup de choses. Lecteurs, vous avez une chance assez rare, car il y a un point que j'ai oublié dans le portrait des hackers que vous avez lu ci-dessus: si un hacker aime partager ce qu'il sait, il n'aime généralement guère écrire. Bravo à Toolbox d'avoir su convaincre les auteurs.

Ce livre ne dit pas tout, loin de là: il a choisi l'approfondissement plutôt que le survol. Chacun a creusé son sillon: il a labouré profond, et vous trouverez dans cet ouvrage des informations que vous ne trouverez nulle part ailleurs.

Mais il reste encore des zones en friche: le port ADB (clavier, souris, etc..) et son microprocesseur spécifique, le port SCSI de la carte SCSI Apple, les ports série. Appletalk, les ROMs 03. etc...: il y a beaucoup de choses à comprendre et à maîtriser dans le GS, beaucoup d'autres volumes du IIGS épluché à écrire.

Espérons que celui-ci saura donner à certains de ses lecteurs le goût d'éplucher aussi, pour leur part, leur GS, et de maîtriser eux-mêmes leur machine. Apple II for ever, la fameuse devise de Wozniak, n'a pas d'autre sens que ce souhait que l'ordinateur reste à jamais ce qu'il doit être: non pas l'auxiliaire de Big Brother, mais un instrument - oh combien puissant - de la liberté des individus.

J.Y. Bourdin

Professeur agrégé de philosophie.

Ecrit dans la revue «Pom's» où il tient une chronique régulière sur l'Apple II appelée «Apple II for ever»

Le 5 Février 1990.

Le moins que l'on puisse dire, c'est que la littérature française sur l'Apple II GS, n'est pour l'instant guère étendue. Entre les clefs pour Apple II GS de Madame Nicole Bréaud-Pouliquen (prenez la deuxième édition), et le livre sur la Toolbox de Monsieur Curcio, il n'y avait rien. C'est un peu ce vide que nous tentons, à travers ce livre, de combler.

Nous avons volontairement délimité notre livre. Nous ne traitons pas de la Toolbox, pour plusieurs raisons : la première étant que les tools ne sont pas encore arrivés à «maturité» et qu'ils sont sujet à modification de la part d'Apple. Le nouveau GS risque de les voir modifier.

D'autre part nous voulons contrôler notre GS à l'octet prêt, et nous ne voulons pas par choix, confier certaines tâches à des outils, qui prennent énormément de temps machine, énormément de place sur disque, - quoique le nouveau GS a les tools en rom - et qui obligent à respecter des normes élaborées par Apple, et par là même à apprendre ces normes. Cependant nous comprenons fort bien que pour certaines applications professionnelles, les tools peuvent s'avérer fort utiles.

Ensuite le livre de Monsieur Curcio, traite fort bien du sujet, même si certaines parties seraient à réactualiser. Par conséquent pour la partie tools, voyez son livre.

Enfin, parce que même si le célèbre cri «NO TOOLS» n'est pas de nous, nous y adhérons.

Ce livre s'adresse à toute les personnes possédant un Apple II GS, mais il est certain que dans ce livre nous ne vous dirons pas comment installer une carte dans un slot ou comment programmer en Basic, pour cela il existe les manuels Apple qui vous sont vendus avec votre GS.

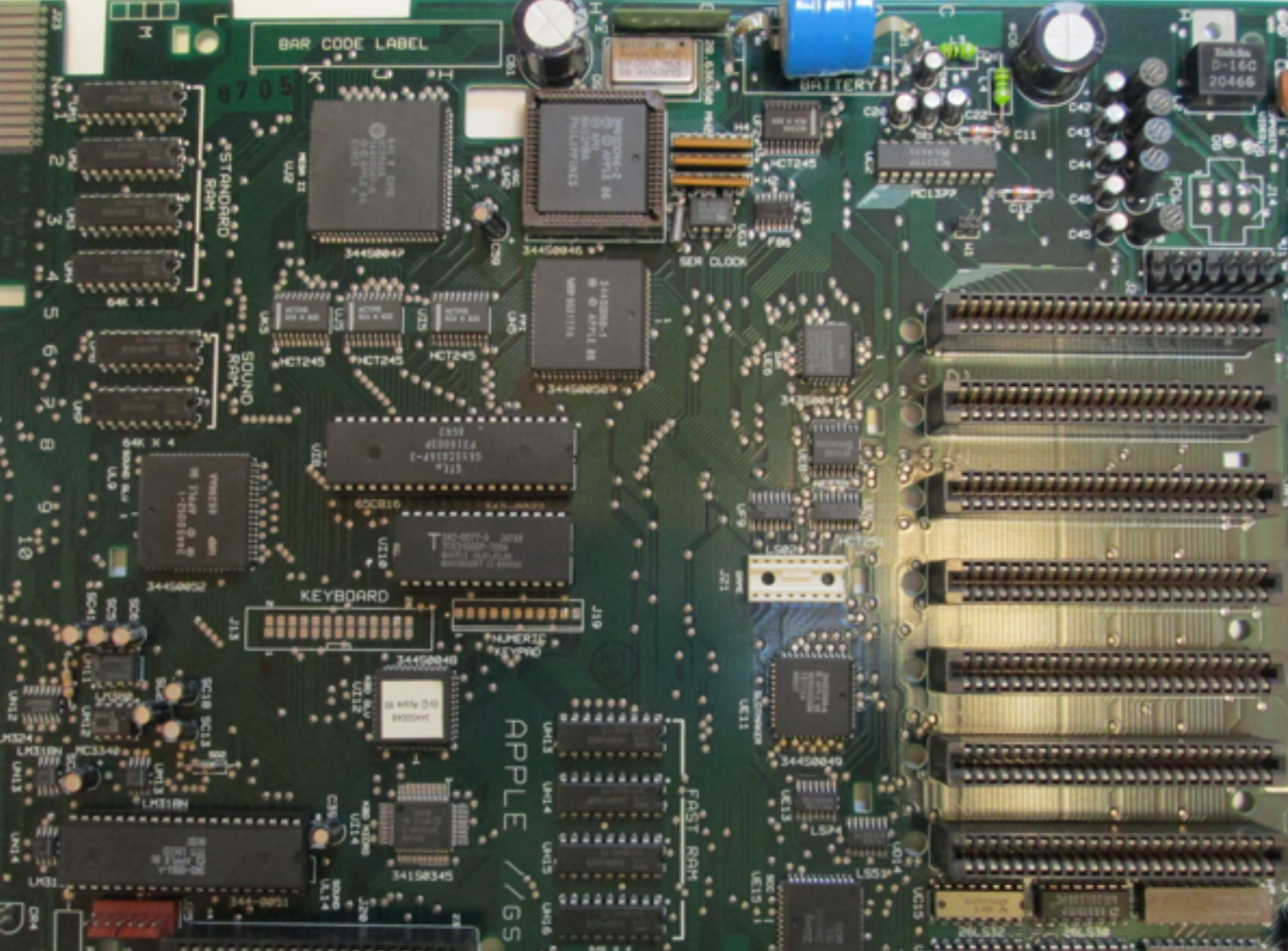
S'il était relativement facile de maîtriser l'Apple 2e et 2c, - facilité qui a permis a quelques pseudo-informaticiens mégalomanes, parce qu'ils savaient modifier un octet, de se faire un nom sur 2e - il est beaucoup plus difficile de maîtriser complètement le II GS, et il est d'ailleurs amusant de constater que ces pseudos-informaticiens ont disparu avec l'évolution technique qu'a représenté le II GS. Si nous avons un dernier conseil à vous donner avant de lire ce livre, ce serait de vous munir d'un assembleur. Tous les auteurs de ce livre utilisent l'assembleur «Merlin 16» de chez «Roger Wagner Publishing Inc».

Sachez que ce livre a nécessité 4 mois de travail à 5. Nous sommes prêts à en écrire un second, notamment sur les périphériques, les extensions, si ce livre connaît dans le milieu du II GS, un réel succès.

Nous espérons simplement que vous l'apprécierez, et que vous comprendrez mieux le fonctionnement de l'Apple II GS. Ceci à fin que le II GS ait la même durée de vie que le 2e. C'est notre seul souhait.

Pour toutes questions, remarques, ou suggestions n'hésitez pas à nous joindre via notre éditeur - Toolbox - 6 Rue Henri Barbusse, 95100 Argenteuil. Nous tenterons de répondre dans la mesure de nos connaissances à tout votre courrier.

Les auteurs



Le 65C816

1.0 Avant propos sur les microprocesseurs

Les microprocesseurs tels que nous les connaissons à l'heure actuelle, sont nés en 1972. A cette date le développement de la technologie était devenu tel qu'il fut possible de fabriquer des circuits comprenant plusieurs milliers de transistors.

Le microprocesseur est composé de deux éléments principaux :

- 1) un processeur, c'est à dire l'élément capable de traiter des informations.
- 2) un circuit intégré, c'est à dire un ensemble indissociable de transistors, réalisant différentes fonctions, enfermés en un même boîtier.

Le microprocesseur réalise des opérations arithmétiques (+, - / * ,/) ainsi que les opérations logiques (ET, OU, OU EXCLUSIF). Il est capable d'effectuer des décalages et des rotations a droite, à gauche, des comparaisons, des masques.

Il peut même prendre des décisions suivant le résultat d'une opération (si le résultat est inférieur, égal, ou supérieur à la donnée comparée).

1.0.1 L'organisation d'un microprocesseur.

Un microprocesseur se compose de 3 parties principales :

1.0.1.1 L'unité de contrôle

L'unité de contrôle décode les instructions envoyées par la mémoire et élabore les signaux de commande nécessaires à l'exécution d'un programme.

1.0.1.2 L'unité arithmétique et logique

L'unité arithmétique et logique se charge de l'exécution des opérations arithmétiques et logiques.

1.0.1.3 Les registres

Les registres se classent en 2 catégories

Ceux qui sont accessibles par le programmeur et ceux qui ne le sont pas. Nous n'étudierons que les registres accessibles par les programmeurs.

Ces registres se classent eux-mêmes en trois sortes :

- Les registres de données
- Les registres d'adresse
- Le compteur ordinal et le registre d'état du processeur

1.0.1.3 /1 Les registres de données

Ces registres assurent le stockage intermédiaire de données allant vers ou venant de l'unité arithmétique ou logique, ou de la mémoire.

1.0.1.3/2 les registres d'adresses

Ce sont en fait les pointeurs qui stockent l'adresse d'une position mémoire.

1.0.1.3/3 le compteur ordinal et le registre d'état du processeur.

Le compteur ordinal suit pas à pas l'exécution d'un programme. Il indique au microprocesseur l'adresse de la prochaine instruction devant être exécutée.

Le registre d'état du processeur contient un certain nombre de bits positionnés à 0 ou 1, suivant le résultat obtenu après l'exécution de certaines instructions.

1.0.2 Le langage du processeur.

Le microprocesseur ne connaît que le langage binaire. Le bit se caractérise par 2 états 0 ou 1 qui au niveau du processeur se matérialise par deux tensions (0 et 5 V).

8 bits forment un octet. Avec un octet on peut distinguer 2^8 états, c'est à dire que l'on peut sélectionner une instruction parmi 256 positions de mémoire.

1.1.1 Introduction

Le microprocesseur de l'Apple IIgs est le 65C816 conçu par David D. Mensch Jr. de chez Western Design Center. C'est un successeur 16 bits du 65C02. Il peut adresser 16 Mo de \$00/0000 à \$FF/FFFF, possède un bus d'adresse de 24 bits et porte à 16 bits tous les registres existants du 65C02. C'est en fait un faux 16 bits car son bus de données est de 8 bits. Mais vu les faibles temps d'accès mémoire, cela n'est pas vraiment pénalisant. Le 65C816 n'est pas le seul successeur du 65C02, il existe aussi le 65C802, qui est un 65C816 qui n'opère que dans le banc 0, mais qui a l'avantage d'être compatible broche à broche avec le 65C02. Ce microprocesseur est dit «hybride» car il a la particularité de pouvoir opérer en 2 modes, et de passer de l'un à l'autre :

- en mode natif soit en 65C816
- en mode émulation soit en 65C02

Contrairement au 65C02 qui fut fabriqué en NMOS, le 65C816 est fabriqué en CMOS (Complementary Metal Oxide Semiconductor), et tire tous les avantages de cette technique.

Le 65C816 peut être cadencé à deux vitesses à 2,8 Megahertz, ou à 1 Megahertz. En fait 2,8 Megahertz est la vitesse théorique, la vitesse réelle du microprocesseur est de 2,5 Megahertz, soit 2.500.000 opérations arithmétiques ou logiques par seconde, car un certain nombre de cycles sont utilisés pour le rafraichissement de la mémoire, et pour les opérations de resynchronisation.

Lorsque des données de 16 bits sont déplacées de la mémoire au microprocesseur ou inversement cette opération est effectuée en 2 cycles. Pendant le 1er cycle le 65C816 écrit ou lit les 8 bits de poids faible de la valeur. Pendant le second cycle il lit ou écrit les 8 bits de poids fort de la valeur. Les 8 bits de poids faible sont toujours stockés l'endroit effectif de la mémoire. Les 8 bits de poids fort sont stockés l'endroit effectif de la mémoire plus un.

1.12 les registres

Les registres peuvent être considérés comme des mémoires, mais très court terme, destinés à mémoriser des données lorsque des calculs sont effectués sur celles-ci.

Le 65C816 contient bien entendu tous les registres que l'on trouve dans le 65C02 (A,X,Y,P,S) mais ils ont été étendus à 16 bits. Le concepteur du 65C816 lui a donné 3 nouveaux registres par rapport au 65C02. Tous ces registres y compris bien évidemment les 3 nouveaux seront vus dans le détail ci après.

Schéma des registres du 65C816

1.12.1 L'accumulateur

L'accumulateur A est un registre de 16 bits ce qui permet d'y garder un nombre compris entre 0 et 65535. L'avantage de l'accumulateur est qu'il peut être utilisé pour réaliser toutes sortes d'opérations arithmétiques ou logiques, contrairement aux registres d'index X et Y dont l'usage est nettement plus limité.

LDA	#\$20	; On met la valeur \$20 (32 en décimal) dans le
		; registre A
CLC		; On met la retenue à zéro
ADC	#\$01	; On ajoute 1 au contenu de l'accumulateur
		; A vaut maintenant \$21 (33 en décimal)

En mode natif (E=0) quand le bit M du registre d'état du processeur est égal à 0, l'accumulateur est en mode 16 bits. Quand le bit M du registre d'état du processeur est égal à 1, l'accumulateur fonctionne en mode 8 bits utilisant uniquement la partie basse de l'accumulateur (les 8 bits de poids faible).

L'accumulateur est souvent divisé en 2 parties:

- Partie basse ou registre A (8 bits) - bits de poids faible
- Partie haute ou registre B (8 bits) - bits de poids fort

Ces 2 registres A et B forment l'accumulateur et sont dénommés C ($A+B=C$).

En mode natif (E=0) quand le bit M du registre d'état du processeur est égal à 0, l'accumulateur est en mode 16 bits. Quand le bit M du registre d'état du processeur est égal à 1, l'accumulateur fonctionne en mode 8 bits utilisant uniquement la partie A de l'accumulateur.

Dans ce dernier cas la partie B peut servir de mémoire temporaire, notamment avec l'instruction XBA.

Les 16 bits du registre sont disponibles dans les 2 modes du 65C816, aussi bien en mode émulation qu'en mode natif.

1.1.2.2 Les registres d'index X et Y

Le 65C816 a deux registres d'index, X et Y, c'est à dire que leur contenu de ces deux registres d'index sont susceptibles de s'ajouter à une adresse mémoire. (Voir la partie sur l'adressage indexé). Ces deux registres peuvent être en mode 8 bits ou en mode 16 bits. Si le bit X du registre d'état du processeur est à 1, les deux registres seront en mode 8 bits; si le bit X du registre d'état du processeur est à 0, les deux registres seront en mode 16 bits. En mode émulation 65C02, ces deux registres sont toujours en 8 bits, contrairement à l'accumulateur.

1.1.2.3 Le registre direct ou Registre D

C'est un nouveau registre par rapport au 65C02. Ce registre est en fait une extension de la page zéro du 65C02. Le registre D permet que cette page zéro de 256 octets (dans le 2e / 2c de \$00 / 0000 à 00FF) puisse se trouver n'importe où dans le banc zéro de \$00/0000 à \$00/FFFF. Cette page zéro relogeable est appelée la page directe.

1.1.2.4 Le registre banc de données (Data bank register)

C'est un registre 8 bits. Il contient en mode natif les bits de poids les plus forts - bits de 16 à 24 - de l'adresse d'une donnée spécifiée par le registre d'index X ou Y. En mode émulation les 8 bits de ce registre sont mis à 0 et ne peuvent être modifiés.

1.1.2.5 Le registre PBR (program bank register).

C'est un registre de 8 bits, contenant les bits de poids les plus fort bits 16 à 24 - de l'adresse de l'instruction suivante à exécuter. Ces 8 bits sont accolés aux 16 bits du compteur ordinal pour former une adresse sur 24 bits.

1.1.2.6 Le registre S ou le pointeur de pile.

Tous les microprocesseurs gèrent une pile c'est à dire une zone de mémoires consécutives obéissant à la règle LIFO (last in first out) ou DEPS (dernier entré, premier

sorti). La pile est nécessaire pour contrôler les sous-programmes ainsi que les interruptions. La pile est un ensemble de registres ou bloc mémoire, allouée à un empilement de données. Elle se caractérise par sa structure chronologique, c'est à dire que le premier élément introduit dans la pile est placé au bas de celle-ci, le suivant est placé au dessus, et ainsi de suite. Cette pile peut être étendue jusqu'à 64K et peut se situer n'importe où dans le banc O.

Pour gérer cette pile, il est nécessaire d'avoir un pointeur de pile qui permet de savoir où se trouve le sommet de la pile en mémoire. Ce registre de 16 bits contient donc l'adresse du sommet de la pile.

Il est automatiquement décrémenté d'une unité après chaque transfert d'un mot dans la pile. Il est automatiquement incrémenté d'une unité après chaque lecture d'un mot dans la pile.

Enfin le pointeur de pile joue un rôle particulier lors des demandes d'interruptions et des appels de sous-programmes. (Voir les instructions).

1.1.2.7 Le registre PC (program counter) ou registre compteur ordinal.

Ce registre de 16 bits indique toujours l'emplacement de la prochaine instruction devant être exécutée. Déjà avec le 65C02 ce registre était de 16 bits. Le 65C02 n'était capable d'adresser que 64 kilos, cela constituait une limite, en dehors bien entendu de la technique dit du «bank switching». Le 65C816 est capable d'adresser jusqu'à 16 Mo de mémoire en utilisant 24 bits pour l'adressage (Ajout de 8 bits par le PBR). Certaines instructions agissent directement sur le PC, il s'agit de toutes les instructions de branchement et de saut. Lors de l'initialisation, le compteur ordinal est chargé avec l'adresse de départ du programme.

1.1.2.8 Le registre d'état du microprocesseur ou registre P

Le registre d'état du microprocesseur est un registre de 8 bits. La version du registre P du processeur 65816 est une version étendue de celle du 65C02.

65C02



65816



Voyons maintenant bit par bit le contenu du registre P:

- Bit 0 ou C → (Carry generated) c'est le bit de report ou de retenue. Ce bit a un rôle double. Il indique principalement si une opération de soustraction ou d'addition a engendré une retenue et sert également de neuvième bit dans les opérations de rotation ou de décalage dans le registre d'état du processeur (instructions ASL, LSR, ROR et ROL).
- Bit 1 ou Z → (Zero result): Ce bit sera à zéro si la dernière opération n'a pas engendré un résultat nul.
- Bit 2 ou I → (Interrupts flag): si ce bit est à 1 les interruptions masquables seront inhibées (IRQ)
- Bit 3 ou D → (Decimal mode): sert à déterminer le mode de calcul :0 si binaire ou 1 si DCB (décimal codé binaire).
- Bit 4 ou X → (index register size): si ce bit est à 0 les registres d'index X et Y sont des registres en 16 bits.
En mode émulation (65C02) ce bit s'appelle B et il est mis à 1 lorsque une interruption logicielle (BRK) se produit, ce qui permet de distinguer entre BRK et une interruption hardware (IRQ et NMI).
- Bit 5 ou M → (Memory and accumulator flag) : Ce bit sert à déterminer la longueur de l'accumulateur. Si ce bit est à 1 l'accumulateur est en mode 8 bits, si ce bit est à 0 l'accumulateur est en mode 16 bits.
En mode émulation (6502) ce bit est toujours à 1
- Bit 6 ou V → (accumulator overflow) : Ce bit indique si au cours d'une soustraction ou d'une addition le signe du résultat a changé à cause du dépassement du résultat dans le bit de signe.
- Bit 7 ou N → (Negative result): Ce bit détermine le signe du dernier résultat. Si ce bit est à 1, le résultat est négatif, si ce bit est à 0 le résultat est positif.

1.1.3 Les Instructions du 65C816

Les instructions du microprocesseur ont été définies une fois pour toutes par le fabricant du microprocesseur, et ne peuvent être modifiées par l'utilisateur qui ne peut les exploiter que pour écrire un programme. Le 65C816 possède 256 instructions. Ce sont ces instructions par ordre alphabétique que nous allons maintenant détailler.

Pour chaque instruction nous allons inclure une table, comme celle que nous montrons ci-dessous, illustrant l'effet qu'elle provoque sur le registre d'état du processeur, les registres du processeurs (A, X et Y) ainsi que la mémoire.

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.	0	.	1	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	.				

Légende:

. = Valeur non modifiée

/ = Valeur modifiée

0 = Bit mis à zéro

1 = Bit mis à un

Les commandes du processeur 65C816

ADC

ADd memory to accumulator with Carry

La fonction ADC additionne le contenu de la mémoire spécifiée par l'opérande à l'accumulateur avec le bit de report. L'opération peut être effectuée en mode binaire ou décimal. Le résultat de cette opération est mis dans l'accumulateur.

Lors d'une addition si vous ne voulez pas que le bit de report influe sur le résultat, vous devez mettre ce bit de report à zéro en utilisant avant l'instruction ADC l'instruction CLC.

Fonction: $A \leftarrow A + M + C$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	ADC addr	\$6D	3	4 ¹
Absolu long	ADC long	\$6F	4	5 ¹
Absolu long indexé par X	ADC long,X	\$7F	4	5 ¹
Absolu indexé par X	ADC addr,X	\$7D	3	4 ^{1,3}
Absolu indexé par Y	ADC addr,Y	\$79	3	4 ^{1,3}
Direct	ADC dp	\$65	2	3 ^{1,2}
Direct indirect	ADC (dp)	\$72	2	5 ^{1,2}
Direct indirect indexé par Y	ADC (dp),Y	\$71	2	5 ^{1,2,3}
Direct indirect long	ADC <dp>	\$67	2	6 ^{1,2}
Direct indirect long indexé par Y	ADC <dp>,Y	\$77	2	6 ^{1,2,4}
Direct indexé indirect	ADC (dp,X)	\$61	2	6 ^{1,2}
Direct indexé par X	ADC dp,X	\$75	2	4 ^{1,2}
Immédiat en émulation 65C02 (M=1)	ADC #const	\$69	2	2
Immédiat en mode natif 65816 (M=0)	ADC #const	\$69	3	3

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile relative	ADC sr,S	\$63	2	4 ¹
Pile relative indirecte indexée	ADC (sr,S),Y	\$73	2	7 ¹

- ¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)
- ² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro
- ³ Ajouter 1 au nombre de cycles si l'addition du registre d'index fait changer de page mémoire

AND

AND accumulator with memory

Cette instruction effectue un ET logique entre le contenu de la mémoire et l'accumulateur. Cette opération est effectuée bit à bit entre l'accumulateur et les bits correspondants en mémoire. Le résultat est stocké dans l'accumulateur.

L'opération ET logique, suit la table de vérité suivante:

Première opérande	0	0	1	1
Seconde opérande	0	1	0	1
Résultat	0	0	0	1

Fonction: $A \leftarrow A \wedge M$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat en émulation 65C02 (M=1)	AND #const	\$29	2	2 ¹
Immédiat en mode natif 65816 (M=0)	AND #const	\$29	3	3 ¹
Absolu	AND addr	\$2D	3	4 ¹
Absolu long	AND long	\$2F	4	5 ¹
Absolu indexé par X	AND addr,X	\$3D	3	4 ^{1,3}
Absolu indexé par Y	AND addr,Y	\$39	3	4 ^{1,3}
Absolu long indexé par X	AND long,X	\$3F	4	5 ¹
Direct	AND dp	\$25	2	3 ^{1,2}
Direct indirect	AND (dp)	\$32	2	5 ^{1,2}
Direct indirect indexé par Y	AND (dp),Y	\$31	2	5 ^{1,2,3}
Direct indirect long	AND <dp>	\$27	2	6 ^{1,2}
Direct indexé indirect	AND (dp,X)	\$61	2	6 ^{1,2}

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Direct indirect long indexé par Y	AND <dp>,Y	\$37	2	6 ^{1,2}
Direct indexé par X	AND dp,X	\$35	2	4 ^{1,2}
Pile relative	AND sr,S	\$23	2	4 ¹
Pile relative indirecte indexée	AND (sr,S),Y	\$33	2	7 ¹

- ¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)
- ² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

Exemple

```
SEP    #$20        ; A en mode 8 bits
```

```
LDA    #$43        ; A contient la valeur %01000011
```

```
AND    #$81        ; On applique un filtre (%10000001)
                        ; A contient maintenant $01 (%00000001)
```

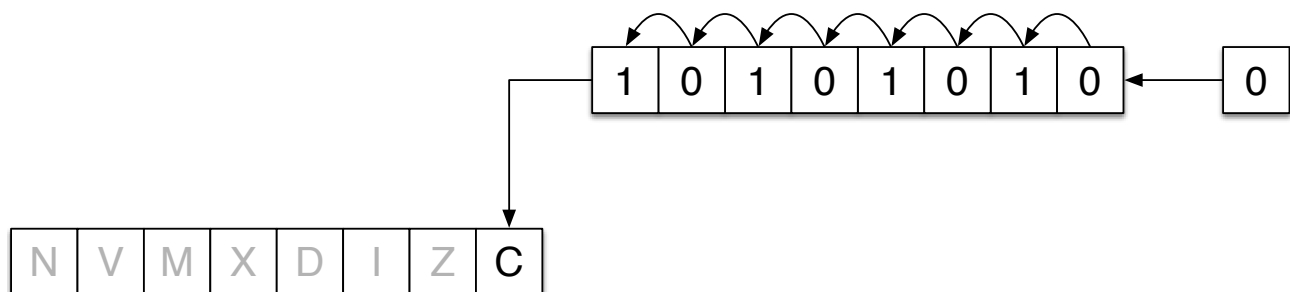
ASL

Arithmetic Shift Left

On décale à gauche l'accumulateur ou le contenu d'une mémoire. En mode 8 bits le bit 0 est mis à zéro, tandis que le bit sortant (le 7ème) gauche est transféré dans la retenue. En mode 16 bits le bit 0 est mis à zéro, tandis que le 15ème bit est transféré dans la retenue.

Cette opération correspond en fait à une multiplication par 2.

Fonction:



Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Accumulateur	ASL A	\$0A	1	2
Absolu	ASL addr	\$0E	3	6 ¹
Absolu indexé par X	ASL addr,X	\$1E	2	7 ¹
Direct	ASL dp	\$06	2	5 ^{1,2}
Direct indexé par X	ASL dp,X	\$16	2	6 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

Exemple

```
SEP # $20 ; A en mode 8 bits
```

```
LDA  #$81      ; A contient la valeur %10000001
ASL  A         ; A contient maintenant $03 (%00000010)
BCS  OK       ; Le programme saute vers OK car le bit C vaut 1
```

BCC

Branch if Carry Clear

Le bit C (retenue ou carry) du registre d'état du processeur est testé. Si le bit C est à zéro dans ce cas un branchement relatif au registre PC (Program Counter) est alors exécuté. Dans le cas contraire c'est l'instruction suivante au BCC qui est exécutée. Ce branchement permet d'accéder à toute instruction située entre -128 et + 127 octets à partir de l'instruction qui suit le BCC. BCC est utilisé dans plusieurs cas : Pour tester un changement de résultat dans la carry, pour déterminer si le résultat d'une comparaison est «plus petit que» (dans ce cas le branchement s'effectue) ou «plus grand que» ou «égal à» (dans ces deux derniers cas le branchement ne s'effectue pas).

Fonction: Si $C=0$ alors $PC \leftarrow PC+N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Relatif	BCC adr	\$90	2	2 ^{1,2}

- ¹ Ajouter 1 au nombre de cycles si le branchement est effectué
- ² Ajouter 1 au nombre de cycles si le branchement conduit à changer de page mémoire lorsque l'instruction s'exécute en mode émulation 65C02 (E=1)

BCS

Branch if Carry Set

Le bit C (retenue ou carry) du registre d'état du processeur est testé. Si C est al dans ce cas un branchement relatif au registre PC (program counter) est alors exécuté. Dans le cas contraire c'est l'instruction suivante au BCS qui est exécutée. Ce branchement permet d'accéder toute instruction située entre -128 et + 127 octets à partir de l'instruction qui suit le BCS. BCS est utilisé dans plusieurs cas : Pour tester un changement de résultat dans la carry, pour déterminer si le résultat d'une comparaison est «plus grand que» ou «égal à» dans ce cas le branchement est effectuée, ou bien «plus petit que» dans ce cas le branchement n'est pas effectué.

Fonction: Si $C=1$ alors $PC \leftarrow PC+N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Relatif	BCS adr	\$B0	2	2 ^{1,2}

- ¹ Ajouter 1 au nombre de cycles si le branchement est effectué
- ² Ajouter 1 au nombre de cycles si le branchement conduit à changer de page mémoire lorsque l'instruction s'exécute en mode émulation 65C02 (E=1)

BEQ

Branch if EQual

Le bit Z (zero flag) du registre d'état du processeur est testé. S'il est à 1 un branchement relatif au registre PC (program counter) est alors effectué. Sinon c'est l'instruction suivante qui est exécutée. Ce branchement permet d'accéder à toute instruction située entre -128 et + 127 octets à partir de l'instruction qui suit le BEQ. BEQ est surtout utilisée pour déterminer si le résultat d'une comparaison est zéro (les deux valeurs comparées sont égales).

Fonction: Si $Z=1$ alors $PC \leftarrow PC+N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Relatif	BEQ adr	\$F0	2	2 ^{1,2}

- ¹ Ajouter 1 au nombre de cycles si le branchement est effectué
- ² Ajouter 1 au nombre de cycles si le branchement conduit à changer de page mémoire lorsque l'instruction s'exécute en mode émulation 65C02 (E=1)

BIT

Compare accumulator BITs with contents of memory

Cette instruction effectue un ET logique entre l'accumulateur et le contenu de la mémoire spécifiée par l'opérande. Cette opération n'affecte pas l'accumulateur. Le résultat de cette opération sera soit nul soit non nul, le bit Z (Zero flag) du registre d'état du processeur sera donc positionné suivant ce résultat.

En mode 16 bits (65C816) les bits 14 et 15 sont recopiés dans respectivement V (overflow flag) et N (sign flag). En mode 8 bits (Emulation) ce sont les bits 6 et 7 qui sont recopiés respectivement dans V et N.

Aucune des opérandes n'est affectée par cette opération. Le registre d'état lui par contre est affecté.

En mode d'adressage immédiat, les bits N et V ne sont pas affectés.

Fonction: $A \leftarrow A \wedge M$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat en émulation 65C02 (M=1)	BIT #const	\$89	2	2
Immédiat en mode natif 65816 (M=0)	BIT #const	\$89	3	3
Absolu	BIT addr	\$2C	3	4 ¹
Absolu indexé par X	BIT addr,X	\$3C	3	4 ^{1,3}
Direct	BIT dp	\$24	2	3 ^{1,2}
Direct indexé par X	BIT dp,X	\$34	2	4 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (Direct page register) est différent de zéro

³ Ajouter 1 au nombre de cycles si l'addition du registre d'index fait changer de page mémoire

BMI

Branch if Minus

Le bit N (sign flag) du registre d'état du processeur est testé. Si le bit N est à 1 un branchement relatif au registre PC (program counter) est alors effectué. Sinon c'est l'instruction suivante qui est exécutée. Ce branchement permet d'accéder à toute instruction située entre -128 et +127 octets à partir de l'instruction suivante BMI.

Fonction: Si $N=1$, alors $PC \leftarrow PC + N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Relatif	BMI adr	\$30	2	2 ^{1,2}

- ¹ Ajouter 1 au nombre de cycles si le branchement est effectué
- ² Ajouter 1 au nombre de cycles si le branchement conduit à changer de page mémoire lorsque l'instruction s'exécute en mode émulation 65C02 (E=1)

BNE

Branch if Not Equal

Le bit Z (zero flag) du registre d'état du processeur est testé. Si le bit Z est à zéro un branchement relatif au registre PC (Program Counter) est alors effectué. Dans le cas contraire c'est l'instruction suivante BNE qui est exécutée. Ce branchement permet d'accéder à toute instruction située entre -128 et +127 octets à partir de l'instruction suivante BML.

L'instruction BNE est utilisée dans plusieurs cas : pour déterminer si le résultat d'une comparaison n'est pas nul (les deux valeurs comparées ne sont pas égales) ou pour savoir si la valeur venant d'être chargée de la pile est nulle ou non.

Fonction: Si $Z=0$, alors $PC \leftarrow PC + N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Relatif	BNE adr	\$D0	2	2 ^{1,2}

- ¹ Ajouter 1 au nombre de cycles si le branchement est effectué
- ² Ajouter 1 au nombre de cycles si le branchement conduit à changer de page mémoire lorsque l'instruction s'exécute en mode émulation 65C02 (E=1)

BPL

Branch if Plus

Le bit N (sign flag) du registre d'état du processeur est testé. Si le bit N est à zéro un branchement relatif au registre PC (Program Counter) est alors effectué. Dans le cas contraire c'est l'instruction suivante qui est exécutée. Ce branchement permet d'accéder à toute instruction située entre -128 et +127 octets à partir de l'instruction suivante BPL.

Fonction: Si $Z=0$, alors $PC \leftarrow PC + N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Relatif	BPL adr	\$10	2	2 ^{1,2}

- ¹ Ajouter 1 au nombre de cycles si le branchement est effectué
- ² Ajouter 1 au nombre de cycles si le branchement conduit à changer de page mémoire lorsque l'instruction s'exécute en mode émulation 65C02 (E=1)

BRA

BRanch Always

Le branchement est toujours effectué (branchement inconditionnel). Cette instruction est équivalente à IMP mais elle est limitée par l'accès à toute instruction située entre -128 et +127 octets à partir de l'instruction suivante BRA.

Fonction: $PC \leftarrow PC + N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Relatif	BRA adr	\$80	2	3 ¹

¹ Ajouter 1 au nombre de cycles si le branchement conduit à changer de page mémoire lorsque l'instruction s'exécute en mode émulation 65C02 (E=1)

BRK

Software BReaK

Cette instruction simule une interruption, stocke le contenu du registre d'état et du compteur ordinal dans la pile.

En mode 16 bits le registre compteur ordinal est forcé à l'adresse mémoire contenue dans le vecteur d'interruption en 00FFE6,00FE7.

En mode 8 bits le registre compteur ordinal est forcé à l'adresse mémoire contenue dans le vecteur d'interruption en 00FFE,00FFF.

Fonction: $PC \leftarrow (\$00FFE)$ si en mode émulation (65C02, e=1)

$PC \leftarrow (\$00FE6)$ si en mode natif (65816, e=0)

Registre d'état du processeur	N	V	M	X	D	I	Z	C
Émulation 65C02	.	.	/	.	0	1	.	.
Mode natif 65816	0	1	.	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	BRK	\$00	2 ¹	7 ²

- ¹ En fait, l'instruction BRK ne prend qu'un seul octet. Cependant lorsque le registre PC est mis sur la pile, sa valeur est augmentée de deux octets, ce qui permet d'utiliser un octet après le \$00 pour identifier le point où se produit l'interruption
- ² Ajouter 1 au nombre de cycles si le branchement conduit à changer de page mémoire lorsque l'instruction s'exécute en mode émulation 65C02 (E=1)

BRL

BRanch always Long

Branchement inconditionnel long. Cette instruction est similaire BRA ou à JMP, mais BRL spécifie une adresse 16 bits avec l'instruction. C'est une instruction à 3 bits contrairement à BRA qui n'en a que 2. Le principal avantage de cette instruction par rapport à PVIIP est qu'elle est entièrement relogeable. Cependant on gagne un cycle en exécutant un JMP en mode absolu.

Fonction: $PC \leftarrow PC + NN$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Relatif long	BRL adr	\$82	3	4

BVC

Branch if oVerflow Clear

Le bit V (overflow flag ou bit de dépassement de capacité) du registre d'état du processeur est testé. Un branchement relatif au registre PC (Program Counter) est effectué si le bit V est à zéro. Ceci est vrai après une opération sur des valeurs en complément à deux, s'il n'y avait pas de dépassement (Résultat validé). Pour mettre ce bit à zéro, vous pouvez utiliser l'instruction CLV. Ce branchement permet d'accéder à toute instruction située entre -128 et +127 octets à partir de l'instruction suivante BVC.

Fonction: Si $V=0$, alors $PC \leftarrow PC + N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Relatif	BVC adr	\$50	2	2 ^{1,2}

- ¹ Ajouter 1 au nombre de cycles si le branchement est effectué
- ² Ajouter 1 au nombre de cycles si le branchement conduit à changer de page mémoire lorsque l'instruction s'exécute en mode émulation 65C02 (E=1)

BVS

Branch if overflow Set

Un branchement relatif au registre PC (Program Counter) est effectué lorsque le bit V (overflow flag ou bit de dépassement de capacité) est 1. Le bit V est à 1 lorsqu'après une opération sur des valeurs en complément à deux il y avait un dépassement (Résultat invalidé). Ce branchement permet d'accéder à toute instruction située entre -128 et +127 octets à partir de l'instruction suivante BVS.

Fonction: Si $V=1$, alors $PC \leftarrow PC + N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Relatif	BVS adr	\$70	2	2 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le branchement est effectué

² Ajouter 1 au nombre de cycles si le branchement conduit à changer de page mémoire lorsque l'instruction s'exécute en mode émulation 65C02 (E=1)

CLC

CLear Carry flag

Le bit C (carry ou bit de retenu) est mis à zéro. Cette instruction est principalement utilisée avant une addition (ADC) afin que le résultat de cette addition ne soit pas affectée par le bit de retenu.

Lorsque l'instruction CLC est utilisée juste avant BCC (Branch on Carry Clear), cela a pour résultat de transformer l'instruction BCC en BRA (Branch Always). Enfin CLC est aussi utilisé avec l'instruction XCE, afin de remettre le microprocesseur en mode natif.

Fonction: $C \leftarrow 0$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	0
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	CLC	\$18	1	2

CLD

CLear Decimal flag

Le bit D (Decimal mode flag ou bit décimal) du registre d'état du processeur est mis à zéro, afin de passer du mode décimal en mode binaire, pour que les instructions ADC et SBC puissent s'exécuter parfaitement.

Fonction: $D \leftarrow 0$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	0	.	.	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	CLD	\$D8	1	2

CLI

CLear Interrupt disable flag

Le bit I (interrupt flag) du registre d'état du processeur est mis à zéro, ce qui a pour effet de remettre en service les interruptions. Lorsque le bit I est à 1 les interruptions hardware sont ignorées.

Fonction: $I \leftarrow 0$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	0	.	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	CLI	\$58	1	2

CLV

CLear oVerflow flag

Le bit V (overflow flag ou bit de dépassement de capacité) du registre d'état du microprocesseur est mis à 0.

Fonction: $V \leftarrow 0$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	0
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	CLV	\$B8	1	2

CMP

CoMPare accumulator with memory

L'instruction CMP effectue une soustraction virtuelle Accumulateur - Mémoire. Cependant le résultat n'est pas mis dans l'accumulateur qui reste inchangé. Cette soustraction correspond en fait à une comparaison puisque le résultat sera soit nul, soit positif ou négatif, et par là même déterminera qu'elle était la valeur la plus grande.

- Si A est plus grand ou égal à M le bit C est mis à 1.
- Si A = M alors Z est mis à 1

Il y a deux possibilités:

1) L'accumulateur est en mode 8 bits : la comparaison se fait sur 8 bits

2) L'accumulateur est en mode 16 bits (M=0) : La partie A de l'accumulateur est comparée avec l'adresse effective de la mémoire, la partie B de l'accumulateur est comparée avec l'adresse effective de la mémoire plus 1.

Fonction: $A - M \implies A > M$
 $A < M$
 $A = M$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat en émulation 65C02 (M=1)	CMP #const	\$C9	2	2
Immédiat en mode natif 65816 (M=0)	CMP #const	\$C9	3	3
Absolu	CMP addr	\$CD	3	4 ¹
Absolu long	CMP long	\$CF	4	5 ¹
Absolu indexé par X	CMP addr,X	\$DD	3	4 ^{1,3}
Absolu indexé par Y	CMP addr,Y	\$D9	3	4 ^{1,3}
Absolu long indexé par X	CMP long,X	\$DF	4	5 ¹

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Direct	CMP dp	\$C5	2	3 ^{1,2}
Direct indirect	CMP (dp)	\$D2	2	5 ^{1,2}
Direct indirect indexé par Y	CMP (dp), Y	\$D1	2	5 ^{1,2,3}
Direct indirect long	CMP <dp>	\$C7	2	6 ^{1,2}
Direct indexé indirect	CMP (dp,X)	\$C1	2	6 ^{1,2}
Direct indirect long indexé par Y	CMP <dp>, Y	\$D7	2	6 ^{1,2}
Direct indexé par X	CMP dp,X	\$D5	2	4 ^{1,2}
Pile relative	CMP sr,S	\$C3	2	4 ¹
Pile relative indirecte indexée	CMP (sr,S), Y	\$D3	2	7 ¹

- ¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)
- ² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro
- ³ Ajouter 1 au nombre de cycles si l'addition du registre d'index fait changer de page

COP

CO Processor enable

C'est une instruction similaire à BRK (break). Tout comme BRK il s'agit d'une interruption logicielle mais utilisant un vecteur différent. En mode natif - 16 Bits (E=0):

Le registre compteur ordinal est forcé alors à l'adresse indiquée par le vecteur en \$00/FFE4,FFE5. Les opérandes de la chaîne entre \$80 et \$FF ont été réservées par le concepteur du 65C816 (Western Design Center).

Fonction: $PC \leftarrow (\$00FFF4)$ si en mode émulation (65C02, e=1)

$PC \leftarrow (\$00FFE6)$ si en mode natif (65816, e=0)

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	0	/	.	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	COP	\$02	2	7 ¹

¹ Ajouter 1 au nombre de cycles lorsque l'instruction s'exécute en mode natif 65C816 (E=0)

CPX

CoMpare index register X with memory

L'instruction CPX compare le contenu du registre d'index X avec le contenu d'une mémoire. Cette comparaison correspond en fait à une soustraction entre la valeur contenue dans le registre X et le contenu de la mémoire, mais le résultat est ignoré. Le bit Z est mis à 1 lorsqu'il y a égalité dans la comparaison. Le bit C est mis à 1 si $X >$ ou $=$ à M.

Il y a deux possibilités:

1) Le registre d'index est en mode 8 bits : la comparaison se fait sur 8 bits

2) Le registre d'index X est en mode 16 bits ($X=0$): Les bits de poids faible du registre d'index X sont comparés avec l'adresse effective de la mémoire, les 8 bits de poids fort du registre d'index X sont comparés avec l'adresse effective de la mémoire plus 1.

Fonction: $X - M \implies X > M$
 $X < M$
 $X = M$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat en émulation 65C02 (M=1)	CPX #const	\$E0	2	2
Immédiat en mode natif 65816 (M=0)	CPX #const	\$E0	3	3
Absolu	CPX addr	\$EC	3	4 ¹
Direct	CPX dp	\$E4	2	3 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit X du registre d'état du processeur vaut 0 (X et Y en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (Direct page register) est différent de zéro

CPY

CoMpare index register Y with memory

L'instruction CPY compare le contenu du registre d'index Y avec le contenu d'une mémoire. Cette comparaison correspond en fait à une soustraction entre la valeur contenue dans le registre Y et le contenu de la mémoire, mais le résultat est ignoré. Le bit Z est mis à 1 lorsqu'il y a égalité dans la comparaison. Le bit C est mis à 1 si $Y >$ ou $=$ à M.

Il y a deux possibilités:

1) Le registre d'index est en mode 8 bits : la comparaison se fait sur 8 bits

2) Le registre d'index Y est en mode 16 bits ($X=0$): Les bits de poids faible du registre d'index Y sont comparés avec l'adresse effective de la mémoire, les 8 bits de poids fort du registre d'index Y sont comparés avec l'adresse effective de la mémoire plus 1.

Fonction: $Y - M \implies Y > M$
 $Y < M$
 $Y = M$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat en émulation 65C02 (M=1)	CPY #const	\$C0	2	2
Immédiat en mode natif 65816 (M=0)	CPY #const	\$C0	3	3
Absolu	CPY addr	\$CC	3	4 ¹
Direct	CPY dp	\$C4	2	3 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit X du registre d'état du processeur vaut 0 (X et Y en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (Direct page register) est différent de zéro

DEC

DECrement

Une décrémentation d'une unité est effectuée sur le contenu d'un emplacement mémoire spécifié par l'opérande ou sur l'accumulateur. Si le contenu original de la mémoire est de # \$00, le résultat sera alors de # \$FF

Fonction: $M \leftarrow M - 1$ ou $A \leftarrow A - 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	DEC addr	\$CE	3	6 ¹
Absolu indexé par X	DEC add,X	\$DE	3	7 ¹
Accumulateur	DEC A	\$3A	1	2
Direct	DEC dp	\$C6	2	5 ^{1,2}
Direct indexé par X	DEC dp,X	\$D6	2	6 ^{1,2}

¹ Ajouter 2 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (Direct page register) est différent de zéro

DEX

DEcrement index register X

Le registre d'index X est décrémenté d'une unité. Si le contenu original du registre X est de # $\$00$, le résultat sera alors de # $\$FF$.

Fonction: $X \leftarrow X - 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	DEX	$\$CA$	1	2

DEY

DEcrement index register Y

Le registre d'index Y est décrémenté d' une unité. Si le contenu original du registre Y est de #\$00, le résultat sera alors de #\$FF.

Fonction: $Y \leftarrow Y - 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	/	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	DEY	\$88	1	2

EOR

Exclusive OR accumulator with memory

Cette instruction effectue un OU exclusif entre le contenu de la mémoire et l'accumulateur. Cette opération est effectuée bit à bit entre l'accumulateur et les bits correspondants en mémoire. Le résultat est stocké en A.

OU exclusif, suit la table de vérité suivante :

Première opérande	0	0	1	1
Seconde opérande	0	1	0	1
Résultat	0	1	1	0

Fonction: $A \leftarrow A \text{ EOR } M$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	/					

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat en émulation 65C02 (M=1)	EOR #const	\$49	2	2
Immédiat en mode natif 65816 (M=0)	EOR #const	\$49	3	3
Absolu	EOR addr	\$4D	3	4 ¹
Absolu long	EOR long	\$4F	4	5 ¹
Absolu indexé par X	EOR addr,X	\$5D	3	4 ^{1,3}
Absolu indexé par Y	EOR addr,Y	\$59	3	4 ^{1,3}
Absolu long indexé par X	EOR long,X	\$5F	4	5 ¹
Direct	EOR dp	\$45	2	3 ^{1,2}
Direct indirect	EOR (dp)	\$52	2	5 ^{1,2}
Direct indirect indexé par Y	EOR (dp),Y	\$51	2	5 ^{1,2,3}
Direct indirect long	EOR <dp>	\$47	2	6 ^{1,2}
Direct indexé indirect	EOR (dp,X)	\$41	2	6 ^{1,2}

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Direct indirect long indexé par Y	EOR <dp>,Y	\$57	2	6 ^{1,2}
Direct indexé par X	EOR dp,X	\$55	2	4 ^{1,2}
Pile relative	EOR sr,S	\$43	2	4 ¹
Pile relative indirecte indexée	EOR (sr,S),Y	\$53	2	7 ¹

- ¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)
- ² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

Exemple

```
SEP    #$20        ; A en mode 8 bits
```

```
LDA    #$F0        ; A contient la valeur %11110000
```

```
EOR    #$AA        ; On applique un filtre (%10101010)
```

```
        ; A contient maintenant $5A (%01011010)
```

INC

INCrement

Une incrémentation de 1 est effectuée sur le contenu d'un emplacement mémoire spécifié par l'opérande ou sur l'accumulateur. Si le contenu original de la mémoire est de #\$FF, le résultat sera alors de #\$00

Fonction: $M \leftarrow M + 1$ ou $A \leftarrow A + 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	INC addr	\$EE	3	6 ¹
Absolu indexé par X	INC addr,X	\$FE	3	7 ¹
Accumulateur	A	\$1A	1	2
Direct	INC dp	\$E6	2	5 ^{1,2}
Direct indexé par X	INC dp,X	\$F6	2	6 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

INX

INcrement index register X

Le registre d'index X est incrémenté d' 1. Si le contenu original du registre X est de #\$FF, le résultat sera alors de #\$00.

Fonction: $X \leftarrow X + 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	INX	\$E8	1	2

INY

INcrement index register Y

Le registre d'index Y est incrémenté d' 1. Si le contenu original du registre Y est de #\$FF, le résultat sera alors de #\$00.

Fonction: $Y \leftarrow Y + 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	/	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	INY	\$C8	1	2

JML

JuMp Long

Cette instruction permet d'exécuter un saut dans un autre banc mémoire à l'adresse spécifiée par l'opérande. Le registre PC (Program Counter) est chargé avec l'adresse de destination. Le registre PCB (Program Counter Bank) est chargé avec le 3ème octet de l'adresse de destination spécifiée par l'opérande.

Fonction: $PC \leftarrow \text{addr}$
 $PCB \leftarrow \text{addr} + 2$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu long	JML long	\$5C	4	4
Absolu indirect	JML (addr)	\$DC	3	6

Faites attention si vous utilisez le mode d'adressage absolu indirect. En effet, ce mode oblige à ce que l'adresse utilisée soit située dans le banc 00. Ceci n'est pas surprenant, vu que l'adresse n'est codée que sur deux octets, mais cela limite quand même l'utilité de ce mode d'adressage.

JMP

JuMP to new location

Cette instruction permet d'exécuter un saut à une adresse fixe spécifiée dans le même banc mémoire. Dans le cas d'une adresse relative, il faut utiliser l'instruction BRA qui permet d'écrire des programmes entièrement relogeables.

Fonction: $PC \leftarrow \text{addr}$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	JMP addr	\$4C	3	3
Absolu indirect	JMP (addr)	\$6C	3	5
Absolu indirect indexé par X	JMP (addr,X)	\$7C	3	6

JSL

Jump to Subroutine Long / Inter-bank

Cette instruction permet un branchement à un sous-programme n'importe quelle adresse de la mémoire.

Fonction: $(S) \leftarrow PBR, S \leftarrow S - 1$
 $(S) \leftarrow PCH, S \leftarrow S - 1$
 $(S) \leftarrow PCL, S \leftarrow S - 1$
 $PC \leftarrow ADDR, PBR \leftarrow ADDR + 2$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu long	JSL long	\$22	4	8

JSR

Jump to SubRoutine

Cette instruction permet un branchement à un sous-programme n'importe quelle adresse dans le banc utilisé.

Fonction: $(S) \leftarrow PCH, S \leftarrow S - 1$
 $(S) \leftarrow PCL, S \leftarrow S - 1$
 $PC \leftarrow ADDR$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	JSR add	\$20	3	6
Absolu indexé indirect	JSR (addr,X)	\$FC	3	8

LDA

LoaD Accumulator from memory

Chargement de l'accumulateur avec une valeur spécifique ou avec la valeur située à l'adresse donnée par l'opérande.

Fonction: $A \leftarrow M$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat en émulation 65C02 (M=1)	LDA #const	\$A9	2	2
Immédiat en mode natif 65816 (M=0)	LDA #const	\$A9	3	3
Absolu	LDA addr	\$AD	3	4 ¹
Absolu long	LDA long	\$AF	4	5 ¹
Absolu indexé par X	LDA addr,X	\$BD	3	4 ^{1,3}
Absolu indexé par Y	LDA addr,Y	\$B9	3	4 ^{1,3}
Absolu long indexé par X	LDA long,X	\$BF	4	5 ¹
Direct	LDA dp	\$A5	2	3 ^{1,2}
Direct indirect	LDA (dp)	\$B2	2	5 ^{1,2}
Direct indirect indexé par Y	LDA (dp),Y	\$B1	2	5 ^{1,2,3}
Direct indirect long	LDA <dp>	\$A7	2	6 ^{1,2}
Direct indexé indirect	LDA (dp,X)	\$A1	2	6 ^{1,2}
Direct indirect long indexé par Y	LDA <dp>,Y	\$B7	2	6 ^{1,2}
Direct indexé par X	LDA dp,X	\$B5	2	4 ^{1,2}
Pile relative	LDA sr,S	\$A3	2	4 ¹
Pile relative indirecte indexée	LDA (sr,S),Y	\$B3	2	7 ¹

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

³ Ajouter 1 au nombre de cycles si l'addition du registre d'index fait changer de page mémoire

LDX

Load index register X from memory

Chargement du registre d'index X avec une valeur spécifique ou avec la valeur située à l'adresse donnée par l'opérande.

Fonction: $X \leftarrow M$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat en émulation 65C02 (M=1)	LDX #const	\$A2	2	2
Immédiat en mode natif 65816 (M=0)	LDX #const	\$A2	3	3
Absolu	LDX addr	\$AE	3	4 ¹
Absolu indexé par Y	LDX addr,Y	\$BE	3	4 ^{1,3}
Direct	LDX dp	\$A6	2	3 ^{1,2}
Direct indirect indexé par Y	LDX (dp),Y	\$B6	2	4 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit X du registre d'état du processeur vaut 0 (X et Y en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

³ Ajouter 1 au nombre de cycles si l'addition du registre d'index fait changer de page mémoire

LDY

Load index register Y from memory

Chargement du registre d'index Y avec une valeur spécifique ou avec la valeur située à l'adresse donnée par l'opérande.

Fonction: $Y \leftarrow M$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	/	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat en émulation 65C02 (M=1)	LDY #const	\$A0	2	2
Immédiat en mode natif 65816 (M=0)	LDY #const	\$A0	3	3
Absolu	LDY addr	\$AC	3	4 ¹
Absolu indexé par X	LDX addr,X	\$BC	3	4 ^{1,3}
Direct	LDY dp	\$A4	2	3 ^{1,2}
Direct indexé par X	LDY dp,X	\$B4	2	4 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit X du registre d'état du processeur vaut 0 (X et Y en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

³ Ajouter 1 au nombre de cycles si l'addition du registre d'index fait changer de page mémoire

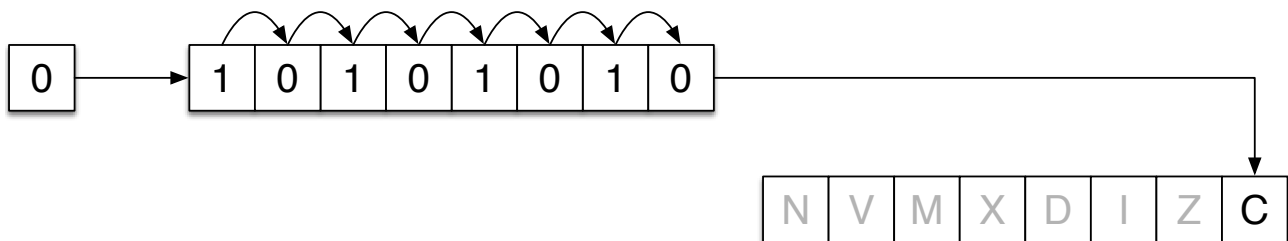
LSR

Logical Shift memory or accumulator Right

On décale à droite l'accumulateur ou le contenu d'une mémoire. En mode 8 bits (émulation 65CO2) le bit 7 est mis à zéro, tandis que le bit 0 est transféré dans la retenue. En mode 16 bits le bit 15 est mis à zéro, tandis que le bit 0 est transféré dans la retenue.

Cette opération correspond en fait à une division par 2.

Fonction:



Registre d'état du processeur	N	V	M	X	D	I	Z	C
	0	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	LSR addr	\$4E	3	6 ¹
Absolu indexé par X	LSR addr,X	\$5E	3	7 ¹
Accumulateur	LSR A	\$4A	1	2
Direct	LSR dp	\$46	2	5 ^{1,2}
Direct indexé par X	LSR dp,X	\$56	2	6 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

Exemple

SEP #20 ; A en mode 8 bits

LDA #81 ; A contient la valeur %10000001

LSR A ; A contient maintenant \$40 (%01000000)

BCS OK ; Le programme saute vers OK car le bit C vaut 1

MVN

block MoVe Next

Cette instruction déplace un bloc mémoire d'un endroit dans la mémoire à un autre. Ce déplacement est effectué en commençant par les adresses les plus basses du bloc à déplacer. Lors de cette opération, le registre d'index X contient l'adresse de départ, et le registre d'index Y contient l'adresse de destination du bloc. Ces registres sont incrémentés après chaque déplacement.

L'accumulateur contient le nombre d'octets à déplacer moins 1. Il est évidemment décrémenté après chaque transfert.

Fonction: $M \leftarrow M$, $X \leftarrow X+1$, $Y \leftarrow Y+1$, $A \leftarrow A-1$, $DBR \leftarrow N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Bloc d'origine → destination	MVN srcbk,destbk	\$54	3	*

* 7 cycles par octet déplacé

MVP

block MoVe Previous

Cette instruction déplace un bloc mémoire d'un endroit dans la mémoire à un autre. Ce déplacement est effectué en commençant par les adresses les plus hautes du bloc à déplacer. Lors de cette opération, le registre d'index X contient l'adresse de départ, et le registre d'index Y contient l'adresse de destination du bloc. Ces registres sont décrémentés après chaque déplacement.

L'accumulateur contient le nombre d'octets à déplacer moins 1. Il est décrementé après chaque transfert.

Fonction: $M \leftarrow M$, $X \leftarrow X-1$, $Y \leftarrow Y-1$, $A \leftarrow A-1$, $DBR \leftarrow N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	/	/	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Bloc d'origine → destination	MVP srcbk,destbk	\$44	3	*

* 7 cycles par octet déplacé

NOP

No Operation

Le microprocesseur n'effectue aucune opération pendant deux cycles. Seul le registre PC (Program Counter) est affecté, puisqu'il est incrémenté de façon à exécuter la prochaine opération. Cette instruction est utilisée dans la correction de programmes, ou pour synchroniser des animations.

Fonction: $M \leftarrow M$, $X \leftarrow X-1$, $Y \leftarrow Y-1$, $A \leftarrow A-1$, $DBR \leftarrow N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	NOP	\$EA	1	2

ORA

OR Accumulator with memory

Cette instruction effectue un OU logique entre le contenu de la mémoire et l'accumulateur. Cette opération est effectuée bit à bit entre l'accumulateur et les bits correspondants en mémoire. Le résultat est stocké en A.

L'opération logique OU inclusif suit la table de vérité suivante:

Première opérande	0	0	1	1
Seconde opérande	0	1	0	1
Résultat	0	1	1	1

Fonction: $A \leftarrow A \vee M$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat en émulation 65C02 (M=1)	ORA #const	\$9	2	2
Immédiat en mode natif 65816 (M=0)	ORA #const	\$9	3	3
Absolu	ORA addr	\$0D	3	4 ¹
Absolu long	ORA long	\$0F	4	5 ¹
Absolu indexé par X	ORA addr,X	\$1D	3	4 ^{1,3}
Absolu indexé par Y	ORA addr,Y	\$19	3	4 ^{1,3}
Absolu long indexé par X	ORA long,X	\$1F	4	5 ¹
Direct	ORA dp	\$05	2	3 ^{1,2}
Direct indirect	ORA (dp)	\$12	2	5 ^{1,2}
Direct indirect indexé par Y	ORA (dp),Y	\$11	2	5 ^{1,2,3}
Direct indirect long	ORA <dp>	\$07	2	6 ^{1,2}
Direct indexé indirect	ORA (dp,X)	\$01	2	6 ^{1,2}

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Direct indirect long indexé par Y	ORA <dp>,Y	\$17	2	6 ^{1,2}
Direct indexé par X	ORA dp,X	\$15	2	4 ^{1,2}
Pile relative	ORA sr,S	\$03	2	4 ¹
Pile relative indirecte indexée	ORA (sr,S),Y	\$13	2	7 ¹

- ¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)
- ² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (Direct page register) est différent de zéro

Exemple

```
SEP    #$20        ; A en mode 8 bits
```

```
LDA    #$F0        ; A contient la valeur %11110000
```

```
ORA    #$3C        ; On applique un filtre (%00111100)
```

```
                ; A contient maintenant $FC (%11111100)
```

PEA

Push Effective Absolute address on the stack

Les 16 bits (2 octets) de données suivant immédiatement PEA sont stockés au sommet de la pile, dans l'ordre suivant : d'abord le troisième octet puis le second. Le pointeur de pile est mis à jour.

Fonction: $(S) \leftarrow PC + 1, S \leftarrow S - 1$
 $(S) \leftarrow PC + 2, S \leftarrow S - 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PEA addr	\$F4	3	5

PEI

Push Effective Indirect address on the stack

Les bits du registre direct (Registre D) et l'octet de données suivant l'instruction sont additionnés, et stockés ensuite au sommet de la pile, les bits de poids forts sont placés en premier lieu, et les bits de poids faible en second. Le registre D n'est pas affecté par cette opération. Le pointeur de pile est mis à jour.

Fonction: $(S) \leftarrow (D+(PC+1)), S \leftarrow S-1$
 $(S) \leftarrow (D+(PC+1)+1), S \leftarrow S-1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile (direct indirect)	PEI (dp)	\$D4	2	6 ¹

¹ Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (Direct page register) est différent de zéro

PER

Push Effective program counter Relative address on the stack

Cette instruction additionne le contenu du registre PC (Program Counter) et les deux octets de données suivant l'instruction. La valeur obtenue est stockée au sommet de la pile, et le pointeur de pile est mis à jour.

Fonction: $(S) \leftarrow PC + NN + 2, S \leftarrow S - 2$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PER label	\$62	3	6

PHA

Push Accumulator

Le contenu de l'accumulateur est placé au sommet de la pile. Le pointeur de pile est mis à jour. L'accumulateur n'est pas modifié. Si le processeur est en mode natif, les deux octets sont placés au sommet de la pile dans l'ordre suivant : les bits de poids fort sont placés en premier lieu et les bits de poids faible en second. Le pointeur de pile est décrémenté de 2.

Fonction: $(S) \leftarrow A, S \leftarrow S - 1$ ou $S \leftarrow S - 2$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PHA	\$48	1	3 ¹

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

PHB

Push data Bank register on the stack

Le contenu du registre DBR (Data Bank Register), est placé au sommet de la pile. Le pointeur de pile est mis à jour.

RAPPEL: le registre DBR est un registre 8 bits.

Fonction: $(S) \leftarrow \text{DBR}, S \leftarrow S - 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PHB	\$8D	1	3

PHD

Push Direct page register on the stack

Cette instruction place le contenu du registre D (registre Direct) au sommet de la pile, le pointeur de pile S est mis à jour.

RAPPEL: le registre Direct est un registre 16 bits.

Fonction: $(S) \leftarrow D, S \leftarrow S - 2$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PHD	\$0B	1	4

PHK

Push program bank register on the stack

Cette instruction place le contenu du registre PBR (Program Bank Register) au sommet de la pile, le pointeur de pile est mis à jour.

RAPPEL: le registre PBR est un registre 8 bits.

Fonction: $(S) \leftarrow PBR, S \leftarrow S - 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PHK	\$4B	1	3

PHP

Push Processor status register on the stack

Cette instruction place le contenu du registre d'état du processeur P au sommet de la pile. Le registre d'état P n'est pas affecté par cette instruction. Le pointeur de pile est mis à jour.

RAPPEL: le registre d'état du processeur P est un registre 8 bits.

Fonction: $(S) \leftarrow P, S \leftarrow S - 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PHP	\$08	1	3

PHX

Push index register X on the stack

Cette instruction place le contenu du registre d'index X au sommet de la pile. Le registre d'index X n'est pas affecté par cette instruction. Le pointeur de pile (S) est mis à jour.

Si le processeur est en mode natif, les deux octets sont placés au sommet de la pile et le pointeur de pile est décrémenté de 2.

Fonction: $(S) \leftarrow X, S \leftarrow S - 1$ (en mode émulation 8 bits)
 $(S) \leftarrow X, S \leftarrow S - 2$ (en mode natif 16 bits)

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
			

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PHX	\$DA	1	3 ¹

¹ Ajouter 1 au nombre de cycles si le bit X du registre d'état du processeur vaut 0 (X et Y en 16 bits)

PHY

Push index register Y on the stack

Cette instruction place le contenu du registre d'index Y au sommet de la pile. Le registre d'index Y n'est pas affecté par cette instruction. Le pointeur de pile (S) est mis à jour.

Si le processeur est en mode natif, les deux octets sont placés au sommet de la pile et le pointeur de pile est décrémenté de 2.

Fonction: $(S) \leftarrow Y, S \leftarrow S - 1$ (en mode émulation 8 bits)
 $(S) \leftarrow Y, S \leftarrow S - 2$ (en mode natif 16 bits)

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PHY	\$5A	1	3 ¹

¹ Ajouter 1 au nombre de cycles si le bit X du registre d'état du processeur vaut 0 (X et Y en 16 bits)

PLA

PuLL Accumulator

Cette instruction place le contenu du sommet de la pile dans l'accumulateur. Le pointeur de pile (S) est mis à jour.

Si le 65C816 est en mode natif ce sont les deux octets (16 bits) qui sont transférés de la pile vers l'accumulateur. Le pointeur de pile est alors incrémenté de 2.

Fonction: $A \leftarrow (S)$, $S \leftarrow S + 1$ (en mode émulation 8 bits)
 $A \leftarrow (S)$, $S \leftarrow S + 2$ (en mode natif 16 bits)

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PLA	\$68	1	4 ¹

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

PLB

Pull data Bank register

Cette instruction transfère le contenu du sommet de la pile, au registre DBR (Data Bank Register). Le pointeur est mis à jour.

RAPPEL: le registre DBR est un registre 8 bits.

Fonction: $S \leftarrow S + 1, DBR \leftarrow (S)$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PLB	\$AB	1	4

PLD

PuLL Direct page register

Cette instruction transfère le contenu du sommet de la pile, au registre D (registre Direct).
Le pointeur est mis à jour.

Fonction: $S \leftarrow S + 2, D \leftarrow (S)$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PLD	\$2B	1	5

PLP

Pull status register P

Cette instruction transfère le contenu du sommet de la pile, au registre d'état du processeur (Registre P). Le pointeur est mis à jour.

Fonction: $S \leftarrow S + 1, P \leftarrow (S)$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	/	/	/	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PLP	\$28	1	4

PLX

Pull index register X

Cette instruction transfère le contenu du sommet de la pile au registre d'index X. Le pointeur de pile (S) est mis à jour.

Si le 65C816 est en mode natif ce sont les deux octets (16 bits) qui sont transférés de la pile vers le registre d'index X. Le pointeur de pile est alors incrémenté de 2.

Fonction: $S \leftarrow S + 1, X \leftarrow (S)$ (en mode émulation 8 bits)
 $S \leftarrow S + 2, X \leftarrow (S)$ (en mode natif 16 bits)

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PLX	\$FA	1	4 ¹

¹ Ajouter 1 au nombre de cycles si le bit X du registre d'état du processeur vaut 0 (X et Y en 16 bits)

PLY

Pull index register Y

Cette instruction transfère le contenu du sommet de la pile au registre d'index Y. Le pointeur de pile (S) est mis à jour.

Si le 65C816 est en mode natif ce sont les deux octets (16 bits) qui sont transférés de la pile vers le registre d'index Y. Le pointeur de pile est alors incrémenté de 2.

Fonction: $S \leftarrow S + 1, Y \leftarrow (S)$ (en mode émulation 8 bits)
 $S \leftarrow S + 2, Y \leftarrow (S)$ (en mode natif 16 bits)

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	/	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	PLY	\$7A	1	4 ¹

¹ Ajouter 1 au nombre de cycles si le bit X du registre d'état du processeur vaut 0 (X et Y en 16 bits)

REP

REset Processor status bits

Cette instruction effectue un ET bits à bits entre le registre d'état du processeur et le complément de l'octet suivant immédiatement l'instruction.

Fonction: $P \leftarrow P \wedge N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
Émulation 65C02	/	/	.	.	/	/	/	/
Mode natif 65816	/	/	/	/	/	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat	REP #const	\$C2	2	3

ROL

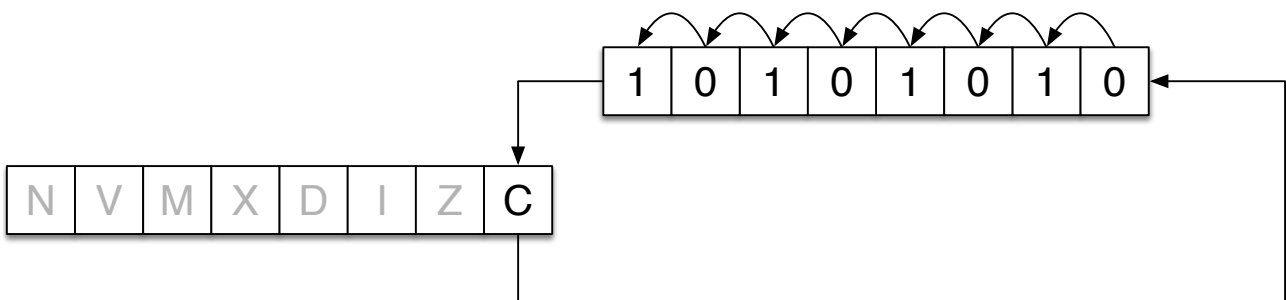
ROtate memory or accumulator Left

Cette instruction décale tous les bits de l'accumulateur ou du contenu de l'adresse mémoire spécifiée par l'opérande de 1 vers la gauche.

En mode 16 bits le bit 0 est mis à la place du bit 1, le bit de report est mis à la place du bit 0, le bit 15 est mis à la place du bit de report, le bit 14 est mis à la place du bit 15 et ainsi de suite.

En mode 8 bit le bit 0 est mis à la place du bit 1, le bit de report est mis a la place du bit 0, le bit 7 est mis à la place du bit de report et ainsi de suite.

Fonction:



Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	ROL addr	\$2E	3	6 ¹
Absolu indexé par X	ROL addr,X	\$3E	3	7 ¹
Accumulateur	ROL A	\$2A	1	2
Direct	ROL dp	\$26	2	5 ^{1,2}
Direct indexé par X	ROL dp,X	\$36	2	6 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

Exemple

SEP #20 ; A en mode 8 bits

LDA #81 ; A contient la valeur %1000001

SEC ; On met le bit de retenue à 1

ROL A ; A contient maintenant \$03 (%0000011)

BCS OK ; Le programme saute vers OK car le bit C vaut 1

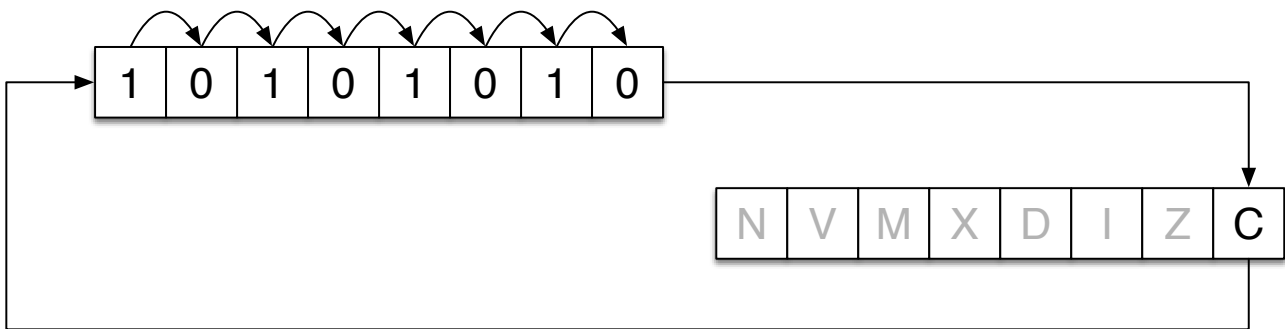
ROR

ROtate memory or accumulator Right

Cette instruction décale tous les bits de l'accumulateur ou du contenu de l'adresse mémoire spécifiée par l'opérande d'un vers la droite. En mode 16 bits le bit 1 est mis à la place du bit 0, le bit 0 est mis à la place du bit de report, le bit de report est mis à la place du bit 15, le bit 15 est mis à la place du bit 14 et ainsi de suite.

En mode 8 bit le bit 1 est mis à la place du bit 0, le bit 0 est mis à la place du bit de report, le bit de report est mis à la place du bit 7 et ainsi de suite.

Fonction:



Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	ROR addr	\$6E	3	6 ¹
Absolu indexé par X	ROR addr,X	\$7E	3	7 ¹
Accumulateur	ROR A	\$6A	1	2
Direct	ROR dp	\$66	2	5 ^{1,2}
Direct indexé par X	ROR dp,X	\$76	2	6 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

Exemple

```
SEP  #$20      ; A en mode 8 bits
```

```
LDA  #$80      ; A contient la valeur %10000000
```

```
CLC              ; On met le bit de retenue à 0
```

```
ROR  A         ; A contient maintenant $40 (%01000000)
```

```
BCS  OK       ; Le programme ne saute pas vers OK car le bit C
```

```
              ; vaut 0
```

RTI

ReTurn from Interrupt

Cette instruction charge le registre d'état du processeur, le registre compteur ordinal, et le registre PBR (Program Bank Register) à partir de la pile.

Fonction: $S \leftarrow S + 1, P \leftarrow (S)$
 $S \leftarrow S + 1, PCL \leftarrow (S)$
 $S \leftarrow S + 1, PCH \leftarrow (S)$
 $S \leftarrow S + 1, PBR \leftarrow (S)$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
Émulation 65C02	/	/	.	.	/	/	/	/
Mode natif 65816	/	/	/	/	/	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	RTI	\$40	1	6 ¹

¹ Ajouter 1 au nombre de cycles lorsque l'instruction s'exécute en mode natif 65C816 (E=0)

RTL

ReTurn from subroutine Long

Cette instruction équivaut à un RETURN en Basic. Cette instruction restitue au registre PC (program counter) l'adresse incrémentée d'un, contenue dans la pile. Cette instruction est utilisée avec JSL. L'instruction RTL permet un retour dans n'importe quel banc mémoire.

Fonction: $S \leftarrow S + 1, PCL \leftarrow (S)$
 $S \leftarrow S + 1, PCH \leftarrow (S)$
 $S \leftarrow S + 1, PBR \leftarrow (S)$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	RTL	\$6B	1	6

RTS

ReTurn from Subroutine

Cette instruction équivaut à un RETURN en Basic. Cette instruction restitue au registre PC (program counter) l'adresse incrémentée d'un, contenue dans la pile. Cette instruction est utilisée avec JSR. L'instruction RTS ne permet un retour que dans le banc mémoire courant.

Fonction: $S \leftarrow S + 1, PCL \leftarrow (S)$
 $S \leftarrow S + 1, PCH \leftarrow (S)$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile	RTL	\$60	1	6

SBC

Subtract with Carry

L'accumulateur est soustrait avec le complément du bit de report au contenu d'une adresse mémoire ou d'une valeur spécifiée par l'opérande. Le résultat est stocké dans l'accumulateur. Afin que le bit de report n'ait pas d'influence sur la soustraction, il faut utiliser l'instruction SEC (SEt Carry flag) avant d'utiliser SBC.

Fonction: $A \leftarrow A - M - C$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat en émulation 65C02 (M=1)	SBC #const	\$E9	2	2
Immédiat en mode natif 65816 (M=0)	SBC #const	\$E9	3	3
Absolu	SBC addr	\$ED	3	4 ¹
Absolu long	SBC long	\$EF	4	5 ¹
Absolu long indexé par X	SBC long,X	\$FF	4	5 ¹
Absolu indexé par X	SBC addr,X	\$FD	3	4 ^{1,3}
Absolu indexé par Y	SBC addr,Y	\$F9	3	4 ^{1,3}
Direct	SBC dp	\$E5	2	3 ^{1,2}
Direct indirect	SBC (dp)	\$F2	2	5 ^{1,2}
Direct indirect indexé par Y	SBC (dp),Y	\$F1	2	5 ^{1,2,3}
Direct indirect long	SBC <dp>	\$E7	2	6 ^{1,2}
Direct indirect long indexé par Y	SBC <dp>,Y	\$F7	2	6 ^{1,2}
Direct indexé indirect	SBC (dp,X)	\$E1	2	6 ^{1,2}
Direct indexé par X	SBC dp,X	\$F5	2	4 ^{1,2}
Pile relative	SBC sr,S	\$E3	2	4 ¹
Pile relative indirecte indexée	SBC (sr,S),Y	\$F3	2	7 ¹

- 1 Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)
- 2 Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro
- 3 Ajouter 1 au nombre de cycles si l'addition du registre d'index fait changer de page mémoire

SEC

SEt Carry flag

Le bit C (Carry flag ou bit de retenue) du registre d'état du processeur est mis à 1. Cette Instruction est souvent utilisée avant SBC ou XCE.

Fonction: $C \leftarrow 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	1
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	SEC	\$38	1	2

SED

SEt Decimal flag

Le bit D (Decimal flag) du registre d'état du processeur est mis à 1., ce qui fait passer le 65C816 en mode décimal codé binaire (BCD - Binary coded Decimal).

Fonction: $D \leftarrow 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	1	.	.	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	SED	\$F8	1	2

SEI

SEt Interrupt disable flag

Le bit I (Interrupt flag) du registre d'état du processeur est mis à 1, ce qui le désactive. Cette instruction a pour but de masquer les interruptions IRQ (Interrupt ReQuest). Cette instruction ne met pas hors d'usage les interruptions non masquables (MNI - non masquable interrupt ou RESET).

Fonction: $I \leftarrow 1$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	1	.	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	SEI	\$78	1	2

SEP

SEt Processor status bits

Cette instruction permet de modifier n'importe quel(s) bit(s) dans le registre d'état du processeur (P) avec le bit correspondant de l'opérande de l'instruction.

Fonction: $P \leftarrow P \vee N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
Émulation 65C02	/	/	.	.	/	/	/	/
Mode natif 65816	/	/	/	/	/	/	/	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Immédiat	SEP #const	\$E2	2	3

STA

STore Accumulator

Cette instruction stocke la valeur contenue dans l'accumulateur en mémoire. En fonction de la valeur du bit e (mode émulation) il existe deux possibilités:

- L'accumulateur est en mode 8 bits (e=1): La valeur dans l'accumulateur est mise à l'adresse mémoire spécifiée par l'opérande.
- L'accumulateur est en mode 16 bits (e=0): Les 8 bits de poids faible de l'accumulateur sont alors mis à l'adresse mémoire spécifiée par l'opérande, et les 8 bits de poids forts sont mis à l'adresse mémoire spécifiée par l'opérande plus une.

Fonction: $P \leftarrow P \vee N$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	STA addr	\$8D	3	4
Absolu long	STA long	\$8F	4	5
Absolu long indexé par X	STA long,X	\$9F	4	5 ¹
Absolu indexé par X	STA addr,X	\$9D	3	5 ¹
Absolu indexé par Y	STA addr,Y	\$99	3	5 ¹
Direct	STA dp	\$85	2	3 ^{1,2}
Direct indirect	STA (dp)	\$92	2	5 ^{1,2}
Direct indirect indexé par Y	STA (dp),Y	\$91	2	6 ^{1,2}
Direct indirect long	STA <dp>	\$87	2	6 ^{1,2}
Direct indirect long indexé par Y	STA <dp>,Y	\$97	2	6 ^{1,2}
Direct indexé indirect	STA (dp,X)	\$81	2	6 ^{1,2}
Direct indexé par X	STA dp,X	\$95	2	4 ^{1,2}
Pile relative	STA sr,S	\$83	2	4 ¹

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Pile relative indirecte indexée	STA (sr,S),Y	\$93	2	7 ¹

- ¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)
- ² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

STP

STop the Processor

Cette instruction arrête le microprocesseur. RESET est la seule façon pour reprendre les opérations.

Fonction: Arrêt du microprocesseur

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	STP	\$DB	1	3

STX

STore X register

Cette instruction stocke en mémoire la valeur contenue dans le registre d'index X. En fonction de la valeur du bit e (mode émulation) il existe deux possibilités:

- L'accumulateur est en mode 8 bits (e=1): La valeur dans le registre d'index X est mise à l'adresse mémoire spécifiée par l'opérande.
- L'accumulateur est en mode 16 bits (e=0): Les 8 bits de poids faible du registre d'index X sont alors mis à l'adresse mémoire spécifiée par l'opérande, et les 8 bits de poids forts sont mis à l'adresse mémoire spécifiée par l'opérande plus une.

Fonction: $M \leftarrow X$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	STX addr	\$8E	3	4 ¹
Direct	STX dp	\$86	2	3 ^{1,2}
Direct indexé par Y	STX addr,Y	\$96	2	4 ¹

¹ Ajouter 1 au nombre de cycles si le bit X du registre d'état du processeur vaut 0 (X et Y en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

STY

STore Y register

Cette instruction stocke en mémoire la valeur contenue dans le registre d'index Y. En fonction de la valeur du bit e (mode émulation) il existe deux possibilités:

- L'accumulateur est en mode 8 bits (e=1): La valeur dans le registre d'index Y est mise à l'adresse mémoire spécifiée par l'opérande.
- L'accumulateur est en mode 16 bits (e=0): Les 8 bits de poids faible du registre d'index Y sont alors mis à l'adresse mémoire spécifiée par l'opérande, et les 8 bits de poids forts sont mis à l'adresse mémoire spécifiée par l'opérande plus une.

Fonction: $M \leftarrow Y$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	STY addr	\$8C	3	4 ¹
Direct	STY dp	\$84	2	3 ^{1,2}
Direct indexé par X	STY addr,X	\$94	2	4 ¹

¹ Ajouter 1 au nombre de cycles si le bit X du registre d'état du processeur vaut 0 (X et Y en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

STZ

STore Zero

Cette instruction stocke la valeur zéro, à l'adresse mémoire spécifiée par l'opérande si le microprocesseur est en mode 8 bits.

Si le microprocesseur est en mode 16 bits (M=0), cette instruction stocke la valeur zéro à l'adresse mémoire spécifiée par l'opérande et l'adresse mémoire spécifiée par l'opérande plus une.

Fonction: $M \leftarrow 0$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	STZ addr	\$9C	3	4 ¹
Absolu indexé par X	STZ ,X	\$9E	3	5 ¹
Direct	STZ dp	\$64	2	3 ^{1,2}
Direct indexé par X	STZ addr,X	\$74	2	4 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

TAX

Transfer A to X

Cette instruction transfère la valeur contenue dans l'accumulateur dans le registre d'index X. Le contenu de l'accumulateur ne varie pas. Il y a 4 possibilités puisque l'accumulateur et le registre d'index X peuvent être de tailles différentes:

- 1) Registre d'index X en 8 bits et Accumulateur en 8 bits: Les 8 bits de l'accumulateur sont transférés dans le registre d'index X
- 2) Registre d'index X en 16 bits et Accumulateur en 8 bits (X=0 ; M=1): Les 8 bits du registre A de l'accumulateur sont transférés dans les bits de poids faible du registre d'index X. Les 8 bits cachés du registre B de l'accumulateur deviennent les bits de poids fort du registre d'index.
- 3) Registre d'index X en 8 bits et Accumulateur en 16 bits (X=1 ; M=0): Seuls les 8 bits de poids faible (A) de l'accumulateur sont transférés dans le registre d'Index.
- 4) Registre d'index X en 16 bits et Accumulateur en 16 bits (X=0 ; M=0): Les 16 bits de l'accumulateur (A+B) sont transférés dans le registre d'index X.

Fonction: $X \leftarrow A$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TAX	\$AA	1	2

TAY

Transfer A to Y

Cette instruction transfère la valeur contenue dans l'accumulateur dans le registre d'index Y. Le contenu de l'accumulateur ne varie pas. Il y a 4 possibilités puisque l'accumulateur et le registre d'index Y peuvent être de tailles différentes:

- 1) Registre d'index Y en 8 bits et Accumulateur en 8 bits: Les 8 bits de l'accumulateur sont transférés dans le registre d'index Y
- 2) Registre d'index Y en 16 bits et Accumulateur en 8 bits (X=0 ; M=1): Les 8 bits du registre A de l'accumulateur sont transférés dans les bits de poids faible du registre d'index Y. Les 8 bits cachés du registre B de l'accumulateur deviennent les bits de poids fort du registre d'index.
- 3) Registre d'index Y en 8 bits et Accumulateur en 16 bits (X=1 ; M=0): Seuls les 8 bits de poids faible (A) de l'accumulateur sont transférés dans le registre d'Index.
- 4) Registre d'index Y en 16 bits et Accumulateur en 16 bits (X=0 ; M=0): Les 16 bits de l'accumulateur (A+B) sont transférés dans le registre d'index Y.

Fonction: $Y \leftarrow A$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	/	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TAY	\$A8	1	2

TCD

Transfer C to D

Cette instruction transfère les 16 bits de l'accumulateur (A+B) dans le registre direct D. Le contenu de l'accumulateur ne varie pas.

Si l'accumulateur est en mode 8 bits (microprocesseur, les bits cachés de l'accumulateur (B) sont quand même transférés dans le registre direct D.

Fonction: $D \leftarrow C$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TCD	\$5B	1	2

TCS

Transfer C to S

Cette instruction transfère les 16 bits de l'accumulateur dans le registre du pointeur de pile S. Le contenu de l'accumulateur ne varie pas. Cette instruction est la seule avec TCS qui modifie le pointeur de pile.

Fonction: $S \leftarrow C$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TCS	\$1B	1	2

TDC

Transfer D to C

Cette instruction transfère les 16 bits du registre direct D dans l'accumulateur même si celui ci est en mode 8 bits (les 8 bits sont transférés dans la partie B de l'accumulateur qui reste cependant occulte). Le contenu du registre direct D ne varie pas.

Fonction: $C \leftarrow D$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TDC	\$7B	1	2

TRB

Test and Reset memory Bits with Accumulator

Cette instruction n'agit que sur le banc 0 et opère à partir d'un masque. Elle effectue un test bit à bit entre l'accumulateur et l'adresse spécifiée.

Le résultat de ce test est stocké dans l'opérande en mémoire. Tout bit étant à 1 dans l'accumulateur sera mis à 0 dans l'opérande en mémoire.

Fonction: $M \leftarrow M \wedge A$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	TRB addr	\$1C	3	6 ¹
Direct	TRB dp	\$14	2	5 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

TSB

Test and Reset memory Bits with Accumulator

Cette instruction n'agit que sur le banc 0 et opère à partir d'un masque. Elle effectue un test bit à bit entre l'accumulateur et l'adresse spécifiée.

Le résultat de ce test est stocké dans l'opérande en mémoire. Tout bit étant à 1 dans l'accumulateur sera mis à 1 dans l'opérande en mémoire.

Fonction: $M \leftarrow M \vee A$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Absolu	TSB addr	\$0C	3	6 ¹
Direct	TSB dp	\$04	2	5 ^{1,2}

¹ Ajouter 1 au nombre de cycles si le bit M du registre d'état du processeur vaut 0 (A en 16 bits)

² Ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro

TSC

Transfer S to C

Cette instruction transfère les 16 bits du pointeur de pile dans l'accumulateur. Le contenu du pointeur de pile S ne varie pas.

Fonction: $C \leftarrow S$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TSC	\$3B	1	2

TSX

Transfer S to X

Le contenu du registre d'index X est transféré dans le pointeur de pile S (Stack pointer).
Le contenu de X ne varie pas.

Il y a 2 possibilités:

- 1) Le registre d'index X est en 8 bits ($X=1$) ; seuls les bits de poids faible du pointeur de pile S sont transférés dans le registre d'index X.
- 2) Le registre d'index X est en 16 bits ($X=0$) ; la totalité des 16 bits du pointeur de pile sont transférés dans le registre d'index X.

Fonction: $X \leftarrow S$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TSX	\$BA	1	2

TXA

Transfer X to A

Cette instruction transfère la valeur contenue dans le registre d'index X dans l'accumulateur. Le contenu du registre d'index X lui ne change pas.

Il y a 4 possibilités puisque l'accumulateur et le registre d'index X peuvent être de tailles différentes:

- 1) Registre d'index X en 8 bits et Accumulateur en 8 bits: Les 8 bits du registre d'index X sont transférés dans l'accumulateur.
- 2) Registre d'index X en 16 bits et Accumulateur en 8 bits (X=0 ; M=1): Seuls les 8 bits de poids faible du registre d'index X sont transférés. Seule la partie A de l'accumulateur est affectée par cette opération. La partie B de l'accumulateur ne change pas.
- 3) Registre d'index X en 8 bits et Accumulateur en 16 bits (X=1 ; M=0): Les 8 bits du registre d'index X sont transférés dans la partie A de l'accumulateur. La partie B de l'accumulateur est mise à zéro.
- 4) Registre d'index X en 16 bits et Accumulateur en 16 bits (X=0 ; M=0): Les 16 bits du registre X sont transférés dans l'accumulateur.

Fonction: $A \leftarrow X$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TXA	\$8A	1	2

TXS

Transfer X to S

Le contenu du registre d'index X est transféré dans le pointeur de pile S(Stack pointer). Le contenu de X ne varie pas. Cette instruction est la seule avec TCS qui modifie le pointeur de pile.

Il y a 3 possibilités :

- 1) En mode émulation (E=1) : Le pointeur de pile a seulement 8 bits, et le registre X également. Les 8 bits contenus dans le registre d'index X sont transférés.
- 2) Le registre X est en mode 8 bits mais le 65C816 est en mode natif (X=1) : Les 8 bits du registre X sont transférés dans les bits de poids faible du pointeur de pile. Les bits de poids fort du pointeur de pile sont mis à zéro.
- 3) Le registre X est en mode 16 bits et le 65C816 est en mode natif (X=0)

Tous les 16 bits du registre d'index X sont transférés dans le pointeur de pile.

Fonction: $S \leftarrow X$

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TXS	\$9A	1	2

TXY

Transfer X to Y

Cette instruction transfère les bits du registre d'index X dans le registre d'index Y. Le registre d'index X lui ne change pas. Ces deux opérations peuvent être effectuées dans les deux modes 8 bits (X=1) et 16 bits (X=0). Comme les deux registres ne sont jamais de tailles différentes, cette opération ne pose aucun problème.

Fonction: $Y \leftarrow X$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	.	/	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TXY	\$9B	1	2

TYA

Transfer Y to A

Cette instruction transfère le registre d'index Y dans l'accumulateur. Le registre d'index Y lui ne change pas.

Il y a 4 possibilités puisque l'accumulateur et le registre d'index Y peuvent être de tailles différentes:

- 1) Registre d'index Y en 8 bits et Accumulateur en 8 bits: Les 8 bits du registre d'index Y sont transférés dans l'accumulateur.
- 2) Registre d'index Y en 16 bits et Accumulateur en 8 bits (X=0 ; M=1): Seuls les 8 bits de poids faible du registre d'index Y sont transférés. Seule la partie A de l'accumulateur est affectée par cette opération. La partie B de l'accumulateur ne change pas.
- 3) Registre d'index Y en 8 bits et Accumulateur en 16 bits (X=1 ; M=0): Les 8 bits du registre d'index Y sont transférés dans la partie A de l'accumulateur. La partie B de l'accumulateur est mise à zéro.
- 4) Registre d'index Y en 16 bits et Accumulateur en 16 bits (X=0 ; M=0): Les 16 bits du registre Y sont transférés dans l'accumulateur.

Fonction: $Y \leftarrow A$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TYA	\$98	1	2

TYX

Transfer Y to X

Cette instruction transfère les bits du registre d'index Y dans le registre d'index X. Le registre d'index Y lui ne change pas. Ces deux opérations peuvent être effectuées dans les deux modes 8 bits (X=1) et 16 bits (X=0). Comme les deux registres ne sont jamais de tailles différentes, cette opération ne pose aucun problème.

Fonction: $X \leftarrow Y$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	TYX	\$BB	1	2

WAI

Wait for Interrupt

Attente d'une interruption. Le microprocesseur arrête toute opération jusqu'à ce qu'intervienne une interruption hardware externe (NMI - ABORT - RESET - IRQ).

Fonction: Ready \leftarrow 0

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	WAI	\$CB	1	3

WDM

William D Mensch, Jr.

Cette instruction a été réservée par Mr. William D. Mensch, Jr, le concepteur du 65C816 chez Western Design Center, afin d'accroître plus tard les possibilités du 65C816.

Fonction: /

Registre d'état du processeur	N	V	M	X	D	I	Z	C

Modifie la mémoire ou les registres	A	X	Y	Mémoire				
				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	WDM	\$42	?	?

XBA

eXchange B and A

Comme nous l'avons vu dans la partie sur l'accumulateur, ce dernier est composé de 2 registres A et B. Cette instruction permet d'échanger les bits de A avec ceux de B.

Fonction: $A \leftrightarrow B$

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	/	.	.	.				

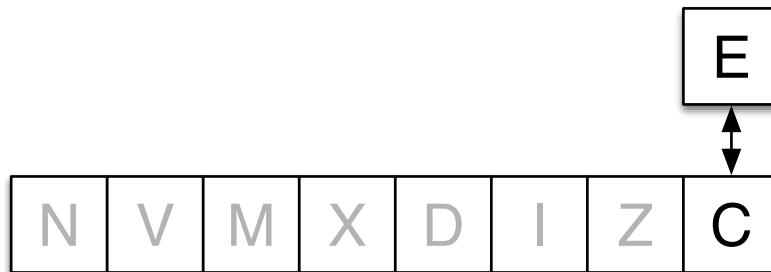
Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	XBA	\$EB	1	3

XCE

eXchange Carry and E bits

Cette instruction sert à échanger dans le registre d'état du processeur le bit de la retenue (carry) et le bit d'émulation E. En programmation on se servira de cette instruction pour passer en mode émulation 65C02 ou en mode 65C816, suivant la valeur du bit C. Il est important de noter que le bit E ne fait pas part du registre P et que la seule façon de modifier ou de connaître la valeur de ce bit est en passant par l'instruction XCE.

Fonction: $X \leftrightarrow E$



Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	/	/	.	.	.	/
Modifie la mémoire ou les registres	A	X	Y	Mémoire				
	.	/	/	.				

Modes d'adressage	Syntaxe	Code	Octets	Cycles
Implicite	XCE	\$FB	1	2

Exemple

CLC

XCE ; Passe en mode 65816

SEC ; Retourne en mode 65C02

XCE

1.2 Le tableau suivant vous donnera la liste des instructions par code hexadécimal croissant.

Code Hexa	Mnémonique	Mode d'adressage
\$00	BRK	Pile
\$01	ORA	Direct indirect indexé par X
\$02	COP	Pile
\$03	ORA	Relatif à la pile
\$04	TSB	Direct
\$05	ORA	Direct
\$06	ASL	Direct
\$07	ORA	Direct indirect long
\$08	PHP	Pile
\$09	ORA	Inmédiate
\$0A	ASL	Accumulateur
\$0B	PHD	Pile
\$0C	TSB	Absolu
\$0D	ORA	Absolu
\$0E	ASL	Absolu
\$0F	ORA	Absolu long
\$10	BPL	Relatif
\$11	ORA	Direct indirect indexé par Y
\$12	ORA	Direct indirect
\$13	ORA	Relatif à la pile indirect indexé par Y
\$14	TRB	Direct
\$15	ORA	Direct indexé par X
\$16	ASL	Direct indexé par X
\$17	ORA	Direct indirect long indexé par Y
\$18	CLC	Implicite
\$19	ORA	Absolu indexé par Y
\$1A	INC	Accumulateur
\$1B	TCS	Implicite
\$1C	TRB	Absolu

Code Hexa	Mnémonique	Mode d'adressage
\$1D	ORA	Absolu indexé par X
\$1E	ASL	Absolu indexé par X
\$1F	ORA	Absolu long indexé par X
\$20	JSR	Absolu
\$21	AND	Direct indexé indirect
\$22	JSR	Absolu long
\$23	AND	Pile relative
\$24	BIT	Direct
\$25	AND	Direct
\$26	ROL	Direct
\$27	AND	Direct indirect long
\$28	PLP	Pile
\$29	AND	Immédiat
\$2A	ROL	Accumulateur
\$2B	PLD	Pile
\$2C	BIT	Absolu
\$2D	AND	Absolu
\$2E	ROL	Absolu
\$2F	AND	Absolu long
\$30	BMI	Relatif
\$31	AND	Direct indirect indexé par Y
\$32	AND	Direct indirect
\$33	AND	Pile relative indirect indexé
\$34	BIT	Direct indexé par X
\$35	AND	Direct indexé par X
\$36	ROL	Direct indexé par X
\$37	AND	Direct indirect long indexé par X
\$38	SEC	Implicite
\$39	AND	Absolu indexé par Y
\$3A	DEC	Accumulateur
\$3B	TSC	Implicite
\$3C	BIT	Absolu indexé par X

Code Hexa	Mnémonique	Mode d'adressage
\$3D	AND	Absolu indexé par X
\$3E	ROL	Absolu indexé par X
\$3F	AND	Absolu long indexé par X
\$40	RTI	Pile
\$41	EOR	Direct indexé indirect
\$42	WDM	Implicite
\$43	EOR	Pile relative
\$44	MVP	Bloc de départ → Bloc de destination (source, destination)
\$45	EOR	Direct
\$46	LSR	Direct
\$47	EOR	Direct indirect long
\$48	PHA	Pile
\$49	EOR	Immédiat
\$4A	LSR	Accumulateur
\$4B	PHK	Pile
\$4C	JMP	Absolu
\$4D	EOR	Absolu
\$4E	LSR	Absolu
\$4F	EOR	Absolu long
\$50	BVC	Relatif
\$51	EOR	Direct indirect indexé par Y
\$52	EOR	Direct indirect
\$53	EOR	Pile relative indirecte indexée
\$54	MVN	Bloc de départ → Bloc de destination (source, destination)
\$55	EOR	Direct indexé par X
\$56	LSR	Direct indexé par X
\$57	EOR	Direct indirect long indexé par Y
\$58	CLI	Implicite
\$59	EOR	Absolu indexé par Y
\$5A	PHY	Pile
\$5B	TCD	Implicite
\$5C	JMP	Absolu long

Code Hexa	Mnémonique	Mode d'adressage
\$5D	EOR	Absolu indexé par X
\$5E	LSR	Absolu indexé par X
\$5F	EOR	Absolu long indexé par X
\$60	RTS	Pile
\$61	ADC	Direct indexé indirect
\$62	PER	Pile
\$63	ADC	Pile relative
\$64	STZ	Direct
\$65	ADC	Direct
\$66	ROR	Direct
\$67	ADC	Direct indirect long
\$68	PLA	Pile
\$69	ADC	Immédiat
\$6A	ROR	Accumulateur
\$6B	RTL	Pile
\$6C	JMP	Absolu indirect
\$6D	ADC	Absolu
\$6E	ROR	Absolu
\$6F	ADC	Absolu long
\$70	BVS	Pile relative
\$71	ADC	Direct indirect indexé par Y
\$72	ADC	Direct indirect
\$73	ADC	Pile relative indirecte indexée
\$74	STZ	Direct indexé par X
\$75	ADC	Direct indexé par X
\$76	ROR	Direct indexé par X
\$77	ADC	Direct indirect long indexé par X
\$78	SEI	Implicite
\$79	ADC	Absolu indexé par Y
\$7A	PLY	Pile
\$7B	TDC	Implicite
\$7C	JMP	Absolu indexé indirect

Code Hexa	Mnémonique	Mode d'adressage
\$7D	ADC	Absolu indexé par X
\$7E	ROR	Absolu indexé par X
\$7F	ADC	Absolu long indexé par X
\$80	BRA	Pile relative
\$81	STA	Direct indexé indirect
\$82	BRL	Relatif long
\$83	STA	Pile
\$84	STY	Direct
\$85	STA	Direct
\$86	STX	Direct
\$87	STA	Direct indirect long
\$88	DEY	Implicite
\$89	BIT	Immédiat
\$8A	TXA	Implicite
\$8B	PHB	Pile
\$8C	STY	Absolu
\$8D	STA	Absolu
\$8E	STX	Absolu
\$8F	STA	Absolu long
\$90	BCC	Pile relative
\$91	STA	Direct indirect indexé par Y
\$92	STA	Direct indirect
\$93	STA	Pile relative indirecte indexée
\$94	STY	Direct indexé par X
\$95	STA	Direct indexé par X
\$96	STX	Direct indexé par Y
\$97	STA	Direct indirect long indexé par Y
\$98	TYA	Implicite
\$99	STA	Absolu indexé par Y
\$9A	TXS	Implicite
\$9B	TXY	Implicite
\$9C	STZ	Absolu

Code Hexa	Mnémonique	Mode d'adressage
\$9D	STA	Absolu indexé par X
\$9E	STZ	Absolu indexé par X
\$9F	STA	Absolu long indexé par X
\$A0	LDY	Immédiat
\$A1	LDA	Direct indexé indirect
\$A2	LDX	Immédiat
\$A3	LDA	Pile relative
\$A4	LDY	Direct
\$A5	LDA	Direct
\$A6	LDX	Direct
\$A7	LDA	Direct indirect long
\$A8	TAY	Implicite
\$A9	LDA	Immédiat
\$AA	TAX	Implicite
\$AB	PLB	Pile
\$AC	LDY	Absolu
\$AD	LDA	Absolu
\$AE	LDX	Absolu
\$AF	LDA	Absolu long
\$B0	BCS	Pile relative
\$B1	LDA	Direct indirect indexé par Y
\$B2	LDA	Direct indirect
\$B3	LDA	Pile relative indirecte indexée
\$B4	LDY	Direct indexé par X
\$B5	LDA	Direct indexé par X
\$B6	LDX	Direct indexé par Y
\$B7	LDA	Direct indirect long indexé par Y
\$B8	CLV	Implicite
\$B9	LDA	Absolu indexé par Y
\$BA	TSX	Implicite
\$BB	TYX	Implicite
\$BC	LDY	Absolu indexé par X

Code Hexa	Mnémonique	Mode d'adressage
\$BD	LDA	Absolu indexé par X
\$BE	LDX	Absolu indexé par Y
\$BF	LDA	Absolu long indexé par X
\$C0	CPY	Immédiat
\$C1	CMP	Direct indexé indirect
\$C2	REP	Immédiat
\$C3	CMP	Pile relative
\$C4	CPY	Direct
\$C5	CMP	Direct
\$C6	DEC	Direct
\$C7	CMP	Direct indirect long
\$C8	INY	Implicite
\$C9	CMP	Immédiat
\$CA	DEX	Implicite
\$CB	WAI	Implicite
\$CC	CPY	Absolu
\$CD	CMP	Absolu
\$CE	DEC	Absolu
\$CF	CMP	Absolu long
\$D0	BNE	Pile relative
\$D1	CMP	Direct indirect indexé par Y
\$D2	CMP	Direct indirect
\$D3	CMP	Pile relative indirecte indexée
\$D4	PEI	Pile
\$D5	CMP	Direct indexé par X
\$D6	DEC	Direct indexé par X
\$D7	CMP	Direct indirect long indexé par Y
\$D8	CLD	Implicite
\$D9	CMP	Absolu indexé par Y
\$DA	PHX	Pile
\$DB	STP	Implicite
\$DC	JMP	Absolu indirect long

Code Hexa	Mnémonique	Mode d'adressage
\$DD	CMP	Absolu indexé par X
\$DE	DEC	Absolu indexé par X
\$DF	CMP	Absolu long indexé par X
\$E0	CPX	Immédiat
\$E1	SBC	Direct indexé indirect
\$E2	SEP	Immédiat
\$E3	SBC	Pile relative
\$E4	CPX	Direct
\$E5	SBC	Direct
\$E6	INC	Direct
\$E7	SBC	Direct indirect long
\$E8	INX	Implicite
\$E9	SBC	Immédiat
\$EA	NOP	Implicite
\$EB	XBA	Implicite
\$EC	CPX	Absolu
\$ED	SBC	Absolu
\$EE	INC	Absolu
\$EF	SBC	Absolu long
\$F0	BEQ	Pile relative
\$F1	SBC	Direct indirect indexé par Y
\$F2	SBC	Direct indirect
\$F3	SBC	Pile relative indirecte indexée
\$F4	PEA	Pile
\$F5	SBC	Direct indexé par X
\$F6	INC	Direct indexé par X
\$F7	SBC	Direct indirect long indexé par Y
\$F8	SED	Implicite
\$F9	SBC	Absolu indexé par Y
\$FA	PLX	Pile
\$FB	XCE	Implicite
\$FC	JSR	Absolu indexé indirect

Code Hexa	Mnémonique	Mode d'adressage
\$FD	SBC	Absolu indexé par X
\$FE	INC	Absolu indexé par X
\$FF	SBC	Absolu long indexé par X

1.3 Les différents modes d'adressage.

Les différents modes d'adressage caractérisent les possibilités d'accès du microprocesseur à l'information en mémoire.

1.3.1 Adressage implicite

Cet adressage est aussi dit «adressage inhérent» ou «adressage de registre». Les instructions implicites n'ont pas d'opérande et s'assemblent en un seul octet. L'opérande est en fait inhérent au code - opération. Toute l'information nécessaire à l'exécution de l'instruction est en fait contenue dans l'instruction et ne nécessite pas d'adresse.

Ce mode d'adressage concerne les instructions d'échange entre registre tels que TAY, des instructions forçant des bits du registre d'état du processeur tel que CLC, ou aux instructions de manipulations de la pile tel que PEA, ou aux retours de sous-programmes tel que RTS

1.3.2 Adressage Immédiat.

Ce mode d'adressage consiste à mettre directement derrière le code de l'opération la donnée sur laquelle on veut opérer, et non une adresse.

1.3.3 Adressage Accumulateur.

Ce mode d'adressage concerne les instructions ne modifiant que l'accumulateur. Ces instructions sont les décalages, les rotations, l'incrémentation ou la décrémentation de l'accumulateur.

1.3.4 Adressage absolu

Le second et le troisième octet de l'instruction, en adressage absolu forment les 16 bits de poids faible de l'adresse effective. Le registre DBR (Data Bank register) contient les 8 bits de poids fort de l'adresse.

1.3.5 Adressage absolu long

Le second, troisième et quatrième octets de l'instruction forment les 24 bits de l'adresse effective.

1.3.6 Adressage direct

Le second octet de l'instruction est additionné avec le registre D (Registre Direct) pour former l'adresse effective.

1.3.7 Adressage direct indirect indexé par Y

Ce mode d'adressage est aussi appelé «Indirect Y». Le second octet de l'instruction est additionné avec le contenu du registre D (Registre direct). Les 16 bits alors obtenus sont associés avec les bits du registre DBR pour former une adresse de 24 bits. A cette adresse de 24 bits il suffit d'ajouter le contenu du registre d'index Y pour former l'adresse effective.

1.3.8 Adressage indirect long indexé par Y

Le second octet de l'instruction est additionné au contenu du registre D (Registre direct). Les 24 bits ainsi obtenus sont additionnés au registre d'index Y pour obtenir l'adresse effective.

1.3.9 Adressage direct indexé indirect

Ce mode d'adressage est aussi appelé «Indirect X». Le second octet de l'instruction est additionné à la somme du registre D (Registre direct) et du registre d'index X. Le résultat constitue les 16 bits de poids faible de l'adresse effective. Les 8 bits de poids fort sont constitués par les 8 bits du registre DBR (Data bank register).

1.3.10 Adressage direct indexé par X.

Le second octet de l'instruction est additionné avec la somme du registre D (Registre direct) et du registre d'index X. Les 16 bits ainsi obtenus forment l'adresse effective.

1.3.11 Adressage direct indexé par Y.

Le second octet de l'instruction est additionné avec la somme du registre D (Registre direct) et du registre d'index Y. Les 16 bits ainsi obtenus forment l'adresse effective.

1.3.12 Adressage absolu indexé par X.

Le second et le troisième octet de l'instruction sont additionnés au registre d'index X pour former les 16 bits de poids faible de l'adresse effective. Les 8 bits de poids fort sont contenus dans le registre DBR (Data bank register).

1.3.13 Adressage absolu long indexé par X.

Le second, troisième et quatrième octet de l'instruction forment les 24 bits de l'adresse de base. L'adresse effective est obtenue en additionnant cette adresse de base au registre d'index X.

1.3.14 Adressage absolu indexé par Y.

Le second et le troisième octet de l'instruction sont additionnés au registre d'index Y pour former les 16 bits de poids faible de l'adresse effective. Les 8 bits de poids fort sont contenus dans le registre DBR (Data bank register).

1.3.15 Adressage relatif

Ce mode d'adressage est utilisé uniquement avec les branchements.

Si la condition est remplie le second octet de l'instruction est additionné au compteur ordinal, afin que la prochaine instruction puisse être exécutée.

Le branchement ne peut s'effectuer qu'entre 128 et + 127 octets à partir de l'instruction suivant le branchement.

1.3.16 Adressage relatif long

Ce mode d'adressage n'est utilisé qu'avec 2 instructions BRL et PER.

Le second et le troisième octet de l'instruction sont ajoutés au compteur ordinal, qui est situé une fois que cette instruction se déroule l'instruction immédiatement suivante de BRL ou de PER, pour former l'adresse effective.

1.3.17 Adressage absolu indirect.

Le second et le troisième octet de l'instruction forment une adresse de 16 bits. Le compteur ordinal est chargé avec le premier et le second octet à cette adresse.

1.3.18 Adressage direct indirect.

Le second octet de l'instruction est ajouté au Registre D, pour former les 16 bits de poids faible l'adresse effective. Le registre DBR contient les 8 bits de poids fort de l'adresse effective.

1.3.19 Adressage indirect long.

Le second octet de l'instruction est ajouté au registre D pour former les 24 bits de l'adresse effective.

1.3.20 Adressage absolu indexé indirect.

Le second et le troisième octet de l'instruction sont ajoutés au registre d'index X, pour former les 16 bits de l'adresse effective.

1.3.21 Pile

Seules les instructions manipulant la pile utilisent ce type d'adressage. L'adresse du banc est toujours 0.

1.3.22 Pile relative

Les 16 bits de poids faible de l'adresse effective sont formés par la somme du second octet de l'instruction et du pointeur de pile. Les bits de poids fort de l'adresse effective sont toujours 0.

1.3.23 Pile relative indirect indexé.

Le second octet de l'instruction est ajouté au registre pointeur de pile pour former les 16 bits de poids faible de l'adresse de base. Le registre DBR contient les 8 bits de poids fort de l'adresse de base. On ajoute à cette adresse de base le contenu du registre Y, pour former l'adresse effective.

1.3.24 Déplacement de bloc mémoire.

Ce mode d'adressage n'est utilisé que par les instructions de déplacement de bloc mémoire. Le second octet de l'instruction contient les 8 bits de poids fort de l'adresse de destination. Le registre d'index Y contient les 16 bits de poids faible de l'adresse de destination.

Le troisième octet de l'instruction contient les 8 bits de poids fort de l'adresse de départ. Le registre d'index X contient les 16 bits de poids faible de l'adresse de départ.

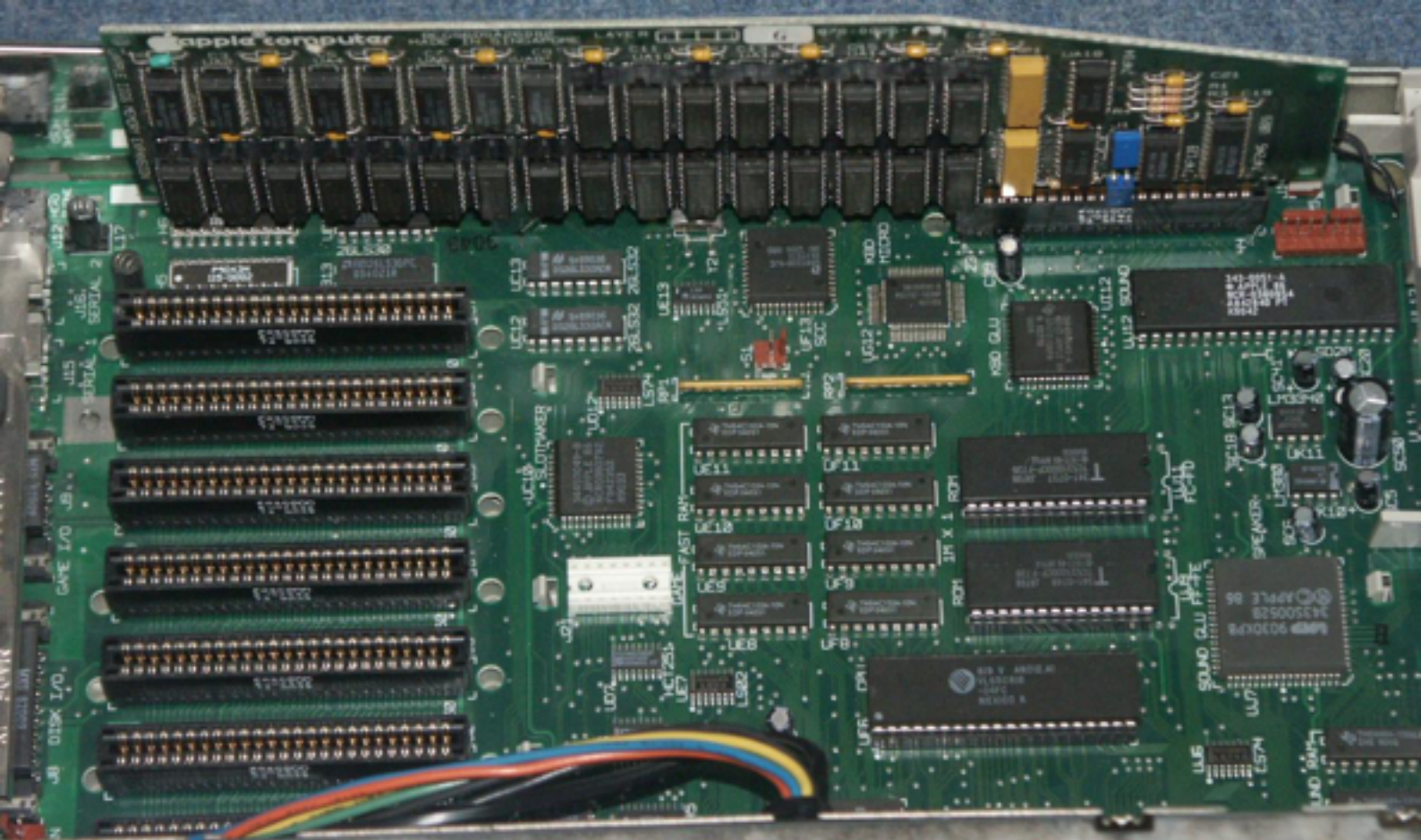
1.4 Conclusion sur le microprocesseur.

En conclusion sur le microprocesseur, je dirais que le 65C816 a l'avantage que ses instructions n'utilisent que peu de cycles, par rapport à d'autres microprocesseurs.

On peut cependant regretter que le changement d'adressage ne puisse pas comprendre des incréments à 2, ainsi qu'une auto incrémentation du compteur ordinal entre deux bornes, notamment pour les consultations de tables.

De plus, la longueur d'un cycle de ce microprocesseur peut varier dans le temps, ce qui peut être gênant dans certaines applications qui nécessitent une synchronisation parfaite.

Enfin mon plus grand regret est que le microprocesseur de l'Apple IIgs n'est cadencé qu'à 2.8 Mhz, alors qu'un minimum de 8Mhz semblerait nécessaire.



La RAM de l'Apple II GS

2.0 Nouvelle résolution graphique (Super-hires)

La mémoire du GS peut être divisée en trois zones distinctes:

- La mémoire principale accessible directement au micro-processeur.

- La mémoire du DOC (Ensoniq) Ram de 64k accessible directement par l'Ensoniq et via une série de Softswitches (commutateurs logiciels) par le micro-processeur.
- La mémoire CMOS (BRAM) sauvegardée par batterie et contenant la configuration de l'Apple IIGS. (paramétrage du tableau de contrôle : Control Panel). Cette mémoire est gérée directement par le circuit horloge, et accessible via une série de Softswitches.

La mémoire principale est elle même divisée en plusieurs zones :

*La mémoire rapide, 128k de base sur la carte mère. Cette zone mémoire est composée des bancs \$00 à \$79.

*La mémoire lente, mémoire système à laquelle on accède à la vitesse de 1Mhz (vitesse lente). Cette zone mémoire est composée de 128k formant les bancs \$E0 à E1. Dans cette zone est incluse la zone des entrées sorties de \$C000 à \$CFFF, ainsi que la mémoire utilisée par l'affichage vidéo.

2.1 L'effet miroir ou shadowing

Le GS possède donc deux types de mémoires fonctionnant à des vitesses différentes. La présence d'une mémoire accédée uniquement à 1Mhz (vitesse lente) est nécessaire pour les entrées sorties et l'affichage vidéo.

Mais la présence d'un mode émulation Apple II sur l'Apple IIGS implique qu'il est nécessaire de transférer des informations entre la mémoire rapide, simulant les 64K de la mémoire principale et les 64k de la mémoire auxiliaire, vers la mémoire lente contenant notamment les entrées sorties et la mémoire vidéo (page texte, graphique) D'où la présence d'un mécanisme transférant automatiquement des valeurs écrites en certains emplacements de la mémoire rapide vers les mêmes emplacements de la mémoire lente. Ce mécanisme à pour nom le Shadowing, ou effet Miroir.

L'effet Miroir peut affecter différentes zones de la mémoire des bancs \$00 et \$01. Ces zones sont les suivantes :

\$0400-07FF Texte page1

\$2000-3FFF Page Hires 1

\$4000-5FFF Page Hires 2

\$2000-9FFF Super Hires

\$2000-3FTE Page Hires mémoire auxiliaire et carte langage

\$C000-FFFF E/S et carte language

On remarque que la page Texte 2 ne peut pas bénéficier de l'effet Miroir, on est donc obligé de le simuler de forme logicielle par la copie périodique de la page 2 du bank \$00 dans le bank \$E0; cette option est validée via le tableau de contrôle (Alternate display mode).

L'état de l'effet Miroir est contrôlé par le registre se trouvant en \$C035 (quagmire state).

Il s'agit d'un registre d'un octet indiquant, outre la vitesse de la machine (lente /rapide) les zones mémoire affectées par l'effet Miroir.

Il faut noter que l'utilisation de l'effet Miroir ralentit la machine. En effet, lors de chaque accès en écriture à une zone Refletée, la valeur écrite est automatiquement reportée dans la zone correspondante de la mémoire lente. Ce phénomène ne se produit évidemment pas lors des opérations de lecture.

De même une écriture directe en \$E0 ou \$E1 ne génère pas une recopie de la valeur dans les bancs à vitesse rapide. L'effet Miroir ne marche que dans un sens mémoire rapide vers mémoire lente et seulement pour les opérations d'écriture.

2.1 La page zéro 00/0000 - 00/00FF

Dans la page zéro se trouvent la plupart des variables utilisées en mode émulation, par l'Applesoft, le Prodos 8 et le moniteur.

En mode CS/OS la mémoire (dont la page zéro, positionable dans une page quelconque du banc 0) est gérée par le Gestionnaire de la mémoire et aucune variable à emplacement fixe n'y est définie. Label

\$00-\$02	JMP \$D43C	Entrée à chaud de l'Applesoft
\$03-\$05	JMP \$DB3A	STROUT
\$06-\$09		libre
\$0A-\$0C		JMP adresse appelée par la fonction USR de l'Applesoft.
\$0D-\$10		Variables Applesoft
\$11	VALTYP	0 si FAC nombre 1 si chaîne
\$12-\$13		Variables Applesoft
\$14	SUBFLAG	si 00 variable dimensionnée, \$80 sinon
\$15-\$17		Variables Applesoft
\$18-\$1F		libre
\$20	WNDLFT	Colonne de début de la fenêtre
\$21	WNDWDTH	Nombre de colonnes de la fenêtre
\$22	WNDTOP	Première ligne de la fenêtre

\$23	WNDBTM	
\$24	CH	Colonne du curseur
\$25	CV	Ligne du curseur
\$26-\$27	GBASL	Adresse du début de la ligne GBASH courante dans la page Basse résolution calculée par GBASCALC Pointeur temporaire (DOS 3.3 RWTS)
\$28-\$29	BASL	Adresse de début de la ligne de texte BASH courante calculée par BASCALC Pointeur temporaire (DOS 3.3)
\$2A - \$2B	BAS2L	Pointeur ligne de destination lors BAS2H d'un scroll. Pointeur temporaire (DOS 3.3)
\$2C		H2 Zone temporaire graphisme basse résolution LMNEN Zone temporaire décodage mnémoniques Mini assembleur RTNL
\$2D		Checksum de l'entête du secteur (DOS 3.3 RWTS) V2 Zone temporaire graphisme basse résolution RMNEM Zone temporaire décodage mnémoniques Mini assembleur RTNH Numéro de secteur contenu dans l'entête (DOS 3.3 RWTS)
\$2E	MASK FORMAT	Masque couleur pour graphismes basse résolution Zone temporaire décodage de l'opcode Mini assembleur. Numéro de piste contenu dans l'entête (DOS 3.3 RWTS)
\$2F	LASTIN LENGTH	Utilisé lors de la lecture cassette sur Apple IIe et II+. Zone temporaire décodage de opcode Mini assembleur Numéro de volume contenu dans l'entête (DOS 3.3 RWTS)
\$30	COLOR	Couleur graphisme basse résolution
\$31	MODE	Mode dans lequel se trouve le moniteur
\$32	INVFLG	Masque utilisé lors de l'affichage \$3F: lettres en vidéo inverse \$FF: lettres normales \$7F: lettres clignotantes Toute autre valeur donne un affichage incohérent.
\$33	PROMPT	Caractère utilisé comme prompt, est affiché par la routine GETLIN: \$FD6A \$AA pour le moniteur code Ascii de *

		\$DD pour l'Applesoft en entrée de ligne code Ascii de]
\$34-\$35	YSAV	Sauvegarde YSAV1
\$35		Numéro du drive dans le bit de poids fort (DOS 3.3 RWTS) \$80 drive 1- \$00 drive 0
\$36-\$37	CSW	Adresse de la routine de sortie de CSWH caractères.
\$38-\$39	KSW	Adresse de la routine d'entrée de KSWH caractères.
\$3A-\$4F		— Zone utilisée par PRODOS
\$3A-\$3B	PCL	Sauvegarde du compteur programme PCH
\$3C-\$3D		Adresse du DCT(DOS3.3RWTS) Device characteristics table.
\$3E-\$3F		Adresse du buffer (DOS 3.3 RWTS)
\$40-\$41		Adresse du buffer fichier (DOS 3.3)
\$41		Compteur de formatage (DOS3.3RWTS)
\$42-\$43		Adresse d'un buffer de travail (DOS 3.3)
\$44-\$45		Opérande numérique (DOS3.3)
\$46-\$47		Zone de travail (DOS 3.3 RWTS)
\$48-\$49		Adresse de l'IOB (DOS3.3 RWTS)
\$3C		XQT Zone pour le pas à pas (Step) et la trace
\$3C-\$45		A1L Zone des paramètres pour A1H l'appel de sous programmes du A2L moniteur. A2H, A3L,A3H,A4L,A4H,A5L,A5H Exemple appel de la routine Auxmove
\$44-\$49		Sauvegarde des registres après un BRK
\$46-\$47		Zone de travail (DOS 3.3 RWTS)
\$48-\$49		Adresse de l'IOB (DOS3.3RWTS)
\$44	MACCSTAT	Etat de la machine après un BRK
\$45	ACC	Sauvegarde de l'accumulateur
\$46	XREG	Sauvegarde de X
\$47	YREG	Sauvegarde de Y
\$48	STATUS	Sauvegarde du registre d'état
\$49	SPNT	Sauvegarde du pointeur de pile
\$42-\$47		Paramètres du «DEVICE DRIVER» Prodos 8
\$42		Code de la commande: \$00 Status \$01 Lecture \$02 Ecriture \$03 Formatage
\$43		Numéro de l'unité sous la forme DSSS0000 où D=numéro du drive et SSS=numéro du slot Exemple: 01100000 soit \$40 Drive 1 slot 6.
\$44-\$45		Zone destination ou source.

\$46-\$47		Numéro du bloc.
\$4A-\$4D		Utilisé par ProDOS
\$4C-\$4D	HIMEM	himem du basic (INTEGER)
\$4E-\$4F		Nombre aléatoire
\$50-\$51		Applesoft : Numéro de la ligne après un LINGET \$DA0C
\$52-\$61		Variables utilisées par l'Applesoft
\$62-\$66		Nombre flottant
\$67-\$68	TXTTAB	Adresse du début du programme basic
\$69-\$6A	LOMEM	Adresse de début de la zone des variables simples (entiers et réels)
\$6B-\$6C	ARYTAB	Adresse de la fin de la zone des variables simples, début de la zone des tableaux
\$6D-\$6E	STREND	Adresse de début de la zone libre fin des tableaux
\$6F-\$70	FRETOP	Adresse de début de la zone des chaînes de caractères et de la fin de la zone libre
\$71-\$72		Réservé Applesoft
\$73-\$74	HIMEM	Fin de la zone disponible pour l'Applesoft
\$75-\$76	CURLIN	Numéro de la ligne en cours d'exécution \$FFFF si mode direct
\$77-\$78	OLDLIN	Numéro de la dernière ligne exécutée ou interrompue
\$79-\$7A	OLDTXPTR	Adresse utilisée par CONT pour continuer l'exécution d'un programme basic interrompu.
\$7B-\$7C		Numéro de la ligne de l'instruction DATA courante
\$7D-\$7E		Adresse de l'élément à lire dans cette ligne
\$7F-\$80		\$201 lors d'un INPUT sinon égal à la ligne de data lors d'un READ
\$81-\$82		Nom, en Ascii de la dernière variable utilisée
\$83-\$84	VARPNT	Adresse de la valeur ou longueur de chaîne de la dernière variable utilisée
\$85-\$86	FORPNT	
\$87-\$89		Utilisé par l'Applesoft
\$8A-\$8E	TEMP3	Registre flottant sur 5 octets
\$8F-\$92		Utilisé par l'Applesoft
\$93-\$97	TEMP1	Registre flottant sur 5 octets
\$98-\$9C	TEMP2	Registre flottant sur 5 octets
\$9D-\$A2	FAC	Registre flottant sur 5 octets: Accumulateur
\$A3-\$A4		
\$A5-\$AA	ARG	Registre flottant contient le 2ème argument des fonctions
\$AB-\$AC	STRGN1	Adresse d'une chaîne devant être déplacée par MOVINS
\$AD-\$AE	STRGN2	

\$AF-\$B0	PRGEND	Adresse de la fin du programme
\$B1-\$C8	CHRGET	Sous-programme pour obtenir le prochain caractère du programme, s'auto patch en TXTPTR
\$B8-B9	TXTPTR	Adresse du dernier caractère obtenu par CHRGET
\$C9-\$CD		Nombre aléatoire flottant
\$CA-\$CB	START	Début du programme basic (INTEGER)
\$CA-\$CD	VAREND	Fin des variables (INTEGER)
\$CE-\$CF		Libre
\$D0-\$D5		Utilisé par la haute résolution
\$D6		Flag auto-exécution si différent de 0
\$D7		Libre
\$D8-\$D9		Numéro de ligne (INTEGER)
\$D8	ERRFLG	\$80 si ONERR est actif
\$D9	RUNMOD	\$80 si en cours d'exécution
\$DA-\$DB	ERRLIN	Numéro de la ligne où l'erreur a eu lieu
\$DC-\$DD	ERRPOS	Adresse du caractère où a eu lieu l'erreur
\$DE	ERRNUM	Code de l'erreur
\$DF	ERRSTR	Sauvegarde registre pointeur de pile
\$E0-\$E2		Coordonnées du curseur HGR
\$E4		Code de la couleur HGR
\$E6	HPAG	Numéro de la page HGR active \$20=page 1, \$40=page 2
\$E7	SCALE	Echelle des formes
\$E8-\$E9		Adresse de la table des formes
\$EA		Zone utilisée par la haute résolution
\$EB-\$EF		Libre
\$F0	FIRST	
\$F1		SPDBYT Vitesse d'affichage 1 correspond à speed = 255 en basic, 0 à speed = 0 en général SPEED = X correspond à SPDBYT = 256-X
\$F2-\$F3		Libre
\$F4-\$F8		Utilisé par l'Applesoft pour les traitements ONERR
\$F9-\$FF		Libre

Note : toutes les adresses, pointeurs sont stockées sous la forme poids faible poids fort, par exemple si TXTTAB = \$801 en \$67 on trouvera \$01 et en \$68 \$08

2.2 La zone 00/0100-00/03FF

\$100-\$1FF	Pile du 65C816 en mode émulation
\$200-\$2FF	Buffer d'entrée de ligne sous basic pour GETLN

\$300-\$3CF		Libre
\$3D0-\$3D2	WARM	Un saut (JMP) à l'entrée à chaud du DOS. Cette routine relance le DOS mais n'efface pas le programme basic en mémoire et ne change pas Maxfile (DOS 3.3)
\$3D3-\$3D5	COLD	Un saut (JMP) vers l'entrée à froid du DOS. Réinitialise le DOS, remet HIMEM à sa valeur initiale, supprime le programme Basic de la mémoire. (DOS 3.3)
\$3D6-\$3D8		Appel du file manager (DOS 3.3)
\$3D9-\$3DB		Appel de la RWTS (DOS 3.3) (Read/Write/Track/Sector ou lecture écriture de pistes et secteurs)
\$3DC-\$3E2		Localise la liste de paramètres du file manager Adresse dans A et Y, A contenant le poids fort.
\$3E3-\$3E9		Localise l'IOB utilisée pour les appels à la RWTS Adresse retournée dans A et Y, poids fort dans A
\$3EA-\$3EC		Un saut vers la routine reconnectant le DOS aux vecteurs entrée clavier sortie écran (DOS 3.3)
\$3F0-\$3F1	BRKV	Adresse de la routine traitant les interruptions BRK. Normalement OLDBRK soit \$FA59, routine affichant les registres.
\$3F2-\$3F3	SOFTEV	Adresse de la routine traitant le reset.
\$3F4	PWREDUP	Valide l'adresse du reset si cette valeur égale (SOFTEV+1) EOR \$A5

Exemple de revectorisation du Reset:

```

LDA  #$59 ; poids faible de $FF59 soit OLDRST
STA  $3F2 ; SOFTEV
LDA  #$FF ; poids fort de ; $FF59
STA  $3F3 ; SOFTEV+1
JSR  $FB6F ; SETPWRC routine du moniteur
      ; effectuant le EOR $A5

```

\$3F5-\$3F7	AMPERV	Routine traitant l'ampersand en APPLESOFT. Normalement \$3F5 contient \$4C pour l'instruction JMP
\$3F8-\$3F	AUSRADR	Routine traitant C I RL-Y pour le moniteur et la fonction USR pour l'Applesoft Normalement on trouve un JMP \$FF65 (MON) dans le moniteur, si basic.system à été chargé, pointe sur l'entrée à chaud du basic.

\$3F13-\$3F	DNMI	Routine traitant les NMI. Normalement on trouve un \$4C en \$3FB (opcode de JMP) Normalement JMP \$FF59 soit le OLDRST du moniteur
\$3FE-\$3F	FIRQLOC	Adresse de la routine traitant les IRQs. Normalement \$FF65

2.3 Carte mémoire Applesoft

Zones occupées par l'Applesoft en mode émulation.

TXTTAB	\$67-\$68	TEXTE DU PROGRAMME BASIC
PRGEND	\$AF-\$B0	
LOMEM	\$69-\$6A	VARIABLES SIMPLES, ENTIERS REELS
ARYTAB	\$6B-\$6C	TABLEAUX
STREND	\$6D-\$6E	ZONE LIBRE
FRETOP	\$6F-\$70	CHAINES DE CARACTERES
HIMEM	\$73-\$74	DOS OU PRODOS

2.4 Occupation des bancs \$E0-\$E1

Dans ces bancs, composés de mémoire lente (à 1Mhz), se trouvent la plupart des vecteurs du GS. De même on y trouve les buffers vidéo, et la zone des entrées sortie.

2.4.1 Occupation du bank \$E0

\$0000-\$02FF	Réservé.
\$0300-\$03FF	Buffer des accessoires de bureau.
\$0400-\$07FF	Texte page 1.
\$0800-\$0BFF	Texte page 2.
\$0C00-\$1DFF	Buffer des accessoires de bureau.
\$1E00-\$1FFF	Vecteurs Quickdraw II.
\$2000-\$3FFF	Page graphique 1.
\$4000-\$5FFF	Page graphique 2.
\$6000-\$BFFF	Mémoire libre gérée par le gestionnaire mémoire.
\$C000-\$CFFF	Zone des entrées sorties
\$D000-\$DFFF	Buffers AppleTalk
\$E000-\$FFFF	ou loader Prodos

2.4.2 Occupation du banc \$E1

2.4.2.1 Vecteurs du banc \$E1

\$0000-\$0003	DISPATCH1: Saut au «TOOL LOCATOR» localisateur d'Outils Normalement sous la forme JMIL\$xx/xxxxx Type 1
\$0004-\$0007	DISPATCH2: Idem DISPATCH1 mais pour tools type 2

\$0008-\$000B	UDISPATCH1: copie de DISPATCH1 modifiable pour installer sa propre version de tool locator type 1
\$000C-\$000F	UDISPATCH2: copie de DISPATCH2 modifiable pour installer sa propre version de tool locator type 2
\$0010-\$0013	INTMGRV: Saut vers le gestionnaire d'interruption. Instruction jump long JML \$xx/xxxx
\$0014-\$0017	COPMGRV: Saut vers le gestionnaire des instructions COP. Instruction jump long JML \$xx/xxxx
\$0018-\$001B	ABORTMGRV: Saut vers le gestionnaire de ABORT. Actuellement traite comme le break affichage des registres.
\$001C-\$001F	SYSDMGRV: Saut vers le gestionnaire d'échec système system failure» L'appel de cette routine suppose l'état suivant

On doit se trouver en mode 16 bit natif

Le bit carry doit être à 1 si l'adresse d'un message personnalisé se trouve sur la pile, 0 sinon.

La pile contient les paramètres suivants :

Code erreur poids fort 9 , S

Code erreur poids faible 8,S 0 7,S

Numéro de la bank de l'adresse du message 6,S

Poids fort de l'adresse du message 5,S

Poids faible de l'adresse du message 4,S

Adresse de retour inutilisée 3,S

Adresse de retour inutilisée 2,S

Adresse de retour inutilisée 1,S

\$0023 -\$0023	IRQ.APTALK: Saut vers le gestionnaire d'interruption Appletalk
\$0024-\$0027	IRQ.SERIAL: Saut vers le gestionnaire d'interruption port série
\$0028-\$002B	IRQ.SCAN: Saut vers le gestionnaire d'interruption de fin de ligne «scan file interrupt»
\$002C-\$002F	IRQ.SOUND: Saut vers le gestionnaire d'interruption processeur sonore (Ensoniq)
\$0030-\$0033	IRQ.VBL: Saut vers le gestionnaire d'interruption de balayage vertical.
\$0034-\$0037	IRQ.MOUSE: Saut vers le gestionnaire d'interruption souris.

\$0038-\$003B	IRQ.QTR: Saut vers le gestionnaire d'interruption quart de seconde utilisé par AppleTalk
\$003C-\$003F	IRQ.KBD Saut vers le gestionnaire d'interruption du clavier. Actuellement le clavier ne génère pas d'interruption. Il est possible de simuler leur présence par l'intermédiaire du Miscellaneous Tool set en installant une tâche «heartbeat» (fonction SetHeartBeat) générant une interruption lors de la scrutation du clavier par le VBL. Si une touche est enfoncée la tâche «heartbeat» appellera cette routine via un JSL
\$0040-\$0043	IRQ.RESPONSE: Saut vers le gestionnaire d'interruptions réponse ADB Saut vers le gestionnaire de réponse de l'Apple Desktop Bus.
\$0044-\$0047	IRQ.SRQ: Saut vers le gestionnaire d'interruptions SRQ Saut vers le gestionnaire des interruptions SRQ (Requête de service) de l'ADB (Apple Desktop Bus)
\$0048-\$004B	IRQ.DSKACC: Saut vers le gestionnaire d'interruptions d'accès au gestionnaire de bureau. Saut vers la routine invoquée lors de la pression des touches CTRL- POMME- ESC.En installant un RTL(\$6B) à la place du JML (\$5C) on interdit l'accès au panneau de contrôle et aux CDA.
\$004C-\$004F	IRQ.FLUSH: Saut vers le gestionnaire d'interruptions vidange du clavier. Cette interruption est provoquée par la frappe des touches CTRL-POMME-BACKSPACE
\$0050-\$0053	IRQ.MICRO: Saut vers le gestionnaire d'interruptions d'abort du micro processeur clavier. Cette interruption a lieu lors d'une défaillance critique du microprocesseur du clavier. Si une telle erreur apparaît le gestionnaire essaye de resynchroniser et de réinitialiser le microprocesseur.
\$0054-\$0057	IRQ.1SEC: Saut vers le gestionnaire d'interruption seconde.
\$0058-\$005B	IRQ.EXT: Saut vers le gestionnaire d'interruptions VGC Normalement saut vers le gestionnaire des interruptions générées par le Video Graphic Chip (circuit video). Actuellement la patte du circuit générant cette interruption est forcée à l'état haut empêchant toutes interruptions.
\$005C-\$005F	IRQ.OTHER: Saut vers le gestionnaire des autres interruptions. Ce gestionnaire

- traite toutes les interruptions non traitées par ailleurs par le logiciel.
- \$0060-\$0063** CUPDATE:
Saut vers la routine de mise à jour du curseur dans QuickDraw II (Cursor Update)
- \$0064-\$0067** INCBUSYFLG:
Saut vers la routine d'incrémentation du flag occupé.
- \$0068-\$006B** DECBUSYFLG:
Saut vers la routine de décrémentation du flag occupé.
- \$006C-\$006F** BELLVECTOR:
Saut vers une routine <personnelle> de BEEP. Cette routine est appelée par le moniteur à chaque demande d'impression d'un caractère \$87 via les vecteurs de sortie (CSWL/CSWH \$36/\$37) et lors de l'appel des routines BELL1, BELL1.2 et BELL2 (\$FBDD, \$FBE2, \$FBE4).
La routine doit satisfaire aux caractéristiques suivantes:
- Elle est appelée en mode 8bit natif et doit revenir au moniteur en mode 8bit natif.
 - Les registres Data Bank et Direct doivent être préservés.
 - Le bit carry doit être à zéro (clear) ou le moniteur générera son propre beep.
 - Le registre X doit être préservé.
 - A la sortie de la routine Y doit contenir 0
- \$0070-\$0073** BREAK VECTOR:
Saut vers une routine <personnelle> de traitement de l'opcode BRK. Cette routine doit obéir aux caractéristiques suivantes:
- Elle est appelée en mode 8bit natif, vitesse rapide, avec le registre Data Bank égal à zéro et le registre Direct égal à zéro.
 - La valeur des registres Data Bank et Direct doit être préservée.
 - La routine doit se terminer par un RTL et revenir en mode 8 bit natif vitesse rapide.
 - Le bit carry doit être à zéro au retour de cette routine sinon le moniteur appelle la routine pointée par \$00/\$3F0
 - Si le bit carry est à zéro le programme interrompu par le BRK est poursuivi 2 octets après le BRK. Ce vecteur est prévu à l'usage des logiciels de débogage.
- \$0074-\$0077** TRACE VECTOR:
Saut vers la routine de trace. Cette routine doit obéir aux caractéristiques suivantes:
- Elle est appelée en mode 8 bit natif, vitesse rapide, avec le

- registre Data Bank égal à zéro et le registre Direct égal à zéro.
- La valeur des registres Data Bank et Direct doit être préservée.
- La routine doit se terminer par un RTL et revenir en mode 8 bit natif vitesse rapide.
- Le bit carry doit être à zéro au retour de cette routine sinon le moniteur appelle la routine pointée par \$00/\$3F0-\$3F1. Si le bit carry est à zéro lors du retour de la routine usager, le programme interrompu est continué, sinon le message Trace est affiché sur l'écran avant continuation du programme. Ce vecteur est prévu l'usage des logiciels de débogage.

\$0078-\$007B

STEP VECTOR:

Vecteur step. Cette routine doit obéir aux caractéristiques suivantes:

- Elle est appelée en mode 8 bit natif, vitesse rapide, avec le registre Data Bank égal à zéro et Le registre Direct égal à zéro.
- La valeur des registres Data Bank et Direct doit être préservée.
- La routine doit se terminer par un RTL et revenir en mode 8 bit natif vitesse rapide.
- Le bit carry doit être à zéro au retour de cette routine sinon le moniteur appelle la routine pointée par \$00/\$3F0-\$3F1. Si le bit carry est à zéro lors du retour de la routine usager, le programme interrompu est continué, sinon le message Step est affiché sur l'écran avant continuation du programme. Ce vecteur est prévu à l'usage des logiciels de débogage.

\$007C-\$007F

Réservé pour des extensions futures.

L'adresse de ces vecteurs est garantie pour toutes les versions des systèmes d'exploitation de l'Apple IIGS. Ces vecteurs ne doivent en aucun cas être altérés par une application.

\$0080-\$0083

TOWRITEBR:

Vecteur sur la routine d'écriture dans la BRAM .Ce vecteur pointe sur une routine copiant le buffer de la BRAM se trouvant dans le bank \$E1 BATTERYRAM dans la RAM du circuit horloge avec la bonne somme de contrôle. Cette routine est appelée par le Miscellaneous Tool Set.

\$0084-\$0087

TOREADBR:

Vecteur sur la routine de lecture de la BRAM. Ce vecteur pointe sur la routine copiant le contenu de la RAM du circuit horloge, dans le buffer du bank \$E1. Si la somme de contrôle est invalide, ou l'un des

paramètres est hors limite, les paramètres par défaut sont recopiés et dans le buffer et dans la RAM du circuit horloge.

\$0088-\$008B

TOWRITETIME:

Ce vecteur pointe sur une routine écrivant dans le registre des secondes du circuit horloge. Elle transfère les valeurs se trouvant dans le buffer CLKWDATA dans le bank \$E1 dans le circuit horloge.

\$008C-\$008F

TOREADTIME:

Ce vecteur pointe sur une routine lisant les registres des secondes du circuit horloge et le recopiant dans le buffer CLKWDATA du bank \$E1. En cas d'échec, au retour le bit carry est à 1.

\$0090-\$0093

TOCTRLPANEL:

Ce vecteur pointe sur le programme du panneau de contrôle. Il suppose qu'il a été appelé par le Desk Manager, et suppose une initialisation correcte de plusieurs zones en page zéro.

\$0094-\$0097

TOBRAMSETUP:

Ce vecteur pointe sur une routine initialisant le système - en fonction des paramètres contenus dans le buffer BATTERYRAM. De plus si on appelle cette routine avec le bit carry à 0 il positionne la configuration des slots (internes externes). Le buffer BATTERYRAM dans le bank \$E1 ne peut être mis jour que grâce au Miscellaneous Tool Set.

\$0098-\$009B

TOPRINTMSG8:

Ce vecteur pointe vers une routine affichant la chaîne ASCII pointée en multipliant le contenu de l'accumulateur par 2 et en s'en servant comme index dans la table pointée par MSGPOINTER (3 bytes). Cette routine est utilisée par le panneau de contrôle intégré, par les panneaux de contrôle en RAM, et par le moniteur.

\$009C-\$009F

TOPRNTMSG 16:

Ce vecteur pointe vers une routine affichant la chaîne ASCII pointée par le contenu de l'accumulateur 16 bits et en s'en servant comme index dans la table pointée par MSGPOINTER (3 bytes). Cette routine est utilisée par le panneau de contrôle intégré, par les panneaux de contrôle en RAM, et par le moniteur.

\$00A0-\$00A3

CTRLVECTOR:

Saut inconditionnel vers une routine usager traitant le CTRL-Y. Cette routine est appelée en mode 8 bit natif, avec le registre Data Bank initialisé à zéro et le registre Direct à \$0000. Cette routine doit préserver le registre Data bank, le registre Direct, la vitesse et revenir en mode émulation avec un RTS de la bank \$00. Si aucun vecteur n'est installé le moniteur exécutera la routine pointée en bank \$00 /USRADR. Ce vecteur est utilisé par les debuggueurs.

\$00A4-\$00A8	TOTEXTPG2DA: Ce vecteur pointe vers l'accessoire «Alternate Display Mode». Il suppose qu'il a été appelé par le Desk Manager. Il retourne au Desk Manager via un RTL lors de la pression d'une touche. Ce vecteur était détourné par les anciennes versions du Diversi Cache.
\$00A9-\$00AC	PRO16MLI: Point d'entrée du MLI ProDOS/16, avec code commande et adresse des paramètres suivant le JSL. Ex: JSL \$E1/00A8 DA cmd DDW parms
\$00A9-\$00AC \$00B0-\$00B3	PRO16MLI : Point d'entrée du MLI ProDOS/16 avec paramètres poussés sur la pile. PushL parms PushW cmd JSL \$E1/00B0
\$00B4-\$00B7 \$00B8-\$00BD	
\$00BC	OSKIND: \$00 ProDOS/8 \$01 GS/OS ou ProDOS/16
\$00BD	OS_BOOT: Cette valeur indique quel système à été initialement booté: \$01 GS/OS ou ProDOS/16 \$00BE-\$00BF bit 15 a 1 si Prodos 16 en cours d'exécution d'un appel
\$00C0-\$00C2	MSGPOINTER: Pointeur vers la tables des adresses des chaines utilisées par le Panneau de controle le moniteur forme de ce pointeur est poids faible/poids fort/no de banc.
\$00FF	BUSYFLAG (Localisation non suportée par Apple)

2.4.2.2 Autres variables en \$E1

\$02B8-\$02BF	Buffer souris.
\$02C0-\$03BF	BATTERYRAM: Buffer où est recopié le contenu de la Ram sauvegardée du circuit horloge.

\$03C0-\$03CF	Variables du Localisateur de Tools.
\$03D0-\$03DF	Listes des interruptions de l'ADB (Apple Desktop Bus).
\$03E0-\$03FF	Buffer horloge.
\$0400-\$07FF	Texte page 1.
\$0800-\$0BFF	Texte page 2.
\$0C00-\$0FCF	Buffer de transfert disque.
\$0FD0-\$0FD5	Utilisé par les Tools.
\$0FD6-\$0FFA	Zone de stockage de l'ADB.
\$0FFB-\$0FFF	Zone de stockage du port série.
\$1000-\$14E1	Zone de stockage AppleTalk.
\$14E2-\$1549	Zone de stockage SmartPort.
\$154A-\$1589	Liste des adresses/attributs ADB.
\$158A-\$15A9	Variables du port série.
\$15AA-\$15C0	Zone de stockage du TOOL texte.
\$15C1-\$15CC	Variables du port série.
\$15CD-\$15FD	Réservé.
\$15FE-\$19B7	Buffer du gestionnaire mémoire.
\$19B8-\$1DAF	Réservé.
\$1DB0-\$1DCF	Buffer des variables Son.
\$1DD0-\$1DD7	Utilisé par les Tools Buffer de la sortie série
\$1DD8-\$1FFF	Buffer de la sortie surie
\$2000-\$3FFF	Page graphique 1
\$4000-\$5FFF	Page graphique 2
\$6000-\$9FFF	Zone super-Hires
\$A000-\$BFFF	Mémoire libre gérée par le gestionnaire mémoire.
\$C000-\$CFFF	Zone des entrées sorties
\$D000-\$DFFF	Buffers AppleTalk
\$E000-\$FFFF	Code AppleTalk

2.5 BRAM ou BATTERYRAM

La BRAM contient tous les paramètres, sauvegardés entre deux extinctions du GS. Allocation des slots, état du clavier, couleur de l'écran, heure et date, format de celle-ci etc...

2.5.1 Contenu de la BRAM

2.5.1.1 Configuration du Port 1:

Octet	Fonction	Par défaut	Limites
-------	----------	------------	---------

\$00	Indique l'usage du slot 1 \$00 imprimante \$01 modem	\$00	\$00 ou \$01
\$01	Longueur de ligne \$00 non limité \$01 40 caractères \$02 72 caractères \$03 80 caractères \$04 132 caractères	\$00	\$00-\$04
	Nombre de caractères reçus ou transmis avant rajout d'un retour chariot.		
\$02	Supprimer le premier saut de lignes après le retour chariot. (supprimer le Line Feed \$0A après le Carriage Return \$0D)	\$00	\$00-\$01
	\$00 Non \$01 Oui		
\$03	Ajouter un saut de ligne après le retour chariot.	\$01	\$00-\$01
	\$00 Non \$01 Oui		
\$04	Echo \$00 Absent \$01 Présent (affiche les caractères transmis sur l'écran)	\$00	\$00-\$01
\$05	Buffer \$00 Non \$01 Oui	\$00	
\$06	Vitesse en bauds \$00 50 bauds \$01 75 bauds \$02 110 bauds \$03 134.5 bauds \$04 150 bauds \$05 300 bauds \$06 600 bauds \$0E 19200 bauds	\$0D \$07 1200 \$08 1800 \$09 2400 \$0A 3600 \$0B 4800 \$0C 7200 \$0D 9600	\$00-\$0E
\$07	Transmission Nombre de bits de données et de stop \$00 5 bits de données 1 de stop \$01 5 bits de données 2 de stop	\$06	\$00-\$07

\$02 6 bits de données 1 de stop
 \$03 6 bits de données 2 de stop
 \$04 7 bits de données 1 de stop
 \$05 7 bits de données 2 de stop
 \$06 8 bits de données 1 de stop
 \$07 8 bits de données 2 de stop

\$08	Parité	\$02	\$00-\$02
	\$00 Impaire		
	\$01 Paire		
	\$02 Aucune		
\$09	DCD Handshake	\$01	\$00-\$01
	\$00 Non		
	\$01 Oui		
\$0A	DSR/DTR Handshake	\$01	\$00-\$01
	\$00 Non		
	\$01 Oui		
\$0B	XON/XOFF Handshake	\$00	\$00-\$01
	\$00 Non		
	\$01 Oui		

2.5.1.2 Configuration du Port 2:

Octet	Fonction	Par défaut	Limites
\$0C	Indique l'usage du slot 2	\$00	\$00 ou \$01
	\$00 imprimante		
	\$01 modem		
\$0D	Longueur de ligne	\$00	\$00-\$04
	\$00 non limité		
	\$01 40 caractères		
	\$02 72 caractères		
	\$03 80 caractères		
	\$04 132 caractères		

Nombre de caractères reçus ou transmis avant rajout d'un retour chariot.

\$0E	Supprimer le premier saut de lignes après le retour chariot. (supprimer le Line Feed \$0A après le Carriage Return \$0D)	\$00	\$00-\$01
------	-----------------------------------------------------------------------------------------------------------------------------	------	-----------

\$00 Non

\$01 Oui

\$0F Ajouter un saut de ligne après le retour chariot.
\$01 \$00-\$01

\$00 Non

\$01 Oui

\$10 Echo \$00 \$00-\$01

\$00 Absent

\$01 Présent (affiche les caractères transmis sur l'écran)

\$11 Buffer \$00

\$00 Non

\$01 Oui

\$12 Vitesse en bauds \$0D \$00-\$0E

\$00 50 bauds \$07 1200

\$01 75 bauds \$08 1800

\$02 110 bauds \$09 2400

\$03 134.5 bauds \$0A 3600

\$04 150 bauds \$0B 4800

\$05 300 bauds \$0C 7200

\$06 600 bauds \$0D 9600

\$0E 19200 bauds

\$13 Transmission \$06 \$00-\$07

Nombre de bits de données et de stop

\$00 5 bits de données 1 de stop

\$01 5 bits de données 2 de stop

\$02 6 bits de données 1 de stop

\$03 6 bits de données 2 de stop

\$04 7 bits de données 1 de stop

\$05 7 bits de données 2 de stop

\$06 8 bits de données 1 de stop

\$07 8 bits de données 2 de stop

\$14 Parité \$02 \$00-\$02

\$00 Impaire

\$01 Paire

\$02 Aucune

\$15 DCD Handshake \$01 \$00-\$01

\$00 Non

\$01 Oui

\$16	DSR/DTR Handshake	\$01	\$00-\$01
	\$00 Non		
	\$01 Oui		

\$17	XON/XOFF Handshake	\$00	\$00-\$01
	\$00 Non		
	\$01 Oui		

2.5.1.3 Divers:

Octet	Fonction	Par défaut	Limites
\$18	Mode d'affichage	\$00	\$00-\$01
	\$00 Couleur		
	\$01 Monochrome		
\$19	Affichage 40/80 colonnes	\$00	\$00-\$01
	\$00 40 colonnes		
	\$01 80 colonnes		
\$1A	Couleur pour du texte	\$0F	\$00-\$0F
	\$00 Noir	\$08 Marron	
	\$01 Rouge sombre	\$09 Orange	
	\$02 Bleu sombre	\$0A Gris clair	
	\$03 Pourpre	\$0B Rose	
	\$04 Vert foncé	\$0C Vert pâle	
	\$05 Gris sombre	\$0D Jaune	
	\$06 Bleu	\$0E Aquamarine	
	\$07 Bleu clair	\$0F Blanc	
\$1B	Couleur de fond	\$06	\$00-\$0F
\$1C	Couleur du bord	\$06	\$00-\$0F
\$1D	50 ou 60 Hertz	\$00	\$00-\$01
	\$00 50 hertz		
	\$01 60 hertz		
\$1E	Volume du haut parleur	\$06	\$00-\$0E
\$1F	Tonalité du haut parleur	\$07	\$00-\$0E

\$20	Vitesse du processeur	\$00	\$00-\$01	
	\$00 Rapide			
	\$01 Lente			
\$21	Utilisation du slot 1	\$00	\$00-\$01	
	\$00 Imprimante			
	\$01 Votre carte			
\$22	Utilisation du slot 2	\$00	\$00-\$01	
	\$00 Modem			
	\$01 Votre carte			
\$23	Utilisation du slot 3	\$00	\$00-\$01	
	\$00 Carte 80 colonnes interne			
	\$01 Votre carte			
\$24	Utilisation du slot 4	\$00	\$00-\$01	
	\$00 Port souris			
	\$01 Votre carte			
\$25	Utilisation du slot 5	\$00	\$00-\$01	
	\$00 Port intelligent (smart port)			
	\$01 Votre carte			
\$26	Utilisation du slot 6	\$00	\$00-\$01	
	\$00 Port disque			
	\$01 Votre carte			
\$27	Utilisation du slot 7	\$01	\$00-\$01	
	\$00 Appletalk			
	\$01 Votre carte			
\$28	Slot de démarrage	\$00	\$00-\$09	
	\$00 Recherche des slots pour le démarrage			
	Commence la recherche A partir du slot 7			
	\$01-\$07 slot de démarrage			
	\$08 Démarre A partir du disque RAM			
	\$09 Démarre A partir du disque ROM			
\$29	Langue de l'affichage	\$00	\$00-\$1F	
	\$00 Anglais (USA)			*
	\$01 Anglais (UK)			*
	\$02 Français			*

\$03 Danois *
 \$04 Espagnol *
 \$05 Italien *
 \$06 Allemand *
 \$07 Suédois *
 \$08 «Dvorak» *
 \$09 Canadien (français) *
 \$0A Hollandais (Flamand)
 \$0B Hébreux
 \$0C Japonais
 \$0D Arabe
 \$0E Grec
 \$0F Turc
 \$10 Finlandais
 \$11 Portugais
 \$12 Tamal
 \$13 Hindi
 \$14-\$1F Reservé aux extensions futures.

* Valeurs correspondantes à un jeu de caractères valide.

\$2A Langue du clavier \$00 \$00-\$1F
 Disposition des touches (Codes identiques à la table ci-dessus)
x Valeurs correspondant à un clavier existant

\$2B Bufférisation du clavier \$00 \$00-\$01
 \$00 Non
 \$01 Oui

\$2C Vitesse de répétition du clavier \$03 \$00-\$07
 \$00 4 caractères par seconde
 \$01 8 caractères par seconde
 \$02 11 caractères par seconde
 \$03 15 caractères par seconde
 \$04 20 caractères par seconde
 \$05 24 caractères par seconde
 \$06 30 caractères par seconde
 \$07 40 caractères par seconde

\$2D Délai avant répétition d'une touche \$02 \$00-\$04
 \$00 1/4 de seconde
 \$01 1/2 seconde
 \$02 3/4 de seconde

\$03 1 seconde
\$04 pas de répétition

\$2E Vitesse du double clic \$02 \$00-\$04
\$00 50 tics (un tic = 1 / 60me de seconde)
\$01 40 tics
\$02 30 tics
\$03 20 tics
\$04 10 tics

Durée entre deux clics souris au bout de laquelle on considère qu'il ne s'agit plus d'un double clic, mais de deux clics distincts.

\$2F Vitesse de clignotement du curseur \$02 \$00-\$04
\$00 0 tics (ne clignote pas)
\$01 60 tics
\$02 30 tics
\$03 15 tics
\$04 10 tics

\$30 Shift caps/Lovercase \$00 \$00-\$01
\$00 Non
\$01 Oui

Si la touche de verrouillage majuscule est enfoncée, la touche shift permet d'obtenir les minuscules, les chiffres et les symboles ne sont pas affectés.

\$31 Barre d'espace et touche d'effacement rapide. \$00 \$00-\$01
\$00 Non
\$01 Oui

Permet une répétition accélérée de la touche d'effacement et de la barre d'espace lorsque la touche CONTROL est maintenue enfoncée simultanément.

— > Touches à deux vitesses.

Nota : les flèches obéissent déjà à cette technique.

\$32 Double vitesse \$00 \$00-\$01
\$00 Prendre en compte les paramètres du panneau de contrôle pour la vitesse de répétition des touches à deux vitesses.
\$01 Prendre comme vitesse de répétition la vitesse maximale possible.

\$33	Souris rapide	\$00	\$00-\$01
	\$00	Ne pas prendre en compte la vitesse de déplacement dans le calcul de la position du pointeur de la souris.	
	\$01	Le pointeur souris se déplacera plus loin pour le même mouvement de la souris suivant la vitesse de déplacement de celle ci.	
\$34	Format de la date	\$00	\$00-\$02
	\$00	Mois / Jour / Année	
	\$01	Jour / Mois / Année	
	\$02	Année / Mois / Jour	
	Mois, Jour et Année sur 2 chiffres chacun		
\$35	Format de l'heure	\$00	\$00-\$01
	\$00	sur 12 heures	
	\$01	sur 24 heures	
\$36	Taille minimum du RAM disk	\$00	\$00-\$20
	Valeur = nombre de blocs de 32k		
\$37	Taille maximum du RAM disk	\$00	\$00-\$20
	Valeur = nombre de blocs de 32k		
\$38-\$40	Liste des langues d'affichage disponibles.		
\$41-\$51	Liste des dispositions de clavier disponibles.		
\$52-\$7F	Réservé		
\$80	Numéro de noeud Appletalk		
\$81-\$A1	Variables du système d'exploitation		
\$A2-\$FB	Réservé		
\$FC-\$FF	Somme de contrôle.		
	Si cette somme est invalide, la configuration par défaut est installée		

2.5.2 Lecture Ecriture dans la Bram

On utilise pour cela le Miscellaneous Tool Set.

Quelques Macros:

Tool

MAC

```
LDX #]1
JSL $E10000 ; Appel au dispatcher
```

```
<<<
```

```
WriteBParam MAC
Tool $B03
```

```
<<<
```

```
ReadBParam MAC ; Lecture d'un paramètre en BRAM
Tool $C03 ; attention on doit être en mode natif et 16 bits
```

```
<<<
```

```
PEA $0000 ; Réserve de la place pour le résultat sur la
; pile
```

```
PEA $0020 ; On veut la vitesse $20 en paramètre de la BRAM
```

```
_ReadBParam
```

```
PLA ; Résultat dans A
```

```
; Ecriture d'un paramètre en BRAM
```

```
PEA $0001 ; Vitesse rapide
```

```
PEA $0020 ; vitesse
```

```
_WriteBParam
```

2.6 Les SOFTSWICHES (commutateurs logiciels)

Résidents de \$E0/\$C000 a \$E0/\$C0FF et de \$E1 /\$C000-\$E1 /\$C0FF en émulation Apple II et suivant le registre d'effet Miroir résident aussi en \$00/\$C000-\$C0FF et \$01/\$C000-\$C0FF.

Même remarque pour l'espace des entrées sorties résidant de \$C100- \$CFFF.

Adresse	Opération valide	Description
\$C000	Lecture	Données du clavier bits 7 à 1 si une touche a été pressée Exemple: loop LDA \$C000 ; Attend la pression d'une BPL loop ; touche. Résultat dans A
\$C010	Écriture	

Adresse	Opération valide	Description
\$C001	Écriture	SET80COL permet l'utilisation de la mémoire auxiliaire, on sélectionne la mémoire auxiliaire ou la mémoire principale en écrivant en TXTPAGE1 ou TXTPAGE2 (mémoire auxiliaire) avec le commutateur HIRES off on ne commute que \$400-\$7FF (text page 1) sinon on commute la page graphique 1 (\$2000-\$3FFF) en plus.
\$C002	Écriture	RDMAINRAM lecture à partir de la mémoire principale
\$C003	Écriture	RDCARDRAM lecture à partir de la mémoire auxiliaire
\$C004	Écriture	WRMAINRAM écriture en mémoire principale.
\$C005	Écriture	WRMAINRAM écriture en mémoire auxiliaire.
\$C006	Écriture	SETSLOTcROM valide la ROM se trouvant sur les cartes d'interfaces. (SC100-\$CFFF espace mémoire des entrées sorties). Fonctionne conjointement a RDCxROM
\$C007	Écriture	SETINTcROM valide la ROM interne \$C100-\$CFFF
\$C008	Écriture	SETSTDZP Page Zéro et Pile en mémoire principale
\$C009	Écriture	SETALTZP Page Zéro et Pile en mémoire auxiliaire
\$C00A	Écriture	SETINTC3ROM valide la ROM interne \$C300. (ROM de la carte 80 colonnes)
\$C00B	Écriture	SETSLOTc3ROM valide la ROM de la carte 80 colonnes.
\$C00C	Écriture	CLR80VID désactive l'affichage 80 colonnes. (attention ne désactive que le mode physique non le logiciel qui continue à afficher en 80 colonnes la méthode la plus propre pour repasser en 40 colonnes consiste à demander l'affichage d'un CTRL-(2).
\$C00D	Écriture	SET80VID active l'affichage 80 colonnes (même remarque).
\$C00E	Écriture	CLRALTCHAR lettres minuscules normales Lettres majuscules clignotantes
\$C00F	Écriture	SETALTCHAR Jeu de caractères normaux et inversés pas de lettres clignotantes.
\$C00F	Lecture	idem \$C010
\$C010	Lecture/Écriture	KBDSTRB Remet à zéro le flag Touche enfoncée. Le contenu de \$C000 devient < 128 Exemple: LDA \$C000 ; A =\$C1 (Code ASCII de 'A', la ; touche qui a été pressée) BIT \$C010 LDA \$C000 ; A = \$41

Adresse	Opération valide	Description
\$C011	Lecture	<p>RDLCBNK2 le bit 7 de cette adresse indique le banc mémoire actif de la carte language</p> <p>1 \$D000 bank 2 0 \$D000 bank 1</p> <p>Exemple:</p> <pre> LDA \$C011 BPL bank1 ; le banc 1 est actif dans la ; carte language bank1 ; le banc 2 est actif dans la ;carte language </pre>
\$C012	Lecture	<p>RDLCRAM le bit 7 de cette adresse indique si on lit sur la carte language ou sur la ROM</p> <p>1 Carte language 0 Rom.</p>
\$C013	Lecture	<p>RDRAMRD Le bit 7 indique si on lit dans les 48k de la mémoire auxiliaire.</p> <p>1 Mémoire auxiliaire 0 Mémoire principale</p>
\$C014	Lecture	<p>RDRAMWRT le bit 7 indique si on écrit dans les 48k de la mémoire auxiliaire.</p> <p>1 Mémoire auxiliaire 0 Mémoire principale</p>
\$C015	Lecture	<p>RDCxROM le bit 7 indique l'état du commutateur SLOTcxROM (1 si la mémoire des cartes est utilisée, 0 si la mémoire interne est utilisée). Cet indicateur est utilisé conjointement à SETSLOTcxROM et SETINTcxROM</p> <p>Exemple:</p> <pre> STA SETSLOTcxROM LDA RDCxROM ; A >= \$80 BMI xxx ; le branchement ser a pris STA SETINTcxROM LDA RDCxROM ; A < \$80 BPL xxx ; le branchement sera pris </pre>
\$C016	Lecture	<p>RDALTZP Le bit 7 est à 1 si la page zéro et la pile de la mémoire auxiliaire sont activées. (Commutateur SETALTZP)</p>

Adresse	Opération valide	Description
\$C017	Lecture	RDC3ROM le bit 7 indique l'état du commutateur SLOT3ROM (1 si la mémoire des cartes est utilisée, 0 si la mémoire interne est utilisée). Cet indicateur est utilisé conjointement à SETSLOT3ROM et SETINT3ROM
\$C018	Lecture	RD8OCOL Le bit 7 est à 1 si utilisation de la mémoire auxiliaire pour l'affichage. (Commutateur SET8OCOL/CLR8OCOL)
\$C019	Lecture	RDVBLBAR le bit 7 est à 1 si pas en VBL
\$C01A	Lecture	RDTEXT Le bit 7 est à 1 si on se trouve en mode texte. (Commutateurs TXTCLR/TXTSET)
\$C01B	Lecture	RDMIX Le bit 7 est à 1 si on se trouve en mode mixte texte et graphique. (Commutateurs MIXCLR/MIXSET)
\$C01C	Lecture	RDPAGE2 Le bit 7 est à 1 si on se trouve sur la page 2. (Commutateurs TXTPAGE1 /TXTPAGE2)
\$C01D	Lecture	RDHIRES Le bit 7 est à 1 si on se trouve en mode haute résolution. (Commutateurs HIRES)
\$C01E	Lecture	ALTCHARSET Le bit 7 est à 1 si on affiche dans le jeu de caractères alternatifs, (Commutateurs CLRALTCHAR/SETALTCHAR)
\$C01F	Lecture	RD80VID Le bit 7 est à 1 si l'affichage est en 80 colonnes. (Commutateurs CLR80VID/SET80VID)
\$C020		Réservé
\$C021	Lecture/Écriture	L/ E MONOCOLOR: Bit 0-6 : Réserve Bit 7 : Si bit à 1 on affiche des nuances de gris sinon l'affichage est en couleurs.

Adresse	Opération valide	Description																
\$C022	Lecture/Écriture	<p>TBCOLOR: Registre de couleur du texte.</p> <p>Bit 0-3 Couleur de fond Bit 4-7 Couleur du texte</p> <table> <tr> <td>\$00 Noir</td> <td>\$08 Marron</td> </tr> <tr> <td>\$01 Rouge sombre</td> <td>\$09 Orange</td> </tr> <tr> <td>\$02 Bleu sombre</td> <td>\$0A Gris clair</td> </tr> <tr> <td>\$03 Pourpre</td> <td>\$0B Rose</td> </tr> <tr> <td>\$04 Vert foncé</td> <td>\$0C Vert pâle</td> </tr> <tr> <td>\$05 Gris sombre</td> <td>\$0D Jaune</td> </tr> <tr> <td>\$06 Bleu</td> <td>\$0E Aquamarine</td> </tr> <tr> <td>\$07 Bleu clair</td> <td>\$0F Blanc</td> </tr> </table>	\$00 Noir	\$08 Marron	\$01 Rouge sombre	\$09 Orange	\$02 Bleu sombre	\$0A Gris clair	\$03 Pourpre	\$0B Rose	\$04 Vert foncé	\$0C Vert pâle	\$05 Gris sombre	\$0D Jaune	\$06 Bleu	\$0E Aquamarine	\$07 Bleu clair	\$0F Blanc
\$00 Noir	\$08 Marron																	
\$01 Rouge sombre	\$09 Orange																	
\$02 Bleu sombre	\$0A Gris clair																	
\$03 Pourpre	\$0B Rose																	
\$04 Vert foncé	\$0C Vert pâle																	
\$05 Gris sombre	\$0D Jaune																	
\$06 Bleu	\$0E Aquamarine																	
\$07 Bleu clair	\$0F Blanc																	
\$C023	Lecture/Écriture	<p>VGCINT registre d'interruption du contrôleur vidéo</p> <p>Bit 0 Réserve</p> <p>Bit 1 Autorise interruption de fin de ligne</p> <p>Bit 2 Autorise les interruptions de 1 seconde</p> <p>Bit 3 Réserve</p> <p>Bit 4 Réserve</p> <p>Bit 5 Status de l'interruption de fin de ligne</p> <p>Bit 6 Status de l'interruption de 1 seconde</p> <p>Bit 7 Status de l'interruption VGC</p>																
\$C024	Lecture	<p>MOUSEDATA Registre de donnée de la souris. Les données générées par la souris lors des déplacements ainsi que l'état du bouton sont accessibles via ce registre. Ce registre doit être lu deux fois successivement, la première lecture renvoyant la coordonnée en Y la seconde celle en X.</p> <p>Bit 7: État du bouton (0 si le bouton enfoncé)</p> <p>Bit 6 : Variation de mouvement (si 1, déplacement négatif)</p> <p>Bit 5-0 : Mouvement relatif de la souris</p>																

Adresse	Opération valide	Description
\$C025	Lecture	<p>KEYMODREG: Registre de status des touches mortes du clavier. (Shift, Control, Pomme et CapsLock)</p> <p>Bit 7: Á 1 si la touche pomme a été enfoncée. Bit 6: Á 1 si la touche Option a été enfoncée. Bit 5: Á 1 si ce registre a été modifié. Bit 4: Á 1 si une touche du pavé numérique a été enfoncée. Bit 3: Á 1 si une touche est enfoncée. Bit 2: Á 1 si la touche CapsLock a été enfoncée. Bit 1: Á 1 si la touche Contrôle a été enfoncée. Bit 0: Á 1 si la touche Shift a été enfoncée.</p>
\$C026	Lecture/Écriture	<p>DATAREG: Registre de donnée/commande du clavier. Lors d'une interruption ce registre est défini de la façon suivante:</p> <p>Bit 7: Á 1 si un octet de réponse, sinon octet de donnée. Bit 6: Á 1 ABORT valide tous les autres bits de ce registre à 0. Bit 5: Á 1 si on a appuyé sur CTRL-POMME-RESET Bit 4: Á 1 si on a appuyé sur une séquence de vidange clavier. (CTRL-POMME-DEL) Bit 3: Á 1 si SRQ valide Bit 2-0: Nombre d'octets de données reçus (doit valoir 1, si 0 pas de données valides).</p>
\$C027	Lecture/Écriture	<p>KMSTATUS registre de status de l'ADB</p> <p>Bit 7: Á 1 si le registre de données de la souris est plein. Bit 6: Á 1 si les interruptions de la souris sont autorisées Bit 5: Á 1 si le registre DATAREG contient des données valides. Bit 4: Á 1 si une interruption est générée lorsque le contenu du registre DATAREG est valide. Bit 3: Á 1 si le registre de données du clavier est plein. Bit 2: Á 1 si une interruption est générée lorsque le contenu du registre de données du clavier est valide Bit 1: Á 1 si coordonnée X de la souris dans MOUSEDATA à 0 si coordonnée Y Bit 0 : à 1 si le registre de DATAREG est plein</p>
\$C028		<p>ROMBANK: Bascule de sélection de banc de ROM (inutilisé sur l'Apple IIGS)</p>

Adresse	Opération valide	Description
\$C029	Lecture/Écriture	<p>WVIDEO: Ce registre contrôle les capacités vidéo supplémentaire de l'Apple IIGS.</p> <p>Bit 0-4 : Réserve ne pas modifier</p> <p>Bit 5: Si ce bit est à 0 la Double Haute Résolution est en couleur (140 par 192 en 16 couleurs) sinon la double haute résolution est en monochrome (560 par 192)</p> <p>Bit 6: Si ce bit est à 0 la carte mémoire des 128k est la même que sur l'Apple IIe (nécessaire pour utiliser la Double Haute Résolution) Si ce bit est à zéro. Le buffer vidéo devient une seule et même zone mémoire contiguë linéaire de \$2000 à \$9D00 dans le bank \$E1</p> <p>Bit7: Si ce bit est à 0 tous les modes graphiques Apple II sont validés. Si ce bit est à 1 tous les modes graphiques Apple II sont inhibés. L'état du bit 6 est (considéré comme à 1, mémoire linéaire) La super résolution est en action.</p>
\$C02A		Réserve
\$C02B	Lecture/Écriture	<p>LANGSEL: Registre contrôlant le générateur de caractères.</p> <p>Bits 7-5 : Sélection de la langue du générateur de caractères:</p> <ul style="list-style-type: none"> 0: anglais (USA) 1: anglais (Uk) 2: Français 3 : Danois 4 : Espagnol 5 : Italien 6 : Allemand 7 : Suédois <p>Bit 4: À 1 si mode PAL sinon sortie NTSC</p> <p>Bit 3: À 0 si jeu de caractères primaire sélectionné.</p> <p>Bits 2-0 : Réserve à 0 .</p>
\$C02C	Lecture/Écriture	CHARROM adresse pour les tests de lecture de la rom caractère.

Adresse	Opération valide	Description
\$C02D	Lecture/Écriture	<p>SLTROMSEL</p> <p>Bits 0-4: Réservé, ne pas modifier</p> <p>Bit 7: À 1 valide la carte du slot 7. 0 valide la ROM Appletalk</p> <p>Bit 6: À 1 valide la carte du slot 6. 0 valide la Rom des lecteurs 5.25</p> <p>Bit 5: À 1 valide la carte du slot 5. 0 valide la ROM des lecteurs 3.5</p> <p>Bit 4: À 1 valide la carte du slot 4. 0 valide la ROM de la souris</p> <p>Bit 3: Réservé ne pas modifier</p> <p>Bit 2: À 1 valide la carte du slot 2. 0 valide la Rom de la sortie série 2</p> <p>Bit 1: À 1 valide la carte du slot 1. 0 valide la Rom de la sortie série 1</p> <p>Bit 0: Réservé, ne pas modifier</p>
\$C02E	Lecture	VERTCNT: Adresse pour la lecture des bits du contrôleur vidéo V5-VB
\$C02F	Lecture	HORIZCNT: Adresse pour la lecture des bits contrôleur vidéo VA-HO
\$C030	Lecture/Écriture	SPKR: L'accès à ce registre produit un clic du haut parleur.
\$C031	Lecture/Écriture	<p>DISKREG: Registre d'interface disque</p> <p>Bit 7: À 1 sélectionne la tête 1 sur le disque 3.5 À 0 sélectionne la tête 0</p> <p>Bit 6: À 1 on utilise un lecteur 3.5" À 0 un lecteur 5.25"</p> <p>Bits 5-0: Réservés.</p>
\$C032	Lecture/Écriture	<p>SCANINT: Registre de remise à zéro des interruptions VGC.</p> <p>Bit 7: Réservé</p> <p>Bit 6: Remise à zéro des ITs 1 secondes</p> <p>Bit 5: Remise à zéro des ITs fin de ligne</p> <p>Bits 4-0: Réservés.</p> <p>Pour remettre à zéro les interruptions, il suffit de mettre à zéro le bit.</p>
\$C033	Lecture/Écriture	CLOCKDATA: Registre de donnée de l'horloge.

Adresse	Opération valide	Description
\$C034	Lecture/Écriture	<p>CLOCKCTL: Permet de sélectionner la couleur du bord et de contrôler l'horloge</p> <p>Bits 0-3: Couleur du bord</p> <p>Bit 5: Une fois le transfert du dernier octet, ou sa lecture, ce bit doit être mis à 1. Ce bit doit être mis à 0 avant tout transfert avec l'horloge.</p> <p>Bit 6 : Mettre ce bit à 1 avant une lecture de l'horloge, et à 0 avant une écriture</p> <p>Bit 7 : Ce bit doit être mis à 1 avant toute opération avec l'horloge. Celle-ci le met à 0 automatiquement à la fin de l'opération.</p>
\$C035	Lecture/Écriture	<p>SHADOW: Registre shadowing</p> <p>Bit 0: Pas de shadowing page text 1 si le bit est à 1</p> <p>Bit 1: Pas de shadowing Hires page 1 si le bit est à 1</p> <p>Bit 2: Pas de shadowing Hires page 2 si le bit est à 1</p> <p>Bit 3: Pas de shadowing Buffer super hires si le bit est à 1</p> <p>Bit 4: Pas de shadowing bank auxiliaire hires si le bit est à 1</p> <p>Bit 5: Réserve</p> <p>Bit 6: Pas de shadowing pour la zone E/S si le bit est à 1</p> <p>Bit 7: Réserve</p>
\$C036	Lecture/Écriture	<p>CYAREG registre de contrôle de la vitesse</p> <p>Bit 0: slot 4 si bit à 1 vitesse lente sur motor-on</p> <p>Bit 1: Slot 5</p> <p>Bit 2: Slot 6</p> <p>Bit 3: Slot 7</p> <p>Bit 4: si bit à 1 autorise les shadowing dans tous les bancs \$00-\$7F, à 0 seulement dans les bancs \$00 et \$01</p> <p>Ne pas utiliser le mode tous bancs.</p> <p>Bit 5:</p> <p>Bit 6: Réserve</p> <p>Bit 7: si bit à 1 vitesse rapide</p> <p>Bits 0-3 si une adresse motor-on (\$C0F9, \$C0E9 \$COD9, \$C0C9) est accédée la vitesse passe automatiquement à 1Mhz (vitesse lente compatible avec les périphériques Apple II) et sur l'accès à une adresse motor-off (\$C0F8, \$C0E8, \$COD8, \$C0C8) la vitesse revient à sa valeur précédente. Utilisé pour permettre l'utilisation des unités de disquettes 5 pouces ne fonctionnant qui à 1MHz.</p>
\$C037	Lecture/Écriture	<p>DMAREG: Utilisé lors des accès DMA comme adresse de banc.</p>

Adresse	Opération valide	Description
\$C038	Lecture/Écriture	SCCBREG: Registre de commande du port série numéro 2
\$C039	Lecture/Écriture	SCCAREG: Registre de commande du port série numéro 1
\$C03A	Lecture/Écriture	SCCBDATA: Registre de donnée du port série numéro 2
\$C03B	Lecture/Écriture	SCCADATA: Registre de donnée du port série numéro 1
\$C03C	Lecture/Écriture	SOUNDCTL : Registre de contrôle du son. Bit7: à 1 le DOC est occupé ce bit doit être à zéro pour pouvoir travailler avec le DOC Bit6 : à 1 tous les accès sont dans la ram dédiée de 64k si bit à 0 on accède aux registres du DOC. Bit 5 : à 1 on autorise l'auto incrémentation des adresses. Bit 4 : Réserve ne pas modifier Bits 3-0: Volume sonore de 0 à \$F (\$F le plus fort).
\$C03D	Lecture/Écriture	SOUNDDATA: Registre de donnée du DOC. Données venant de la Ram de 64k ou des registres du DOC suivant le bit 6 du registre précédent.
\$C03E	Lecture/Écriture	SOUNDADRL: Poids faible de l'adresse dans la ram du DOC
\$C03F	Lecture/Écriture	SOUNDADRH: Poids fort de cette même adresse.
\$C040		Réserve
\$C041	Lecture/Écriture	INTEN Bit 7-5 : Réserve. Bit 4 : à 1 autorise les interruptions quart de seconde Bit 3 : à 1 autorise les interruptions VBL Bit 2 : à 1 autorise les interruptions boutons de la souris en mode Apple II Bit 1 : à 1 autorise les interruptions mouvements de la souris en mode Apple II Bit (Mega 2)
\$C042		Réserve
\$C043		Réserve
\$C044	Lecture	MMDELTA X variation du mouvement de la souris mode Apple 2 axe des X. (complément à 2)
\$C045	Lecture	MMDELTA Y variation du mouvement de la souris mode Apple 2 axe des Y. (complément à 2)
\$C046	Écriture	DIAGTYPE Bit 7: 1 si Diagnostics ROM

Adresse	Opération valide	Description
\$C047	Lecture	INTFLAG: Bit 7: à 1 si le bouton de la souris est enfoncé Bit 6: à 1 si le bouton de la souris était enfoncé lors de la dernière lecture Bit 5: Status de la sortie AN3 Bit 4: à 1 s'il y a eu une interruption quart de seconde Bit 3: à 1 s'il y a eu une interruption VBL Bit 2: à 1 s'il y a eu une interruption due à la pression du bouton de la souris Bit 1: à 1 s'il y a eu une interruption due à un déplacement de la souris
\$C047	Écriture	CLRVLINT: Réinitialise l'interruption VBL
\$C048	Écriture	CLRXINT: Réinitialise les interruptions de la souris mode Apple 2.
\$C049		Réservé
\$C04A		Réservé
\$C04B		Réservé
\$C04C		Réservé
\$C04E		Réservé
\$C04F		Réservé
\$C050	Lecture/Écriture	TXTCLR: Sélectionne le mode graphique «standard» Apple II
\$C051	Lecture/Écriture	TXTSET: Sélectionne le mode texte.
\$C052	Lecture/Écriture	MIXCLR: Positionne en mode plein graphique.
\$C053	Lecture/Écriture	MIXSET: Positionne en mode mixte graphisme plus 4 lignes de texte.
\$C054	Lecture/Écriture	TXTPAGE1: Sélectionne la page 1 texte ou graphique
\$C055	Lecture/Écriture	TXTPAGE2: Sélectionne la page 2 ou si on a utilisé SET80COL (mémoire auxiliaire pour l'affichage) la page 1 en mémoire auxiliaire.
\$C056	Lecture/Écriture	LORES: Sélectionne le mode graphique basse résolution.
\$C057	Lecture/Écriture	HIRES: Sélectionne le mode haute résolution ou si on a utilisé SETAN3 la Double Haute résolution Exemple: STA SETAN3 ; passe en double haute STA HIRES

Adresse	Opération valide	Description
\$C058	Écriture	SETANO: Positionne l'annunciator 0
\$C059	Écriture	CLRANO: Efface l'annunciator 0
\$C05A	Écriture	SETAN1: Positionne l'annunciator 1
\$C05B	Écriture	CLRAN1: Efface l'annunciator 1
\$C05C	Écriture	SETAN2: Positionne l'annunciator 2
\$C05D	Écriture	CLRAN2: Efface l'annunciator 2
\$C05E	Lecture/Écriture	SETAN3: autorise la double haute résolution.
\$C05F	Lecture/Écriture	CLRAN3: inhibe le mode double haute résolution.
\$C060	Lecture	BUTN3: Lire le bouton 3 du joystick
\$C061	Lecture	BUTN0: Lire le bouton 0 du joystick
\$C062	Lecture	BUTN1: Lire le bouton 1 du joystick
\$C063	Lecture	BUTN2: Lire le bouton 2 du joystick
\$C064	Lecture	Lecture paddle 0
\$C065	Lecture	Lecture paddle 1
\$C066	Lecture	Lecture paddle 2
\$C067	Lecture	Lecture paddle 3

Adresse	Opération valide	Description
\$C068	Lecture/Écriture	<p>STATEREG: Registre regroupant 8 commutateurs communément utilisés sur l'Apple II.</p> <p>Bit 7: 1 si diagnostics ROM</p> <p>Bit 0: INTCXROM \$C007,\$C006, Si bit à 1 la rom interne en SCx00 est sélectionnée.</p> <p>Bit 1: ROMBANK Ce bit doit toujours être à 0. Ne pas modifier.</p> <p>Bit2: LCBNK2 \$C083,\$C011 Si bit à 1 Bank 1 de la carte langage sélectionné. Bank 2 sinon.</p> <p>Bit 3: RDROM \$C080,\$C012 Si bit à 1 la ROM de la carte langage est validée. Sinon, il s'agit de la RAM.</p> <p>Bit 4: RAMWRT \$C009,\$C008 Si bit à 1, la RAM auxiliaire est validée en écriture.</p> <p>Bit 5: RAMRD \$C009,\$C008 Si bit à 1 la RAM auxiliaire est validée en lecture.</p> <p>Bit6: PAGE 2 \$C054,\$C055 Si bit est à 1, la page 2 est sélectionnée.</p> <p>Bit 7: ALTZP \$C009,\$C016 Si ce bit est à 1 la page zéro et la pile se trouve en mémoire principale.</p>
\$C069		Réservé
\$C06A		Réservé
\$C06B		Réservé
\$C06C		Réservé
\$C06D	Lecture/Écriture	TESTREG: Registre de mode de test
\$C06E	Écriture	CLRTM: Invalide le mode test.
\$C06F	Écriture	ENTM: Valide le mode test.
\$C070	Écriture	PTRIG: Réinitialise la lecture des Paddles
\$C071	Lecture	Code de traitement des interruptions.
\$C07F	Lecture	<p>RDDHIRES :</p> <p>Le bit 7 est à 1 si le mode double haute résolution est actif. (Commutateurs SETAN3/CLRAN3)</p>
\$C080	Lecture	Lire cette adresse pour passer en lecture RAM de la carte langage utiliser \$D000 bank 2 et protéger la RAM en écriture.

Adresse	Opération valide	Description
\$C081	Lecture/Écriture	ROMIN: Lire deux fois cette adresse pour lire la ROM (carte langage) valider en écriture la RAM en utilisant \$D000 bank 2.
\$C082	Lecture	Lire cette adresse pour valider la ROM en lecture empêcher réécriture en RAM et utiliser \$D000 bank 2
\$C083	Lecture/Écriture	Lire deux fois cette adresse pour valider la RAM en lecture et en écriture avec pour \$D000 bank 2
\$C084	Lecture	Idem \$C080
\$C085	Lecture/Écriture	Idem \$C081
\$C086	Lecture	Idem \$C082
\$C087	Lecture/Écriture	Idem \$C083
\$C088	Lecture	Lire cette adresse pour passer en lecture RAM de la carte langage utiliser \$D000 bank 1 et protéger la RAM en écriture.
\$C089	Lecture/Écriture	Lire deux fois cette adresse pour lire la ROM (carte langage) valider en écriture la RAM en utilisant \$D000 bank 1.
\$C08A	Lecture	Lire cette adresse pour valider la ROM en lecture, empêcher l'écriture en RAM et utiliser \$D000 bank 1
\$C08B	Lecture/Écriture	Lire deux fois cette adresse pour valider la RAM en lecture et en écriture avec pour \$D000 bank 1
\$C08C	Lecture	Idem \$C088
\$C08D	Lecture/Écriture	Idem \$C089
\$C08E	Lecture	Idem \$C08A
\$C08F	Lecture/Écriture	Idem \$C08B

(*) Commutateurs utiles généralement seulement en mode émulation Apple 2.

Zone mémoire	Description
\$C090-\$C09F	Commutateurs réservés au slot 1
\$C0A0-\$C0AF	Commutateurs réservés au slot 2
\$C0B0-\$C0BF	Commutateurs réservés au slot 3
\$C0C0-\$C0CF	Commutateurs réservés au slot 4
\$C0D0-\$C0DF	Commutateurs réservés au slot 5
\$C0E0-\$C0EF	Commutateurs réservés au slot 6
\$C0F0-\$C0FF	Commutateurs réservés au slot 7

ROM des cartes d'extensions, la ROM en présence dépend de \$CO2D SLTROMSEL

Zone mémoire	Opération	Description
\$C100-\$C1FF	Lecture	Rom carte slot 1 ou ROM interne
\$C200-\$C0AF	Lecture	Rom carte slot 2 ou ROM interne
\$C300-\$C0BF	Lecture	Rom carte slot 3 ou ROM interne
\$C400-\$C0CF	Lecture	Rom carte slot 4 ou ROM interne
\$C500-\$C0DF	Lecture	Rom carte slot 5 ou ROM interne
\$C600-\$C0EF	Lecture	Rom carte slot 6 ou ROM interne
\$C700-\$C7FF	Lecture	Rom carte slot 7 ou ROM interne
\$C800-\$CFFF	Lecture	ROM carte ou ROM interne
\$CFFF	Lecture	Invalide la ROM en \$C800



Drives

3.0 Les mémoires de masse

Dans le GS on trouve principalement quatre types de mémoires de masse. Les disques Ram et Rom, les unités de disques souples ou floppy et les disques durs. Ces mémoires

sont gérées, plus ou moins bien, par le système d'exploitation disque soit DOS, PRODOS ou GS/OS ou etc...

Dans ce chapitre nous allons tenter de décrire les différents moyens d'accès aux mémoires de masse de l'Apple IIgs.

Le GS peut supporter quatre types de lecteurs de disquettes. Les deux premiers, acceptent des disquettes 3.5", d'une capacité de 800KB. Les deux autres, des disquettes 5.25", d'une capacité de 140KB par face.

Les deux versions 3.5" se différencient principalement par la présence ou non d'un processeur à l'intérieur qui les rend plus ou moins "intelligents". Il s'agit des Apple 3.5 (sans processeur) et des UniDisk 3.5 (avec processeur). Seuls les UniDisk 3.5 peuvent fonctionner indifféremment sur l'Apple IIe, IIc et IIgs.



En ce qui concerne les lecteurs 5.25" il s'agit soit du DuoDisk ou de l'UniDisk 5.25" hérité de l'Apple II.



3.1 Quelques rappels sur l'organisation des disques

3.1.1 DOS 3.3

Le DOS 3.3 (à plus forte raison les versions antérieures) utilise uniquement les disquettes 5.25" (l'utilisation des disquettes 3.5" nécessitant un DOS modifié non fourni par Apple).

Le formatage (opération consistant à préparer le support magnétique à recevoir des données en le divisant en pistes et secteurs) est contrairement à certaines machines (suivez mon regard) réalisé en grande partie par voie logicielle.

La structure d'une disquette 5.25" après formatage est la suivante :

35 pistes numérotées de 0 à 34, chacune composée de 16 secteurs eux mêmes numérotés de 0 à 15. Chaque secteur contient 256 octets d'information utile. On obtient ainsi une capacité de 4 kilooctets par piste soit 140 k par disquette (rappelons qu'une seule face de la disquette est généralement utilisée, mais qu'il est possible de gagner 140K de plus en retournant simplement la disquette)

Chaque piste sur le disque est formée d'une suite d'octets représentant les données.

Ces octets doivent satisfaire certaines conditions propres à la logique du format:

- Bit de poids fort à 1
- Au plus une paire de bits consécutifs à zéro.

Le format d'une piste est le suivant:

Gap1

Zone composée d'un certain nombre d'octets \$FF «synchronisés» permettant le remplissage de la piste (12 à 85 suivant la piste).

Secteurs 0 à 15:

Chacun de ces secteurs se décomposent de la façon suivante

Champ adresse :

- Marque d'adresse Prologue soit D5 AA 96
- Volume soit deux octets
- Numéro de piste soit deux octets
- Numéro de secteur soit deux octets
- Somme de contrôle soit deux octets
- Marque d'adresse Epilogue soit DE AA EB

Gap 2 :

Idem Gap 1 mais généralement 5-10 octets à \$FF

Champ de donnée:

- Marque de donnée Prologue soit D5 A A A D
- 342 octets de données

- Somme de contrôle soit un octet
- Marque de donnée Epilogue soit DE A A EB

Gap 3:

Idem Gap 2 longueur 16-28

Les informations de volume, piste, secteur ainsi que le checksum (somme de contrôle) sont encodées sur deux octets suivant un codage appelé 4 4.

Soit d7d6d5d4d3d2d1d0 les bits de l'octet à encoder suivant la méthode 4 4, en sortie on obtient 1d71d51d31d1 et 1d61d41d21d0 deux octets satisfaisant aux conditions exprimées.

Exemple : pour la piste 00 on a les octets \$AA et \$AA soit 1010 1010 et 1010 1010, pour le secteur \$F on a les octets \$AF \$AF soit 1010 1111 et 1010 1111.

La somme de contrôle est calculée de telle façon que le ou exclusif de toutes les informations, elle même y compris, donne zéro.

Les données sont elles encodées au format 62. Trois octets de données utiles donnant quatre octets sur disque.

Les octets de synchronisation sont des octets écrits de telle façon que à la suite des huit bits les composants on trouve deux bits à zéro. Ce sont des octets à dix bits.

Exemple un \$FF synchronisé sera composé sur le disque de :

1111 1111 00 Le contrôleur ne pouvant lire que les octets commençant par un bit à 1, ces bits permettent à celui-ci de se synchroniser lors de la lecture, et ainsi de déterminer le début d'une suite d'octets valides.

Je ne vais pas m'appesantir sur l'organisation des fichiers sous DOS, ce système étant maintenant remplacé par ProDOS et GS/OS. Passons à la suite: ProDOS.

3.1.2 ProDOS, GS/OS

ProDOS, contrairement au DOS supporte plusieurs types d'unités de stockage. Celle ci sont des disques durs, des unités de disquettes 3.5" et 5.25", des cartes mémoire RAM et ROM. La base de stockage pour ces différentes unités à été normalisée et est le bloc. Un bloc contient exactement 2 secteurs d'une disquette 5.25" soit 512 octets.

On a donc les capacités suivantes en blocs:

Type d'unité	Capacité (en blocs)
Disquette 5.25"	280

Type d'unité	Capacité (en blocs)
Disquette 3.5"	1600
Carte mémoire RAM (1MB)	2048
ROM disk	1024

Le format des disquettes 5.25" est exactement le même que sous DOS, seulement l'organisation des fichiers change. Par contre, le format des disquettes 3.5" diffère largement :

Les deux faces de la disquette sont utilisées. Chacune contient 80 pistes, composées d'un nombre variable de secteurs en fonction de la dimension de la circonférence de la piste (plus on s'éloigne du centre, plus elle augmente):

Pistes	Nombre de secteurs
0 à 15	12
16 à 31	11
32 à 47	10
48 à 63	9
64 à 79	8

Chaque secteur a la structure suivante :

Marque d'adresse : D5 A A 96
 Numéro de piste sur un octet codé en 62
 Numéro de secteur «>»
 Numéro de la face
 Type de format sur un octet
 Checksum
 Marque de fin DE AA

 Gap de 5-10 octets FF synchronisés

 Marque de donnée : D5 AA AD
 Numéro du secteur sur un octet
 Données
 Checksum
 Marque de fin DE AA FF
 Gap

Pour l'organisation des fichiers sur les disquettes, disque ProDOS et GS/OS se reporter à la partie sur le GS/OS.

3.2 L'accès via Prodos - GS /OS

Cette méthode est celle à préférer, car la plus indépendante du périphérique. En effet on accède exactement de la même façon à une carte RAM émulant un disque, qu'à un disque dur.

De cette manière on accède à des BLOCS, soit en général 512 octets, chaque périphérique étant divisé en un certain nombre de blocs.

3.2.1 ProDOS 8

Deux commandes du MLI (Machine Language Interface, ou Interface langage machine) permettent d'accéder aux blocs:

READ BLOCK	code \$80	Lecture d'un bloc.
WRITE BLOCK	code \$81	Ecriture d'un bloc

READ BLOCK :
WRITE BLOCK :

Ces fonctions appellent le «device handler» d'une unité donnée pour la lecture ou l'écriture d'un bloc de 512 octets. L'appel à cette fonction est équivalent à l'appel direct du «device handler» en ayant respecté les conditions d'entrées suivantes :

- La validité du buffer est vérifiée.
- Les interruptions sont inhibées
- Le numéro d'unité est vérifié.
- La carte langage est validée, et restaurée dans son état initial.

Liste de paramètres:

+0 \$03
+1 Numéro de l'unité sous la forme DSSS0000
(où D est le numéro de Drive et SSS le numéro de slot).
+2/3 Adresse du buffer de donnée
+4/4 Numéro du bloc.

Code erreur retourné dans l'accumulateur. Le bit de carry est positionné en cas d'erreur.

Code	Description de l'erreur
\$00	Pas d'erreur

Code	Description de l'erreur
\$04	Nombre de paramètres différent de \$03
\$27	Erreur d'E/S ou numéro de bloc invalide
\$28	Pas de périphérique correspondant cette unité
\$2B	Le disque est protégé en écriture
\$56	Buffer invalide

Exemple :

MLI	EQU	\$BF00	
	JSR	MLI	
	DFB	\$80	; Lecture du bloc \$0000 du drive 1 slot 6
	DW	parm_liste	
	BCS	erreur	
	JSR	MLI	
	DFB	\$81	; Écriture du bloc \$0000 du drive 1 slot 6
	DW	parm_liste	
	BCS	erreur	
parm_liste	DFB	\$03	
	DFB	\$60	
	DW	donnée	
	DW	\$0000	
donnée	DS	512	

3.2.2 GS/OS

Quatre fonctions permettent l'accès aux unités de stockage:

- \$202CDInfo: Permet d'obtenir des informations a propos d'un Device connecté au système.
- \$202F DRead: Permet de lire un bloc ou plus.
- \$202DDStatus: Pour déterminer l'état d'un Device GS/OS
- \$2030 DWrite: Permet d'écrire un bloc ou plus.
E Paramètres en entrée.
R Paramètres en retour.

Index	Taille		Paramètres de DInfo
0	2	E	Nombre de paramètres (10)
2	2	E	Numéro de référence du dispositif (device)
4	4	R	Pointeur sur le buffer devant recevoir le nom du dispositif (device)
8	2	R	Caractéristiques du dispositif (device) Bit 15: Si ce bit vaut 1, 'il s'agit d'un disque RAM ou ROM Bit 14: Le bit est à 1 si un «device driver» a été généré Bit 13: Réserve Bit 12: Le bit est à 1 si le dispositif est occupé Bit 11: Réserve Bit 10: Réserve Bit 9-8: Vitesse à laquelle peut fonctionner le dispositif 00 1 Mhz 01 2.6 Mhz 10 > 2.6 Mhz 11 Vitesse sans importance. Bit 7: Si 1, le dispositif utilise des blocs. Bit 6: Si 1, l'écriture est autorisée Bit 5: Si 1, la lecture est autorisée Bit 4: Réserve Bit 3: Si 1, le formatage est autorisé Bit 2: Si 1, le support du dispositif est amovible (ex une disquette pour une unité de disquette) Bit 1: Réserve Bit 0: Réserve
10	4	R	Capacité du Device (en nombre de blocs)
14	2	R	Numéro du slot du Device
16	2	R	Numéro d'unité du Device pour le Smartport.
18	2	R	Numéro de version du «device driver»
20	2	R	Numéro d'identification du Device Exemples: \$0000 Disquette 5.25" \$0003 Disquette 3.5"
22	2	R	Premier numéro de dispositif dans la liste
24	2	R	Numéro suivant de dispositif

La liste contient tous les numéros de dispositifs en rapport, comme par exemple les partitions d'un disque dur.

Pour montrer le fonctionnement de DInfo, voyons l'exemple suivant où nous allons afficher la liste de tous les dispositifs en ligne:

```
                LDA    #$01
Encore          STA    NumDev
                _DInfo part
                BCS    Termine

                LDA    NomDev      ; Transforme le mot de longueur en octet
                XBA
                STA    NomDev

                PushPtr _NomDev+1 ; Début de la chaîne
                _DrawString      ; Invoque la fonction Quickdraw pour imprimer la
                                ; chaîne sur l'écran
                JSR    CRLF      ; Passe à la ligne suivante

                INC    NumDev      ; Tente le numéro suivant
                BRA    Encore

Termine        RTS

parm          DW    10

NumDev        DW    1
                DDW    NOM
                DDW    0
                DDW    0
                DW    00
                DW    00
                DW    00
                DW    00
                DW    00
                DW    00
nom           DW    35      ; un buffer de 35 octets pour le nom
NomDev        DS    33
```

Index	Taille		Paramètres de DRead/DWrite
0	2	E	Nombre de paramètres (6)
2	2	E	Numéro de référence du dispositif (device)
4	4	R	Pointeur sur le buffer
8	4	E	Nombre d'octets à lire/écrire
12	4	E	Bloc de départ
16	2	E	Nombre d'octets par bloc
18	4	R	Nombre d'octets réellement lus/écrits

Index	Taille		Paramètres de DStatus
0	2	E	Nombre de paramètres (5)
2	2	E	Numéro de référence du dispositif (device)
4	2	E	Type de requête \$0000 Status du dispositif \$0001 Paramètre de configuration \$0002 Wait / no wait \$0003 Options de formatage \$0004 Status de la partition
6	4	R	Pointeur vers le résultat
10	4	E	Taille théorique du résultat
14	4	E	Taille réelle du résultat

3.3 L'accès via le Smartport

Le logiciel du SmartPort correspond à la ROM interne du slot 5. Il est composé d'une série de routines contrôlant des périphériques au format caractère (imprimantes, modem ...) ou au format bloc (disque dur, disquette, ...) connectés à la prise disquette du IIGS (ou insérés dans un slot). Ce logiciel transforme les appels en un format intelligible par les unités intelligentes (unités pouvant interpréter des flots de commandes circulant sur le Bus du SmartPort, comme par exemple l'UniDisk 3.5). De même le SmartPort fournit une interface pour plusieurs Devices inintelligentes comme l'Apple 3.5, les RAM cards, les disques RAM et ROM.

On trouve aussi une version du SmartPort dans l'Apple IIc et IIe pour le contrôle de la carte mémoire (extension) de 1Mo et de l'UniDisk 3.5.

L'appel du SmartPort ressemble à un appel au MLI de ProDOS/8. Chaque séquence d'appel consiste en un JSR vers l'entrée du SmartPort suivi par l'octet de commande du SmartPort et l'adresse de la liste de paramètres.

Exemple:

SR	SmartPort
DFB	\$00 ; Status
DW	Paris

Les appels au SmartPort existent sous deux formes, une version ne permettant les accès que dans le banc \$00 (version compatible avec le SmartPort sur Ile et Ilc) et une version étendue permettant l'accès à toute la mémoire.

3.2.1 Localisation du SmartPort et de l'adresse d'appel.

On peut déterminer la présence du SmartPort en examinant les octets de signature des roms des cartes.

\$Cn01 doit être égal à \$20
\$Cn03 doit être égal à \$00
\$Cn05 doit être égal à \$03
\$Cn07 doit être égal à \$00

Avec n le numéro du Slot, soit pour le logiciel SmartPort en slot 5 du CS, \$C501, \$C503, \$C505 et \$C507. En effet si on examine à partir du moniteur la rom en \$00/ C500 on obtient la séquence suivante :

\$00/C500	A2 20	LDX #\$20
\$00/C502	A2 00	LDX #\$00
\$00/C504	A2 03	LDX #\$03
\$00/C506	C9 00	CMP #\$00

Donnant bien les valeurs caractérisant le SmartPort en \$C501, \$C503, \$C505, \$C507.

L'adresse d'appel est calculée de la façon suivante : On additionne la valeur trouvée en \$CnFF à \$Cn00 (ce qui nous donne le point d'entrée du «Device handler» de ProDOS /8) et on ajoute 3.

Exemple :

\$C5FF = \$0A ce qui donne comme point d'entrée du Device Handler ProDOS/8 \$C50A et comme point d'entrée du SmartPort \$C50D.

3.2.2 Liste des commandes SmartPort

Il y a deux types de commandes pour le SmartPort, les commandes normales (héritées de l'Apple IIe, IIc) et les commandes étendues tirant avantage des nouvelles possibilités d'adressage du 65C816 (plus de mémoire, adressage 16 bits, etc...). Généralement chaque commande existe sous les deux formes.

Chaque appel au SmartPort à la forme d'un appel au MLI de ProDOS/8, soit un JSR au point d'entrée du SmartPort suivi du code de la commande, et d'un pointeur sur la liste de paramètres de la commande.

Pour les appels normaux, le pointeur est sur deux octets et représente une adresse dans le banc 00. Pour les appels étendus le pointeur est codé sur 4 octets et référence une liste de paramètres dans n'importe quel banc.

Le code d'un appel étendu est égal au code d'un appel normal plus \$40.

Exemple:

Status normal \$00
Status étendu \$40

Les appels au SmartPort subissent les contraintes suivantes :

On ne peut rien transférer vers ou de la zone se trouvant dans la Page Zéro du mode émulation. (soit \$00/0000 à \$00 / 00FF)

- On doit disposer d'un minimum de 35 octets de pile libre avant l'appel du SmartPort.
- On ne peut appeler le SmartPort qu'à partir du mode émulation du 65C816.

En cas d'erreur lors de l'exécution d'une commande le bit carry est positionné et un code erreur est transmis dans l'accumulateur. Lors de l'exécution correcte d'une fonction le bit carry est effacé et l'accumulateur contient 0.

Si des données ont été transférées d'un Device au CPU on trouve dans les registres X et Y leur nombre (poids fort en Y)

Note: Tous les buffers doivent être alloués par l'appelant, avant l'appel.

3.2.3 Commandes

3.2.3.1 Status : \$00 | \$40

Renvoi des informations à propos d'un Device particulier ou si le numéro d'unité est à zéro, du SmartPort lui-même.

Index	Taille		Paramètres de DStatus
0	2	E	Nombre de paramètres (3)
2	2	E	Numéro d'unité (Les numéros d'unité sont alloués de façon unique à chaque dispositif lors de l'initialisation du SmartPort)
4	4	R	Pointeur vers un buffer où sera placé le résultat qui occupera deux ou quatre octets en fonction du mode (normal ou étendu)
8	2	E	Type de status demandé \$0000 Status d'un dispositif \$0001 Bloc de contrôle d'un dispositif \$0002 État du caractère nouvelle ligne \$0003 Bloc d'information d'un dispositif

En retour on obtient dans le buffer les informations suivantes, en fonction du type de status demandé:

- Type de status: \$0000

4-5 octets d'informations: 1 octet de status général

- Bit 7 1: Device de type bloc
 0: Device de type caractère
- Bit 6 1: Ecriture autorisée
 0: Écriture non-autorisée
- Bit 5 1: Lecture autorisée
 0: Lecture non-autorisée
- Bit 4 1: Device en ligne ou disquette présente.
- Bit 3 1: Formatage autorisé
- Bit 2 1: Protection en écriture (Device de type bloc seulement)
- Bit 1 1: Device en cours d'interruption (Apple IIc seulement)
- Bit 0 1: Device actuellement ouvert (dispositif de type caractère)

S'il s'agit d'un Device de type bloc les octets suivants représentent le nombre de blocs du Device (3 octets pour un appel normal, 4 octets pour un appel étendu).

Dans le cas d'un Device de type caractère, les octets suivants sont zéro.

Si le numéro d'unité est de zéro on obtient les informations sur le SmartPort soit 7 octets :

- octet 0 Nombre de dispositifs
- octet 1 État d'interruption (si bit 6 à alors pas d'interruption)

octets 2-3	Société ayant réalisé le logiciel SmartPort \$0000 Indéterminé \$0001 Apple \$0002 Autre
octets 4-5	Version de l'Interface
octets 6-7	Reservé (doit être \$0000)

- Type de status: \$0001

Le bloc de contrôle est spécifique à chaque dispositif. La seule chose commune entre les différents blocs de contrôle est la présence d'un octet au début du buffer indiquant la longueur de celui-ci. (Note: Lorsque cet octet vaut 0, cela signifie que la longueur du buffer est de 256 octets et non pas 0. Il est également important de signaler que l'octet contenant la longueur du buffer contenant le résultat n'est pas compté).

- Type de status: \$0002

Étant donné qu'aucun dispositif de type caractère n'a encore été implémenté, ce type de status est pour l'instant indéfini.

- Type de status: \$0003

Cet appel retourne dans le buffer le bloc d'information d'un dispositif.

- Status du Device (Idem octet de status général) 1 octet.
- Taille en bloc du device 3 octets si appel normal, 4 octets sinon.
- Taille de la chaîne d'identification 1 octet.
- Chaîne d'identification 16 octets.
- Octet de type du Device
- Octet de sous type du Device

Bit 7 : support du mode étendu

Bit 6 : support des erreurs de changement de disque

Bit 5 : Media amovible.

Bits 4 à 0 réservés.

- Mot de version: 2 octets

Exemples de couples (type, sous-type) définis:

Type	Sous-Type	Description
\$00	\$00	Carte d'extension mémoire Apple
\$00	\$C0	Carte d'extension mémoire Apple IIgs configurée en disque RAM.
\$01	\$00	Unité de disquette UniDisk 3.5"
\$01	\$C0	Unité de disquette Apple 3.5"
\$03	\$E0	SCSI ave média non-amovible

3.2.3.2 ReadBlock: \$01 I \$41 et WriteBlock: \$02 I \$42

Lit / Écrit un bloc de 512 octets à partir de l'unité spécifiée en paramètre. Le bloc est lu/écrit vers/à partir de l'emplacement mémoire spécifié en paramètre.

Index	Taille		Paramètres de ReadBlock et WriteBlock
0	2	E	Nombre de paramètres (3)
2	2	E	Numéro d'unité (Les numéros d'unité sont alloués de façon unique à chaque dispositif lors de l'initialisation du SmartPort)
4	4	ER	Pointeur vers un buffer où sera placé le résultat qui occupera deux ou quatre octets en fonction du mode (normal ou étendu)
8	2	E	Numéro du bloc (3 à 4 octets)

Codes d' erreur reçus en retour:

Code	Nom	Description de l'erreur
\$06	BUSERR	Erreur de communication sur le SmartPort
\$27	IOERROR	Erreur d'entrée sortie
\$28	NODRIVE	Pas de dispositif connecté
\$2B	NOWRITE	Disque protégé contre l'écriture
\$2D	BADBLOCK	Numéro de bloc invalide
\$2F	OFFLINE	Dispositif offline ou pas de disquette dans l'unité

3.2.3.3 Format : \$03/\$43

Formate un Device de type bloc. Ce formatage est sans rapport avec le système d'exploitation (on installe la structure pistes/secteurs sur une disquette, mais pas le

catalogue ou la liste des blocs libres), il prépare tous les blocs du dispositif pour permettre l'écriture et la lecture.

Paramètres:

Index	Taille		Paramètres de Format
0	2	E	Nombre de paramètres (1)
2	2	E	Numéro d'unité (Les numéros d'unité sont alloués de façon unique à chaque dispositif lors de l'initialisation du SmartPort)

Codes d' erreur reçus en retour:

Code	Nom	Description de l'erreur
\$06	BUSERR	Erreur de communication sur le SmartPort
\$27	IOERROR	Erreur d'entrée sortie
\$28	NODRIVE	Pas de dispositif connecté
\$2B	NOWRITE	Disque protégé contre l'écriture
\$2F	OFFLINE	Dispositif offline ou pas de disquette dans l'unité

3.2.3.4 Control : \$04/\$44

Envoi des informations de contrôle à un dispositif (par exemple demande d'éjection de disque). La demande peut être de type général ou spécifique à un type de dispositif.

Index	Taille		Paramètres de Control
0	2	E	Nombre de paramètres (3)
2	2	E	Numéro d'unité (Les numéros d'unité sont alloués de façon unique à chaque dispositif lors de l'initialisation du SmartPort)
4	4	ER	Pointeur vers la liste de contrôle qui occupera deux ou quatre octets en fonction du mode (normal ou étendu)
8	2	E	Type de contrôle sollicité

Note : les deux premiers octets de la liste de contrôle spécifient sa taille.

Le type de contrôle dépend du dispositif adressé. Les types suivants sont prédéfinis:

Type	Description
\$00	Reset logiciel du dispositif
\$01	Positionnement du bloc de contrôle
\$02	Sélection du caractère nouvelle ligne
\$03	Contrôle des interruptions

Codes d' erreur reçus en retour:

Code	Nom	Description de l'erreur
\$06	BUSERR	Erreur de communication sur le SmartPort
\$21	IOERROR	Type de contrôle invalide
\$22	BADCTLPARM	Liste de paramètres invalides
\$30-3F	UNDEFINED	Erreurs spécifiques d'un type de dispositif

3.2.3.5 Init: \$05/\$45

Permet à l'application de réinitialiser le SmartPort.

Paramètres:

Index	Taille	Paramètres de Init	
		E	Nombre de paramètres (1)
		E	Numéro d'unité (\$00)

Codes d' erreur reçus en retour:

Code	Nom	Description de l'erreur
\$06	BUSERR	Erreur de communication sur le SmartPort
\$28	NODRIVE	Pas de dispositif connecté

3.2.3.6 Open : \$05/\$45

Ouvre un Device de type caractère en lecture/écriture. Un Device de type bloc recevant cet appel renverra le code BADCMD.

Paramètres:

Index	Taille	Paramètres de Open	
		E	Nombre de paramètres (1)
		E	Numéro d'unité (1 octet, différent de \$00) (Les numéros d'unité sont alloués de façon unique à chaque dispositif lors de l'initialisation du SmartPort)

Codes d' erreur reçus en retour:

Code	Nom	Description de l'erreur
\$01	BADCMD	Commande invalide
\$06	BUSERR	Erreur de communication sur le SmartPort
\$28	NODRIVE	Pas de dispositif connecté

3.2.3.7 Close: \$07/\$47

Cet appel indique à un Device de type caractère qu'une séquence d'écriture/ Lecture est terminée (pour une imprimante cet appel peut demander de vider le buffer d'impression)

Un Device de type bloc recevant cet appel renverra le code erreur BADCMD.

Paramètres:

Index	Taille	Paramètres de Close	
		E	Nombre de paramètres (1)
		E	Numéro d'unité (1 octet, différent de \$00) (Les numéros d'unité sont alloués de façon unique à chaque dispositif lors de l'initialisation du SmartPort)

Codes d' erreur reçus en retour:

Code	Nom	Description de l'erreur
\$01	BADCMD	Commande invalide
\$06	BUSERR	Erreur de communication sur le SmartPort
\$28	NODRIVE	Pas de dispositif connecté

3.2.3.8 Read: \$08/\$48

Permet de lire un nombre d'octets donné.

L'adresse référence une localisation sur un dispositif de type caractère. Dans le cas d'un dispositif de type bloc l'adresse correspond à un numéro de bloc (cet appel sert dans ce cas à lire des blocs de taille différente de 512 octets comme par exemple les blocs d'une disquette Macintosh).

Paramètres:

Index	Taille	Paramètres de Read	
		E	Nombre de paramètres (4)
		E	Numéro d'unité (1 octet, différent de \$00) (Les numéros d'unité sont alloués de façon unique à chaque dispositif lors de l'initialisation du SmartPort)
		ER	Pointeur vers le buffer qui occupera deux ou quatre octets en fonction du mode (normal ou étendu)
		E	Nombre d'octets à transférer 2 octets (poids faible poids fort)
		E	Adresse dans le dispositif 3 à 4 octets

Codes d' erreur reçus en retour:

Code	Nom	Description de l'erreur
\$06	BUSERR	Erreur de communication sur le SmartPort
\$27	IOERROR	Erreur d'entrée sortie
\$28	NODRIVE	Pas de dispositif connecté
\$2B	NOWRITE	Disque protégé contre l'écriture
\$2D	BADBLOCK	Numéro de bloc invalide
\$2F	OFFLINE	Dispositif offline ou pas de disquette dans l'unité

3.2.3.9 Write: \$09/\$49

Permet d'écrire un nombre d'octets donné.

L'adresse référence une localisation dans un dispositif de type caractère. Dans le cas d'un dispositif de type bloc l'adresse correspond à un numéro de bloc (cet appel sert dans ce cas à lire des blocs de taille différente de 512 octets comme par exemple les blocs d'une disquette Macintosh).

Paramètres:

Index	Taille	Paramètres de Write	
		E	Nombre de paramètres (4)
		E	Numéro d'unité (1 octet, différent de \$00) (Les numéros d'unité sont alloués de façon unique à chaque dispositif lors de l'initialisation du SmartPort)
		ER	Pointeur vers le buffer qui occupera deux ou quatre octets en fonction du mode (normal ou étendu)
		E	Nombre d'octets à transférer 2 octets (poids faible poids fort)
		E	Adresse dans le dispositif 3 à 4 octets

Codes d' erreur reçus en retour:

Code	Nom	Description de l'erreur
\$06	BUSERR	Erreur de communication sur le SmartPort
\$27	IOERROR	Erreur d'entrée sortie
\$28	NODRIVE	Pas de dispositif connecté
\$2B	NOWRITE	Disque protégé contre l'écriture
\$2D	BADBLOCK	Numéro de bloc invalide
\$2F	OFFLINE	Dispositif offline ou pas de disquette dans l'unité

3.2.3.10 Types de contrôle de l'unité de disquettes Apple 3.5"

Il existe sept types de contrôle spécifiques à l'Apple 3.5". Ceux-ci permettent notamment d'éjecter une disquette, de changer le nombre de faces (simple ou double face) ainsi que son formatage.

Les paramètres donnés doivent se trouver dans la liste de contrôle avant l'appel du SmartPort.

3.4 L'Accès via les devices handlers de Prodos 8

ProDOS /8 contrairement au DOS 3.3, est indépendant des périphériques, mais nécessite pour permettre leur utilisation un Device handler, équivalent de la RWTS du DOS 3.3. Ce device handler existe en standard pour les unités 5.25" et pour le volume /RAM, de même ce device handler est défini en ROM pour le disque dur Profile et les Apple 3.5" et Unidisk 3.5".

Ces devices handlers sont utilisés par le MLI et permettent d'accéder de façon adéquate aux périphériques. Ces handlers peuvent supporter quatre fonctions de base : FORMAT, READ, WRITE, STATUS. Cependant, il faut noter que tous les handlers ne supportent pas l'intégralité de ses fonctions, notamment le driver pour le Disk II ne supporte pas la fonction FORMAT.

Les appels READ BLOCK et WRITE BLOCK du MLI sont la méthode permettant d'accéder aux blocs à partir de ProDOS comme les appels DInfo, DStatus, DRead, DWrite pour Cs/OS, mais il est également possible d'accéder directement (totalement conseillé) au device handler.

L'adresse des devices handlers se trouve sous ProDOS/ 8 dans la page globale de \$BF10 à \$13F2F (par exemple pour appeler le device handler du slot 6 drive 1 on fait un JSR (\$BF1C)). Le passage de paramètres aux devices handlers se fait via des emplacements en page zéro:

- \$42 Code de la commande
- \$43 Numéro d'unité sous la forme DSSS0000
où D=numéro de Drive et SSS numéro de slot
- \$44-\$45 adresse du buffer
- \$46-\$47 numéro du bloc.

3.4.1 Status: \$00

Cet appel retourne l'état d'un Device particulier. Généralement il est utilisé pour déterminer si un Device est présent et s'il est protégé en écriture ou non. Certains handlers retournent en plus le nombre de blocs supportés par le Device.

- \$42 Numéro de la commande (\$00 pour Status).
- \$43 Unité concernée.
- \$44-\$45 Inutilisé.
- \$46-\$47 Inutilisé doit être à \$0000

En retour le bit carry est positionné en cas d'erreur et l'accumulateur contient le code d'erreur :

- \$00 Pas d'erreur.
- \$27 Erreur d'Entrée/Sortie (I/O)
- \$28 Pas de Device connecté à cette unité.
- \$2B Disque protégé en écriture.

3.4.2 Read: \$01

Cet appel lit un bloc de 512 octets et le stocke en mémoire à l'emplacement spécifié. Aucun test n'est effectué sur la validité de l'adresse destination, donc prudence.

\$42	Numéro de la commande (\$01 pour Read).
\$43	Unité concernée.
\$44-\$45	Adresse de destination.
\$46-\$47	Numéro du bloc.

En retour le bit carry est positionné en cas d'erreur et l'accumulateur contient le code d'erreur :

\$00	Pas d'erreur.
\$27	Erreur d'Entrée/Sortie (I/O)
\$28	Pas de dispositif connecté à cette unité.
\$2B	Disque protégé en écriture.

3.4.3 Write: \$02

Cet appel écrit un bloc de 512 octets à partir de l'emplacement mémoire spécifié.

\$42	Numéro de la commande (\$02 pour Write).
\$43	Unité concernée.
\$44-\$45	Adresse source.
\$46-\$47	Numéro du bloc.

En retour le bit carry est positionné en cas d'erreur et l'accumulateur contient le code d'erreur:

\$00	Pas d'erreur.
\$27	Erreur d'Entrée/Sortie (I/O)
\$28	Pas de dispositif connecté à cette unité.
\$2B	Disque protégé en écriture.

3.4.3 Format: \$03

Cet appel formate le média dans l'unité spécifié. Il s'agit comme dans le cas des appels SmartPort juste d'un formatage bas niveau, les structures de fichiers de ProDOS (catalogue, bit map) ne sont pas installées.

\$42	Numéro de la commande (\$03 pour Format).
\$43	Unité concernée.
\$44-\$45	Inutilisé.
\$46-\$47	Inutilisé.

En retour le bit carry est positionné en cas d'erreur et l'accumulateur contient le code d'erreur:

\$00	Pas d'erreur.
\$27	Erreur d'Entrée/Sortie (I/O)

\$28 Pas de dispositif connecté à cette unité.

\$2B Disque protégé en écriture.

Exemple :

Recopie du bloc \$0001 sur le bloc \$0000 pour le floppy en slot 6 drive 1

```
STA $C080 ; Valide la carte langage
LDA #$01 ; Numéro de la commande READ
STA $42
LDA #%0110000
STA $43 ; Unité
LDA #<buffer ; Adresse du buffer (poids faible)
STA $44
LDA #>buffer ; Adresse du buffer (poids fort)
STA $45
LDA #$01 ; Numéro du bloc
STA $46 ; poids faible
LDA #$00
JSR call
BCS erreur
LDA #$02 ; Numéro de la commande WRITE
STA $42
LDA #%0110000
STA $43 ; Unité
LDA #<buffer ; Adresse du buffer (poids faible)
STA $44
LDA #>buffer ; Adresse du buffer (poids fort)
STA $45
LDA #$00
STA $46
LDA #$00
STA $47
```



```
JSR call
```

```
BCS erreur
```

```
call JMP ($13F1C) ; Adresse dans la page globale
```

```
buffer DS 512
```

3.5 L'accès direct aux disquettes

Via les commutateurs logiciels situés en SC000/\$COFF il est possible de simuler entièrement les Devices handlers de ProDOS/8. L'accès aux unités de disquettes via cette méthode est extrêmement délicat, car il suffit d'une petite erreur de programmation pour effacer définitivement le contenu d'une piste. En contre partie cette méthode laisse une liberté totale (dans la limite des capacités du lecteur de disquettes) quant à l'organisation des informations sur le disque, d'où son emploi intensif par les programmes de copie et les systèmes de protections.

3.5.1 Le lecteur 5 1/4

16 commutateurs logiciels (softswitches) permettent l'accès aux disques 5 1/4. Huit autres contrôlent le déplacement de la tête, 8 autres contrôlent la lecture/écriture, la sélection de l'unité et la mise en marche du moteur.

Phase 0	\$C0E0 Off	\$C0E1 On
Phase 1	\$C0E2 Off	\$C0E3 On
Phase 2	\$C0E4 Off	\$C0E5 On
Phase 3	\$C0E6 Off	\$C0E7 On

Sélection du Drive1	\$C0EA
Sélection du Drive2	\$C0EB
Mise en route du moteur	\$C0E9
Arrêt du moteur	\$C0E8

Q6	\$C0EC Off	\$C0ED On
Q7	\$C0EE Off	\$C0EF On

3.5.1.1 Déplacement du bras

Chacune des phases doit être accédée séquentiellement pour déplacer le bras (Mise On

puis Off). Dans un ordre ascendant, 0 à 3 le bras se déplace vers les pistes supérieures, dans l'ordre inverse le bras se déplace vers la piste 0.

Pour se déplacer d'une piste entière, il faut accéder à deux phases. L'accès à une seule phase ne déplaçant le bras que d'une demi piste.

Pour l'obtention de performances optimum le délai entre l'allumage et l'extinction des phases, est critique. Il faut laisser le temps au bras de se positionner sur la piste suivante, sans lui laisser le temps de s'y arrêter, (pour profiter de son élan) s'il ne s'agit pas de la piste voulue.

De même un délai suffisant doit être prévu pour permettre la prise de vitesse du moteur du drive après sa mise en route.

```
;  
; SEEK  
;  
; En entrée n de piste souhaitée fois 2  
; Old_Trk piste actuelle  
; X slot fois 16 soit $60  
  
STX Slot_Nb  
STA $2A  
CMP Old_Trk  
BEQ LFC ; On est déjà sur la piste  
LDA #$00  
STA $26  
  
LAD LDA Old_Trk  
STA $27  
SEC  
SBC $2A  
BEQ Fin  
BCS Inf  
EOR #$FF  
INC Old_Trk  
BCC LC5  
  
Inf ADC #$FE
```

	DEC	Old_Trk
LC5	CMP	\$26
	BCC	LCB
	LDA	\$26
LCB	CMP	#\$0C
	BCS	LD0
	TAY	
LD0	SEC	
	JSR	Move1
	LDA	TblTmp1,Y
	JSR	Wait
	LDA	\$27
	CLC	
	JSR	Move11
	LDA	TblTmp2,Y
	JSR	Wait
	INC	\$26
	BNE	LAD
Fin	JSR	WAIT
	CLC	
Move1	LDA	Old_Trk
Move11	AND	#\$03
	ROL	A
	ORA	Slot_Nb
	TAX	
	LDA	\$C080,x
	LDX	\$2B
LFC	RTS	

; Routine délai lors du déplacement

; A contient le nombre de 100 micro sec à attendre

```
Wait          LDX  #$11          ; Routine de temporisation
```

```
L02          DEX
```

```
             BNE  L02
```

```
             INC  $46
```

```
             BNE  L0B
```

```
             INC  $47
```

```
L0B          SEC
```

```
             SBC  #$01
```

```
             BNE  Wait
```

```
             RTS
```

; Valeurs d'intervalles de 100 micro sec utilisés lors du déplacement du bras

```
TblTmp1      DC          H'01302824201E1D1C1C1C1C1C'
```

```
TblTmp2      DC          H'702C26221F1E1D1C1C1C1C1C'
```

3.5.1.3 Sélection du drive 1 ou 2

\$C0EA Sélectionne le Drive 1

\$C0EB Sélectionne le Drive 2

3.5.1.4 Lecture d'un Octet

On doit avoir précédemment accédé à \$C0EE pour valider le mode de lecture.

```
             LDA  $C0EE
```

```
Loop         LDA  $C0EC
```

```
             BPL  Loop
```

Tous les octets valides ont le bit de poids fort à 1 (octet de \$80 à \$FF)

3.5.1.5 Détection de la protection en écriture

```
LDA $C0ED
```

```
LDA $C0EE
```

```
BMI prtg ; La valeur dans $C0EE est supérieure à $80 si  
; protégé en écriture
```

3.5.1.6 Ecriture d'un octet

```
LDX #$60
```

```
STA $C08F,X ; Mode écriture
```

```
JSR délai ; Délai de 100msec
```

```
LDA #$AA
```

```
STA $C08D,X ; Charge le registre
```

```
ORA $C08C ; Commence l'écriture
```

Pour écrire un octet, on doit commencer par charger la valeur à écrire dans le registre de donnée (\$C0ED) puis accéder à \$C0EC pour déclencher l'écriture. L'automate contrôlant le disque écrit un bit tous les 4 cycles. Le bit écrit est le bit de poids fort de \$C0ED le registre est décalé ensuite automatiquement d'un bit vers la gauche (\$FF devient \$FE).

On doit donc écrire les octets en 32 cycles exactement. A noter que les cycles sont comptés en vitesse lente soit 1MHz.

Exemple d'écriture en 32 cycles de la marque du champ donnée d'un secteur :

Note : Il existe un deuxième type d'octets, les octets de synchronisation. Ceux-ci permettent au contrôleur de se positionner dans le flux de bits composant une piste. Ces octets sont écrits en 40 cycles. Les 8 cycles supplémentaires générant 2 bits à 0 permettant cette synchronisation.

Soit une chaîne de \$FF écrite comme octets de synchro. Suivi d'un octet valant \$AA

```
$FF $FF $FF $AA
```

```
1111 1111 00 1111 1111 00 1111 1111 00 1010 1010
```

La lecture commence ici

```
1111 00 1 $F9
```

```
111 1111 0 $FE
```

0 Skip un octet valide commence par un 1 1111 1111 \$ F F

On est synchronisé

Premier octet lu (Rappel le bit de poids fort doit être à 1):

1111 1001 \$F9

Octet suivant : \$FE

Octet suivant : \$FF

On lit correctement le \$AA suivant

Cas où les \$FF ne sont pas des octets synchros.

1111 1111 1111 1111 1111 1111 1010 1010

La lecture commence ici

1111 111 \$FF

 1 1111 111 \$FF

 1 1111 101 \$FD

On ne lira pas le \$AA correctement.

On peut se demander pourquoi on n'écrit pas tout simplement une chaîne de zéro avant toute information valide. La raison est toute simple, la présence de plusieurs bits à zéro consécutifs sur le disque est très mal supportée par l'automate qui relit une valeur aléatoire.

3.5.2 Le lecteur 3.5"

Dans cette partie on va parler uniquement du lecteur non-intelligent l'Apple 3.5. En effet celui-ci est contrôlé directement par le GS, alors que l'Unidisk 3.5 dispose de son propre processeur (65C02) et de sa propre RAM/ROM.

Le drive 3.5 utilise les mêmes commutateurs logiciels que le drive 5.25". On doit donc le sélectionner avant toute chose car le 5.25" est validé par défaut.

Pour sélectionner le Drive 3.5 on doit effectuer les opérations suivantes

- Valider les slots internes en 5 et 6 du GS

```
LDA    $C02D
```

```
AND    #$9F
```

```
STA    $C02D
```

- Être en vitesse rapide, et ne pas repasser en vitesse lente au démarrage du moteur

LDA \$C036

AND #%11111011 ; Pas de passage en vitesse lente sur mise en
; marche du moteur

ORA #80

STA \$C036

- Écrire un \$40 en \$C031 pour sélectionner le 3.5, tête 0

3.5.2.2 Allumage extinction du moteur

\$C0E9 Allumage du moteur

\$C0E8 Extinction du moteur

3.5.2.3 Sélection du Drive 1 ou 2

\$C0EA Sélectionne le Drive 1

\$C0EB Sélectionne le Drive 2

3.5.3 Registres propres au 3.5

Le registre d'interface disque:

Permet de sélectionner le lecteur 5 1/4 ou le drive 3 1/2, permet pour ce dernier de sélectionner la tête adressée pour les opérations de lecture écriture.

Bit 7: 1 Sélectionne la tête 1
0 Sélectionne la tête 0

Bit 6: 1 Sélectionne les drives 3.5"
0 Sélectionne les drives 5.25"

Le lecteur 3.5" utilise aussi les mêmes commutateurs logiciels que le lecteur 5.25". Ces commutateurs situés de \$C0E0 à \$COEF portent les mêmes noms que lors de leur emploi pour le 5.25" mais par contre leur fonction est totalement différente.

En effet les commutateurs Q6 à Q7 de concert avec la mise en marche du moteur permettent d'accéder à cinq registres différents.

Q7	Q6	Moteur	Description
Off	Off	On	Lecture registre de donnée
Off	On	xx	Lecture registre de status
On	Off	xx	Lecture registre de handshake
On	On	Off	Écriture registre de mode
On	On	On	Écriture registre de donnée

Après sélection d'un registre, l'écriture à n'importe quelle adresse impaire de l'IWM (\$C0E1 a \$C0EF) écrira dans ce registre, la lecture d'une adresse paire de l'IWM (\$C0E0 à \$C0EE) lira ce registre.

Le registre de Mode :

Attention le lecteur doit être arrêté et non seulement éteint.

Bit 7	Réservé ne pas modifier
Bits 6-5	Réservé écrire toujours 0
Bit 4	Vitesse de l'horloge de lecture 1: 7MHz valeur pour le GS 0: 8MHz
Bit 3	1: Les cellules de bits de l'octet font 2 Usec cas des lecteurs 3.5 0: Les cellules bits de l'octet ont une longueur de 4 Usec, valeur utilisée pour les Devices connectés au SmartPort et les lecteurs 5.25".
Bit 2	1: Le lecteur, après sa désélection, restera allumé une seconde.
Bit 1	1: Protocole de Handshake synchrone (lecteur 5.25") 0: Protocole de Handshake asynchrone (autres cas)
Bit 0	1: Mode latch validé, la donnée reste valide pour la durée d'un octet. (Cellule de 2 Usec alors 16 Usec, cellule de 4 microsec alors 32 Usec) 0: Mode latch invalidé, la donnée lue reste valide pendant 7 Usec.

Le registre de Status

Bit 7	Test de la protection écriture (Lecteur 5.25") Résultat accès registre (Lecteur 3.5").
-------	-------------------------------------------------------------------------------------------

Bit 6	Réservé
Bit 5	1: Le lecteur 1 ou 2 est sélectionné et le moteur est allumé. 0: Pas de lecteur sélectionné
Bit 4-0	Idem registre de mode.

Le registre de Handshake

Bit 7	1: Le registre de données est prêt pour les données. 0: Le registre de données est plein.
Bit 6	1: Le dernier octet écrit à été écrit correctement. 0: Un octet à été raté et non écrit sur le disque.
Bit 5-0	Réservé.

Le registre de Donnée

Suivant l'état des commutateurs Q7 Q6 on accède a l'octet lu à partir du disque, ou on écrit un octet sur le disque.

Un deuxième jeu de registres peut être accédé grâce aux commutateurs correspondant aux phases, et à la sélection de la tête. (SEL) Lecture de ces registres :

Q7 doit être Off et Q6 On le lecteur doit être validé avec le moteur allumé. Puis vérifier que Phase 3 est Off. On positionne ensuite les phase 3,2,1 et SEL suivant le registre auquel on veut accéder. Une fois ceci effectué on peut lire l'information dans le bit de poids fort de \$C0EE. Après lecture du registre on peut accéder à un autre registre simplement en repositionnant les phases et SEL. À la fin des opérations de lectures de registres, repassez en mode normal en accédant à Q6 off.

Phase 2	Phase 1	Phase 0	SEL	Registre
Off	Off	Off 0	DIRTN	Direction tête
Off	Off	Off 1	CSTIN	Présence d'une disquette
Off	Off	On 0	STEP	
Off	Off	On 1	WRTPRT	Protection écriture
Off	On	Off 0	MOTORON	Moteur en marche

Phase 2	Phase 1	Phase 0	SEL	Registre
Off	On	Off 1	TK0	Tête sur piste 0
Off	On	On 1	TACH	Tachomètre
On	Off	Off 0	RDDATA0	Flot de données tête 0
On	Off	Off 1	RDDATA1	Flot de données tête 1
On	On	Off 0	SIDES	Lecteur simple ou double face
On	On	On 1	DRVIN	Lecteur installé

Écriture dans ces registres :

Vérifiez d'abord que Phase 3 est off, ensuite passez On Phase 0 et Phase 2, puis positionnez SEL à 0. Mettez ensuite Phase 1 et Phase 0 dans l'état nécessaire pour accéder aux registres, et positionnez Phase 2 suivant la valeur à écrire (On pour 1, Off pour 0).

Maintenez Phase 3 à On, pendant aux moins 1 Usec, mais moins de 1 msec (sauf si vous éjectez une disquette) puis remettez-le à Off. Soyez sûr que vous ne changez ni Phase1-2 ou SEL pendant que Phase3 est On, et que Phase 0 et Phase 1 sont On avant de modifier SEL.

NOTE : Comme dans le cas d'une lecture le lecteur doit être d'abord sélectionné.

Phase 1	Phase 0	SEL	Registre
Off	Off 0	DIRTN	Sens de déplacement de la tête
Off	On 0	STEP	Déplace la tête d'une piste
On	Off 0	MOTORON	Allume le moteur
On	On 0	EJECT	Éjecte le disque

DIRTN: Indique le sens de déplacement de la tête. À 1 on se déplace vers la piste 0, à 0 vers la piste 79.

CSTIN: A comme valeur 0 quand une disquette se trouve dans le lecteur.

STEP: Mettre ce registre à 0 cause un déplacement de la tête en fonction de DIRTN. Quand le déplacement est terminé (à peu près 12 msec) le lecteur remet STEP à 1 et on peut recommencer.

WRPTR: À 0 si le disque est protégé contre l'écriture.

MOTORON: 0 allume le moteur 1 l'éteint. Ne fonctionne que si le drive est sélectionné et qu'une disquette est présente.

TK0: À 0 si la tête se trouve sur la piste 0.

EJECT: Écrire 1 dans ce registre éjecte la disquette. Attention il faut maintenir Phase 3 On pendant au moins 1 / 2 seconde.

RDDATA0:

RDDATA1: Donne le flot de bits instantané venant de la tête 1 ou 0.

SIDES: 1 car lecteur double faces.

DRVIN: 0 si un lecteur est connecté.

Exemple d'accès aux lecteurs:

Éjection d'une disquette lecteur 1

```
LDA $C036
```

```
AND #$FB
```

```
ORA #$80
```

```
STA $C036
```

```
LDA $C02D
```

```
AND #$9F
```

```
STA $C02D
```

```
LDA #$40
```

```
STA $C031
```

```
; Sélection du drive
```

```
LDA $C0EA
```

```
LDA $C0E9
```

```
JSR Wait ; Attente
```

```
LDA $C0E6 ; Phase 3 Off
```

```
LDA $C0E1 ; Phase 0 On
```

```
LDA $C0E3 ; Phase 1 On
```

```
LDA #$40
```

```
STA $C031 ; SEL à 0
```

; Paramètres pour l'éjection

LDA \$C0E1 ; Phase0 On

LDA \$C0E3 ; Phase1 On

LDA \$C0E5 ; Phase2 On (valeur1)

LDA \$C0E7 ; Phase3 On

JSR WAIT ; Attente d'une demi seconde

LDA \$C0E6 ; Phase3 Off

; Déplacement du bras

* SEEK

* Allume le moteur et se déplace vers une piste

* Cyl: Piste voulue

* CurCY1: piste actuelle

* Drive: Lecteur concerné

SEEK LDX Drive

BIT CurCyl,X

BPL no

JSR Recall ; Recalibre

BCS erreur

no JSR DriveOk

SEC

LDX Drive ; Calcule le nombre de pistes à déplacer

LDA CurCyl,X

SBC Cyl

BEQ fin

LDY tel ; Piste plus petite

BCS inf

LDY #\$00 ; No pst plus grand

	EOR	#\$FF	
	ADC	#\$01	
inf	TAX		
	TYA		
	JSR	EcrREG	; Sens du déplacement
	JSR	seek_n	
fin	LDX	Drive	
	LDA	Cyl	
	STA	CurCyl,X	
	CLC		
	RTS		
erreur	LDA	#\$02	
	ORA	Error	
	STA	Error	
	RTS		
seek_n	LDA	#\$04	; Registre SEEK
	JSR	EcrREG	
Loop	JSR	LitREG	
	BPL	Loop	; Attente fin de seek
	DEX		
	BNE	seek_n	; encore une piste
tmp	DEX		
	BNE	tmp	
	RTS		
DriveOk	LDA	#\$00	
	STA	\$6A	
	LDA	#\$5D	
	STA	\$6B	

```

LDA  #$08          ; MOTORON
JSR  LitReg
BPL  Ok

JSR  Ecr2Reg

LDX  Drive
LDA  $11,X
JSR  Wait

LDA  #$19
STA  $11,X
JSR  Wait

Ok                RTS

Wait              PHA
                  JSR  Wait2
                  PLA

Wait2             STA  $18
                  LDA  $C036
                  PHA
                  AND  #$7F
                  STA  $C036          ; Tempo en vitesse lente

L1                LDA  #$C8
                  STA  $17

L2                DEC  $17
                  NOP
                  BNE  L2
                  DEC  $18
                  BNE  L1
                  PLA
                  STA  $C036
                  RTS

```

```
Recall      LDA  #$01
           JSR  EcrReg
```

```
           LDA  #$50
           JSR  seek_n
```

```
encore     LDX  #$50
           LDA  #$07
           JSR  Wait2
```

```
           LDA  #$0A      ; TK0
           JSR  LitReg
           BPL  fin
           DEX
           BNE  encore
           SEC
           BRA  fin2
```

```
fin        CLC
```

```
fin2      LDX  Drive
           STZ  CurCyl,X
           RTS
```

```
; Lecture d'un registre
```

```
LitREG    JSR  SetReg
           BIT  $C0ED
           LDA  $C0EE
           RTS
```

```
; Écriture d'un registre
```

```
EcrREG    JSR  SetReg
EcrREG2   BIT  $C0E7      ; Phase3 On
           BIT  $C0E6      ; Phase3 Off
           RTS
```

```
; Suivant l'état des bits poids faible de A
```

```
; positionne les Phases et SEL
```

```
; Bit 0: Phase 2          1 = ON
```

```
; Bit 1: SEL             1 = 1
```

```
; Bit 2: Phase 0        1 = ON
```

```
; Bit 3: Phase 1        1 = ON
```

```
SetReg      BIT    $C0E0
```

```
            BIT    $C0E3
```

```
            BIT    $C0E6
```

```
            BIT    $C0E4
```

```
            LSR
```

```
            BCC   pasPhase2
```

```
            BIT    $C0E5
```

```
pasPhase2   LSR
```

```
            PHA
```

```
            LDA   $C031
```

```
            AND   #$7F
```

```
            BCC   pasSEL
```

```
            ORA   #$80
```

```
pasSEL      STA   $C031
```

```
            PLA
```

```
            LSR
```

```
            BCC   pasPhase0
```

```
            BIT    $C0E1
```

```
pasPhase0   LSR
```

```
            BCC   pasPhase1
```

```
            BIT    $C0E2
```

```
pasPhase1   RTS
```


; Écriture d'un octet

HEF03 BIT \$C0ED

LDA #\$FF

STA \$C0EF

LDY #\$07

Loop LDA :DATA_S2,Y ; Marque de champ de donnée

; On regarde le registre de Handshake

L1 BIT \$C0EC ; On peut écrire?

BPL L1

STA \$C0ED ; Écrit

DEY

BPL Loop

;

; Marque d'adresse

;

DATA_S2 HEX AD

HEX AA

HEX D5

HEX FF

HEX FC

HEX F3

HEX CF

HEX 3F

HEX FF

HEX FF

FF 3F CF F3 FC écrit des octets de synchro:

En effet, l'écriture de cette chaîne donne la séquence suivante: 11111111-0011 1111-1100
1111-1111 0011-1111 1100

Ce qui à la lecture, le contrôleur ne pouvant lire que des octets ayant le bit de poids fort à 1, est relue comme:

1111 1111 (saute les 2 0)

1111 1111 (saute les 2 0) soit quatre \$FF

Note: Cette méthode utilise le registre de Handshake, indiquant la possibilité d'écriture d'un octet. Il est également possible d'écrire en se synchronisant en 16 cycles comme pour le lecteur 5.25".

Lecture d'un octet.

```
                BIT    $C0EC        ; Mode lecture
Loop            LDA    $C0EE
                BPL    Loop
```

PLAYER 1
00266300

HIGH SCORE
01387000

PLAYER 2
00000000



4

Graphisme

Le G de GS provient de «GRAPHISME». Avec le IIGS, Apple nous a offert de nouvelles possibilités, puissantes sur certains points, limitées sur d'autres. Je vais essayer de décrire ici les différentes innovations d'un point de vue «FIRMWARE», c'est à dire en restant le plus proche possible du hardware. Beaucoup d'ouvrages traitent déjà de la programmation pratique avec les TOOLS (la boîte à outils), mon but est de rédiger un guide de référence et surtout de programmation du hardware directement, seul moyen, à mon avis, d'atteindre une vitesse d'exécution acceptable sur GS.

La partie programmation ressemblera à un dialogue dans lequel tous les points importants seront analysés. J'utiliserai la langue anglaise pour les termes techniques, dans un esprit d'uniformité avec les ouvrages américains.

Tous les exemples de (courts) programmes présentés dans cet ouvrage sont en assembleur.

Un minimum de connaissance de ce langage est requis, mais le niveau reste assez simple et les commentaires sont nombreux. Ces programmes sont assemblables directement avec MERLIN 16 (tm ROGER WAGNER PUBLISHING) sous PRODOS 8, mais peuvent certainement être adaptés sans peine pour un autre assembleur.

4.1 Nouvelle résolution graphique (Super-hires)

Cette résolution est de 320 ou 640 pixels de large par 200 lignes. Seize couleurs par ligne peuvent être utilisées dans ces 2 modes, avec plus ou moins de restrictions. 32K de mémoire sont alloués pour ce mode graphique, à une adresse fixe: de \$E12000 à \$E19EFF.

A chaque ligne d'écran correspond un byte appelé SCB (Scan Line Control Byte), ce byte définit 4 paramètres pour la ligne en question

1. La résolution horizontale de la ligne en question (320 ou 640 points)
2. Quelle palette de couleurs est utilisée pour cette ligne (16 palettes définissables)
3. Si une interruption IRQ doit être générée lors du rafraichissement de la ligne
4. Si le mode fill (remplissage) est actif ou non

4.1.1 Structure d'un SCANLINE CONTROL BYTE (SCB)

Bits 0-3	Numéro de la palette assignée : Réserve (mis à 0)
Bit 4	Réserve (mis à 0)
Bit 5	Mode fill (1=on, 0=off)
Bit 6	Interruption (1=réclamée, 0=non réclamée)
Bit 7	Nombre de pixels horizontaux (0=320px, 1=640px)

La table contenant tous les SCB commence en \$E19D00 (pour la première ligne) et s'étend jusqu'en \$E19DC7 (ligne 200). La zone mémoire de \$E19DC8 à \$E19DEF est réservée et devrait être remplie de zéros.

Les données concernant les points eux-mêmes commencent en \$E12000, chaque ligne est représentée par \$A0 (160) bytes

Il choses se présentent donc ainsi :

Ligne	SCB	Points
0	\$E19D00	\$E12000 - \$E1209F
1	\$E19D01	\$E120A0 - \$E1213F
2	\$E19D02	\$E12140 - \$E121DF
3	\$E19D03	\$E121E0 - \$E1227F
...
198	\$E19DC6	\$E19BC0 - \$E19C5F
199	\$E19DC7	\$E19C60 - \$E19CFF

Les données pour chaque point dépendent de la résolution de la ligne (320 ou 640).

En mode 320 points par ligne, chaque point peut prendre 16 couleurs différentes, chaque point est donc codé par un nibble (un demi byte, 4 bits) représentant une valeur entre 0 et 15, cette valeur sert de pointeur pour la couleur dans la palette assignée à la ligne.

On y retrouve notre compte puisque :

1. 2 valeurs \wedge 4 bits = 16 couleurs possibles
2. 160 bytes/ligne nous donne : 160 bytes * 2 points /byte = 320 points / ligne

En mode 320, les nibbles les plus significatifs codent pour les points pairs, les nibbles les moins significatifs pour les points impairs.

Exemple : \$E12000:AB signifie que le point 00 de la première ligne est de couleur \$0A (10), le point 01 de couleur \$0B (11). Un byte code donc pour 2 points en mode 320.

Le mode 640 est un peu plus complexe, chaque point est codé par 2 bits seulement ($2^2 = 4$ et 160 bytes * 8/2 = 640 points /ligne). La couleur représentée par ces bits dépend à la fois de leur valeur, et de leur position dans le byte : le point représenté par les 2 bits les plus forts du byte (6-7) peut prendre les 4 premières couleurs de la palette en vigueur (0-3), le point représenté par les 2 bits suivants (4-5) peut prendre les 4 couleurs suivantes de la palette (4-7) etc... (voir la table suivante). Un byte code pour quatre points en mode 640.

Le mode 320 permet donc d'afficher seize couleurs par ligne, sans aucun conflit (comme c'était le cas dans le mode HGR de l'Apple II par exemple), à chaque ligne peut être assignée une palette particulière, jusqu'à concurrence de seize palettes différentes pour une image donnée. Ceci pousse le nombre de couleurs maximum théorique à 256 par

image, tout en le limitant à seize par ligne. Nous verrons dans le chapitre «PROGRAMMATION SPECIALE» comment améliorer cet état de fait.

Le mode 640 nous permet également seize couleurs par ligne, mais comme nous l'avons vu, chaque point particulier ne peut prendre que certaines couleurs précises, impossible dès lors, d'obtenir des images 640*200 en seize vraies couleurs.

Toutefois certains programmes de dessin (et même le FINDER) nous propose de choisir parmi seize couleurs en mode 640! La technique qu'ils utilisent pour obtenir ce résultat s'appelle «DITHERING», elle consiste simplement à aligner deux points de deux couleurs différentes pour en simuler une troisième. Ce système réduit la résolution à 320 points par ligne (2 points 640 pour un point coloré), mais il permet de juxtaposer (p. ex) du texte en 640, deux couleurs, et des motifs colorés. On remarque très bien le DITHERING à l'écran, l'image semble plus «granuleuse».

Une autre technique utilisable est d'avoir des lignes en 320 et des lignes en 640 sur le même écran : les premières permettant des dessins colorés, les autres du texte en haute résolution noir & blanc.

4.1.2 Palettes

Les seize palettes définissables occupent la mémoire de \$E19E00 \$E19FFF, à raison de 32 (\$20) bytes par palette, et de deux valeurs par couleur. Pour chacune des couleurs, on définit l'intensité de ses composantes verte, bleue et rouge. Chaque intensité peut prendre seize niveaux, elle est représentée par un nibble, on a donc la possibilité d'afficher $16 \cdot 16 \cdot 16 = 16^3 = 4096$ couleurs différentes (RGB 12 bits, 4 bits par composante). Deux bytes (un Word) sont alloués pour chaque couleur de chaque palette, les intensités sont distribuées de la façon suivante : la composante verte occupe le nibble le plus fort du premier byte, la composante bleue le nibble le plus faible de ce même byte, enfin, la composante rouge occupe le nibble le plus faible du deuxième byte, le nibble le plus fort du deuxième byte est en principe réservé et devrait être laissé à zéro. La couleur zéro de chacune des palettes est définie par les deux premiers bytes de celle-ci, la couleur une par les deux suivants, exemple : la troisième couleur de la deuxième palette est définie en \$E19E46-\$E19E47...

N° Palette	Adresse
0	\$E19E00
1	\$E19E20
2	\$E19E40
3	\$E19E60
...	

N° Palette	Adresse
14	\$E19FC0
15	\$E19FE0

L'adresse d'une couleur particulière d'une palette particulière est égale à : \$E19E00 + \$20 * le numéro de la palette + 2 * le numéro de la couleur.

16 BITS DEFINISSANT UNE COULEUR

VVVVBBBB 0000RRRR — 4 bits pour l'intensité du rouge

4 bits pour l'intensités du bleu

4 bits pour l'intensité du vert

Exemples de couleurs définissables :

Binaire	Hexadécimal	Couleur
0000000000000000	\$0000	Noir
1111000000000000	\$F000	Vert
0000111100000000	\$0F00	Bleu
0000000000001111	\$000F	Rouge
0000111100001111	\$0F0F	Violet
1111000000001111	\$F00F	Jaune
1111111100000000	\$FF00	Bleu clair
1111111100001111	\$FF0F	Blanc
1000100000001000	\$8808	Gris 8

4.1.3. Interruptions SCANLINE

Le IIGS, contrairement à l'Apple II, autorise l'utilisateur à contrôler les interruptions software de type IRQ, des interruptions de ce type peuvent être générées par le processeur vidéo, appelé VGC (Vidéo Graphic Controller). Une interruption «SCANLINE» est générée en début de rafraichissement d'une ligne donnée, si son SCB a le bit 6 mis à 1. Ce type d'interruption est crucial pour les animations graphiques, nous en reparlerons en détails, avec de nombreux exemples lorsque nous parlerons de programmation d'animations diverses... Pour l'instant, la seule chose à retenir, est le fait que le bit 6 d'un SCB détermine si l'interruption sera générée ou non au niveau de la ligne correspondante (d'autres conditions sont bien sûr nécessaires !).

4.1.4. MODE FILL (REPLISSAGE)

Ce mode semble être l'exclusivité de l'Apple IIgs, je ne connais en effet pas d'autres machines possédant un système même rapproché. Ce mode permet de remplir instantanément de grandes surfaces à l'écran, il n'est accessible qu'en mode 320 points par ligne. Lorsqu'il est actif, la couleur 00 disparaît, ne nous laissant plus qu'un choix de 15 couleurs. Chaque point de couleur 00 aura, à l'écran, la couleur du point qui le précède !

Ainsi, la séquence suivante :

```
2000000 400000000 60000000000 70000
```

apparaîtra comme :

```
2222222 444444444 66666666666 77777
```

La seule restriction est que le premier point d'une ligne en mode «FILL» doit avoir une couleur différente de 00, ou alors il apparaîtra avec une couleur indéterminée (dépendante des couleurs de la ligne précédente).

Ce mode est actif pour une ligne, si le bit 5 de son SCB est à 1. Il peut s'avérer extrêmement utile pour les animations graphiques rapides d'objets, où l'on peut se contenter de redessiner seulement les contours des objets, le remplissage se faisant par le hardware

4.1.5 Mémoire graphique (BANC \$E1)

Total : \$7D00 + \$300 = \$8000 bytes = 32k

Cette mémoire se trouve en banc \$E1, un banc de mémoire «lente» (appelée ainsi car à chaque accès à cette mémoire, le processeur est ramené à sa vitesse d'origine : 1 MHz, consulter le chapitre sur la mémoire pour de plus amples renseignements). Pour accélérer les accès à la mémoire, on peut utiliser le «SHADOWING». Décrit en détails dans un autre chapitre, ce mécanisme nous permet, lorsqu'il est actif, de lire et d'écrire les données graphiques en banc \$01, celles-ci étant recopiées à l'écriture en banc \$E1. Cette technique ne résout pas tous les problèmes de vitesse, car il ralentit également le processeur, mais lors de l'écriture uniquement (les valeurs doivent être recopiées automatiquement par le hardware du banc \$01 au banc \$E1). Nous reparlerons abondamment du SHADOWING avec les techniques d'animation, pour l'instant, citons juste le softswitch à modifier pour «SHADOWER» la page SUPER-HIRES : il s'agit de \$C035.

Lorsque ce registre est à 00, toute valeur écrite à une adresse du banc \$01 est automatiquement copiée à l'adresse correspondante dans le banc \$E1.

Par exemple, la séquence suivante :


```
LDA  #$FF
STAL $E19E1F
```

est équivalente à:

```
STZ  $C035
LDA  #$FF
STAL $019E1F
```

Pour visualiser cette mémoire graphique, il faut manipuler un softswitch, il s'agit de NEWVIDEO, à l'adresse \$CO29. 2 bits de ce softswitch sont importants à connaître :

- Le bit 7 : s' il est mis a 1, permet l'affichage de la page Superes, s' il est à 0, permet l'affichage de tous les autres modes

- Le bit 6 : doit être mis à 1 si l'on désire lire ou écrire dans la mémoire graphique sans affichage. En effet, ce bit permet de linéariser la mémoire graphique, si on oublie de le mettre à 1 avant d'écrire, les valeurs écrites risquent de ne pas se retrouver au bon endroit.

Le bit 5 a également une fonction :

- Il détermine si la page double haute résolution est en mode couleur ou en mode noir & blanc. Bit 5 : 0 = double haute résolution couleur, 1= double haute résolution N&B.

Les bits 0 à 4 sont réservés et ne doivent pas être modifiés. Lorsque le mode TEXT est sélectionné, \$CO29 contient \$21. Consulter le chapitre sur les softswitchs pour plus d'informations.

En théorie, on devrait modifier ce switch avec ce type de séquences :

```
LDA  $C029
ORA  #$80          ; Affiche la Superes (bit 7 à 1)
STA  $C029
```

```
LDA  $C029
ORA  #$40          ; Linéarise la Superes (bit 6 à 1)
STA  $C029
```

```
LDA  $C029
AND  #$3F          ; Passe en mode TEXT (bits 7,6 et 5 mis à 0)
STA  $C029
```

Mais en pratique, on utilise souvent des stockages directs:

```
LDA  #A1          ; Affiche la Superes
```

```
STA  $C029
```

```
LDA  #A1          ; Linéarise la mémoire graphique
```

```
STA  $C029
```

```
LDA  #A1          ; Repasse en mode TEXT
```

```
STA  $C029
```

Depuis le moniteur, on peut visualiser la page SUPER-HIRES en tapant :

```
C029:A1
```

```
CTRL-T <RETURN> remet le mode texte
```

De même, il est facile de récupérer une image d'écran et de la sauver, sous PRODOS 8, sous moniteur taper :

```
C029:41 : reste en mode TEXT, mais permet l'accès à la mémoire
```

```
0/1000<E1 /2000.A000M : déplace la mémoire graphique en banc $0
```

```
BSAVE IMAGE,A$1000,L$8000 : Sauve l'image sur disque
```

Pour réafficher l'image après coup (sous PRODOS 8 et moniteur toujours)

```
BLOAD IMAGE : charge l'image en banc $00
```

```
CO29:41 : linéarise la mémoire avant de copier
```

```
E1/2000<0/1000.9000M : copie l'image
```

```
C029:A1 : Affiche l'image
```

Voir également dans les exemples de programmes qui suivent, les routines automatiques de copie et d'affichage.

COPIE: Ce programme copie la mémoire graphique en banc 00, d'où on peut la sauver.

```
CLC
```

```
XCE          ; Passe en mode 65C816
```

```
SEP  #A30     ; Passe les registres A,X et Y en 8 bits
```

```
LDA  #A41     ; linéarise la mémoire pour pouvoir
```

```

STA  $C029      ; la lire
REP  #$30      ; Passe les registres A,X et Y en 16 bits
LDX  #$7FFE    ; Le registre X sert de compteur
:1   LDAL  $E12000,X ; On copie les bytes de deux par deux
     STAL  $002000,X
     DEX                      ; et l'on décrémente donc le compteur
     DEX                      ; deux fois
     BPL  :1                  ; On répète la boucle s'il reste des bytes
     SEP  #$30                ; Passe les registres A,X et Y en 8 bits
RTS

```

Après exécution, on peut sauver l'image : BSAVE IMAGE,A\$1000,L\$8000

Affichage: copie une image (données, SCBs et palettes) du banc 00 (ou elle a été chargée) vers la mémoire graphique. Ce programme réalise l'opération inverse du programme précédent

```

CLC
XCE                      ; Passe en mode 65C816
SEP  #$30                ; Passe les registres A,X et Y en 8 bits
LDA  #$41                ; linéarise la mémoire pour pouvoir
STA  $C029                ; la lire
REP  #$30                ; Passe les registres A,X et Y en 16 bits
LDX  #$7FFE              ; Le registre X sert de compteur
:1   LDAL  $002000,X      ; On copie les bytes de deux par deux
     STAL  $E12000,X
     DEX                      ; et l'on décrémente donc le compteur
     DEX                      ; deux fois
     BPL  :1                  ; On répète la boucle s'il reste des bytes
     SEP  #$30                ; Passe les registres A,X et Y en 8 bits
RTS

```

Lorsque l'on travaille sur la page graphique, il est toujours agréable de pouvoir connaître à tout instant l'adresse d'une ligne particulière. La formule en est simple :

Adresse = \$2000 +nligne* \$A0

Mais en pratique, il est assez compliqué de réaliser des multiplications et plutôt long d'additionner des nombre en chaine. On préférera donc créer une tables avec les adresses de toutes les lignes. Le programme suivant s'en occupe :

	CLC	
	XCE	; Passe en mode 65C816
	REP #30	; Passe les registres A,X et Y en 16 bits
	LDX #0000	; Le registre X sert de compteur
	LDA #2000	; Adresse de départ (ligne zéro)
:1	STA	; On copie les bytes de deux par deux
	INX	; et l'on décrémente donc le compteur
	INX	; deux fois
	CLC	
	ADC #A0	
	CMP #9D00	
	BNE :1	; On répète la boucle 200 fois
	SEP #30	; Passe les registres A,X et Y en 8 bits
	RTS	

Il crée une table, à l'adresse TABLE, qui contient les adresses de toutes les lignes (codées en seize bits), sous la forme : byte faible, byte fort.

Pour obtenir l'adresse du premier octet d'une ligne quelconque, il suffit de faire :

	CLC	
	XCE	; Passe en mode 65C816
	REP #30	; Passe les registres A,X et Y en 16 bits
	LDA Ligne_Nb	; Numéro de la ligne dont on veut l'adresse
	ASL	; On le multiplie par deux
	TAX	; et on passe cette valeur dans X
	LDA TABLE,X	; Le registre A contient maintenant l'adresse
		; (16 bits) où commencent les données de cette
	...	; ligne

4.1.6 Registre monochrome / couleur

Ce registre MONOCOLOR (\$C021), option DISPLAY dans le CONTROL PANEL, détermine si la sortie composite du IIGS doit être en couleur ou en tons de gris. Le bit 7 (le bit le plus fort) est le seul modifier: s' il est à 1, la sortie est monochrome, s' il est à 0, la sortie est couleur.

Si vous avez un moniteur monochrome, choisissez l'option monochrome, les images seront plus lisses, les couleurs étant représentées par des dégradés de gris plutôt que par la juxtaposition de points.

\$C021 : REGISTRE «MONOCOLOR»

bit 7: 0 = couleur
 1 = monochrome
bits 0 - 6 : réservés

4.1.7 Couleur en mode TEXT

Avec l'Apple IIGS, nous avons la possibilité de changer la couleur des caractères en mode TEXT, la couleur du fond, ainsi que celle de la bordure d'écran. La couleur pour chacune de ces zones peut être choisie parmi 16 couleurs disponibles (les couleurs sont les mêmes que celles de l'ancien mode GR, la basse résolution). On peut sélectionner ces couleurs soit depuis le CONTROL PANEL, soit directement en modifiant les softswitchs adéquats. Comme chaque couleur ne peut prendre que 16 valeurs, elle seront codées par un nibble (4 bits).

Deux softswitchs sont impliqués dans ces réglages : TBCOLOR (\$C022) et CLOCKCTL (\$C034).

Nom	Adresse	Bits	Description
TBCOLOR	\$C022	7-4 3-0	Couleur du texte Couleur du fond
CLOCKCTL	\$C034	7-4 3-0	Bits horloge, ne pas modifier Couleur du bord

Attention, ne pas toucher au bit 7 de \$C034 lors du changement de la couleur du bord, utiliser des instructions du type :

La table suivante donne les équivalents de chacune des couleurs en système RGB.

N° Couleur	Couleur	Équivalent RGB	Équivalent binaire
\$00	Noir	\$0000	0000000000000000
\$01	Rouge	\$030D	0011000000000000
\$02	Bleu foncé	\$0900	0000000000000000
\$03	Violet	\$2D0D	0000000000000000
\$04	Vert foncé	\$7200	0000000000000000
\$05	Gris foncé	\$5505	0000000000000000
\$06	Bleu moyen	\$2F02	0000000000000000
\$07	Bleu clair	\$AF06	0000000000000000
\$08	Brun	\$5008	0000000000000000

N° Couleur	Couleur	Équivalent RGB	Équivalent binaire
\$09	Orange	\$600F	0000000000000000
\$0A	Gris clair	\$AA0A	0000000000000000
\$0B	Rose	\$9808	0000000000000000
\$0C	Vert clair	\$D000	0000000000000000
\$0D	Jaune	\$F00F	1111000000001111
\$0E	Aquamarine	\$F904	1111100100000100
\$0F	Blanc	\$FFF0F	1111111100001111

Le programme d'exemple qui suit montre comment harmoniser les couleurs de l'écran SUPER HIRES avec celle du bord, en se servant de la table décrite précédemment.

Il est regrettable que l'on ne puisse pas afficher des caractères de couleurs différentes sur un même écran texte. Cet état de fait est certainement dicté par la compatibilité avec les anciens modèles de la série II. Il aurait en effet fallu, le cas échéant, modifier le mode text, ajouter de la mémoire pour la couleur de chaque caractère etc.. C'est dommage, car un mode ressemblant au mode ANSI d'IBM aurait pu nous ouvrir des portes (de plus en plus de serveurs américains utilisent à fond le mode ANSI).

4.1.8 NTSC / PAL

Le système TV américain diffère du système européen de part sa fréquence de rafraichissement d'image. Celle-ci est de 60 HZ aux Etats-Unis pour 50 HZ en Europe. 60 HZ (Hertz) signifie simplement que le processeur vidéo rafraichit l'écran 60 fois par seconde; qu'il envoie, 60 fois par seconde, les données d'affichage de chaque ligne vers l'écran.

Un bit du registre LANGSEL (\$C02B) (qui contient également les données relatives au langage du clavier), nous signale si le GS est en mode PAL ou NTSC. Ce bit est le bit 4. S'il est à 1 le système est en PAL, s'il est à 0, le système est en NTSC. Cette information se règle au RESET général (CTRL-OPTION-RESET) et est visible dans le CONTROL PANEL, option DISPLAY.

Les séquences à utiliser pour changer de mode sont les suivantes :

```
LDA  $C02B          ; Passe en mode PAL
ORA  #$10           ; Met le bit 4 à 1
STA  $C02B
```

...

```
LDA    $C02B          ; Passe en mode PAL
```

```
AND    #$EF          ; Met le bit 4 à 0
```

```
STA    $C02B
```

...

En général, la plupart des moniteurs vidéo acceptent des signaux dans les deux standards. Mais si vous utilisez une télévision munie d'une prise péritel, et que votre GS est en 50 Hz, vous pourriez avoir la désagréable surprise de voir votre téléviseur perdre sa synchronisation lors d'un passage en 60 HZ; ce dernier ne reconnaissant pas le signal NTSC.

Parlons maintenant des différences entre ces 2 modes. Nous avons dit qu'en mode 60 HZ, l'écran était balayé plus souvent; pourtant le temps de balayage d'une ligne doit rester et reste toujours le même. Il en découle qu'en 60 HZ, le processeur vidéo est obligé de balayer moins de lignes, tout simplement !

Mode	Lignes balayées
50 Hz	312
60 Hz	262

Le rapport 50/60 est bien approximativement égal au rapport 262/ 312. Ceci signifie que :

- 1) Si un programme utilise le balayage en général pour sa temporisation, il ira environ 1/6 plus vite sur un système NTSC que sur un système PAL.
- 2) Si un programme utilise à fond le balayage pour se synchroniser et qu'il accapare pratiquement tout le temps d'un balayage 50Hz, il se désynchronisera en 60Hz, car le temps d'un balayage est plus court d'un sixième environ !

Moralité : si on programme des animations synchronisées, mieux vaut le faire en 60 Hz, car elles marcheront toujours en 501-lz (temps de balayage plus long).

Encore une dernière remarque : le processeur vidéo semble déterminer à quelle fréquence l'image sera balayée (50 ou 60HZ) au moment du VBL (instant où le faisceau électronique «remonte» du coin en bas droite vers le coin en haut à gauche). Il est donc inutile d'essayer de changer de mode en milieu d'écran, le changement ne s'effectuant qu'au balayage suivant. Si tel n'avait pas été le cas, on aurait pu imaginer pouvoir afficher plus de 200 lignes en hauteur. En changeant de mode de manière synchronisée, on aurait peut-être pu «répéter» l'affichage d'un certain nombre de lignes !

On peut encore se livrer à quelques petits calculs concernant les temps disponibles durant un balayage. Le 65C816 «tourne» au maximum environ 2.8 MHZ, ce qui signifie qu'il exécute 2800000 cycles/ seconde. En mode NTSC, l'écran est balayé 60 fois par seconde, on peut en déduire que le nombre de cycles disponibles par balayage est de $2800000/60 = 46000$ cycles. Si on divise ce chiffre par 262 (nombre de lignes rafraichies par balayage), on se retrouve avec 178 petits cycles disponibles par ligne d'écran. C'est peu, surtout que ce chiffre est encore trop élevé par rapport à la réalité : tout d'abord le 65C816 tourne toujours un peu plus lentement (il est ralenti par des accès DMA, des interruptions de service etc..). Ensuite, ce calcul présuppose que le processeur reste à vitesse constante, ce qui est loin d'être le cas, puisqu'à chaque accès à de la mémoire lente (banc \$E0 et \$E1), il repasse en mode 1 MHZ.

Ces questions de temps seront réexaminées avec les commentaires du programme «MULTICOULEURS».

4.2. Animations graphiques, Programmation spéciale

A la lumière des constatations qui précèdent, on se rend compte qu'il est plutôt difficile de programmer des animations rapides et de grande envergure sur l'Apple IIgs. On peut néanmoins atteindre un niveau acceptable, en utilisant des astuces de programmation. L'analyse qui suit, illustrée de nombreux exemples de programmes, dévoile quelques unes de ces astuces, utilisées dans la plupart des programmes de jeu vendus dans le commerce.

4.2.1 Synchronisation

Pour que les parties de l'écran graphique en cours de mouvement semblent «glisser» harmonieusement, il est impératif de les faire bouger lorsque le processeur vidéo ne les balaye pas (lorsque le faisceau électronique rafraichit une autre partie de l'écran) Pour s'assurer de ce fait, il existe plusieurs moyens (j'en vois en fait 5) :

(1) Utilisation du registre \$C070 (marche sur Apple II+, IIe, IIc et IIgs)

Ce registre est un écho du byte actuellement lu par le processeur vidéo (en mode Text et Simple High-Res en tout cas). Donc pour synchroniser le 65C02 sur le haut de l'écran High-Res, faire HGR, puis sous moniteur, remplir (p. ex) la zone \$2000-\$2020 avec des \$AB et écrire le programme suivant :

```
:1          LDA    $C070
           CMP    #$AB          ; Attend que le faisceau passe sur la zone
           BNE   :1          ; qui contient les valeurs $AB
```


En étant en mode HGR, il faut plusieurs «AB» car le processeur vidéo est plus rapide que le 65C02 (je reviendrai ultérieurement sur ces problèmes de vitesse et temps avec la description du registre HORIZCNT)

Cette méthode est limitée, par exemple si l'image affichée contient des «AB», la synchro peut sauter de temps en temps.

Ceci n'est pas une innovation de l'Apple IIGS, mais ce softswitch me paraissait intéressant à souligner, car son utilisation dans ce but n'a jamais été décrite (à ma connaissance). C'est dommage, car c'était certainement le seul moyen de synchronisation sur Apple II+ et IIe. Seuls d'anciens programmeurs Californiens semblent le connaître!

(2) Le registre \$C019 (RDVBLBAR) (marche sur certains IIc et sur IIGS) Ce registre est à \$00 lorsque l'écran (Text, graphique.) est balayé et \$80 lorsque le processeur vidéo rafraichit les bordures haut ou bas (VBL). Donc, pour synchroniser le processeur sur le début de la bordure du bas, il suffit d'écrire

```
:1          LDA    $C019          ; Assure que l'on soit sur l'écran
           BMI    :1
:2          LDA    $C019          ; Assure que l'on passe sur le bord inférieur
           BPL    :2
```

Un programme de démonstration de cette technique pourrait être :

```
:1          LDA    $C019          ; 65C02 synchronisé sur le début du bord
           BPL    :1              ; inférieur
           LDA    #$0E
           STA    $C034
:2          LDA    $C019
           BMI    :2
           SZT    $C034
           BRA    :1
```

Ce qui nous donne des bordures haut et bas couleur «aquamarine» et des bordures latérales noires

(3)Le registre \$C02E (HORIZCNT), qui est le reflet de rafraichissement. En fait, ce registre est incrémenté toutes les 2 lignes seulement, il peut prendre les valeurs suivantes :

Mode	Valeurs de HORIZCNT (\$C02E)
50 Hz	\$7D à \$FF
60 Hz	\$64 à \$FF

Ce registre permet une synchronisation verticale relativement fine, il suffit pour cela d'attendre une valeur particulière à l'aide d'une séquence LDA, CMP, BNE, exemple:

```

                CLC    SmartPort
XCE    $00      ; Passe en mode 65C816
SEP    #$30     ; Passe les registres A,X et Y en mode 8 bits
DEBUT  LDA    #$00
:1     LDA    $C02E ; Position verticale
        CMP    POS,X ; attend une position particulière
        BNE   :1

                LDA    $C034
                AND    #$F0
                ORA    COUL,X
                STA    $C034 ; Change la couleur du bord et celle du fond
                LDA    COUL,X
                STA    $C022 ; couleur des caractères
                INX          ; = 00
                CPX    #$04 ; Si X=4, on le remet à zéro
                BNE   :1
                BRA   DEBUT

POS     HEX    A0,B0,C0,D0 ; Position des changements de couleur
COUL    HEX    02,0F,01,00 ; Couleurs: bleu, blanc, rouge et noir pour finir

```

Le registre \$C02F (HOR1ZCNT) est un reflet de la position horizontale du faisceau électronique, cette valeur change très rapidement, de sorte que le 65C816 n'est pas assez rapide pour se synchroniser à coup sûr avec une séquence de type LDA CMP BNE. Ce registre peut néanmoins être utilisé à cette fin, si par exemple on teste une valeur sur plusieurs balayages de suite. Il faut alors garder précieusement la synchronisation acquise. Une autre technique consiste à chercher plusieurs valeurs de positions horizontales possible, chacune conduisant à une temporisation différente, aboutissant toutes à la même synchronisation. Cette discussion est purement pratique, Apple ne donnant d'ailleurs pas beaucoup d'informations sur ces registres. Enfin, vu leurs

changements de valeurs très rapides, ces registres sont souvent utilisés comme générateurs de nombres aléatoires.

Exemple : le programme suivant fournit des nombres relativement aléatoires entre \$00 et \$FF

```
TEMP = $00
```

```
CLC
```

```
XCE          ; Passe en mode 65C816
```

```
SEP  #$30    ; Passe les registres A, X et Y en 8 bits
```

```
ASL          ; On extrait le nibble le plus faible
```

```
ASL
```

```
ASL
```

```
ASL
```

```
EOR  $C02F   ; HORIZCNT change très rapidement
```

```
STA  TEMP
```

```
LDA  $C02E
```

```
EOR  TEMP
```

```
EOR  $C02F
```

Après l'exécution de cette routine, le registre A contient un nombre aléatoire entre \$00 et \$FF

(4) L'interruption Scanline. Cette interruption est l'une des deux que le VGC (Video Graphic Controller) puisse générer, l'autre étant l'interruption «une seconde». Lorsqu'une interruption software de type IRQ a lieu, le 65C816 interrompt l'exécution du programme en cours, sauve l'adresse à laquelle il se trouvait ainsi que le registre d'état (P) dans la pile et saute dans un programme de la mémoire morte (ROM) appelé INTERRUPT MANAGER. L'INTERRUPT MANAGER commence par sauver les données minimums du moment (valeurs des registres...) puis il détermine ensuite la source de l'interruption (graphique, sonore, en provenance de la souris...) et finalement le contrôle est donné à la routine d'interruption au travers d'un vecteur. Lorsque cette routine a terminé son travail, le chemin se fait en sens inverse, le contrôle est redonné à l'INTERRUPT MANAGER, qui remet en place les données sauvegardées et termine par une instruction RTI (Return from Interruption) qui remet P et le Compteur de Programme à leur valeurs initiales, continuant ainsi l'exécution du programme interrompu comme si rien ne s'était passé.

La mise en oeuvre d'une interruption SCANLINE requiert les étapes suivantes :

- Le mode graphique Super Hires doit être actif

- Au moins une ligne d'écran doit avoir le bit d'interruption de son SCB à 1
- Le registre d'interruption VGC (\$C023) doit avoir le bit d'autorisation d'interruption SCANLINE mis à 1 (bit 1)
- Le vecteur d'interruption SCANLINE doit pointer sur une routine de gestion d'interruption (JMPL en \$E10028)
- Cette routine doit signaler qu'elle s'est chargée de l'interruption en mettant le bit 6 du registre SCANLNT (\$C032) à 0
- La routine d'interruption peut maintenant rendre le contrôle L'INTERRUPT MANAGER, par l'intermédiaire d'un CLC, RTL.

Reprenons ces étapes en détail :

1) Pour activer le mode SUPER-HIRES, il suffit de mettre #\$A1 dans le registre NEWVIDEO (\$C029)

2) Le bit d'interruption dans un SCB est le bit 6, pour le mettre à un, il suffit d'écrire :

```

LDX  #$10          ; X = Numéro de la ligne où l'on désire
LDAL $E19D00,X    ; l'interruption
ORA  #$40
STAL $E19D00,X

```

On peut également, à titre préventif, remettre les bits d'interruption des autres SCB à 0, pour être sûr de n'avoir qu'une seule interruption par balayage d'écran. On peut le faire comme ceci :

```

LDX  #$C8
:1   LDAL $E19CFF,X
AND  #$BC
STAL $E19CFF,X
DEX
BPL  :1

```

3) VGCINT : \$C023

Bit 1 - Autorise l'interruption SCANLINE s'il est à 1

Bit 2 - Autorise l'interruption «Une Seconde» s'il est à 1

Bit 5 - Statut de l'interruption SCANLINE, mis à 1 si l'interruption a eu lieu

Bit 6 - Statut de l'interruption une seconde, mis à 1 si l'interruption a eu lieu

Bit 7 - Statut d'interruption VGC, est mis à 1 si à la fois un bit d'autorisation et un bit de statut d'une même interruption sont à 1

Les bits 0,3 et 4 sont inutilisés et ne doivent pas être modifiés.

En pratique, les bits 5,6 et 7 servent à détecter si une interruption VGC a eu lieu ou non (L'INTERRUPT MANAGER s'en sert pour déterminer quel type d'interruption a été déclenchée). Ce qui nous intéresse pour le moment, c'est d'autoriser l'interruption SCANLINE, il suffit pour cela de forcer le bit 1 à 1 par l'instruction suivante :

```
LDA  $C023
ORA  #$02
STA  $C023
```

4) Le vecteur d'interruption SCANLINE se trouve à l'adresse \$E10028 (avec les autres vecteurs d'interruption IRQ, interruption une seconde en \$E10054 p.ex). Si on désassemble cette zone sur un Apple IIGS équipé d'une ROM 01, on trouve :

```
$E10028:  JMPL  $FFB5DE
```

À l'adresse \$FFB5DE, on trouve simplement le code suivant:

```
$FFB5DE:  SEC
$FFB5DF:  RTL
```

Cela signifie, que par défaut, lorsqu'une interruption est générée, rien ne se produit (sauf que l'on perd un peu de temps).

Pour modifier ce vecteur, il faut changer l'adresse suivant le JMP (n de banc y compris). Par exemple, pour faire pointer le vecteur vers l'adresse \$0320, banc \$00, la séquence suivante fait très bien l'affaire:

```
LDA  #$02
STAL $E10029
LDA  #$03
STAL $E1002A
LDA  #$00
STAL $E1002B
```

5) Pour signaler que l'interruption a été prise en compte, le programme de gestion doit effacer un bit du registre SCANINT (\$C032)

Bit 5 - doit être remis à zéro pour que les interruptions SCANLINE continuent à avoir lieu.

Bit 6 - doit être remis à zéro pour que les interruptions Une Seconde continuent à avoir lieu.

La séquence adéquate pour remettre SCANLINE à zéro est donc :

```
LDA  $C032
AND  #$DF
STA  $C032
```

6) Le programme de gestion rend la main à l'INTERRUPT MANAGER à l'aide d'un :

```
CLC
RTL
```

Le CLC (mise à zéro de la retenue) indique à l'INTERRUPT MANAGER que l'interruption a été correctement traitée, L'instruction RTL (Retour Long) est utilisée, car les vecteurs sont appelés par des JSL (JSR longs). Le vecteur par défaut pointe en ROM, sur un SEC, RTL qui indique à l'INTERRUPT MANAGER qu'aucune routine de gestion n'est installée.

Remarques :

- il faut bien sûr que le bit d'interruption du registre d'état (P) soit zéro (interruptions autorisée) pour que cette séquence d'événements puisse avoir lieu. Ceci est accompli par une seule instruction : CLI (CLear Interrupt bit)

- L'interruption demandée pour une ligne a lieu au début du rafraichissement de cette dernière. C'est à dire une ligne plus haut,

l'interface entre l'écran graphique proprement dit et la bordure droite. Malheureusement, l'INTERRUPT MANAGER prend énormément de temps pour déterminer la source de l'interruption, sauver les registres.. .etc si bien que lorsque la routine d'interruption prend le contrôle, plusieurs lignes auront déjà été balayées (environ 3). Cette perte de temps peut être réduite si l'on prend directement le contrôle de l'interruption en sautant l'INTERRUPT MANAGER. Il suffit pour cela de modifier le vecteur en \$E10010 (INTERRUPT MANAGER de la mémoire morte). Dès cet instant, il faut gérer soi-même toutes les interruptions IR() (vidéo, son, clavier, souris, interface série...) ce qui est loin d'être facile ! En pratique, il est

fortement déconseillé de toucher à ce vecteur, à moins peut-être, d'avoir une tâche très précise et rapide à accomplir.

Le programme qui suit est un exemple d'utilisation de l'interrupt SCANLINE. Il dessine un point blanc qui rebondit entre les bords gauche et droit de l'écran le point bouge de deux positions à chaque balayage. Le programme est entièrement autonome et lorsqu'il est lancé, on se retrouve sous moniteur (avec la page SUPER-HIRES affichée). CTRL-T permet de revenir au mode text, en arrêtant l'interruption puisque la SUPER-HIRES est alors inactivée. Tout peut être réactivé en tapant : C029:A1 (retour en SUPER-HIRES). On peut aussi lancer un programme basic en parallèle, ce dernier étant interrompu 50 ou 60 fois par seconde Les possibilités sont innombrables.

Le programme BASIC, lance le programme "POINT" en interruption, puis extrait les racines carrées des nombres de 1 A 256 en incrémentant la couleur de la bordure (adresse \$C034 = 49204). Lorsque ceci est terminé le mode text est forcé (49193 = \$C029).

Une démonstration d'un mode pseudo multi-tâche, ou 2 programmes semblent s'exécuter en même temps.

```
10 PRINT CHR$(4);BRUN"POINT"  
20 FOR N=1 TO 256  
30 PRINT N,SQR(N)  
40 POKE 49204,PEEK(49204)+1  
50 NEXT  
60 POKE 49193,33
```

PROGRAMME «POINT»

```
ORG $A00
```

```
OLDPOS=$00
```

```
NEWPOS=$01
```

```
DIREC=$02
```

```
CLC
```

```
XCE
```

```
SEP #$30
```

```
LDA #$A1
```

	STA	C029	
	STZ	OLDPOS	; Ancienne position à 00
	STZ	NEWPOS	; Nouvelle position à 00
	LDA	#\$01	; Direction vers la droite
	STA	DIREC	; (+1)
	REP	#\$30	
	LDA	#\$0000	; Efface l'écran graphique
	LDX	#\$7FFE	
CLR	STAL	\$E12000,X	
	DEX		
	DEX		
	BPL	CLR	
	LDA	#\$0FFF	; Couleur 0F de la palette
	STAL	\$E19E1E	; 00 mise en blanc
	SEP	#\$30	
	LDX	#\$C8	; Mise à zéro des bits d'interruption de tous
CLILOP	LDAL	\$E19D00-1,X	; les SCB
	AND	#\$BF	
	STAL	\$E19D00-1,X	
	DEX		
	BNE	CLILOP	
	SEI		
	LDAL	\$E19D66	; Interruption à la ligne 66
	ORA	#\$40	
	STAL	\$E19D66	
	LDA	#\$5C	; Assure la présence d'un JMPL au début du
	STAL	\$E10028	; vecteur
	LDA	#\$00	; Banc mémoire de la routine d'interruption
	STAL	\$E1002B	
	LDA	#INTER	; Adresse de la routine d'interruption, partie
	STAL	\$E10029	; basse
	LDA	#>INTER	; Adresse de la routine d'interruption, partie
	STAL	\$E1002A	; haute
	LDA	\$C023	; Mise à 1 du bit 1 du registre VGCINT


```

ORA   #$02
STA   $C023      ; Autorise SCANLINE
CLI                       ; Assure que le bit INT du registre d'état soit
SEC                       ; à zéro
XCE                       ; La séquence SEC XCE est nécessaire avant le
RTS                       ; retour au BASIC

```

```

INTER   LDA   $C032
        AND   #$DF
        STA   $C032

```

```

PHB
LDA   #$E1
PHA
PLB

```

```

LDX   OLDPOS
LDA   #$00
STA   $5DE0,X
STA   $5E80,X
STA   $5E81,X
STA   $5F20,X

```

```

LDX   NEWPOS      ; Dessine le nouveau point
LDA   #$0F
STA   $5DE0,X
STA   $5F20,X
LDA   #$FF      ; Byte de motif
STA   #$5E80    ; Adresse de la ligne
LDA   #$F0
STA   $5E81,X

```

```

PLB      ; Récupère le banc mémoire
LDA   NEWPOS      ; pour l'effacement ultérieur
STA   OLDPOS
CLC
ADC   DIREC      ; Direction (+1 ou -1)

```

	TAX		
	BEQ	CHNG	; Changement si POS = 00
	CMP	#\$9E	; ou si POS = \$9E
	BNE	OK	
CHNG	LDA	DIREC	; Changement de direction
	EOR	#\$FE	; +1 ou -1 (\$FF)
	STA	DIREC	
OK	STX	NEWPOS	
	CLC		; Tout s'est bien passé
	RTL		; Retour à l'INTERRUPT MANAGER

(5) L'interruption VBL. Cette interruption est générée 50 ou 60 fois par seconde, en fin de balayage de l'écran. Elle se contrôle pratiquement comme l'interruption SCANLINE, une des seules différences notables est le fait qu'elle est active quel que soit le mode vidéo sélectionné. Voici les vecteurs et softswitches impliqués :

Le vecteur se trouve à l'adresse \$E10030

Le switch autorisant le VBL et INTEN (\$C041), voici la signification de ses bits

- Bits 0,1,2 - Autorisent le contrôle de la souris par le chip MEGA II
- Bit 3 - Si mis à un, autorise les interruptions VBL
- Bit 4 - Si mis à un, autorise les interruptions «quart de seconde»
- Bits 5, 6 et 7 - Réservés, mis à zéro

Après le traitement de l'interruption, il faut remettre le switch CLRVLINT (\$C047) à zéro pour que l'interruption ait lieu à nouveau.

Le switch INTFLAG (\$C046) nous renseigne si une interruption VBL a eu lieu ou non. Ses bits les plus importants sont :

- Bit 7- est à un si le bouton de la souris est enfoncé
- Bit 6 - est à un si le bouton de la souris était enfoncé à la dernière lecture
- Bit 4 - est à un si une interruption quart de seconde a eu lieu
- Bit 3 - est à un si une interruption VBL a eu lieu
- Bit 2 - est à un si le switch souris du chip MEGA II a généré une interruption
- Bit 1 - est à un si un mouvement de la souris au travers du chip MEGA II à généré une interruption

Voici un petit programme de démonstration de l'utilisation de l'interruption VBL, qui utilise cette interruption pour changer la couleur de la bordure à chaque balayage.

```

                ORG    $900

                CLC
                XCE
                SEP    #$30

                SEI
                LDA    #$5C          ; Assure un JMPL
                STAL   $E10030
                LDA    #$00          ; Vecteur, banc de mémoire de l'adresse
                STAL   $E10033
                LDA    #INTER        ; Vecteur, partie basse de l'adresse
                STAL   E10031
                LDA    #>INTER       ; Vecteur, partie haute de l'adresse
                STAL   $E10032
                LDA    $C041         ; Interruption VBL autorisée
                ORA    #$08
                STA    $C041
                CLI
                RTS                  ; Revient au moniteur ou au BASIC

INTER          STZ    $C047         ; Régénère l'interruption VBL
                INC    $C034         ; Incréméte la couleur du bord de l'écran
                CLC                  ; Retour à l'INTERRUPT MANAGER
                RTL

```

Note : L'interruption VBL n'est pas tellement utilisée dans un but de synchronisation vidéo, elle agit plutôt comme compteur de temps, comptant les 50èmes ou 60èmes de seconde Elle est utilisée pour la gestion du pointeur de la souris et rythme le HeartBeat» du IIGS (une fonction du système qui permet de chainer des taches exécutées tous les VBL)

Tous ces modes de synchronisation se ressemblent(du moins deux à deux: interruptions VBL avec \$C019-RDVBLBAR et interruptions SCANLINE avec \$C02E-VERTCNT). Il faut choisir lequel utiliser d'après les besoins de son programme. Si une tache doit impérativement être exécutée tous les balayages, mieux vaut utiliser une interruption,

cette dernière étant exécutée même si le programme principal calcule pendant plus d'un balayage.

Encore deux dernières remarques à propos des interruptions en général sur le IIGS:

- Dans les exemples de programmes ci-dessus, les vecteurs sont modifiés en stockant les nouvelles valeurs directement en banc \$E1, ce qui n'est pas une pratique de programmation très "propre". Apple nous met en garde, ces vecteurs pourraient être déplacés dans des versions futures du CS. Nous devrions utiliser les outils «GetVector» (n 1103) et «SetVector» (n 1003) pour respectivement lire et écrire les valeurs des vecteurs d'interruption. Chacun fait ce qu'il veut.
- Il n'y a pas, sur le IIGS, de niveau d'interruption. Ou, autrement dit, toutes les interruptions ont le même niveau d'importance. Ce qui signifie que si une interruption est en cours de traitement, toute interruption générée, quelle qu'elle soit, sera mise en ligne d'attente. Ceci peut s'avérer très gênant si l'on gère des interruptions en provenance de plusieurs sources. Admettons qu'une interruption SCANLINE ait lieu alors qu'une interruption sonore est en cours de traitement, l'interruption vidéo sera retardée de quelques lignes rendant peut-être l'effet recherché caduc ! (voir programme MULTICOULEUR» à la fin du chapitre. La solution à ce problème consiste peut-être à n'utiliser qu'un seul type d'interruption à la fois, remplaçant le travail de l'autre par des switches. Dans notre cas, il suffirait de laisser l'interruption sonore active, d'avancer un peu l'interruption graphique (de quelques lignes), puis, une fois cette dernière générée, de se synchroniser parfaitement à l'aide du switch VERTCNT (\$C02E). La réécriture d'un nouvel INTERRUPT MANAGER pourrait résoudre ce problème, mais à un prix cher payé.

4.2.2 SCROLLINGS tout azimut

Avant la présentation d'un exemple, voici quelques faits relatifs aux mouvements d'images en général :

Si le mouvement est de plus d'un point entre deux positions alors il faut impérativement tout recopier à chaque balayage. Si le temps manque, passer en mode un point entre 2 positions et observer. Il y aura de toute façon des scintillements, mais le résultat peut s'avérer acceptable.

- Pour bouger de grandes zones mémoire, on est obligé de recourir à des boucles en assembleur. Avec l'arrivée du 65C816, on serait tenté d'utiliser les instructions MVP et MVN, qui permettent de copier des blocs de mémoire d'un seul coup. Ces instructions sont intéressantes, car permettant la copie d'un banc de mémoire vers un autre, mais elles sont relativement lentes : il leur faut 7 cycles pour copier un seul byte ! On peut faire nettement plus rapide en utilisant une autre particularité du 65C816 : la possibilité de ce

dernier d'avoir une page zéro et une pile mobiles dans des limites de 64K. Il est dès lors très facile de positionner la page zéro en début d'une ligne graphique, par exemple, et de copier des valeurs sur l'image en utilisant des instructions page zéro, moins gourmandes en cycles.

Instruction	Nombre de cycles	Page zéro
LDA	(\$AD) 4	(\$A5) 3
STA	(\$8D) 4	(\$85) 3
LDA,X	(\$BD) 4+	(\$B5) 3
STA,X	(\$9D) 5+	(\$95) 4

Le + Signifie qu'il faut rajouter un cycle en cas de saut de page, c'est à dire si l'adresse + X dépasse la limite de la page en cours (\$1000, \$1100..., voir chapitre sur le microprocesseur). Encore une chose à savoir : lorsque le 65C816 est en mode 16 bits, toutes les instructions ayant un rapport avec les registres sont majorées d'un cycle. Cette même majoration a lieu lorsque la page zéro est placée à une adresse dont le byte le plus faible n'est pas égal à zéro. Les scrolls les plus rapides seront donc ceux ou l'on bouge l'image quatre points par quatre (16 bits par 16 bits) et dont la page zéro est fixée à une adresse multiple de \$100. Le temps d'exécution d'un couple LDA page zéro, STA page zéro devient alors : $(3 + 1) + (4 + 1) = 9$ cycles, ce qui nous ramène à 4.5 cycles par byte, soit 9/14 seulement du temps pris par une instruction MVN.

Les instructions permettant de changer les valeurs du registre D (position de la page zéro) sont les suivantes :

- TCD Transfert de C (A 16 bits) dans D
- TDC Transfert de D dans l'accumulateur 16 bits
- PHD Empile le registre D (16 bits)
- PLD - Dépille le registre D

Pour permettre à la page zéro et à la pile de se trouver en banc \$01 (la page graphique en puissance si le shadowing est fonctionnel), il faut toucher au registre STATREG (\$C068). Plus précisément, il faut forcer les bits 4 et 5 de ce switch à 1. Ces deux bits permettent respectivement l'écriture et la lecture de la page zéro et de la pile en banc 01. Il faut, en plus, que les routines effectuant ces opérations ne soient pas en banc 00! Elles peuvent être dans n'importe quel autre banc, bien que le banc 01 semble être le plus logique.

Séquence fixant le début de la page zéro en \$2000, (banc 01, p.ex)

Cette routine doit impérativement se trouver en banc 01.

L'exemple qui suit utilise des pseudo codes opératoires de MERLIN 16, qui permettent de générer une série d'instructions. La séquence suivante :

```
LDA  $02
STA  $00
LDA  $04
STA  $02
LDA  $06
STA  $04
```

...

est le fruit de l'assemblage de ce code source :

```
]POS = $00 ; La variable d'assemblage 1POS est mise à zéro
LUP  $7F ; Boucle répétée $7F fois
LDA  ]POS+2 ; Assemble l'instruction LDA page zéro avec
; l'adresse POS
STA  ]POS ; idem pour STA avec POS+2
]POS =]POS+2 ; POS est incrémenté de 2 avant la création de
; l'instruction suivante
_^ ; Signifie que l'on complète la boucle
```

Scrolling Horizontal, de droite à gauche. 4 points par balayage, 32 lignes de hauteur. Le principe en est le suivant :

- 1- Sauver les points de l'extrême gauche
- 2- Bouger l'image vers la gauche de 4 points
- 3- copier les points sauvés a l'extrême droite

Ce programme utilise HORIZCNT (\$C02E) pour se synchroniser. Le lancer avec une image quelconque déjà en mémoire.

```
ORG  $900
CLC
XCE
SEP  # $30
LDA  # $A1
```

	STA	\$C029	
	STZ	\$C035	; Shadowing actif
	BIT	\$C010	
	REP	#\$30	
	LDX	#AUX	; Va copier la routine AUX en banc 01
	LDY	#\$1800	
	LDA	#AUXEND-AUX	
	MVN	\$000000,\$010000	
	PHK		
	PLB		; Restore B, perturbé
			; par le MVN
	LDX	#\$7FFE	; Copie de l'image du
COPY	LDAL	\$E12000,X	; banc \$E1 en banc 01
	STAL	\$012000,X	; pour pouvoir
	DEX		; travailler en banc 01
	DEX		; avec le shadowing on
	BPL	COPY	
	SEP	#\$30	
SYNC	LDA	\$C02E	
	CMP	#\$D8	; Attend une position
	BNE	SYNC	; balayage
	REP	#\$30	; Mode 16 bits
	JSL	\$11800	; Appel de la routine AUX
	SEP	#\$30	; Mode 8 bits
	LDA	\$C000	; Teste le clavier
	BMI	QUIT	
	JMP	SYNC	
QUIT	LDA	#\$21	; Sortie, on affiche le
	STA	\$C029	; mode text
	RTS		; et on sort

```
MX 00 ; Obligatoire, car la suite doit être  
; assemblée en 16 bits
```

```
AUX LDA $C068 ; Page Zéro et Pile en banc 01
```

```
ORA #$30
```

```
STA $C068
```

```
PHD
```

```
PHB
```

```
PHK
```

```
PLB
```

```
LDA #$A000 ; Crée la séquence qui
```

```
TCD ; copie la bande de gauche
```

```
]FROM = $8900 ; dans le buffer ($A000)
```

```
]TO = $00
```

```
LUP $20 ; $20 (32) instructions générées
```

```
LDA ]FROM ; du type LDA $8900
```

```
STA ]TO ; STA $A000
```

```
]FROM = ]FROM + $A0
```

```
]TO = ]TO + $2
```

```
-^
```

```
LDX #$02 ; Scroll vers la gauche
```

```
LDA #$8900 ; en lui-même
```

```
; D commence à $8900 (un multiple de $100)
```

```
]POS = $00
```

```
LUP $7F
```

```
LDA ]POS+2
```

```
STA ]POS
```

```
-^
```

```
LDA ]POS,X ; Obligatoire au saut
```

```
STA ]POS ; de page:
```

```
; LDA $FE,X (=$100)
```

```
TDC ; STA $FE
```



```

        CLC
        ADC  #$100
        CMP  #$9D00
        BEQ  END

        BRL  LOOP

END      LCA  #$A000      ; Recopie du Buffer
        TCD      ; vers la droite de
]TO     = $899E      ; la bande du scroll
]FROM   =$00
        LUP  $20
        LDA  ]FROM
        STA  ]TO
]FROM   = ]FROM +$2
]TO     = ]TO +$A0
        -^

        LDA  $C068      ; Remet $C068 à sa
        AND  #$CF      ; valeur de départ
        STA  $C068      ; (PZ et Pile en banc 00)

        PLB      ; Récupère B et D
        PLD
AUXEND  RTL      ; Retour en banc 00

```

Cette technique est utile, mais très limitée. La zone couverte par des instructions page zéro n'a pas plus de \$100 bytes de long, les scrolls verticaux sont donc périlleux voir impossibles. On peut améliorer cette technique en utilisant, en plus, la pile. L'exemple suivant est un scroll vers le haut d'une fenêtre de 200 par 65 points. L'instruction au coeur de ce programme est PEI. Cette instruction, spécifique au 65C816, signifie que l'on empile le contenu d'une case mémoire en page zéro. L'instruction :

```
PEI  $30
```

est équivalente à la séquence :

```
LDA  $30
PHA
```

Mais elle ne prend que 6 cycles! L'astuce de ce scroll est de positionner la page zéro au début de la ligne de départ, le pointeur de pile en fin de ligne d'arrivée (à chaque empilement, le pointeur de pile est décrémenté), et d'exécuter une séquence de PEI. Les pseudo codes opératoires (LUP, —^...) de l'exemple précédent sont à nouveau présents.

```

                ORG   $900
                CLC
                XCE
                SEP   #$30
                LDA   #$A1           ; Affiche Super hires
                STA   $C029
                STZ   $C035           ; Shadowing actif
                BIT   $C010
                REP   #$30           ; Copie la routine scroll
                LDX   #AUX           ; en RAM auxiliaire
                LDY   #$1800
                LDA   AUXEND-AUX
                MVN   $000000,$010000
                PHK
                PLB
                LDX   #7FFE           ; Recopie l'image du banc E1 en banc 01
COPY           LDAL  $E12000,X
                STAL  $012000,X
                DEX
                DEX
                BPL   COPY
                CONT           SEP   #$30
                SYNC           LDA   $C02E           ; Attend une valeur du balayage
                CMP   #$92
                BNE   SYNC
                REP   #$30           ; Sauve la ligne la plus haute de la fenêtre

```

```

LDX #\$0062 ; dans une zone tampon (\$9000)
:1 LDAL \$01341E,X
STA \$9000,X
DEX
DEX
BPL :1

JSL \$11800 ; Bouge toutes les lignes d'un cran vers le haut

LDA #\$0000 ; Remet la page zéro au bon endroit
TCD

SEP #\$30

LDA \$C000 ; Si une touch est pressée on arrête tout
BMI END

REP #\$30
LDX #\$0062 ; Recopie la ligne sauvée tout en bas de la
:2 LDA \$9000,X ; fenêtre
STAL \$015C1E,X
DEX
DEX
BPL :2
BRA CONT

END LDA #\$21 ; Fin, on passe en mode TEXT
STA \$C029
RTS

; Routine Scroll en RAM auxiliaire

AUX TSX ; Sauve le pointeur de pile

LDA \$C068 ; Page zéro et pile en banc 01
ORA #\$0030
STA \$C068

```

```

        LDA  #$3481      ; Fin de la ligne destination
        TCS              ; Dans le pointeur de pile
        LDA  #$34BE      ; Début de la ligne source
LOOP    TCD              ; = Début de la page zéro

]POS    =$62

        PEI  ]POS        ; Crée les instructions PEI

]POS    =]POS-2          ; PEI $62
        _^              ; PEI $60
                          ; PEI $5E...
        TSC              ; Fin d'une ligne
        CLC
        ADC  #$A0        ; ligne source suivante
        CMP  #$5D00
        BCC  LOOP        ; Terminé?

        LDA  $C068       ; Remet la page zéro et la
        ADC  #$00CF      ; pile en banc 00
        STA  $C068
        TXS              ; Récupère le pointeur de pile
AUXEND  RTL              ; revient en banc 00

```

L'astuce du changement d'adresse de la pile peut également être utilisée pour changer des valeurs en banc 01 très rapidement. Les couleurs d'une palette peuvent être changées par une boucle du type:

```

        LDX  #$1F
COPY    LDA  TABLE,X
        STAL $E19E00,X
        DEX
        BPL  COPY

```

Cette boucle est lente, et si le temps est compté (en cas de changement de palette à chaque ligne), on lui préférera la routine suivante :

LDA \$C068

ORA #30

STA \$C068

LDA #9E20

TCS

PEA \$0FFF

PEA \$F0F0

...

Le pointeur de pile est placé au sommet de la palette 00 (\$9E00-9E1F), puis les valeurs des couleurs sont empilées, à l'aide de l'instruction PEA (Push Effective absolute Address, consulter le chapitre sur le microprocesseur pour plus d'informations). Auparavant, on aura pris soin de positionner la pile et la page zéro en banc 01, tout en se rappelant que cette routine doit se trouver dans un banc autre que le banc 00!

Ces techniques de scroll, malgré leur ingéniosité, sont limitées par la lenteur du processeur. Arriver à bouger une fenêtre entre trente et quarante lignes par balayage semble être le maximum envisageable pour un scroll horizontal. La fenêtre de l'exemple 2, de dimension 65 * 200 points est certainement la plus grande animable à chaque balayage.

4.3 SPRITES

Après les mouvements de portions d'écran, abordons les mouvements d'objets individuels sur l'écran. L'Apple IIGS ne possède pas, comme certaines machines, de «sprites» hardware. Il va donc falloir que nous les gérions nous-même. Des routines ROM existent pour cela, elles sont non spécifiques et un peu trop lentes pour nous permettre d'atteindre de bons résultats. Lorsque l'on bouge un objet en premier plan devant un décor, il faut non seulement sauver ce dernier pour le redessiner après coup, mais il faut également respecter les zones transparentes de l'objet.

Pour cela, nous utiliseront des techniques classiques d'animation. Il faut tout d'abord créer un «masque». Le masque est une sorte de négatif de l'objet à bouger. A chaque point de couleur 0 de l'objet, on fait correspondre un \$F dans le masque. A chaque point de l'objet de couleur autre que 0, correspond un 0 dans le masque.

Exemple :

Une ligne de points: 0A AA A0 00 12 21 00 30 04

Masque: F0 00 0F FF 00 00 FF 0F F0

Le rôle du masque est de mettre à 0 les points du décor correspondant à des points de couleur de l'objet. Lorsque l'on effectue le ET logique (AND) entre le décor et le masque, une silhouette noire représentant les points non transparents de l'objet apparait. Il suffit ensuite, de réaliser un OU (ORA) entre cette silhouette et les données de l'objet lui-même pour que ce dernier se dessine.

Toutes ces opérations pourraient être réalisées sur des zones de mémoire tampon, par des routines du type:

```
:1          LDA  $2000,X
           AND  $1000,X
           ORA  $1100,X
           STA  $2000,X
           DEX
           BNE  :1
```

Ces routines sont trop lentes, pour atteindre une vitesse optimale, il est impératif de:

- 1) Travailler en 16 bits.
- 2) Réaliser les opérations décrites précédemment en mode immédiat, c'est à dire : charger les points du décor, les sauver, effectuer le AND avec le masque, le ORA avec le motif et finalement stocker les points ainsi traités.
- 3) Programmer tout ceci avec une suite d'instructions, sans boucles

Le programme suivant affiche une petite croix en haut à gauche de l'écran. Cette croix peut être déplacée à l'aide des quatre flèches du clavier dans les quatre directions. Le déplacement effectué est de une case vers le haut et le bas et de deux cases (un byte) vers la gauche et la droite. Il est préférable qu'une image se trouve déjà en mémoire lors de son lancement. La croix est de couleur \$0F.

PROGRAMME «SPRITE»

```
          ORG  $900

NEWPOS    = $00          Contient l'adresse relative en mémoire du point
OLDPOS    = $02

          CLC
          XCE
          SEP  #$30
```

```

LDA  #A1          ; Super hires affichée
STA  C029
STZ  C035          ; Shadowing actif
LDA  #01          ; Le banc «données» est le banc 01
PHA                                ; donc STA $1000 est équivalent à STAL $011000
PLB

REP  #30
STZ  NEWPOS        ; Position du sprite de 00 à $7A7C
LDX  #7FFE
COPY  LDAL $E12000,X ; Copie la mémoire graphique du banc $E1
      STAL $012000,X ; dans le banc $01
      DEX
      DEX
      BPL  COPY

JSR  DRAW          ; Dessine la croix une première fois pour
                                ; sauver le décor
LOOP  SEP  #30
      JSR  WAIT        ; Attend le balayage
      REP  #30
      JSR  ERASE       ; Efface la croix
      JSR  DRAW        ; et la redessine à sa nouvelle position
      SEP  #30

KBD   LDA  C000          ; Lit le clavier
      BPL  KBD
      BIT  C010
      CMP  #95          ; Flèche pointée à droite
      BEQ  RIGHT
      CMP  #88          ; Flèche pointée à gauche
      BEQ  LEFT
      CMP  #8B          ; Flèche pointée vers le haut
      BEQ  UP
      CMP  #8A          ; Flèche pointée vers le bas
      BNE  KBD
DOWN  REP  #30

```

	LDA	NEWPOS	; Change la position un point vers le bas
	CLC		; Il faut donc ajouter \$A0 (160) octets à
	ADC	#\$A0	; l'adresse pour pointer vers la ligne suivante
	BRA	CONT	
UP	REP	#\$30	
	LDA	NEWPOS	; Change la position un point vers le haut
	SEC		; Il faut donc soustraire \$A0 (160) octets à
	SBC	#\$A0	; l'adresse pour pointer vers la ligne précédente
	BRA	CONT	
RIGHT	REP	#\$30	
	LDA	NEWPOS	; Change la position deux points vers la droite
	INC		; Il faut donc ajouter un octet à l'adresse pour
	BRA	CONT	; pointer vers ce nouveau point
LEFT	REP	#\$30	
	LDA	NEWPOS	; Change la position deux points vers la gauche
	DEC		; Il faut donc soustraire un octet à l'adresse
	BRA	CONT	; pour pointer vers ce nouveau point
WAIT	LDA	\$C019	
	BMI	WAIT	
CONT	BPL	OK1	; Vérifie que l'adresse ne soit pas comprise ; entre \$8000 et \$FFFF, parce que ça signifierait ; que la croix est hors de l'écran
	LDA	#\$0000	; Sinon remet la croix au bord supérieur gauche ; de l'écran
OK1	CMP	#\$7A7C	; Vérifie que l'adresse ne soit pas comprise
	BCC	OK2	; entre \$7A7D et \$7FFF, parce que ça signifierait ; que la croix est hors de l'écran
	LDA	#\$7A7C	; Sinon remet la croix au bord inférieur doit
OK2	STA	NEWPOS	
	BRA	LOOP	; On remet ça...
	MX	11	; Nécessaire, car la suite est en 8 bits
WAIT	LDA	\$C019	; Attend que le balayage de l'écran atteigne le
	BMI	WAIT	; bas de l'écran


```

W2          LDA  $C019
           BPL  W2
           RTS          ; et revient

           MX  0          ; La suite en 16 bits

DRAW       LDX  NEWPOS    ; X = position de la croix
           LDA  $2000,X
           STA  $A000
           LDA  #$FFFF
           STA  $2000,X
           LDA  $2002,X   ; Chargement du décor
           STA  $A002     ; Sauvegarde dans le buffer
           AND  #$0F00    ; ET avec le masque
           ORA  #$F0FF    ; OU avec les données
           STA  $2002,X   ; Stockage sur le décor
           LDA  $20A0,X   ; et on continue pour
           STA  $A004     ; chaque série de points
           AND  #$FF0F
           ORA  #$00F0
           STA  $20A0,X
           LDA  $20A2,X
           STA  $A006
           AND  #$0FFF
           ORA  #$F000
           STA  $20A2,X
           LDA  $2140,X
           STA  $A008
           AND  #$F00F
           ORA  #$0FF0
           STA  $2140,X
           LDA  $2142,X
           STA  $A00A
           AND  #$0FFF
           ORA  #$F000
           STA  $2142,X
           LDA  $21E0,X

```

```

STA $A00C
AND #$FF0F
ORA #$00F0
STA $21E0,X
LDA $21E2,X
STA $A00E
AND #$0FFF
ORA #$F000
STA $21E2,X
LDA $2280,X
STA $A010
LDA #$FFFF
STA $2280,X
LDA $2282,X
STA $A012
AND #$0F00
ORA #$F0FF
STA $2282,X
LDA NEWPOS
STA OLDPOS
RTS

```

```

ERASE LDX OLDPOS ; Ancienne position
LDA $A000 ; Recopie les données
STA $2000,X ; du Buffer (Tampon)
LDA $A002 ; sur le décor
STA $2002,X ; Opération inverse de la sauvegarde
LDA $A004
STA $20A0,X
LDA $A006
STA $20A2,X
LDA $A008
STA $2140,X
LDA $A00A
STA $2142,X
LDA $A00C
STA $21E0,X

```

```

LDA  $A00E
STA  $21E2,X
LDA  $A010
STA  $2280,X
LDA  $A012
STA  $2282,X
RTS

```

On ne peut évidemment pas s'amuser à entrer les données des masques et des points de l'objet à la main ! Le mieux est d'écrire un petit programme, qui passe en revue tous les nibbles (tous les points) du futur sprite en construisant, pour chaque groupe de quatre points les instructions LDA, STA sauvegarde, AND masque, ORA données, STA image. Aucun exemple n'est donné, car le programme varie de cas en cas (taille du sprite...). De tels générateurs de programmes représentent un bon compromis place occupée sur disque/ vitesse d'exécution. Il suffit de charger les données des objets et de lancer le créateur, ce dernier créant lui même le programme dessinateur, qui n'a pas besoin d'être sauvegardé ! Seul défaut, si l'on a plusieurs objets, la mémoire se remplit très vite. Pour gagner encore un peu de temps, on peut utiliser la page zéro pour charger et stocker les points du décor. Ce procédé est avantageux si le sprite est large, car à chaque changement de ligne, il faut rajouter SAO au registre D, ce qui mange des cycles...

Dans notre exemple, le sprite bouge de deux positions minimum dans les directions horizontales. Pour que ce mouvement devienne d'une position seulement, il faut avoir deux sprites et deux masques en mémoire, et sélectionner à chaque fois soit le sprite des positions paires, soit le sprite des positions impaires. Estimons-nous heureux, du temps de la résolution HGR de l'Apple II, il ne fallait pas moins de sept sprites pour animer un objet

Quand on gère plusieurs objets, il faut veiller à les effacer dans l'ordre inverse de celui dans lequel ils ont été dessinés. Sinon, des motifs bizarres risquent d'apparaître lorsque deux objets se chevauchent.

Le nombre maximum d'objets gérables avec ce type de routine est très limité. Il est d'environ douze pour des sprites de seize points par seize points redessinés à chaque balayage. Un moyen d'augmenter ce nombre est de réduire à la fois le champ de mouvement des sprites et leur fréquence de modification, tout en utilisant le shadowing pour les redessiner (voir le paragraphe traitant du shadowing en tant qu'outil d'animation).

4.3 SHADOWING dans l'animation

Jusqu'à maintenant, nous avons simplement utilisé le SHADOWING de la page SUPER HIRES pour pouvoir travailler directement en banc 01, mémoire rapide. Le shadowing peut également être utilisé comme outil d'animation, et ceci de deux manières différentes :

- Admettons que nous animons beaucoup d'objets dans une partie seulement de l'écran, et qu'il est impossible de les dessiner tous le temps d'un balayage. Si on les dessine l'un après l'autre directement sur la page affichée, l'animation ne sera pas nette. Une solution consiste à dessiner tous les objets dans la mémoire en banc 01 avec le shadowing inactif, puis d'activer ce dernier et de recopier entièrement ou partiellement la mémoire du banc 01 sur elle-même. Si la zone de recopie est restreinte, cette opération a des chances de se dérouler en moins d'un balayage, améliorant ainsi considérablement l'animation. Cette technique peut être utilisée pour faire apparaître des menus déroulants presque instantanément, ces derniers étant tout d'abord dessinés en banc 01 avec le shadowing inactif.
- Le deuxième cas où le shadowing peut nous être utile est celui où l'on travaille avec un décor de fond. On peut, dans ce cas, avoir les données du décor en bancs 01 et \$E1 et dessiner directement en banc \$E1. C'est un banc de mémoire lente, mais pour certaines routines, ce n'est pas très grave. Ce qui est intéressant, c'est que l'on peut recopier le décor directement en recopiant la banc 01 sur lui-même avec le shadowing actif. Cette technique est utilisable pour créer des génériques, des textes apparaissent sur une image, puis ils sont effacés et d'autres les remplacent, l'image restant intacte.

Voyons maintenant les techniques de recopie rapide de la mémoire graphique du banc 01 sur elle-même. On est tenté d'utiliser des couples LDA/STA page zéro, comme pour les scrolls. Cette manière d'agir est relativement rapide, mais dans ce cas précis, on peut faire encore plus rapide. En utilisant l'instruction TSB (Test and Set Bit). Cette instruction exécute l'équivalent d'un LDA et d'un STA, tout en modifiant, en plus, certains bits du byte de travail, d'après la valeur de ces bits dans l'accumulateur.

Exemple :

```
LDA  $C068
ORA  #$30
STA  $C068
```

Est équivalent à:

```
LDA  #$30
TSB  $C068
```

Tous les bits à 1 de l'accumulateur sont mis à 1 dans la case mémoire (voir le chapitre sur les instructions du microprocesseur pour plus de renseignements).

Nous allons utiliser la version page zéro de cette instruction, qui ne prend que 5 cycles (6 en mode 16 bits), ce qui est tout bonnement ridicule ! Le programme de recopie de toute la page SUPER HIRES sur elle-même serait donc :

PROGRAMME «RECOPIE RAPIDE»

```

                ORG  $1800          ; En banc 01
;
                CLC
                XCE
                SEP  #$30
                STZ  $C035          ; Shadowing actif
                LDA  #$A1          ; Page affichée
                STA  $C029
                LDA  #$30          ; Page zéro en banc 01
                TSB  $C068
                REP  #$30          ; 16 bits
                PHD                    ; Sauve D
;
                LDA  #$2000          ; Début de la mémoire
Loop           TCD
                LDA  #$0000          ; Nécessaire que A=0 pour ne pas perturber les
]POS          = $00                ; bits copiés
                LUP  $80            ; Génère 128 instructions
                TSB  ]POS           ; de type TSB page zéro
;
]POS          = ]POS+2
                _^                    ; Boucle
;
                TDC
                CLC
                ADC  #$100
                CMP  #$9D00          ; Fin de la mémoire à copier
                BEQ  END
                BRL  LOOP
;
END           LDA  #$30            ; Page zéro à nouveau en banc 00
                TRB  $C068

```

PLD	; Récupère D
RTL	; Revient

Cette routine doit être placée en banc 01, on imagine qu'elle est appelée par un JSL depuis le banc 00. Pour copier toute la page, il lui faut environ : 128 instructions * 125 blocs de 256 bytes * 6 cycles/ instructions = 96000 cycles (en fait un tout petit peu plus). C'est très peu, seulement deux balayage et demi, et encore moins si l'on recopie une partie de l'écran seulement.

Cette technique remplace sommairement la 2ème page graphique absente sur le IIGS.

Les deux programmes qui suivent sont des programmes prototypes, qui surpassent les possibilités «normales» du IIGS.

4.4 Programme «MULTICOULEUR»

Ce programme affiche 96 couleurs à l'écran, en utilisant qu'une seule palette !

Il est basé sur l'interruption SCANLINE (voir la description détaillée de ces événements dans le chapitre «SYNCHRONISATIONS»). Une interruption est générée à la ligne \$30, puis le programme change la couleur 00 de la palette 00 à chaque début de nouvelle ligne (les valeurs des couleurs sont stockées dans les tables COLTB1 et COLTB2). Pour ce faire, le programme possède une boucle d'attente dont le temps d'exécution, additionné avec le temps mis par la lecture des tables et le stockage des couleurs, correspond exactement au temps mis par le balayage pour parcourir une ligne. Avec ce principe, on peut imaginer plusieurs types d'effets : animations en modifiant le contenu des tables à chaque interruption, changement de plusieurs couleurs d'une palette (il reste du temps) ou alors carrément changement des 16 couleurs d'une palette à chaque ligne (réalisable en transférant la pile en banc 01 et en utilisant des PEAs, voir le chapitre «SCROLLS»). Le maximum de couleurs affichables simultanément à l'écran passe de 256 (16 palettes de 16 couleurs) à 3200(16 couleurs par ligne * 200 lignes d'écran Super Hires). Ce principe pourrait être utilisé pour afficher sur l'écran du IIGS des images provenant d'autres machines avec plus de 16 couleurs (images AMIGA 32 couleurs, VGA 256 couleurs etc..). Cependant, le IIGS est limité à 16 couleurs par ligne, maximum qui ne pourra vraisemblablement jamais être dépassé, car le processeur video du IIGS semble «lire», au début de chaque ligne, le contenu de la palette assignée. Ce qui signifie que si l'on tente de changer la valeur d'une couleur lorsque le balayage est au milieu d'une ligne, le changement ne sera pris en compte qu'à la ligne suivante, lorsque la palette sera «rechargée».

Cette technique marque une progression par rapport aux effets de couleurs «classiques» (programmables en langages évolués par l'intermédiaire de la boîte à outils, par exemple). Toutefois, elle possède ses inconvénients :

- Ce programme est gourmand en temps d'exécution, pour afficher ce dégradé de 96 couleurs, il monopolise, en 60 HERZ, environ $100/262 = 38$ pour cent du temps du processeur. Mais l'effet en vaut certainement la peine
- La routine centrale de temporisation est calculée par rapport au 65C816 à 2.8 MHZ. Pour tout changement de vitesse, même minime, les calculs sont à refaire. Des problèmes apparaîtront donc forcément avec l'évolution du IIGS, qui se fera, on l'espère, avec un processeur plus rapide. Pour l'instant, le programme s'assure déjà qu'une carte accélératrice «TRANSWARP (tm)» n'est pas présente et la décélère si il en trouve une (voir le chapitre «TRANSWARP»).

PROGRAMME «MULTICOULEUR»

```

                ORG    $900
                CLC
                XCE
                SEP    #$30
                LDA    #$A1
                STA    $C029
                REP    #$30
                LDAL   $BCFF00      ; Cherche si une carte accélératrice
                CMP    #'TW'       ; Transwarp GS est installée
                BNE    NOTR
                LDAL   $BCFF02
                CMP    #'GS'
                BNE    NOTR
                LDA    #$28         ; La carte est présente et on doit donc mettre la
                JSL   $BCFF24      ; vitesse à 2.8MHz, la vitesse normale du IIGS
NOTR            LDA    #$0000      ; Efface l'écran
                LDX    #$7FFE

```

```

CLR          STAL  $E12000,X
            DEX
            DEX
            BPL   CLR

            SEP   #$30
            LDX   #$C8
CLILOP      LDAL  $E19D00-1,X ; Met le bit d'interruption de
            AND   #$BF         ; tous les SCB à zéro pour nous assurer
            STAL  $E19D00-1,X ; qu'aucune interruption soit active
            DEX
            BNE  CLILOP

            SEI
            LDAL  $E19D30      ; Active une interruption à la ligne $30
            ORA   #$40
            STAL  $E19D30
            LDA   #$5C         ; Code de l'instruction JMPL
            STAL  $E10028
            LDA   #$00
            STAL  $E1002B
            LDA   #INTER      ; Adresse de la routine
            STAL  $E10029      ; de gestion
            LDA   #>INTER     ; d'interruption
            STAL  $E1002A
            LDA   $C023        ; Interruption engagée
            ORA   #$02
            STA   $C023
            CLI          ; Réactive les interruptions
            RTS          ; Revient au moniteur

            DS   \

INTER       LDA   $C032        ; Signifie que l'interruption
            AND   #$DF         ; est traitée
            STA   $C032
            LDY   #$00

```


	JSR	WAIT1	; Temporisation préliminaire
BARLOP	LDA	COLTB1,Y	
	STA	\$C051	
	NOP		
	NOP		; Temporisation !
	NOP		
	NOP		
	LDA	COLTB1,Y	
	STAL	\$E19E00	
	LDA	COLTB2,Y	
	STAL	\$E19E01	
	JSR	WAIT2	; Encore temporisation!
	INY		
	CPY	#\$61	; 97 couleurs seulement !
	BNE	BARLOP	
	CLC		; Fin de l'interruption
	RTL		; Retour à l'Interrupt Manager
WAIT1	LDX	#\$14	
W1	DEX		
	BNE	W1	
	NOP		
	NOP		
	RTS		
WAIT2	LDX	#\$14	
W2	DEX		
	BNE	W2	
	RTS		
	DS	\	
COLTB1	HEX	00,10,20,30,40,50,60,70	
	HEX	80,90,A0,B0,C0,D0,E0,F0	
	HEX	F0,F0,F0,F0,F0,F0,F0,F0	
	HEX	F0,F1,F2,F3,F4,F5,F6,F7	
	HEX	F8,F9,FA,FB,FC,FD,FE,FF	
	HEX	FF,EF,DF,CF,BF,AF,9F,8F	

HEX 7F,6F,5F,4F,3F,2F,1F,0F

HEX 0F,0F,0F,0F,0F,0F,0F,0F

HEX 0F,0F,0F,0F,0F,0F,0F,0F

HEX 0F,0E,0D,0C,0B,0A,09,08

HEX 07,06,05,04,03,02,01,00

HEX 0

COLTB2 HEX 0F,0F,0F,0F,0F,0F,0F,0F

HEX 0F,0F,0F,0F,0F,0F,0F,0F

HEX 0F,0E,0D,0C,0B,0A,09,08

HEX 07,06,05,04,03,02,01,00

HEX 00,00,00,00,00,00,00,00

HEX 00,00,00,00,00,00,00,00

HEX 00,00,00,00,00,00,00,00

HEX 00,00,00,00,00,00,00,00

HEX 00,01,02,03,04,05,06,07

HEX 08,09,0A,0B,0C,0D,0E,0F

HEX 0F,0F,0F,0F,0F,0F,0F,0F

HEX 0F,0F,0F,0F,0F,0F,0F,0F

HEX 00

4.5 Programme «ENTRELAÇAGE»

Le nombre de couleurs différentes que peut générer le processeur vidéo du IIGS est de 4096 (RGB 12 bits, 4 bits = 16 valeurs par composante, $16^3 = 4096$). Un moyen pour augmenter ce nombre est de créer des demi intensités. Pour ce faire, il suffit de changer l'intensité d'une couleur un balayage sur deux. Si par exemple on a un rouge d'intensité 2 que l'on fait passer à 3 au balayage suivant, pour la remettre à 2 au balayage d'après, la couleur effectivement perçue sera un rouge 2.5. En utilisant ce procédé, on peut générer jusqu'à 31 intensités différentes pour chaque composante, nous autorisant : $31^3 = 29791$ couleurs!

Ce qui multiplie par plus de 7 le nombre de couleurs de base. Cette technique pourrait s'appeler «ENTRELAÇAGE COULEUR» par analogie avec l'entrelaçage tout court. Un mode graphique entrelacé possède plus de lignes en hauteur qu'un mode traditionnel (en général 400 ou 512), chaque ligne n'est alors rafraichie qu'une fois tous les deux balayages, créant un scintillement généralisé. Le mode entrelacé couleur a simplement un certain nombre de couleurs qui ne retrouvent une même valeur que tous les deux

balayages, un léger scintillement en résulte, plus marqué en 50 HZ qu'en 60 HZ (fréquence de rafraichissement plus faible).

Le programme qui suit est un bon exemple d'entrelaçage couleur, il montre la différence entre un dégradé composé de 16 nuances de rouge et le même dégradé avec 31 nuances. Une fois lancé, un dégradé entrelacé de 31 rouges apparaît, en appuyant sur une touche, on passe du mode 31 au mode 16 et vice-versa. La différence saute aux yeux, les 31 nuances donnent un dégradé beaucoup plus lisse, légèrement scintillant.

Le principe d'exécution est le suivant :

Le programme commence par dessiner deux lignes avec chaque couleur de 1 à 16, ce qui donne les séquences suivantes (en bytes) :

```
11111 11111 11111 11111 ...
11111 11111 11111 11111 ...
22222 22222 22222 22222 ...
22222 22222 22222 22222 ...

... ..
FFFFFFFFFFFFFFFFFFFFFF...
FFFFFFFFFFFFFFFFFFFFFF...
```

En mode 16 intensités, chacune des couleurs possède l'intensité correspondant à son numéro, on a donc deux lignes pour chaque intensité. Lorsque l'entrelaçage est activé, une seconde palette est assignée à la deuxième ligne de chaque couleur, un balayage sur deux. Dans cette seconde palette toutes les couleurs sont plus élevées d'une intensité de rouge. La deuxième ligne de chacune des couleurs aura donc une intensité comprise entre son numéro et un numéro plus élevé. Ce programme est donné plus en tant que curiosité qu'en tant qu'utilitaire. Mais si l'on repense à notre programme 16 couleurs par ligne qui devait nous permettre d'afficher des images en provenance d'ordinateurs aux processeurs vidéos plus performants, on se rend compte que ce mode entrelacé pourrait nous rendre service. Non pas au niveau du nombre de couleurs simultanées à l'écran, mais plutôt au niveau du nombre de couleurs sélectionnables, le RGB 24 bits (8 bits, 256 niveaux par composante, 16 millions de couleurs affichables) étant une réalité sur des machines comme le Macintosh II (TM Apple Computer).

```
ORG    $900
```

```
STATUS    = $00
```

```
LACE      = $01
```

```
CLC
```

```

XCE
SEP   #$30
LDA   #$A1      ; Super hires affichée
STA   $C029
STZ   STATUS     ; Variables du programme
STZ   LACE

REP   #$30
LDX   #$7FFE    ; Efface la page graphique
LDA   #$0000
CLR   STAL      $E12000,X
DEX
DEX
BPL   CLR

SEP   #$20      ; A 8 bits, X,Y 16 bits
LDA   #$11      ; Première ligne, couleur 1
LDX   #$33E0    ; Adresse de la 1ère ligne
LOOP2 LDY   #$0140 ; 2 lignes = $140 bytes
LOOP1 STAL  $E12000,X
      INX
      DEY
      BNE  LOOP1
      CLC
      ADC  #$11      ; Couleur suivante, byte + $11
      CMP  #$10
      BNE  LOOP2

SEP   #$10      ; A,X et Y 8 bits
LDA   #$00      ; Création des deux palettes
LDX   #$01
LOOP3 STAL  $E19E00,X ; Palette 00
      STAL  $E19E22,X ; Palette 01
      INX
      INX
      INC
      CMP  #$10

```

	BNE	LOOP3	
W1	LDA	\$C019	; Attend la fin de
	BMI	W1	; l'écran
W2	LDA	\$C019	; (voir SYNCHROS)
	BPL	W2	
	LDA	\$C000	; Attend une touche au
	BPL	CONT	; clavier
	BIT	\$C010	
	LDA	LACE	; Si une touche est appuyée,
	EOR	#\$01	; change de mode:
	STA		; normal ou entrelacé
	LDA	\$C000	; Attend une touche au
	BPL	CONT	; clavier
	BIT	\$C010	
	LDA	LACE	; Si une touche est appuyée,
	EOR	#\$01	; change de mode:
	STA	LACE	; normal ou entrelacé
CONT	LDX	#\$21	; Remet la palette à 00
	LDA	#\$00	; sur toutes les lignes
PALO	STAL	\$E19D53,X	; du dessin
	DEX		
	BPL	PALO	
	LDA	LACE	; Si lace<>0 alors pas de
	BNE	W1	; mode entrelacé, retour au début
	LDA	STATUS	; Sinon, changement du
	EOR	#\$01	; status pour la prochaine
	STA	STATUS	; fois
	BNE	PAL01	; et peut être changement de palette une ligne
	BRA	W1	; sur deux
PAL01	LDX	#\$00	; Palette 01 assignée une
	LDA	#\$01	; ligne sur deux

```

LOOP4          STAL  $E19D53,X      ; Cette routine n'est
               INX                    ; exécutée qu'un balayage sur deux
               INX
               CPX   #$22
               BNE   LOOP4
               BRA   W1              ; Et ça continue!!!!

```

4.6. Format des images sur disk

Deux formats principaux existent :

- Le format PIC (fichier type \$C1). Ce format est le plus simple, l'image est directement stockée sous forme d'un bloc mémoire non compacté, contenant données, table SCB et palettes. On peut le charger en banc 00, \$1000 et l'afficher avec le programme «COPIE».
- Le format PNT (type \$C0) qui est en fait composé de plusieurs sous-format. Chaque sous-format (identifiable par un type auxiliaire, AUXTYPE, différent: 00,01,02.) a sa particularité d'organisation. Tous les fichiers de type \$C0 sont compactés, complètement ou partiellement, par l'outil N 26 de la TOOLBOX.

Si l'on veut utiliser une image de type \$C0, on peut soit se servir du décompresseur adéquat (disponible dans le domaine public), soit utiliser un programme d'affichage d'images ou un programme de dessin, puis sauver la zone mémoire de l'image (en utilisant le programme «COPIE» par exemple). Pour récupérer les images des programmes du commerce, qui utilisent souvent leur propre compresseur et des images sous forme de fichiers binaires, la technique décrite précédemment fonctionne parfaitement. Il suffit d'interrompre le programme (par un CTRL-RESET, ou à l'aide de l'option VISIT MONITEUR installable dans le CONTROL PANEL), puis de lancer le programme «COPIE» et de sauver l'image maintenant en banc 00.

En exemple, voila un programme de décompression d'images au format \$C0 type GS-PAINT, émulant l'outil numéro 26 . Il est donné assemblé en \$2000, et il attend l'image compactée en \$2100. Ces valeurs peuvent être modifiées très facilement.

Exemple d'un programme BASIC pour charger et décompacter une image :

```

10 PRINT CHR$(4);"BLOAD IMAGE,A$2100,TKO"
20 PRINT CHR$(4);"BRUN DECOMPACTEUR"

```

PROGRAMME «DECOMPACTEUR»

	ORG	\$2000	
	LDA	#\$A1	
	STA	\$C029	
	STZ	\$EB	
	STZ	SED	
	STZ	\$FA	
	LDA	#\$21	; Adresse de l'image
	STA	\$EC	; compactée (\$2100)
	LDA	#\$20	
	STA	\$EE	
	CLC		
	XCE		
	REP	#\$10	
	PHB		
	LDA	#\$00	
	XBA		
	LDA	#\$1F	
	LDX	\$EB	
	LDY	#\$9E00	
	MVN	\$000000,\$E10000	
	LDX	#\$0007	
CLR	STZ	\$9D00,X	
	DEX		
	BPL	CLR	
	PLB		
	LDA	#\$22	
	STA	\$EB	
	CLC		
	LDA	\$EC	
	ADC	#\$02	
	STA	\$EC	
LOOP	LDA	(\$EB)	
	CMP	#\$40	
	BCC	L5	
	CMP	#\$C0	

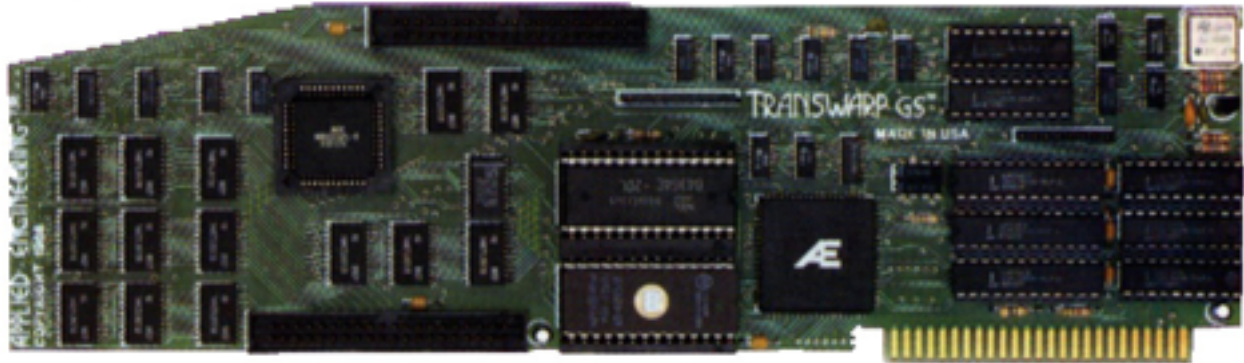
	BCS	L1
	CMP	#\$80
	BCS	L4
	AND	#\$0F
	INC	
	BRA	L3
L1	SEC	
	SBC	#\$3F
	ASL	
	ASL	
L3	STA	\$EF
	JSR	INCR
L2	LDA	(\$EB)
	LDY	#\$00E1
	PHY	
	PLB	
	STA	(\$ED)
	PLB	
	JSR	INCR2
	DEC	\$EF
	BNE	L2
	JSR	INCR
	BRA	LOOP
L4	SEC	
	SBC	#\$80
	STA	\$FA
	LDA	#\$03
L5	STA	\$EF
	JSR	INCR
L6	PHB	
	LDA	#\$00
	XBA	
	LDA	\$EF
	LDX	\$EB
	LDY	\$ED

	MVN	\$000000,\$E10000
	PLB	
L7	JSR	INCR2
	LDA	\$FA
	BNE	L8
	JSR	INCR
L8	DEC	\$EF
	BPL	L7
	LDA	#\$03
	STA	\$EF
	DEC	\$FA
	BPL	L6
	STZ	\$FA
	BRA	LOOP
INCR	INC	\$EB
	BNE	RET
	INC	\$EC
RET	RTS	
INCR2	INC	\$ED
	BNE	NOEND
	INC	\$EE
	LDA	\$EE
	CMP	#\$9D ; Fin si on atteint
	BNE	NOEND ; \$9D00
	PLX	
	SEC	
	XCE	
NOEND	RTS	

4.7 TRANSWARP GS

La carte TRANSWARP GS, commercialisée par la société américaine Applied Engineering, est une carte accélératrice pour l'Apple IIGS. Avec cette carte installée (comportant son propre microprocesseur ainsi que de la mémoire cache), la vitesse de la machine est poussée à 6 ou 7 MHz. Cette vitesse augmentée est fort appréciable pour

certaines applications (calculs en particulier), mais peut s'avérer néfaste pour certaines animations graphiques ayant des temporisations précises (le programme «MULTICOULEUR» en est un des meilleurs exemples).



Il est toujours possible d'inclure aux programmes en question un message préliminaire invitant l'utilisateur ralentir son GS avant exécution. Mais on peut également détecter la carte et la ralentir directement ! Voici un petit programme qui s'en charge :

```

SAVSPEED      = $00          ; Adresse de sauvegarde

                CLC
                XCE          ; Passe en mode 16 bits
                REP    #$30

                STZ    SAVSPEED    ; Met la sauvegarde à zéro

                LDAL  $BCFF00      ; Marque de la présence de la carte TRANSWARP:
                CMP   "TW"        ; la séquence "TWGS" = $54574753
                BNE   :1          ; à l'adresse $BCFF00
                CMP   "GS"
                BNE   :1

                JSL   $BCFF20      ; Si la carte est présente, on regarde la vitesse
                STA   SAVSPEED    ; et on la sauve
                LDA   #$28        ; On force la vitesse à 2.8 MHz, vitesse normale
                JSL   $BCFF24      ; du processeur d'un IIGS en mode FAST

:1              REP    #$30        ; Fin, passage en 16 bits
                LDA   SAVSPEED    ; Si SAVSPEED=00, pas de TRANSWARP et l'on sort
                BEQ   :2

```

	JSL	\$BCFF24	; Sinon, on revient à la vitesse sauvegardée
:2	RTS		; avant de sortir

Les deux routines de la carte TRANSWARP GS utilisées dans ce programme sont les suivantes:

Adresse	Description
\$BCFF20	GETSPEED, On récupère la vitesse actuelle dans le registre A (en mode 16 bits)
\$BCFF24	STORESPEED, On force la vitesse à la valeur contenue dans le registre A

Il existe encore d'autres appels possibles, entre autres pour connaître les vitesses disponibles... Mais ces 2 suffisent amplement dans la plupart des cas. Un dernier mot sur la TRANSWARP : cette carte possède de la mémoire cache, ce qui signifie que l'effet accélérateur sera meilleur lorsque le programme exécuté «bouclera» dans une zone mémoire plus petite que 32k. Il faut éviter, dans ce cas, d'avoir des morceaux de programmes dans tous les bancs et de les appeler les uns après les autres, sous peine de voir le facteur d'accélération chuter. De même, la TRANSWARP n'accélère pas les opérations de stockage en mémoire lente, ni le SHADOWING, elle n'aura donc pas des effets aussi spectaculaires sur les animations graphiques en SUPER HIRES que sur les programmes en mode texte, par exemple.

4.8 CONCLUSIONS

Nous avons fait le tour de toutes les possibilités graphiques de base de l'Apple IIGS, nous avons vu comment aller au delà de ces dernières à l'aide de techniques de programmations spéciales. Il est temps de faire un bilan, et ce comparer le IIGS à ses concurrents, d'un point de vue graphique.

Machine	Résolution	Nombre de couleurs	RGB
Apple IIGS	320 * 200 640 * 200	16 ¹ 4	
Atari ST	320 * 200 640 * 200 640 * 400	16 4 ² 2	
Commodore AMIGA	320 * 200 ³	32 16 4096 ⁴	

*1 Et non pas 256, car la majorité des programmes de dessin ne gèrent que 16 couleurs

*2 Le ST «classique» est considéré ici, et non pas le nouveau

*3 Le nombre de lignes en PAL peut être étendu à 256 et 512 respectivement

*4 avec beaucoup de restrictions (mode HAM)

Il ressort de cette comparaison les faits suivants :

- Le GS est la seule machine à ne pas posséder de mode 400 lignes (que ce soit sur un MULTISYNC en noir & blanc comme pour le ST ou alors sur un moniteur normal en mode entrelacé comme l'AMIGA)
- Le GS semble avoir un avantage sur le ST, ce dernier ne possédant qu'un mode RGB 9 bits (3 bits = 8 valeurs par composantes, 512 couleurs maximum)
- Les possibilités «normales» de ces trois machines en mode 320 sont relativement proches (en excluant le mode HAM de l'AMIGA, qui est une particularité)

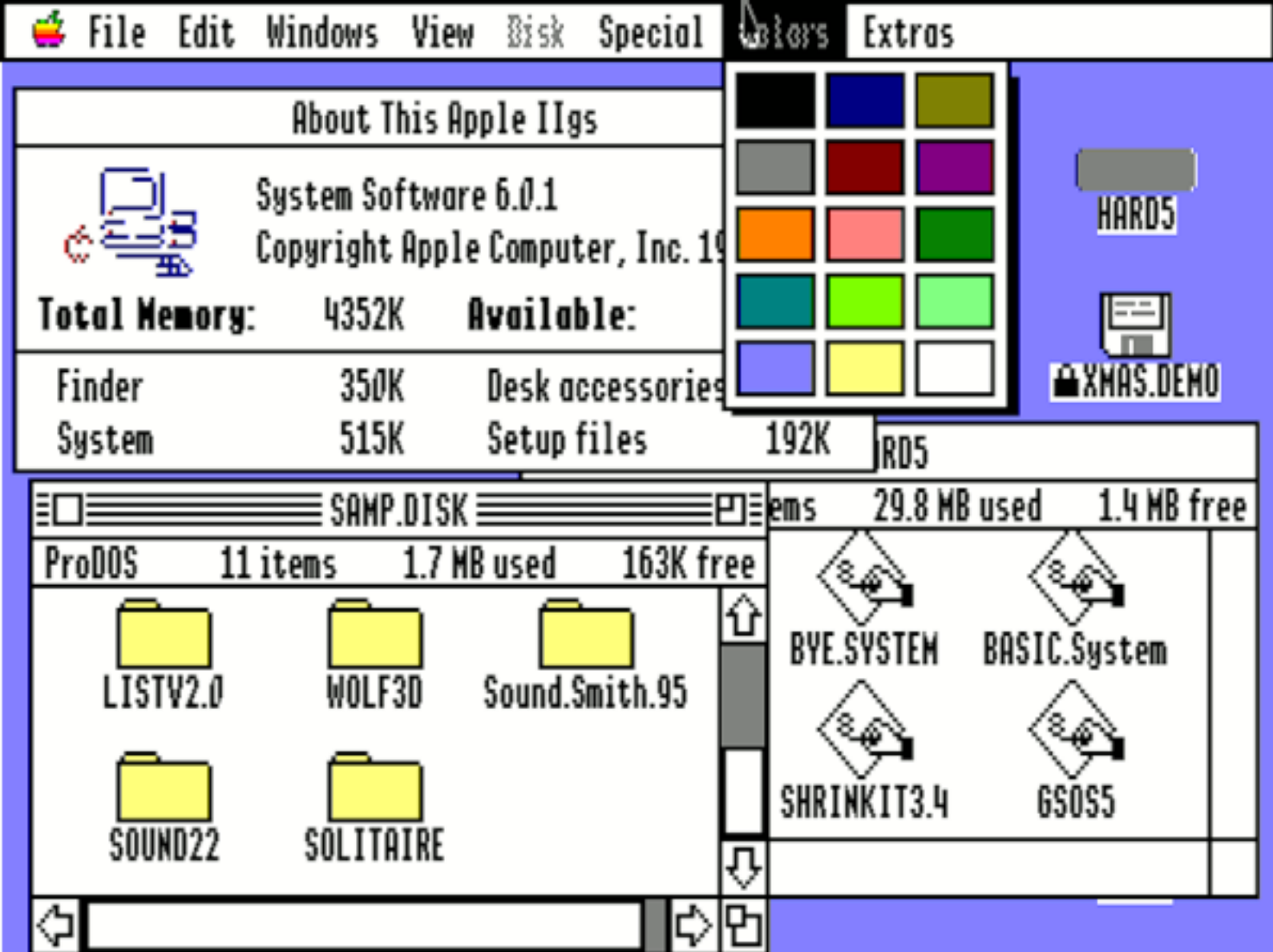
A la lumière de ces considérations, le GS ne semble pas en arrière par rapport à ses concurrents. Alors comment se fait-il que les animations sur le IIGS semblent si dérisoires par rapport aux animations sur AMIGA ou même sur ST?

La réponse est à chercher à la fois dans l'absence d'un co-processeur graphique valable (l'Amiga en possède deux) et dans la lenteur du 65C816. La fréquence de 2.8 MHz représente une augmentation de vitesse de 2.8 fois par rapport à l'Apple II alors que la taille d'une page graphique a quadruplé, passant de 8 à 32K. Tout le monde semble d'accord pour dire que la vitesse de l'Apple IIGS devrait être de 8 MHz

Les choses à regretter quant au processeur graphique sont les suivantes:

- L'impossibilité de changer la position en mémoire de la page graphique, et surtout son positionnement en mémoire lente!
- L'absence de mode 400 lignes
- L'impossibilité d'avoir des caractères de couleurs différentes en mode TEXT
- La limitation du nombre de couleurs de la bordure à seize
- L'absence d'un mode «PAL» 256 lignes
- Le fait que les valeurs des couleurs d'une palette ne soient lues qu'une fois, en début de ligne
- Sa trop grande simplicité.

Interprétons toutes ces limitations comme un encouragement à aller le plus loin possible avec ce qui nous est donné, quelles que soient les techniques employées.



GS/OS

5.0 Introduction à GS/OS & Prodos 8

Nous allons étudier dans cette partie deux systèmes d'exploitation fonctionnant sur Apple IIgs : GS/OS (GS Operating System), le système d'exploitation 16 bits de l'Apple IIgs, et

le ProDOS 8 (Professional Disk Operating System), le système d'exploitation 8 bits de l'Apple II et de l'Apple IIGS.

GS/OS est le premier système d'exploitation qui exploite l'Apple IIGS doté de ROM 01 ou de ROM supérieure. Il ne fonctionne sur aucun autre modèle de la famille Apple que l'Apple IIGS. GS/OS exploite les caractéristiques les plus avancées du microprocesseur 16 bits 65C816 du IIGS. C'est le système d'exploitation qui a remplacé ProDOS 16, un système d'exploitation d'attente qu'Apple a fourni à l'introduction sur le marché de l'Apple IIGS de Septembre 1986 à Septembre 1988. Pour des raisons de compatibilité, GS/OS reconnaît toutes les commandes de ProDOS 16, afin que les applications écrites pour fonctionner sous ProDOS 16 continuent de fonctionner correctement sous GS/OS.

ProDOS 8 fonctionne sur les ordinateurs Apple II+, IIe, et IIc. Il fonctionne aussi sur Apple IIGS quand celui-ci est en mode émulation; vous pouvez passer du GS/OS au ProDOS 8 si GS/OS est le système d'exploitation utilisé pour booter le IIGS. ProDOS 8 est un système d'exploitation écrit en 8 bits fonctionnant avec les micro-processeurs 6502 et 65C02. La plupart des commandes ProDOS 8 ont un équivalent sous GS/OS, mais la méthode pour mettre en oeuvre la commande est différente, du moins pour les programmes en langage machine.

GS/OS et ProDOS 8, comme tous les autres systèmes d'exploitation gèrent les données allant et provenant d'un support magnétique comme par exemple une disquette 5.25", une disquette 3.5" ou un disque dur. Ils font ce travail en traduisant des commandes disque de haut niveau utilisées par un programme d'application en instructions de bas niveau nécessaires pour communiquer directement avec le contrôleur de disque. Le système d'exploitation utilise aussi des groupes de données structurées appelés fichiers.

GS/OS et ProDOS 8 fonctionnent parfaitement avec tous les périphériques disque qu'Apple a vendu pour la famille Apple II: le Drive 5.25" (et ses prédécesseurs), le disque dur HD20SC, l'UniDisk 3.5", le lecteur Apple 3.5", la carte d'extension mémoire Apple II (utilisée en RAMDisque), et le lecteur Apple CD SC (CD-ROM).

5.0.1 Les systèmes d'exploitation sur Apple II - Une longue histoire

Quand Apple s'est lancé dans l'aventure en 1977, un magnétophone était le seul périphérique disponible pour stocker à terme "les données en masse" des utilisateurs. La raison en est simple : le premier Apple II disposait d'un port cassette (une simple entrée audio), ce qui permettait la connexion d'un magnétophone à cassettes.

Travailler avec des cassettes pour stocker ses données n'était pas de tout repos. Les temps de chargement étaient très lents, et l'utilisateur n'était jamais certain de retrouver ses précieuses données. Pour rendre l'Apple II de ces temps héroïques aussi pratique

qu'un ordinateur compatible d'aujourd'hui, on ne pouvait pas donner de nom aux fichiers et pour le chargement d'un fichier en particulier, il fallait au préalable positionner méticuleusement la cassette à l'endroit stratégique du début de l'enregistrement du dit fichier.

Steve Wozniak, l'inventeur de l'Apple II fut bien évidemment le premier à qui cette situation rendait extrêmement nerveux. Au cours de l'hiver 1977-1978, il a conçu une carte d'extension capable d'interfacer un Apple II et un lecteur de disquettes qu'on a appelé plus tard le Disk II. Au même moment, Bob Shepardson, et plus tard Randy Wigginton, Dick Huston, et Rick Auricchio, étaient en train d'écrire un système d'exploitation qui permettrait aux programmeurs de créer, organiser, et accéder à des fichiers sur les disques souples 5.25" que ce lecteur (le Disk II) allait utiliser.

Apple a en fait livré le Disk II, sa carte contrôleur, et la première version du système d'exploitation (DOS 3.1) au tout début de l'été 1978. Ce lecteur, le Disk II a été bien plus tard renommé Unidisk, puis lecteur Apple 5.25". Ce fut probablement l'évènement le plus important de la courte histoire d'Apple, car cela signifiait que pour la première fois des logiciels professionnels pouvaient être écrits pour un ordinateur de type Apple II. De tels logiciels nécessitent de créer et de manipuler des fichiers de données importants rapidement et facilement, une prouesse qu'il était impossible de réaliser avec une cassette et un magnétophone.

Plusieurs modifications ont été apportées au DOS 3.1 dans les mois qui ont suivi sa sortie pour corriger les inévitables bugs qui frétilaient dans tout le système d'exploitation. Le DOS s'est finalement stabilisé à la version 3.2.1 au milieu de l'année 1979. Cette première version du DOS formattait les disquettes en 35 pistes (l'utilisation d'une 36ème piste est une particularité du Disk II qui n'a séduit que quelques partisans avides de succès ...), chacune contenant 13 secteurs de 256 octets (soit donc un total de 133.75 Ko, où 1 Ko = 1024 octets). En fait, le programme situé sur la ROM de cette carte contrôleur ne pouvait booter que les disquettes utilisant ce format spécifique en 13 secteurs.

C'est également en 1979 qu'Apple a sorti son système d'exploitation Pascal. Ce système gère les fichiers d'une manière tout à fait différente des systèmes DOS 3.x ou plus tard ProDOS. Pour transférer des fichiers texte Pascal sur une disquette DOS (ou inversement), vous pouvez utiliser des programmes utilitaires disponibles dans le commerce ou auprès des clubs d'utilisateurs.

Apple a mis à jour le DOS 3.2.1 en 1980 afin qu'il puisse gérer le nouveau système de formatage en 16 secteurs (ce n'est pas Apple qui a inventé le 18 secteurs...) utilisé par son système d'exploitation Pascal. Le résultat fut le DOS 3.3, une version du DOS encore en cours quand Apple a sorti ProDOS 8 au tout début de 1984. Le formatage étant différent, cela a nécessité un changement de ROM sur la carte contrôleur. Le principal avantage de

ce changement de formatage fut de pouvoir utiliser 26.25 Ko supplémentaire sur le disque 5.25" (pour un total de 140 Ko: ceux des lecteurs qui n'ont pas saisis peuvent immédiatement retourner à la lecture des manuels Apple ou s'adresser à E. S. chez Apple France). L'inconvénient de taille était que le DOS 3.3 ne pouvait pas directement lire les fichiers situés sur une disquette formatée en DOS 3.2.1; de plus, il n'était pas possible de booter les disques formatés en DOS 3.2.1 sur une carte contrôleur équipée de cette nouvelle ROM. Heureusement, Apple a créé un programme appelé MUFFIN pour transférer des fichiers de l'ancien format (13 secteurs) au nouveau format (16 secteurs). Il est intéressant de constater que c'est à partir de MUFFIN que les pirates de l'époque ont puisé nombre de connaissances pour déplomber les disquettes utilisant une protection altérant le formatage. Apple a créé un deuxième programme BOOT13 afin que l'utilisateur puisse booter des disques formatés en DOS 3.2.1 avec le nouveau contrôleur de disques.

Apple a sorti ProDOS 8 que l'on appelle maintenant ProDOS en janvier 1984. Il fonctionne sur tous les Apple IIe, Apple IIc, ou Apple IIgs ou sur un Apple II Plus muni d'une carte d'extension mémoire de 16 Ko en slot 0. Il fonctionne aussi sur l'Apple II de l'âge de pierre muni d'une carte d'extension RAM de 16 Ko, seulement si le Basic Applesoft, et non pas le BASIC Integer est présent en ROM. A la sortie de ProDOS 8, Apple a précisé qu'il ne sortirait plus de logiciels utilisant le DOS 3.3; Apple a recommandé aux développeurs indépendants d'en faire de même. 6 ans après, certains éditeurs de logiciels et constructeurs de disques durs ne connaissent toujours pas ProDOS...

Une carte contrôleur compatible ProDOS 8 pour le disque dur ProFile qu'Apple avait réalisé deux ans auparavant pour être utilisé sur Apple III a été mise sur le marché en Janvier 1984. ProDOS 8 est capable de gérer des volumes d'une taille allant jusqu'à 32 Méga-octets. Ce n'est que plus tard qu'Apple a remplacé le ProFile par le disque dur HD20SC d'une capacité de 20 Méga-octets, un périphérique SCSI. On le connecte à l'ordinateur au moyen d'une carte d'interface SCSI.

En Septembre 1985, le lecteur UniDisk 3.5 a fait sa première apparition. Il utilise des disquettes au format 3.5" pouces enveloppées d'une protection en plastique; leur capacité de stockage est de 800 Ko. ProDOS 8 reconnaît automatiquement la carte contrôleur de ce type de lecteur de disquettes, aucun Driver n'est à installer. C'est à cette époque qu'Apple a commencé à fabriquer les Apple IIc avec un contrôleur de lecteur 3.5" intégré. Apple a aussi développé à la même époque une carte d'extension mémoire que ProDOS 8 reconnaît comme un RAM Disque au moment du boot.

Apple a annoncé l'Apple IIgs en Septembre 1986. À ce moment, Apple a renommé ProDOS par ProDOS 8, et a réalisé ProDOS 16 un système d'exploitation spécifique à l'Apple IIgs. Bien que ProDOS 16 formate les disques et stocke les fichiers de la même manière que ProDOS 8 (ce qui signifie que les deux systèmes d'exploitation peuvent co-exister sur le même disque), les deux systèmes sont tout à fait incompatibles au niveau de

la programmation. Apple a réalisé le ProDOS 16 pour exploiter pleinement la capacité d'adressage du 65C816; ProDOS 8 travaille seulement avec une capacité mémoire de 64 Ko. En même temps que l'Apple IIgs, le lecteur Apple 3.5", un autre lecteur utilisant des disquettes 3.5" a fait son apparition. La différence entre l'Apple 3.5" et l'UniDisk 3.5" est que le premier n'a pas de processeur intelligent intégré, et que par conséquent il ne fonctionne que sur l'Apple IIgs.

Une autre version de l'Apple IIc est sortie en Septembre 1986. Il comprend un connecteur sur lequel on peut enficher une carte d'extension mémoire. ProDOS 8 reconnaît cette carte en RAMDisque.

En Septembre 1988, Apple a lancé GS/OS, un nouveau système d'exploitation pour le IIgs destiné à remplacer ProDOS 16. GS/OS est bien entendu compatible avec le ProDOS 16. Il comprend des nouvelles commandes qui sont plus puissantes que celles de ProDOS 16. Une caractéristique très importante est que GS/OS peut accéder à des disques formatés sous ProDOS et à des disques formatés avec un autre système d'exploitation comme par exemple le High Sierra (pour les CD Roms), le HFS (utilisé par le Macintosh), ou le MS-DOS. Pour avoir accès à un autre système d'exploitation, il suffit de placer un module FST (File System Translator) sur le disque système GS/OS. Pour le moment, Apple n'a fourni que les modules FST pour les systèmes ProDOS et High Sierra.

Un autre modèle d'Apple IIc, l'Apple IIc Plus a également vu le jour en Septembre 1988. Il incorpore un lecteur de disquettes 3.5" qui fonctionne avec ProDOS 8.

Les premières versions de ProDOS avaient des bugs mineurs mais embêtants qui furent corrigés dans les versions ultérieures. Au moment de la rédaction de ces lignes, la version ayant cours est la 1.7. GS/OS, système d'exploitation beaucoup plus complexe, n'est pas encore très stable au contraire de ProDOS 8. Apple va en sortir de nouvelles versions environ deux fois par an...

5.0.2 Comparaison du ProDos 8 et du DOS 3.3

Le DOS 3.3 est constitué de deux modules principaux : le driver I/O (Input/Output) qui communique directement avec le contrôleur de disque 5.25", et l'interpréteur de commandes Applesoft qui analyse et exécute les commandes disque du BASIC Applesoft que le DOS 3.3 permet (OPEN, READ, CATALOG, etc ...). Les modules équivalents dans ProDos 8 sont séparés en deux fichiers programme appelés PRODOS (le driver I/O) et BASIC.SYSTEM (l'interpréteur de commandes Applesoft). Dans la plupart des applications, ProDOS 8 charge directement BASIC.SYSTEM au moment où le disque boote. Le tableau suivant décrit brièvement les commandes disque permises sous BASIC.SYSTEM et DOS 3.3. La plupart de ces commandes sont disponibles sous les

deux systèmes, certaines étant spécifiques à l'un ou l'autre des systèmes d'exploitation. En général, les commandes BASIC.SYSTEM sont plus puissantes que leur équivalent en DOS 3.3 car elles acceptent un nombre important de paramètres. Faites attention, car quelques commandes ne jouent pas le même rôle d'un système d'exploitation à l'autre.

5.0.3 Comparaison des commandes disque Applesoft.

Commandes	Description	ProDOS 8	DOS 3.3
APPEND	Ouvrir un fichier pour y ajouter des données	OUI	OUI
BLOAD	Charger un fichier	OUI	OUI
BRUN	Charger et exécuter un programme binaire	OUI	OUI
BSAVE	Sauvegarder un fichier	OUI	OUI
CATALOG	Lister les fichiers du volume (forme longue)	OUI	OUI
CLOSE	Fermer un fichier	OUI	OUI
DELETE	Effacer un fichier	OUI	OUI
EXEC	Exécuter les commandes contenues dans un fichier Texte	OUI	OUI
IN#	Rediriger l'entrée de caractères	OUI	OUI
LOAD	Charger un programme Applesoft	OUI	OUI
LOCK	Verrouiller un fichier	OUI	OUI
NOMON	Permis, mais ignoré sous ProDOS 8	OUI	OUI
OPEN	Ouvrir un fichier	OUI	OUI
POSITION	Bouger le pointeur à une adresse dans le fichier	OUI	OUI
PR#	Rediriger la sortie de caractères	OUI	OUI
READ	Lecture à partir d'un fichier	OUI	OUI
RENAME	Renommer un fichier	OUI	OUI
RUN	Charger et exécuter un programme Applesoft	OUI	OUI
SAVE	Sauvegarder un programme Applesoft	OUI	OUI
UNLOCK	Déverrouiller un fichier	OUI	OUI
VERIFY	Vérifier l'existence d'un fichier	OUI	OUI
WRITE	Écrire un fichier	OUI	OUI
"	Exécuter un fichier Applesoft, BIN, TXT, SYS	OUI	NON
BYE	Transférer le contrôle à un autre programme SYS	OUI	NON
CAT	Lister les fichiers d'un volume (forme courte)	OUI	NON

Commandes	Description	ProDOS 8	DOS 3.3
CHAIN	Transférer le contrôle à un autre programme AppleSoft	OUI	NON
CREATE	Créer un fichier	OUI	NON
FLUSH	Écrire les contenus des buffets fichier	OUI	NON
FRE	Faire le ménage des chaines en mémoire	OUI	NON
PREFIX	Donner le nom du catalogue actif	OUI	NON
RESTORE	Récupérer les variables d'un programme BASIC	OUI	NON
STORE	Sauvegarder les variables d'un programme BASIC	OUI	NON
FP	Initialiser le mode Applesoft	NON	OUI
INIT	Formater une disquette	NON	OUI
INT	Initialiser le mode Integer BASIC	NON	OUI
MAXFILES	Créer de l'espace pour les buffets fichier	NON	OUI
MON	Permet l'affichage des opérations du DOS	NON	OUI

Il n'est pas surprenant de constater, que l'environnement ProDOS + BASIC.SYSTEM est plus gourmand en mémoire que le DOS 3.3. Le système ProDOS + BASIC.SYSTEM utilise presque deux fois plus de mémoire que le DOS 3.3. Heureusement, la plupart du ProDOS 8 réside dans une Bank RAM de 16 Ko commutable et n'est donc pas en conflit avec l'espace mémoire utilisé par l'interpréteur Applesoft. Cet espace mémoire commutable est intégré sur les Apple IIe, IIc et IIGS et peut être ajouté sur les Apple II ou Apple II Plus en installant une carte d'extension mémoire de 16 Ko dans le slot 0 de l'ordinateur: il s'agit de la carte langage, créée au moment du développement du système d'exploitation Pascal sur Apple II qui nécessitait de la mémoire supplémentaire. Il y a deux effets secondaires dus à l'utilisation de la carte langage par le ProDOS. Premièrement ProDOS ne peut pas co-exister avec un programme qui utilise la carte langage par lui même, ou pour y ranger des données. Deuxièmement, le Basic Integer ne peut plus être utilisé, car il se loge en carte langage: paix à ses cendres.

Une autre grande différence entre DOS 3.3 et BASIC.SYSTEM est dans la manière de prendre en charge les buffers pour les fichiers. Un buffer fichier est une zone de mémoire qu'un fichier va utiliser; ce buffer contient les données contenues dans la partie active de ce fichier ainsi que l'information nécessaire à la localisation de ce fichier sur disque.

Quand le DOS 3.3 se lance, il initialise automatiquement trois de ces buffers; un nombre différent de buffers peut être réservé (de 1 à 16) en utilisant la commande MAXFILES. Les buffers fichier du DOS 3.3 ont chacun une longueur de 595 octets et sont situés entre le haut du programme Applesoft (cette adresse est stockée en \$73 /\$74 et est appelée HIMEM) et le début du DOS 3.3 en \$9D00.

ProDOS 8, de son côté n'initialise pas de buffer fichier; il assigne ou dé-assigne dynamiquement des buffers fichiers au moment où les fichiers sont ouverts et fermés. Quand un fichier est ouvert, ProDOS 8 diminue HIMEM de 1024 octets, et attribue ce buffer de 1024 octets à partir de la zone mémoire commençant en HIMEM + 1024. Quand un fichier est fermé, les buffers fichiers sont repositionnés, et HIMEM est augmenté de 1024 octets. Un total de huit fichiers peuvent être ouverts simultanément. Puisque ProDOS utilise cette méthode d'allocation dynamique pour les buffers fichiers, il n'est pas possible d'utiliser la technique qui en DOS 3.3 consiste à réserver un espace mémoire pour y loger un programme machine en faisant diminuer HIMEM et en stockant le programme machine dans la zone mémoire ainsi libérée.

5.0.4 Caractéristiques importantes de ProDOS 8 et de BASIC.SYSTEM

Un environnement PRODOS + BASIC.SYSTEM intègre de nombreuses caractéristiques utiles qui améliorent la vitesse d'exécution d'un programme et qui permettent de prendre en compte facilement des périphériques non Apple dans le système. Voici quelques une de ces importantes caractéristiques.

Interface Langage Machine (MLI)

Il est probable que la caractéristique la plus importante du ProDOS 8 soit l'interpréteur de commandes disque appelé MLI (Machine Language Interface), qui permet un accès facile aux fichiers en utilisant les techniques de programmation du langage assembleur. Le DOS 3.3 ne dispose pas d'une telle interface et il est assez malcommode de programmer en langage assembleur dans son environnement. Avec les commandes MLI, on peut faire de la manipulation de fichiers comme ouvrir, lire, écrire, et fermer un fichier. Les paramètres correspondant à chacune des commandes ont clairement été définis par Apple. Nous examinerons plus tard les diverses commandes du MLI.

5.0.5 Datage des fichiers

A chaque fois que ProDOS 8 crée ou écrit un fichier, il lit la date et l'heure courantes à partir de l'horloge installée dans le système (à supposer qu'il y en ait une), et range ces informations dans l'entrée du catalogue de ce fichier. Quand le disque est catalogué, la date et l'heure de création et la dernière modification apparaissent après le nom de ce fichier. ProDOS 8 travaille avec l'horloge intégrée de l'Apple IIgs et avec toutes les cartes horloge qui émulent le jeu de commandes de la Thunderclock de Thunderware.

5.0.6 Carte contrôleur de disque et protocoles pour les drivers de périphériques

Une des grandes contraintes du DOS 3.3 est qu'il est très difficile de lui intégrer et de lui faire reconnaître des périphériques disque autres que ceux développés par Apple (Lecteurs de disquettes haute densité, Disques durs compatibles, etc ...). Il n'en est pas de même avec ProDOS 8. Apple a publié un protocole de communication reconnu par le ProDOS 8 qui permet de faire reconnaître au système les périphériques disque autres que ceux qu'Apple commercialise. Ce protocole définit les adresses de la ROM de la carte contrôleur faisant référence la taille de ce volume, les caractéristiques de ce volume, et l'adresse de la sous routine du driver responsable des opérations I/O de ce périphérique. Apple a également défini la technique pour passer ces paramètres à une sous routine driver disque ProDOS 8 et comment le driver retourne des codes d'erreurs au programme appelant.

5.0.7 Une gestion améliorée des Interruptions

ProDOS 8 installe automatiquement sa propre routine de gestion des interruptions qui prend le contrôle du système à chaque fois qu'un périphérique I/O génère un signal IRQ (Interrupt Request), Cette sous routine peut alors appeler d'autres sous routines permettant de gérer ces interruptions. Cela signifie qu'il est assez facile d'intégrer une sous routine d'interruption même si une autre est déjà active.

5.0.8 Structure hiérarchisée du catalogue

En utilisant ProDOS 8, il est possible de créer plusieurs catalogues, chacun d'entre eux pouvant contenir plusieurs fichiers, le tout sur le même disque. Cela permet de ranger commodément dans un même catalogue un groupe de fichiers ayant des points communs pour pouvoir y avoir accès plus facilement. Les catalogues sont organisés de manière à ce que chacun d'entre eux soit contenu dans un autre (le catalogue père); le chemin d'accès des catalogues revient finalement au catalogue principal, la racine, que l'on appelle aussi le catalogue Volume. Le catalogue principal est celui que l'on crée et que l'on nomme quand le disque est formaté. Nous analyserons plus tard et en détail la structure hiérarchisée des catalogues.

5.0.9 Périphérique disque /RAM

L'Apple IIGS, l'Apple IIc, et l'Apple IIe (doté d'une carte 80 colonnes étendue) possède une mémoire auxiliaire de 64 Ko en supplément des 64 Ko de mémoire principale utilisée

normalement pour le rangement des programmes. ProDOS 8 utilise cet espace mémoire pour le stockage de fichiers de la même façon qu'il le fait sur un lecteur de disquettes ou un disque dur. Ceci s'appelle un RAM Disque. Les principales différences entre l'utilisation d'un RAM Disque et un lecteur de disques conventionnel sont que les opérations I/O se font beaucoup plus rapidement (il n'y a aucune partie mécanique à déplacer) et que le contenu du RAM Disque disparaît dès que l'alimentation électrique de l'ordinateur est coupée. Nous le verrons plus loin, mais il faut savoir que chaque volume disque du système a un nom, il s'agit du nom de volume. Le nom de volume pour le RAM Disque est /RAM.

5.0.10 Prolongement du BASIC.SYSTEM

Le BASIC.SYSTEM offre une méthode simple que vous pouvez utiliser pour lui ajouter des commandes additionnelles.

5.0.11 Séparation des pouvoirs

Contrairement au DOS 3.3, l'interpréteur de commandes bas niveau de ProDOS 8, celui qui prend en charge toutes les opérations disque I/O fondamentales n'est pas intégré au sein de l'interpréteur BASIC.SYSTEM qui permet de communiquer avec un jeu de commandes disque "en anglais" lorsque l'on programme en BASIC Applesoft. Cela signifie que si vous désirez écrire un autre interpréteur de commandes, ou un programme 100% en langage assembleur, vous pouvez économiser environ 12 Ko de mémoire en le chargeant à la place de BASIC.SYSTEM.

5.0.12.1 La commande RUN intelligente

La commande - du BASIC.SYSTEM est une commande très pratique pour les gens qui n'aiment pas taper au clavier. Elle exécute soit un programme Applesoft (comme le fait un RUN), soit un fichier binaire (BRUN), soit un fichier texte (EXEC) en déterminant automatiquement quel type de fichier a été spécifié puis en exécutant le travail nécessaire au lancement de ce fichier. Cette commande peut également exécuter des fichiers SYStème comme par exemple BASIC.SYSTEM. En gros, un programme SYStème est un programme indépendant écrit en langage assembleur, qui définit un environnement de programmation, ou un programme réalisant des fonctions spécifiques sans tenir compte de la présence d'un autre programme SYStème.

5.0.13 Des paramètres très utiles

Beaucoup des commandes du BASIC.SYSTEM supportent des paramètres très utiles permettant un contrôle plus grand sur la manière dont elles sont exécutées. Par exemple, vous pouvez utiliser le suffixe # (où # représente un numéro de ligne) avec la commande RUN du BASIC.SYSTEM pour charger un programme puis le lancer à partir de ce numéro de ligne. Vous pouvez utiliser aussi le suffixe E# (où # représente une adresse mémoire) pour spécifier une adresse de fin en utilisant une commande agissant sur un fichier BINaire (BLOAD et BSAVE). Vous pouvez aussi utiliser le suffixe type avec BLOAD ou BSAVE pour travailler avec tous les types de fichiers autres que les fichiers BINaires. Le type de fichier est les trois caractères qui suivent le nom d'un fichier: BAS pour BASIC, BIN pour BINaire, TXT pour TeXT, etc ... Un autre paramètre très utile est F#; lors de la lecture d'un fichier texte son utilisation permet de sauter un nombre spécifié d'enregistrements (un enregistrement est un groupe de caractères suivi par un <RETURN>).

5.0.14 Vitesse

ProDOS 8 réalise les opérations disque I/O sur une disquette 5.25" à un débit d'environ 8 Ko par seconde. Cela est beaucoup plus rapide qu'en DOS 3.3 qui a un débit d'environ 1 Ko par seconde. De plus, BASIC.SYSTEM comprend une nouvelle version de la commande FRE qui permet de faire le ménage parmi les variables chaînes du BASIC Applesoft beaucoup plus rapidement que son équivalent en DOS 3.3 (il faut plusieurs minutes au DOS 3.3 pour faire ce travail alors qu'il ne faut que quelques secondes à ProDos 8 pour réaliser le même travail).

5.0.15 Taille des fichiers et taille des volumes

ProDOS 8 peut gérer des fichiers ayant une taille allant jusqu'à 16 Méga-octets, sur un volume pouvant avoir une taille jusqu'à 32 Méga-octets. Les volumes DOS 3.3 ne peuvent pas excéder 400 Ko

5.0.16 Comparaison de GS/OS et de ProDOS 8

La différence fondamentale entre GS/OS et ProDOS 8 est bien sûr que GS/OS ne fonctionne que sur l'Apple IIGS. Ceci parce que GS/OS est écrit en langage assembleur 65C816, et qu'il utilise la Boîte à outils (Toolbox) de l'Apple IIGS comme par exemple le Memory Manager et le System Loader. Quoique la plupart des commandes GS/OS ont un

équivalent sous ProDOS 8, quelques commandes uniques font de GS/OS un environnement de programmation beaucoup plus riche.

1. Une application sous GS/OS peut appeler toutes les commandes GS/OS de n'importe quelle adresse mémoire à l'intérieur des 16 Méga-octets adressables par le 6C816. Une application sous ProDOS 8 peut appeler les commandes ProDOS 8 seulement à partir des premiers 64 Ko de mémoire.
2. Les applications sous GS/OS sont stockées dans des fichiers relogeables; cela signifie qu'elles peuvent être chargées et exécutées à toute adresse mémoire. Les applications ProDOS 8 sont simplement des images binaires du code programme, et en général elles ne peuvent être exécutées qu'à partir d'une adresse mémoire spécifique. Il est bien entendu possible d'écrire des applications sous ProDOS 8 relogeantes, mais cela rend leur programmation particulièrement difficile et la plupart des programmeurs ne s'ennuient pas avec ce problème.
3. Les applications sous GS/OS utilisent le Memory Manager de l'Apple IIgs pour s'assurer qu'elles n'utilisent pas des zones de mémoire déjà utilisées par d'autres programmes ou fichiers ressources du système. Les applications sous ProDOS 8 sont responsables de leur gestion mémoire, les programmeurs doivent donc veiller aux zones mémoire utilisées par le ProDOS 8.
4. GS/OS a 33 préfixes de pathname qui peuvent être référencés par un nom spécial abrégé comme par exemple 1/ ou 28/. ProDOS 8 dispose seulement d'un préfixe pathname que l'on appelle le préfixe par défaut.
5. GS/OS identifie les périphériques disque par un nom alors que ProDOS 8 les identifie par un numero de connecteur (slot) et de lecteur (drive).
6. GS/OS dispose d'une commande intégrée pour formater les disques (Format) et d'une commande permettant de cataloguer un volume (GetDirEntry). ProDOS 8 ne dispose pas de ces commandes.
7. GS/OS a une commande permettant de déplacer des fichiers d'un catalogue à un autre (ChangePath). ProDOS 8 ne dispose pas de cette commande.
8. Sous GS/OS, une application peut déterminer son propre nom avec la commande GetName. ProDOS 8 ne dispose pas de commande similaire; une application peut déduire son nom en inspectant le buffer pathname.
9. GS/OS dispose d'une commande Quit améliorée qu'une application peut utiliser pour directement passer le contrôle au programme SYStème qui l'a appelée, ou passer le contrôle à tout programme système presque comme s'il s'agissait d'une sous-routine.

La commande Quit de ProDOS 8 peut seulement passer le contrôle à un sélecteur de programmes.

10. GS/OS peut créer et gérer des fichiers étendus, ProDOS 8 ne le peut pas. Les fichiers étendus (parfois appelés fichiers ressource) sont constitués de deux parties logiques : une partie données et une partie ressource. La partie data contient généralement les données spécifiques à l'application, la partie ressource contient en général un groupe de données que l'on appelle ressources qui peuvent être des choses comme des icônes, des chaînes de caractères, les messages d'erreurs, etc.
11. GS/OS utilise des FST (File System Translator) pour permettre aux applications l'accès à des volumes disque n'utilisant pas le système d'exploitation ProDOS, comme par exemple le système High Sierra pour le CD Rom ou le système HFS pour le Macintosh. GS/OS utilise aussi un FST pour accéder aux fichiers de type ProDOS. Le ProDOS 8 fonctionne uniquement avec des disques formatés sous ProDOS.
12. GS/OS permet à une application d'avoir accès à des périphériques orientés caractères comme par exemple l'écran vidéo, le clavier, le modem, et l'imprimante en utilisant le même type de commandes que pour accéder aux fichiers disque. Sous ProDOS 8, l'application doit utiliser des techniques tout à fait différentes pour accéder aux périphériques orientés caractères, chacun d'eux nécessitant la compréhension du fonctionnement du hardware en détail.
13. GS/OS accède aux disques plus rapidement que ProDOS 8 car il utilise des techniques de disque cache et un codage assembleur 65C816 optimisé. Il peut aussi formater les disques avec un entrelacement de pistes plus bas (2:1 à la place de 4:1), améliorant ainsi la vitesse de transfert des données.
14. GS/OS permet d'ouvrir un nombre illimité de fichiers et de volumes; il n'impose pas de limites sur le nombre de périphériques par slot. ProDOS 8 autorise seulement 8 fichiers ouverts simultanément, 14 volumes actifs, et deux périphériques par slots.
15. GS/OS utilisant des FST peut accéder à des volumes non ProDOS pouvant avoir une taille jusqu'à 2048 GO (Giga-Octets), et peut gérer des fichiers ayant jusqu'à une longueur de 4096 Méga-octets. La taille des volumes ProDOS ne peut pas excéder 32 Méga-octets, les fichiers ne peuvent pas dépasser 16 Méga-octets.
16. GS/OS n'est pas fourni avec un langage BASIC interprété comme BASIC.SYSTEM sous ProDOS 8.

5.1 Volumes disques et organisation des fichiers

Dans cette partie, nous allons nous familiariser avec le concept de fichier et expliquer comment le système ProDOS organise les fichiers sur la surface du disque. Il est nécessaire de connaître les détails de ce système si vous voulez comprendre parfaitement le fonctionnement des commandes agissant sur les fichiers. GS/OS peut travailler avec des systèmes d'exploitation complètement différents du ProDOS, mais la plupart des utilisateurs utilisent GS/OS avec des disques formatés sous ProDOS.

Le concept de fichier est commun et fondamental à tous les systèmes d'exploitation. Un fichier est tout simplement un ensemble de données qui peut être un programme exécutable, une lettre à votre éditeur, une feuille de calcul pour un tableur, ou tout autre document qu'un programme peut gérer. La structure générale d'un fichier est définie par le système d'exploitation lui-même; le système d'exploitation fournit également les diverses commandes nécessaires pour accéder aux fichiers: création, ouverture, lecture, écriture, fermeture, destruction, etc...

5.1.1 Donner un nom aux fichiers

Lorsque vous sauvez pour la première fois un fichier sur disque, vous devez lui donner un nom qu'un programme pourra identifier par la suite. Un nom de fichier ProDOS peut être constitué d'un maximum de 15 caractères. Il doit commencer par une lettre alphabétique (A à Z), mais les autres caractères peuvent être toute combinaison de lettres, de chiffres (0 à 9), et de points (.). Vous pouvez aussi utiliser les lettres minuscules, mais ProDOS 8 et GS/OS les convertissent immédiatement en majuscules. Voici quelques exemples de nom de fichiers acceptés par ProDOS:

- MODELE.LETTRE
- CONTRAT.3
- CHAPITRE.QUATRE

Voici quelques exemples de noms de fichiers non valides :

- 5.MORCEAUX (Commence par un nombre)
- CHIEN ROUGE (Contient un espace)
- CECI&CELA (Contient un &)
- LE.DANUBE.DELA.PENSEE (Trop long)

Une erreur souvent commise est l'utilisation de l'espace comme séparateur de mots (voir exemple 2 ci dessus). Cela est permis en DOS 3.3 mais pas en ProDOS. Les points (.) et

non les espaces doivent être utilisés pour séparer les mots dans un nom de fichier pour améliorer la lisibilité. Certains programmes, comme Appleworks, permettent l'utilisateur d'entrer des espaces dans le nom des fichiers, mais ils convertissent automatiquement les espaces en points avant d'utiliser les commandes du système d'exploitation. GS/OS peut travailler avec des volumes disque qui ont été formatés avec d'autres systèmes d'exploitation, si le FST approprié se trouve sur le disque système. Les règles pour nommer les fichiers sont différentes pour ces systèmes d'exploitation. Par exemple, le système HFS du Macintosh permet de donner des noms d'une longueur maximum de 31 caractères: ces noms peuvent contenir tout caractère ASCII qui puisse être imprimé sauf les deux points (:). Consultez les manuels de référence du système d'exploitation utilisé pour connaître les règles en ce qui concerne les noms de fichiers.

5.1.2 Catalogues et Sous-Catalogues

Quand vous sauvegardez un fichier ProDOS sur disque, vous pouvez le ranger dans n'importe lequel des catalogues qui a pu être créé sur ce disque. Ces catalogues sont similaires à des dossiers dans lesquels on peut classer des fiches, car on place dans des dossiers des fiches ayant un lien entre elles. En fait le terme dossier est employé en informatique comme un synonyme de catalogue, on emploie aussi le terme répertoire. Par exemple, il est possible de créer un catalogue pour y stocker tous les documents traitement de textes, et un autre catalogue pour y placer tous les programmes Applesoft. La possibilité de créer des catalogues séparés sur le même disque facilite l'organisation et le rangement d'un nombre important de fichiers.

Quand on formate un disque, seul un catalogue, le catalogue principal du volume, existe. On lui donne un nom lorsque l'on formate le disque. Il devient alors le nom de ce volume. Les règles pour nommer les catalogues sont identiques à celles qu'il faut observer pour nommer un fichier. Le catalogue principal d'un volume formaté à l'aide de ProDOS peut contenir le nom d'un maximum de 51 fichiers. Le DOS 3.3 accepte 105 noms de fichiers.

Il est possible de créer des catalogues supplémentaires (on les appelle alors des sous-catalogues) à partir du catalogue principal en utilisant la commande CREATE de GS/OS ou de ProDOS. On peut même créer des sous-catalogues à l'intérieur d'autres sous-catalogues. Un sous-catalogue peut contenir les noms d'autant de fichiers que vous désirez, jusqu'à ce que votre disque soit plein. Ce système de catalogues imbriqués s'appelle une structure de catalogues hiérarchisée.

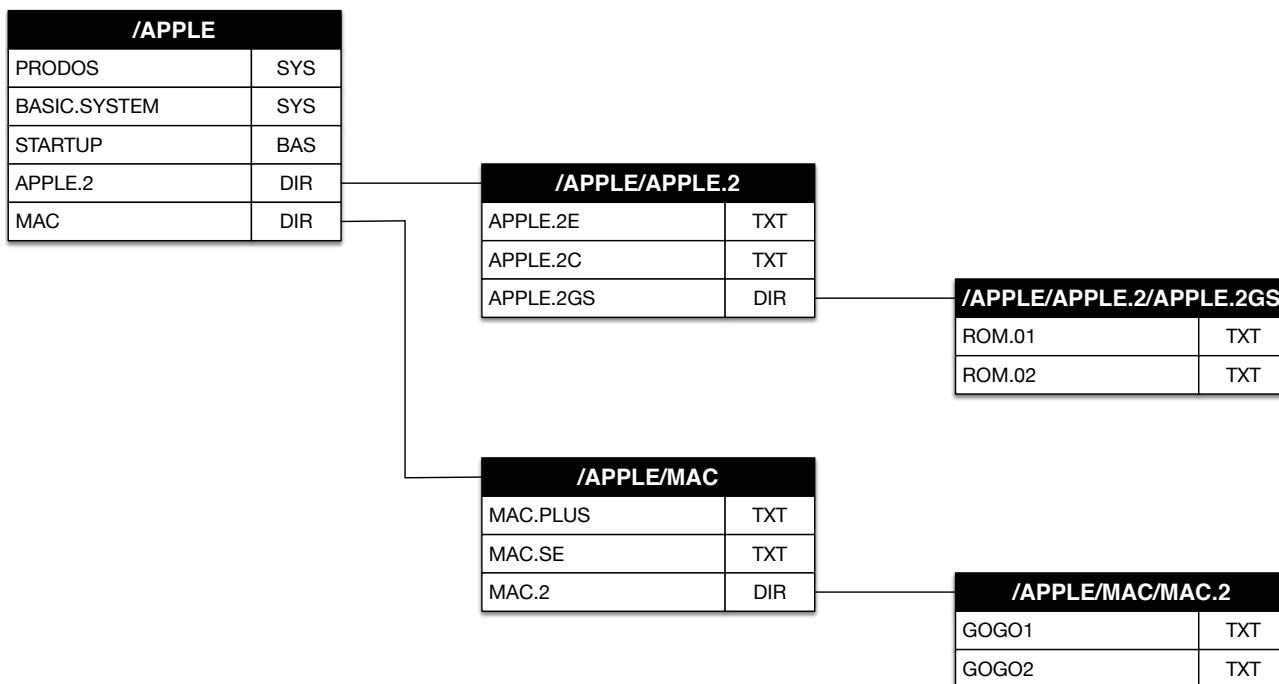
Tous les systèmes d'exploitation modernes utilisent une structure de catalogue hiérarchisée.

Pour indiquer le catalogue où un fichier doit être sauvé, vous devez ajouter à son nom un préfixe spécial afin de créer un identificateur unique appelé le PATHNAME ou Chemin d'accès en français. Un Pathname est constitué des noms d'une série de catalogues commençant par le nom du catalogue principal et continuant avec les noms de tous les catalogues par lesquels il faut passer pour atteindre le catalogue final, suivie par le nom du fichier lui-même. Chaque nom de catalogue est séparé du précédent par un délimiteur spécial, et ce délimiteur doit précéder le nom du catalogue principal.

Sous GS/OS, ce caractère délimiteur peut être soit le slash (/) ou les deux points (:). Sous ProDOS 8, cela doit être un slash (/). Nous utiliserons le / comme délimiteur dans les explications à venir.

Les noms de catalogues dans un Pathname (chemin d'accès) doivent définir un chemin continu; cela signifie que chaque catalogue spécifié doit être contenu dans le catalogue précédent. Par exemple, supposez que nous ayons un disque appelé APPLE; le catalogue principal de ce disque a donc le nom de APPLE. Supposez que le volume APPLE ait deux sous-catalogues appelés APPLE.2 et MAC.

Le tableau suivant montre cette structure de catalogues.



Si vous voulez sauvegarder un fichier appelé MAC.PLUS dans le sous-catalogue MAC, vous devrez indiquer le Pathname suivant:

`/APPLE /MAC/MAC.PLUS`

Si vous aviez seulement indiqué le nom de fichier MAC.PLUS, ce fichier aurait été sauvegardé dans le catalogue courant, qui est en général le catalogue principal à moins

que le catalogue courant ait été changé avec la commande SetPrefix que nous décrirons plus loin.

Sous GS/OS vous pouvez spécifier un nom de périphérique, à la place d'un nom de volume (le nom du volume est le même que le nom de catalogue principal), au moment où l'on forme le pathname. Les noms de périphériques commencent par un point (.) et peuvent avoir un nombre de caractères compris entre 2 et 31.

.SCSI1, .DEV4, .APPLEDISK3.5A sont des exemples de noms de périphériques. Si le fichier MAC.PLUS de l'exemple précédent se trouve sur le périphérique appelé .SCSI1; on pourrait l'identifier avec le pathname suivant :

.SCSI1/MAC/MAC.PLUS

Cette technique ne peut pas être utilisée sous ProDos 8 car ProDos 8 n'utilise pas les noms de périphériques.

Comme indiqué plus haut, sous GS/OS, le délimiteur dans un Pathname peut être / ou : mais vous ne pouvez pas utiliser deux délimiteurs différents pour un seul pathname. GS/OS détermine le délimiteur en examinant le Pathname indiqué de gauche à droite jusqu'à ce qu'il trouve un "/" ou un ":"; le caractère trouvé servira de délimiteur.

Si le délimiteur (sous GS/OS) est : vous pouvez utiliser des / dans les noms de fichiers, ce qui est très important si vous désirez accéder à des fichiers ne se trouvant pas sur un volume ProDos (en utilisant un FST approprié). Par exemple, les fichiers Macintosh peuvent contenir des /. L'inverse n'est pas possible. En effet, si le délimiteur est un "/" vous ne pouvez pas utiliser ":" dans un nom de fichier. Il est préférable dans une application sous GS/OS d'utiliser le ":" comme délimiteur dans les Pathnames.

5.1.3 Les Préfixes

Si la plupart des fichiers que vous allez utiliser se trouvent dans le même sous-catalogue, il devient rapidement ennuyeux de spécifier le même Pathname (chemin d'accès) de catalogues à chaque fois que l'on veut accéder à l'un de ces fichiers. Pour supprimer ce problème, GS/OS et ProDOS 8 ont une commande SetPrefix que vous pouvez utiliser pour former une chaîne de noms de catalogues qui s'ajoutera automatiquement à tout nom de fichier que vous spécifierez dans une commande. Cette chaîne s'appelle le Préfixe par défaut et ne peut pas avoir une longueur supérieure à 64 caractères sous ProDOS 8 ou de 8 Ko sous GS/OS. Si par exemple vous fixez le préfixe par défaut sur /APPLE/MAC/, vous pouvez faire référence à tout fichier de ce sous-catalogue en lui ajoutant simplement le nom de ce fichier (par exemple MAC.PLUS). Un nom qui est la continuation

du préfixe par défaut peut aussi accéder à des niveaux de sous-catalogues plus bas; un tel nom s'appelle un Pathname partiel. Si, par exemple, le préfixe par défaut est /APPLE/MAC/, et si le sous-catalogue MAC contient un sous-catalogue appelé MAC.2 contenant à son tour un fichier appelé GOGOL vous pouvez accéder à ce fichier en accolant au Pathname /APPLE/MAC/ le Pathname partiel MAC.2/GOGOL. Le Pathname partiel n'est pas précédé par un /.

Sous GS/OS (mais pas sous ProDOS 8), on peut utiliser une forme raccourcie du Préfixe par défaut en le remplaçant par 0/ Dans notre exemple, cela signifie que 0/ est équivalent à /APPLE /MAC. Comme le montre le tableau suivant, GS/OS reconnaît 32 préfixes différents auxquels on peut faire référence par un nombre suivi de / (de 0/ à 31/).GS/OS reconnaît aussi le préfixe de boot (le nom du volume à partir duquel vous avez booté) en l'identifiant par */; on ne peut pas modifier */ 1 / et 9/ identifient le catalogue .clans lequel l'application courante réside. 2/ identifie le catalogue qui contient les fichiers système. On peut modifier 1 /, 2/, et 9 / avec la commande GS/OS SetPrefix. Vous pouvez utiliser les autres préfixes quand une application a besoin d'identifier un catalogue particulier en utilisant les préfixes abrégés de GS/OS.

Sous ProDos 8 les préfixes peuvent, au maximum, avoir une longueur de 64 caractères, en comptant le / qui doit les précéder. Les Pathnames partiels ne peuvent eux non plus dépasser 64 caractères. GS/ OS a deux sortes de préfixes : les courts et les longs. Les préfixes courts (*/,et 0/ à 7/) peuvent avoir au maximum une longueur de 64 caractères; les préfixes longs (8/ à 31/) peuvent avoir au maximum une longueur de 8192 caractères.

Les numéros de préfixe sous GS/OS

Numéro de préfixe	Description
*/	Le préfixe de boot. C'est le nom du volume de boot.
0/	Le préfixe par défaut. Il est ajouté automatiquement.
1/	Le préfixe où est localisée l'application.
2/	Le préfixe qui pointe sur la bibliothèque système.
3/ à 8/	Laissés au choix de l'utilisateur.
9/	Identique à /1.
10/ à 31/	Laissés au choix de l'utilisateur.

Une caractéristique intéressante de GS/OS et de ProDOS 8 est qu'à chaque fois qu'une commande doit localiser un fichier décrit par un Pathname, elle effectue cette recherche sur tous les volumes disques connectés au système. Ceci est tout à fait différent du DOS 3.3 où l'on doit expliciter le numéro de slot et de lecteur pour accéder à un fichier (en utilisant les paramètres ,S# et ,D#). Pour des raisons de compatibilité avec le DOS 3.3, on

peut aussi utiliser ces paramètres avec le BASIC.SYSTEM. De toutes façons, BASIC.SYSTEM utilisera automatiquement le nom du volume disque indiqué par les paramètres de numéro de slot et de lecteur pour créer le Pathname.

Les avantages apportés dans l'utilisation des sous-catalogues sont souvent peu évidents aux utilisateurs de lecteurs de disquettes, mais tout à fait clairs aux utilisateurs de disques durs. Sur un disque dur, on peut ranger des centaines de fichiers. Si tous les fichiers étaient placés dans un catalogue unique, il faudrait attendre un temps important pour localiser un fichier en particulier lors du catalogue, de plus il y a de fortes chances pour que vous ne le trouviez pas parmi tous les autres fichiers. Heureusement que ProDOS permet une organisation hiérarchisée des catalogues permettant ainsi de ranger les fichiers de même intérêt dans un même sous-catalogue pour y accéder plus facilement.

5.1.4 Les concepts fondamentaux pour la manipulation d'un fichier

GS/OS et ProDOS 8 disposent d'un interpréteur de commandes qui peut comprendre une variété de commandes permettant de manipuler un fichier.

OPEN	Ouverture d'un fichier pour les opérations I/O.
READ	Lire des données à partir d'un fichier.
WRITE	Écrire des données dans un fichier.
CLOSE	Fermeture d'un fichier pour les opérations I/O.

Quatre commandes similaires sont également disponibles sous Applesoft quand on utilise l'interpréteur BASIC .SYSTEM dans l'environnement ProDOS 8.

5.1.5 Ouverture d'un fichier

Avant d'utiliser un fichier, on doit rouvrir. On réalise cette action en utilisant la commande Open suivie du nom du fichier que l'on désire ouvrir. Le système d'exploitation ouvre un fichier d'abord en le localisant sur le volume disque, puis en lui attribuant un buffer spécial en mémoire. Une partie de ce buffer contient l'information qui indique au système d'exploitation à quel endroit du volume disque sont localisées les données de ce fichier; une autre partie de ce buffer contient la portion du fichier à laquelle on vient d'accéder. A chaque fois que l'on demande une opération I/O pour un fichier, le système d'exploitation détermine si la portion du fichier à laquelle on veut accéder se trouve dans ce buffer. Si c'est le cas, le système d'exploitation n'a pas besoin d'accéder à ce fichier sur le volume disque. Il range simplement les données dans le buffer (dans le cas d'une opération

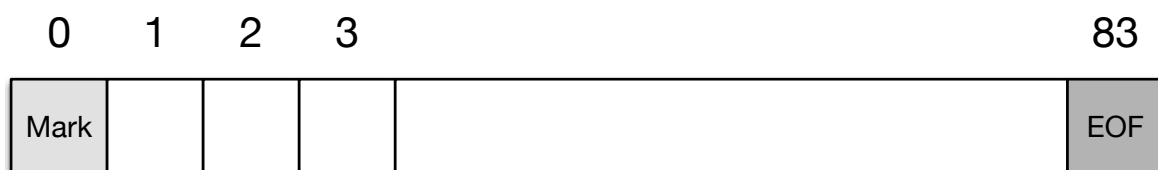
d'écriture), ou lit les données à partir de ce buffer (dans le cas d'une opération de lecture). Ainsi, les opérations portant sur un fichier s'exécutent beaucoup plus rapidement que dans le cas où on utiliserait des techniques ne mettant pas en oeuvre de buffers en mémoire.

ProDOS 8 peut ouvrir un fichier à l'un des 16 différents niveaux de fichiers système (numérotés de 0 à 15); GS/OS reconnaît 256 niveaux différents de fichiers système (numérotés de 0 à 255). Sous ProDOS 8, une application peut spécifier un niveau de fichier système en stockant le numéro de niveau à une adresse mémoire particulière (\$BF94) juste avant d'ouvrir le fichier. Sous GS/OS, l'application doit utiliser la commande SetLevel. Le fichier système par défaut a le niveau 0. L'avantage d'avoir différents niveaux de fichiers permet d'écrire facilement des programmes "superviseurs". Ce type de programmes ouvrent leurs propres fichiers de travail, passent le contrôle à des programmes utilisateur, et reprennent le contrôle quand les programmes utilisateur ont terminé. Si un programme "superviseur" a un niveau supérieur à un programme utilisateur, ses fichiers de travail ne pourront pas être fermés par inadvertance par le programme utilisateur même si ce programme tente de fermer tous les fichiers ouverts (sauf bien sûr si le programme utilisateur ne respecte pas la procédure normale en décrémentant le niveau de fichier).

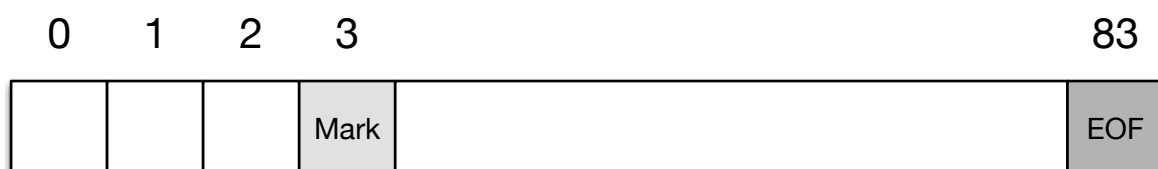
5.1.6 Lecture et Ecriture d'un fichier

Quand le système d'exploitation ouvre un fichier, il initialise deux pointeurs internes importants qu'il utilise pour garder trace de la taille de ce fichier et de la dernière position dans ce fichier référencée par une application. Ces pointeurs sont appelés EOF et MARK.

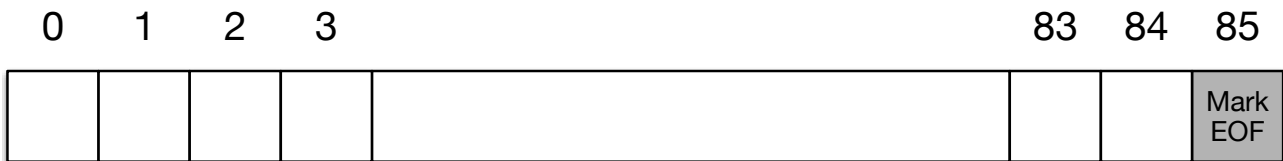
1) MARK & EOF après l'ouverture d'un fichier de 83 octets



2) MARK & EOF après la lecture de 3 octets de ce même fichier



3) MARK & EOF après l'écriture de 2 octets à la fin du fichier



Note : EOF est automatiquement étendu.

EOF (End Of File = Fin De Fichier) est le pointeur qui indique la fin du fichier; il pointe toujours un octet supplémentaire après le dernier octet réel du fichier. Si vous tentez de faire une lecture de données dans ce fichier après cet emplacement, une erreur se produira (l'erreur "fin de données"). EOF se modifie seulement si une application écrit des données à la fin de ce fichier; quand cela arrive, EOF est automatiquement incrémenté du nombre d'octets approprié, et si nécessaire, le système d'exploitation attribue des blocks supplémentaires sur le disque pour ce fichier. GS/OS et ProDOS 8 disposent d'une commande SetEOF permettant d'attribuer à EOF une valeur spécifique.

MARK est un pointeur qui donne la position courante dans le fichier; il indique toujours la position à laquelle la prochaine opération de lecture ou d'écriture prendra place. Il est égal à 0 (le début du fichier) quand on ouvre un fichier pour la première fois, et il est automatiquement incrémenté à chaque fois que des données sont lues et écrites dans ce fichier. Par exemple, si MARK pointe sur 10 (c'est à dire sur le 11ème octet du fichier), et que vous lisez ou écrivez 14 octets supplémentaires d'information, MARK se positionnera alors à 24.

Il est aussi possible de donner une valeur explicite au pointeur MARK permettant ainsi d'accéder à toutes les positions dans un fichier. Cela signifie qu'il est possible d'accéder à un enregistrement d'un fichier contenant des longueurs d'enregistrement fixes très rapidement, car il n'est pas nécessaire de lire tous les enregistrements précédents.

5.1.7 Fermeture d'un fichier

Une fois que l'on a terminé de travailler avec un fichier, il faut le fermer. Cela permet de s'assurer que les données se trouvant dans le buffer fichier, et pas encore rangées sur disque, vont être sauvegardées sur le disque lui même. La fermeture d'un fichier met également à jour dans le catalogue la taille de ce fichier.

Il n'est pas nécessaire de fermer un fichier immédiatement après avoir fini de travailler avec lui. On peut attendre jusqu'à la fin du programme pour procéder à cette fermeture. Il est quand même préférable de procéder à cette fermeture le plus rapidement possible pour réduire le risque de perte de données dans le cas d'un plantage du système ou d'une coupure électrique. Le fait de fermer les fichiers inactifs libère de la mémoire.

5.1.8 GS/OS et la technique du disque cache

Pour accélérer les opérations disque comme celles décrites plus haut, GS/OS effectue un cache des blocs disque. Le cache disque est une zone de mémoire où GS/OS sauvegarde une copie des blocs disque quand il les a lu une fois à partir du disque. GS/OS place aussi dans ce cache des copies des blocs qu'il écrit sur disque. Une fois qu'un bloc est dans le cache, GS/OS peut rapidement le récupérer de la mémoire à chaque fois qu'il a besoin de relire ce bloc disque. GS/OS n'a alors pas besoin de refaire un accès disque (relativement lent) pour relire ce bloc.

L'utilisateur peut régler la taille de ce cache disque avec l'accessoire de bureau (NDA) Disk Cache. (Dans le système 5.0 ce réglage est incorporé dans le NDA plus général Control Panel qui permet de configurer tout le système.) Une application peut régler la taille de ce cache en utilisant la commande GS/OS ResetCache après avoir sauvegardé cette nouvelle taille du cache dans la RAM alimentée (BRAM) en utilisant pour cela la fonction WriteBParam. Plus la taille du cache est importante, plus GS/OS est performant, mais l'espace mémoire utilisable par les applications s'en trouve réduit en proportion.

Dans la plupart des cas, la taille du cache n'est pas suffisamment importante pour contenir tous les blocs que GS/OS voudrait y placer. Quand la mémoire cache est saturée, GS/OS enlève de la mémoire cache le bloc le moins récemment utilisé pour libérer de la place au prochain bloc.

Les commandes GS/OS Read et Write vous permettent de spécifier des blocs disque spécifiques qui pourront ou non être placés dans cette mémoire cache.

5.1.9. Organisation des fichiers sous ProDOS

Les systèmes d'exploitation utilisent différentes méthodes pour organiser les fichiers sur disque, et pour garder trace de quelles parties du disque ont été utilisées pour la sauvegarde des données, de telle manière que les fichiers peuvent facilement et efficacement être créés, effacés, etc... Dans ce passage, nous allons examiner les points suivants:

- La structure d'un volume disque formaté sous ProDOS
- La structure d'une bit map d'un volume ProDOS
- La structure des catalogues et des sous catalogues sous ProDOS
- La structure d'une entrée dans un catalogue ProDOS
- Le processus d'index utilisé par ProDOS pour localiser les fichiers ProDOS utilise la même méthode générale pour organiser les fichiers quel que soit le volume disque utilisé; il s'agit de toute façon de volumes disque divisés en blocs : lecteur Apple 5.25", lecteur Apple 3.5", HD20SC, volume /RAM , etc. Des différences spécifiques peuvent apparaître car la capacité de stockage de ces différents volumes est variable. De plus, les tailles de deux structures très importantes, le catalogue principal, et la bit map (table d'occupation) d'un volume peuvent être différentes.

5.1.10 Formatage du volume disque

Avant de pouvoir utiliser une disquette (ou tout autre support magnétique) avec GS/OS ou ProDOS 8, elle doit être formatée pour que GS/OS ou ProDOS 8 la reconnaisse. Vous pouvez formater un volume disque avec un Filer (ProSel, Copy II+, etc ...), ou avec les utilitaires système qui se trouvent sur la disquette master ProDOS 8, ou encore avec le Finder de GS/OS. GS/OS dispose d'une commande Format utilisable par les applications pour formater un volume disque.

La méthode utilisée pour formater un disque dépend de la nature de ce périphérique disque. Quand on formate une disquette 5.25", 35 pistes sont créées sur le disque (numérotées de 0 à M), chacune d'entre elles contenant 4096 octets d'information. Ces pistes sont arrangées en anneaux concentriques tout autour du trou central de la disquette; la piste 0 se trouvant à l'extérieur, et la piste 34 à l'intérieur. Le système d'exploitation peut accéder à chacune de ces pistes en déplaçant une tête de lecture/écriture sur la piste choisie. Ceci est réalisé en utilisant des emplacements I/O (Entrée/Sortie) particuliers qui activent un moteur pas à pas contrôlant le déplacement de la tête de lecture/écriture.

Chacune de ces 35 pistes est subdivisée en 16 unités plus petites appelées secteurs. Un secteur est la plus petite unité de données qui peut être lue ou écrite en même temps. Les secteurs qui constituent une piste sont numérotés de 0 à 15; chacun d'entre eux pouvant contenir 256 octets d'information. Si vous savez faire une multiplication, vous vous apercevrez qu'une disquette 5.25" peut contenir 560 secteurs d'information (140 Ko).

C'est la dernière fois que vous entendrez parler de secteurs car ProDOS utilise des blocs de 512 octets comme unité de base dans la manipulation des fichiers; chaque bloc est donc constitué de deux secteurs. Une disquette 5.25" est donc constituée de 280 de ces blocs (numérotés de 0 à 279). Heureusement, il est rarement nécessaire de connaître la localisation de ces blocs sur disque; le système d'exploitation se charge de cette gestion.

5.1.11 Volumes disque et Lecteurs de disque

Une disquette formatée qui est en ligne (placée dans un lecteur et prête à être accédée) est souvent appelée un volume disque. Les volumes ProDOS ont des noms qui suivent les mêmes règles de baptême que les fichiers, mais sont précédés d'un / pour les distinguer des noms de fichiers.

Les lecteurs de disque eux-mêmes ont des identificateurs uniques. ProDOS 8 donne un numéro d'unité à chaque périphérique disque connecté au système. La valeur de ce numéro d'unité est formée par le numéro de slot de la carte contrôleur de ce lecteur, et du numéro du lecteur.

7	6	5	4	3	2	1	0
DR	SLOT			Non utilisé			

SLOT peut en fait être le numéro réel ou logique d'un slot si le système contient des périphériques disque comme par exemple un RAM disque. Par exemple, le numéro d'unité d'un volume /RAM connecté en slot 3, Drive 2 sur un Ile, Ilc, ou IIGS est \$B0 soit 1011 0000 en binaire ou 176 en décimal.

DR indique le numéro de Drive (lecteur): 0 pour le lecteur 1 et 1 pour le lecteur 2. Plus de deux lecteurs peuvent être connectés au port 5 du SmartPort. Dans ce cas, ProDOS 8 assigne logiquement les deux prochains lecteurs en Slot 2, Drive 1 et en Slot 2, Drive 2. ProDOS 8 ignore tous les lecteurs connectés au SmartPort après le quatrième.

GS/OS donne des numéros de référence uniques aux périphériques disque (et aux périphériques caractères) qu'il trouve connecté au système. Ces nombres sont des entiers consécutifs commençant par 1.

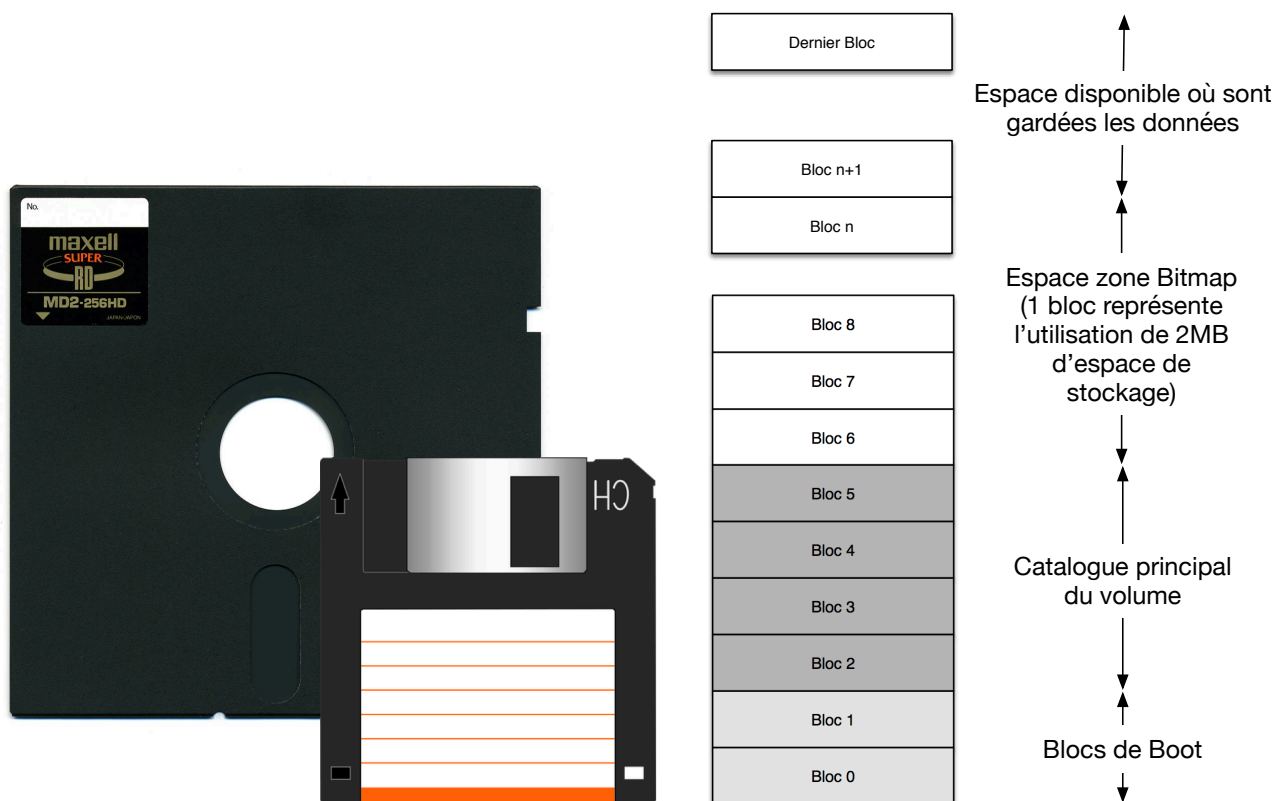
Il leur donne aussi un nom de périphérique; par exemple

.APPLEDISK3.5A,.SCSI1,etc.

Ces noms peuvent être constitués de 2 à 31 caractères de long. GS/OS n'utilise pas le procédé du numéro d'unité utilisé par ProDOS 8. Voir plus loin pour des explications détaillées concernant les périphériques disque et les conventions sur leur nom.

5.1.12 Utilisation des blocs sur un Volume disque

Nous allons maintenant examiner la méthode utilisée par ProDOS pour gérer les fichiers sur disque. Notre discussion induit une analyse des structures des catalogues renfermant l'information sur les fichiers, de la bit map qui renferme l'information concernant l'utilisation des blocs du volume disque, et des blocs d'index qui contiennent les emplacements des blocs de données utilisés par chacun des fichiers du volume. Comme nous l'avons dit plus haut, un total de 280 blocs contenant 140 Ko de données sont disponibles sur une disquette 5.25" formatée sous ProDOS. Si un programme standard de formatage est utilisé, 7 de ces blocs (0 - 6) ne sont pas disponibles pour l'utilisation pour le stockage des fichiers car ProDOS se les réserve. La figure suivante montre l'utilisation des blocs sur des disquettes 5.25" et 3.5" venant d'être formatées.



Chaque bloc contient 512 octets.

Pour une disquette 5.25", la capacité totale de stockage est de 280 blocs (140 Ko).

Pour une disquette 3.5", la capacité totale de stockage est de 1600 blocs (800 Ko).

Les blocs 0 et 1 contiennent un court programme en langage machine que le firmware de la carte contrôleur de disque charge en mémoire et exécute à chaque fois qu'il boote un disque. Ce programme est appelé programme de boot; il localise, charge et exécute un fichier SYStème spécial appelé PRODOS s'il le trouve sur le disque. Un fichier SYStème a un type de fichier ayant pour code \$FF, et le mnémonique SYS. PRODOS est le programme qui va installer et activer le système d'exploitation.

Les blocs 2 à 5 sont les blocs qui contiennent le catalogue principal du volume disque. Nous décrivons plus loin la structure de ce catalogue.

Le bloc 6 est le premier bloc contenant la bit map pour un volume disque. Dans la bit map, chaque bit indique si le block auquel il correspond est libre ou occupé. ProDOS réserve un bloc bit map pour chaque 2 MB d'espace de stockage. Comment arrive-t'on à ce nombre? Un bloc contient 512 octets, soit 4096 bits. Chaque bit indique si un bloc de 512 octets est occupé ou non. Si l'on multiplie 4096 par 512 on obtient 2.097.152 soit 2MB.

Les blocs au delà du bloc de bit map (ou des blocs), soit un total de 273 (280-7) pour une disquette 5.25" ou 1593 (1600-7) pour une disquette 3.5", sont libres et peuvent être utilisés pour le stockage des fichiers sur disque.

5.1.13 La bit map d'un volume (Table d'occupation)

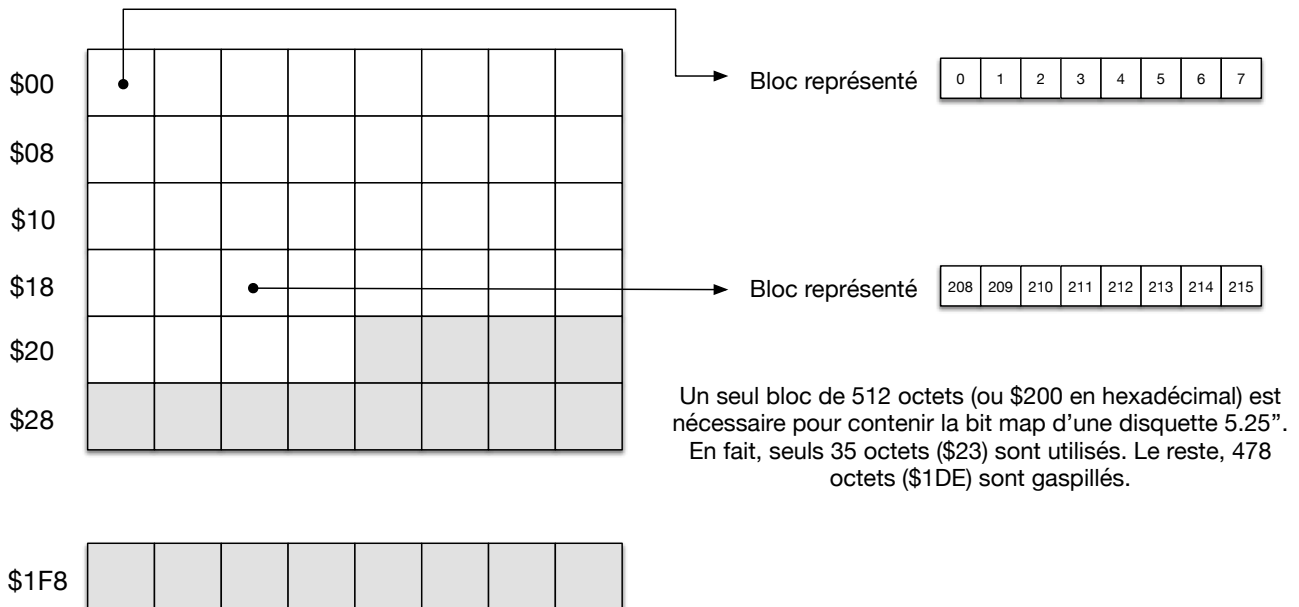
Le système d'exploitation accède à la bit map d'un volume quand il a besoin de déterminer l'état de chaque bloc sur le disque. Il lit la bit map à chaque fois qu'il attribue un nouvel espace disque à un fichier, de telle manière qu'il peut rapidement localiser les blocs libres de ce disque. Il écrit dans la bit map pour réserver des blocs à un fichier; cela arrive quand un fichier déjà existant augmente en taille, ou libère des blocs; cela arrive quand un fichier diminue de taille ou quand il est effacé.

Les routines standards de formatage utilisent le bloc 6 comme premier bloc bit map d'un volume. Le bloc 6 est seulement un emplacement conventionnel pour placer le début de la bit map; il est possible de ranger la bit map n'importe où ailleurs sur un bloc libre du disk. Par exemple, la bit map d'un volume /RAM est en bloc 3. Comme indiqué plus loin, le numéro de bloc du premier bloc de bit map d'un volume apparaît dans l'entête du catalogue principal décrivant les caractéristiques du volume disque.

Sur une disquette 5.25", seuls les 35 premiers octets (280 bits) du bloc de bit map du volume sont utilisés; chacun des bits de chaque octet correspond à un numéro unique de bloc. Un bloc de bit map peut donc faire référence à des volumes constitués d'un

maximum de 4096 blocs (2MB). Donc, même pour une disquette 3.5" on n'a besoin que d'un seul bloc pour représenter la bit map.

Bitmap d'une disquette 5.25"



Pour des volumes permettant de stocker beaucoup plus de données, comme les disques durs, une continuation de la bit map peut être trouvée sur les blocs qui suivent directement celui utilisé en premier.

Par exemple, le vieux disque dur Apple ProFile (9728 blocs ou presque 5MB) nécessite 3 blocs pour constituer sa bit map. Le programme standard de formatage stocke la première partie de la bit map sur le bloc 6 et la continue sur les blocs 7 et 8. Le système d'exploitation détermine la taille de la bit map d'un volume en examinant deux octets dans l'entête du catalogue du volume indiquant la taille de ce volume.



Chaque octet dans la bit map du volume définit l'état de huit blocs contigus. Le bit correspondant à un numéro de bloc donné peut être calculé en divisant d'abord le numéro de bloc par 8 (ce qui peut se faire en effectuant trois fois une opération ASR A). La partie entière du résultat donne le numéro de l'octet correspondant. Pour obtenir le numéro du bit dans cet octet, ôtez le reste de cette division de 7.

Numéro d'octet = INT(Numéro de bloc / 8)

Numéro de bit = 7 - (Numéro de bloc - (8 * Numéro d'octet))

La figure de la page précédente montre la structure d'une bit map pour des disquettes 5.25". Comme vous pouvez le voir, les bits de chacun des octets de cette bit map reflètent l'état de 8 blocs consécutifs; le bit 0 correspond au bloc ayant le numéro le plus haut, le bit 7 correspond au bloc ayant le numéro le plus bas. Si le bit correspondant à un bloc particulier est égal à 0, alors ce bloc est occupé. S'il vaut 1, il est libre.

Exemple de bit map

Index	Données															
\$00	\$00	\$3F	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF
\$10	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF	\$FF
\$20	\$FF	\$FF	\$FF	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00
\$30	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00	\$00

Le première entrée dans la table (octet \$00) a pour valeur \$00 ce qui correspond à 0000 0000 en binaire; cela signifie que les blocs \$00 à \$07 sont occupés. La deuxième entrée dans la table (octet \$01) a pour valeur \$3F ce qui correspond à 0011 1111 en binaire; cela signifie que les blocs \$08 à \$09 sont occupés (valeur binaire 0), et que les blocs \$0A à \$0F sont libres (valeur binaire 1). Les autres blocs de la bit map (\$10 à \$17) sont libres car leur valeur binaire est 1. Dans cet exemple nous avons donc un total de dix blocs occupés sur le volume disque. Ici la bit map s'arrête à partir de l'octet \$23; le système d'exploitation connaît la taille de la bit map car il la retrouve dans l'entête du catalogue principal.

5.1.14 Catalogues et Sous-Catalogues d'un Volume

Un catalogue est un enchevêtrement compliqué de données que ProDOS utilise pour conserver les informations importantes concernant chaque fichier sauvegardé sur le volume. Cela comprend le nom du fichier, son type, sa taille, sa date de création, l'emplacement sur le volume des données de ce fichier, etc. Sans ces informations, il ne serait pas possible de gérer efficacement de nombreux fichiers sur un volume.

Comme nous l'avons dit, ProDOS permet de créer de multiples catalogues sur un volume. A l'exception du catalogue principal (on accède à tous les autres catalogues en faisant référence au catalogue principal), ces catalogues peuvent occuper sur le volume tout l'espace nécessaire car ProDOS les traite comme des fichiers standards. Le catalogue principal débute toujours à partir du bloc 2; si vous utilisez un programme de formatage

standard, ou les commandes Format de GS/OS et EraseDisk, le catalogue principal occupera aussi les blocs 3, 4 et 5.

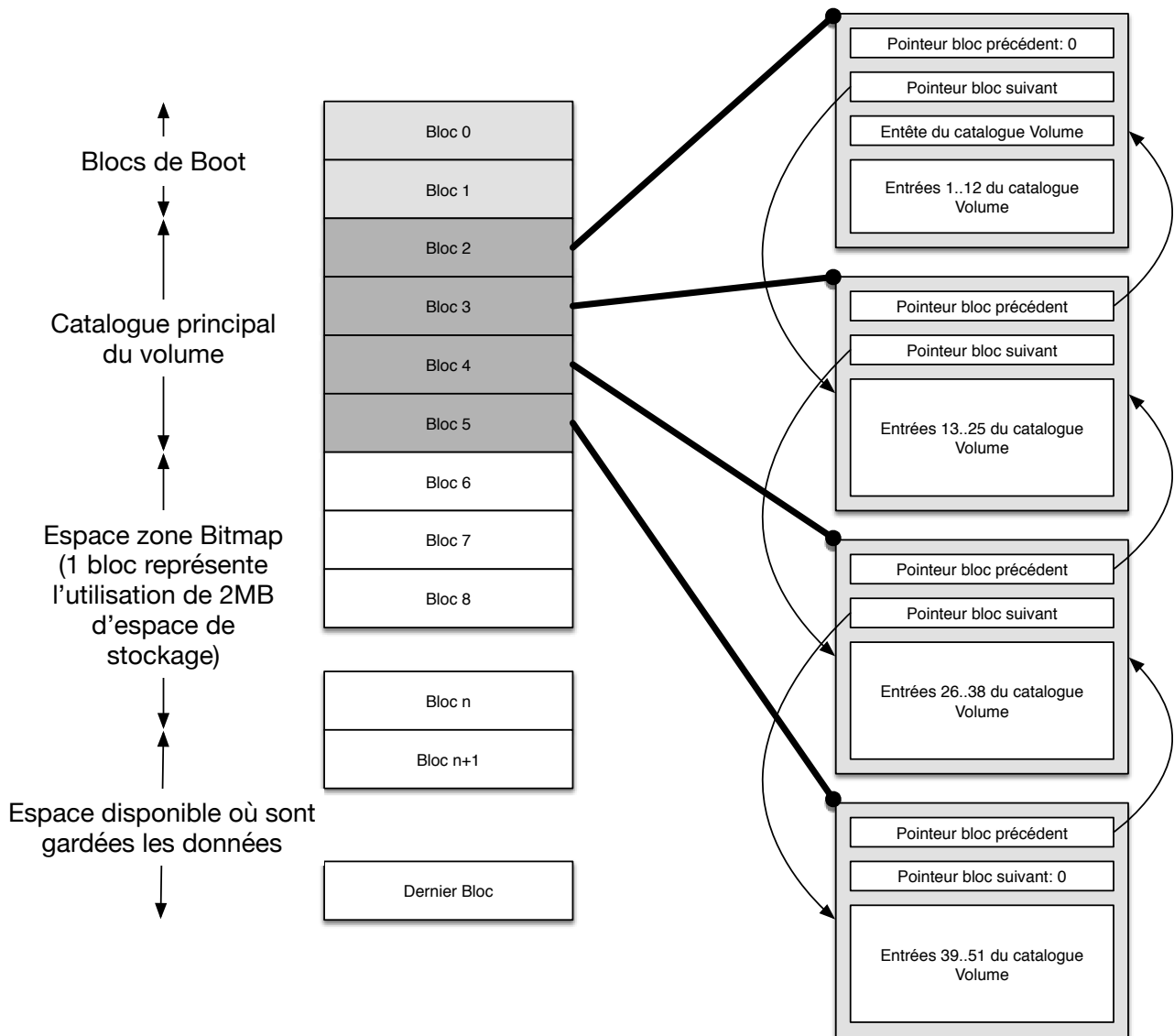
Un catalogue ProDOS est un exemple d'une structure de données double lien. Les liens sont en fait des paires de pointeurs sur 2 octets stockés au début de chaque bloc de catalogue. Un de ces pointeurs (Octets \$00-\$01) contient le numéro du bloc de catalogue précédent dans la chaîne, ou zéro s'il n'y a pas de bloc précédent. L'autre Pointeur (Octet \$02-\$03) contient le numéro du prochain bloc catalogue, ou zéro s'il n'y a pas de bloc catalogue à la suite. Cette technique permet de créer des catalogues de toute taille.

Chaque bloc utilisé par un catalogue peut contenir 13 entrées de fichier; chaque entrée de fichier est constituée de 39 octets. Cela signifie que le catalogue principal (blocs \$02-\$05) peut contenir un maximum de 52 entrées, dont l'une d'elles est l'entrée pour le nom de Volume lui même. Voilà pourquoi vous obtenez parfois le sympathique message d'erreur "Disque Plein" alors que votre disque dur de 100 MO ne contient qu'une poignée de fichiers; vous avez simplement oublié de sauvegarder vos précieuses données ailleurs que dans le catalogue principal qui ne peut contenir que 51 entrées...

Quand ProDOS doit trouver un fichier particulier, il lit en premier le bloc 2 du volume; il s'agit du premier bloc du Catalogue Principal du Volume, celui auquel le système d'exploitation accède le plus souvent (et en conséquence le plus fragile). Si ce fichier n'est pas trouvé dans ce bloc, le bloc de catalogue suivant est lu, et ainsi de suite jusqu'au bloc 5.

5.1.14 Le Catalogue Principal d'un Volume ProDOS

Le catalogue principal d'un volume ProDOS occupe quatre blocs (soit 2Ko) et peut contenir au maximum les données de 51 fichiers. Ces fichiers peuvent être des fichiers de tout type (BAS, BIN, SYS) y compris des sous-catalogues.



Structure d'un bloc Catalogue

Index	Description
\$000-\$001	Numéro de bloc du bloc catalogue précédent (octet de poids faible d'abord). Ce numéro est égal à 0 s'il s'agit du premier bloc catalogue.
\$002-\$003	Numéro de bloc du bloc catalogue suivant (octet de poids faible d'abord). Ce numéro est égal à 0 s'il s'agit du dernier bloc catalogue.

Index	Description
\$004-\$02A	Entrée au catalogue du fichier numéro 1 ou s'il s'agit du premier bloc catalogue (bloc clé), les octets \$00 et \$01 valent 0 et on trouve ici l'entête du catalogue (Directory header en anglais)
\$02B-\$051	Entrée au catalogue du fichier numéro 2
\$052-\$078	Entrée au catalogue du fichier numéro 3
\$079-\$09F	Entrée au catalogue du fichier numéro 4
\$0A0-\$0C6	Entrée au catalogue du fichier numéro 5
\$0C7-\$0ED	Entrée au catalogue du fichier numéro 6
\$0EE-\$114	Entrée au catalogue du fichier numéro 7
\$115-\$13B	Entrée au catalogue du fichier numéro 8
\$13C-\$162	Entrée au catalogue du fichier numéro 9
\$163-\$189	Entrée au catalogue du fichier numéro 10
\$18A-\$1B0	Entrée au catalogue du fichier numéro 11
\$1B1-\$1D7	Entrée au catalogue du fichier numéro 12
\$1D8-\$1FE	Entrée au catalogue du fichier numéro 13
\$1FF	Non utilisé

5.1.15 L'Entête du Catalogue (Directory Header)

Le premier bloc qu'un catalogue (ou qu'un sous-catalogue) utilise est le bloc clé, et il a une structure un peu différente des autres blocs catalogue. La différence réside dans les premiers 39 octets qui normalement décrivent l'entrée au catalogue du fichier numéro 1; dans ce cas ces octets servent à décrire le catalogue lui même; cette entrée est alors appelée entête du catalogue. Le tableau qui suit montre la signification de ces 39 octets constituant l'entête du catalogue.

Entête du Catalogue

Index	Description
\$04	4 Bits de poids fort: Type d'enregistrement \$F = Catalogue Volume \$E = Sous-Catalogue 4 Bits de poids faible: Longueur du nom du volume
\$05-\$13	Nom du volume: 15 octets (ASCII positif) La longueur de ce nom est dans l'octet \$04. Les octets après la fin du nom sont ignorés

Index	Description
\$14-\$1B	Réservé pour une utilisation future (en général rempli avec des 00)
\$1C-\$1D	Date de création du catalogue (formatage) YYYYYYYMMMMDDDD (Y=Année, M=Mois, D=Jour) en forme binaire
\$1E-\$1F	Minute et heure de création du catalogue 000HHHHH00MMMMMM (H=Heure, M=Minute) en forme binaire
\$20	Version de ProDOS utilisée pour formater le volume
\$21	Version minimum de ProDOS pouvant utiliser ce volume
\$22	Code d'accès ProDOS \$80 Le volume peut être reformaté \$40 Le volume peut être renommé \$20 Le catalogue a été modifié depuis le dernier backup (copie de sauvegarde) \$02 Autorisation de sauvegarde d'écriture sur le volume \$01 Autorisation de lecture sur le volume
\$23	Le nombre d'octets utilisés par chaque entrée au catalogue (\$27, soit 39 en décimal)
\$24	Le nombre d'entrées sur chaque bloc de catalogue (\$0D, soit 13 en décimal). Notez que l'entête du catalogue est considérée comme une entrée.
\$25-\$26	Nombre d'entrées actives dans le catalogue. Une entrée active décrit un fichier ou un sous-catalogue actif (ce qui veut dire qu'il n'a pas été effacé). L'entête du catalogue ne compte donc pas comme un fichier actif.
\$27-\$28	Pointeur sur le premier bloc de la bit map (en général le bloc 6). Dans un sous-catalogue c'est le pointeur indiquant le bloc définissant l'entrée de ce sous-catalogue.
\$29-\$2A	Taille du volume (en blocs de 512 Ko) \$0118 Disquette 5.25" (280 blocs) \$0640 Disquette 3.5" (1600 blocs)

5.1.15 Entrée au Catalogue

Toutes les entrées au catalogue, autres que l'entête du catalogue, représentent soit des fichiers de données standards (binaires, texte, programme Basic Applesoft, etc...), soit des sous-catalogues. Les formats de ces différentes entrées sont dans l'essentiel identiques et sont décrits ici.

Descriptif d'une entrée d'un fichier au Catalogue

Index	Description
\$00	<p>4 Bits de poids fort: Type d'enregistrement</p> <ul style="list-style-type: none"> \$0 = Entrée effacée. L'espace est disponible pour un nouveau fichier \$1 = Fichier "SEEDLING" (1 seul bloc de data) \$2 = Fichier "SAPLING" (entre 2 et 256 blocs data) \$3 = Fichier "TREE" (entre 257 et 32768 blocs data) \$4 = Zone Pascal \$5 = Fichier étendu \$D = Sous-catalogue \$E = Réserve pour l'entrée d'un sous-catalogue \$F = Réserve pour l'entête du catalogue principal <p>4 Bits de poids faible: Longueur du nom du fichier. Quand le fichier a été effacé, cet octet vaut zéro.</p>
\$01-\$0F	Nom du fichier: 14 octets (ASCII positif)
\$10	Type de fichier (voir table correspondante)
\$11-\$12	<p>Pointeur sur le bloc clé</p> <p>Numéro de bloc du bloc clé d'un fichier:</p> <ul style="list-style-type: none"> • Dans le cas d'un fichier "SEEDLING" c'est le numéro de bloc du seul bloc de données • Dans le cas d'un fichier "SAPLING" c'est le numéro de bloc du bloc d'index • Pour les fichiers "TREE" c'est le numéro de bloc du bloc d'index master • Pour un sous-catalogue c'est le numéro de bloc de son premier bloc
\$13-\$14	<p>Taille du fichier en nombre de blocs</p> <p>Ce nombre inclus les blocs d'index et les blocs de données (Data) Si le fichier est un sous-catalogue, la taille représente le nombre de blocs de ce sous-catalogue.</p>
\$15-\$17	EOF (End Of File) - Taille du fichier en octets (octet de poids faible en tête)
\$18-\$19	Date de création du fichier YYYYYYMMMMDDDD (Y=Année, M=Mois, D=Jour) en forme binaire
\$1A-\$1B	Minute et heure de création du fichier 00HHHHH00MMMMMM (H=Heure, M=Minute) en forme binaire
\$1C	Version ProDOS lors de la création de ce fichier
\$1D	Version minimum de ProDOS pour ce fichier
\$1E	<p>Code d'accès pour ce fichier</p> <ul style="list-style-type: none"> \$80 Le fichier peut être détruit \$40 Le fichier peut être renommé \$20 Fichier modifié depuis de dernier backup \$02 Autorisation d'écrire dans le fichier \$01 Autorisation de lecture du contenu du fichier <p>En général le code d'accès d'un fichier non verrouillé est \$C3 (\$80 + \$40 +\$02 +\$01) et celui d'un fichier verrouillé vaut \$01</p>

Index	Description
\$1F-\$20	Type de fichier auxiliaire. La signification de cette valeur dépend tu type de fichier TXT Longueur du fichier texte BIN Adresse de chargement d'une image binaire BAS Adresse de chargement d'un programme BASIC (\$801) VAR Adresse de chargement d'un fichier de variables SYS Adresse de chargement d'un programme système (\$2000)
\$21-\$24	Date/Heure de la dernière modification du fichier. Même format que pour la date/heure de création
\$25-\$26	Numéro de bloc du bloc clé du catalogue contenant l'entrée du fichier

Types de fichier

\$00	UNK	Type inconnu ("Unknown") Pas de type de fichier.
\$01	BAD	Fichier de blocs défectueux.
\$02	PCD	Fichier de code Pascal
\$03	PTX	Fichier de texte Pascal
\$04	TXT	Fichier de texte (ASCII positif)
\$05	PDA	Fichier de données Pascal
\$06	BIN	Fichier binaire (Image binaire 8 bits)
\$07	FNT	SOS (Apple 3) Fichier font (police de caractères)
\$08	FOT	Fichier FOTO SOS (Apple 3)
\$09	BA3	Programme Business BASIC (Apple 3)
\$0A	DA3	Fichier de données Business BASIC
\$0B	WPF	Fichier de traitement de textes
\$0C	SOS	Fichier SYSTEM SOS (Apple 3)
\$0F	DIR	Fichier sous-catalogue
\$10	RPD	Fichier de données RPS
\$11	RPI	Fichier index RPS
\$12	AFD	Fichier Discard AppleFile
\$13	AFM	Fichier Model AppleFile
\$14	AFR	Fichier Format de Rapport AppleWorks
\$15	SCL	Fichier Screen Library

\$19	ADB	Fichier de base de données AppleWorks
\$1A	AWP	Fichier de traitement de textes AppleWorks
\$1B	ASP	Fichier tableur AppleWorks
\$B0	SRC	Fichier source APW
\$B1	OBJ	Fichier objet APW
\$B2	LIB	Fichier bibliothèque APW
\$B3	S16	Fichier System GS/OS
\$B4	RTL	Bibliothèque run-time APW
\$B5	EXE	Fichier Executable APW
\$B6	PIF	Fichier Init Permanent GS/OS
\$B7	TIF	Fichier Init Temporaire GS/OS
\$B8	NDA	Fichier New Desk Accessory (NDA)
\$B9	CDA	Fichier Classic Desk Accessory (CDA)
\$BA	TOL	Fichier Outil GS/OS (Tool)
\$BB	GRV	Fichier Driver GS/OS
\$BC	GLF	Fichier Load GS/OS
\$BD	FST	File System Translator GS/OS
\$C0	PNT	Fichier Image SHGR compressé
\$C1	PIC	Fichier Image SHGR
\$C8	FON	GS/OS Font (Police de caractères)
\$C9	FND	Fichier de données utilisées para le Finder
\$CA	ICN	Fichier Icône
\$CB	AIF	Fichier de son Audio Interchange Format
\$EE	R16	Fichier objet relogeable EDASM 816
\$EF	PAS	Partition Pascal sur un volume
\$D5		Fichier de séquence de musique SoundSmith
\$F0	CMD	Fichier commande BASIC.SYSTEM
\$F1		
...		Types de fichiers à définir
\$F8		
\$F9	O.S	Système d'exploitation GS/OS
\$FA	INT	Fichier de programme Integer BASIC

\$FB	IVR	Fichier de variables Integer BASIC
\$FC	BAS	Fichier de programme AppleSoft BASIC
\$FD	VAR	Fichier de variables AppleSoft BASIC
\$FE	REL	Fichier relogante code EDASM
\$FF	SYS	Fichier système ProDOS 8

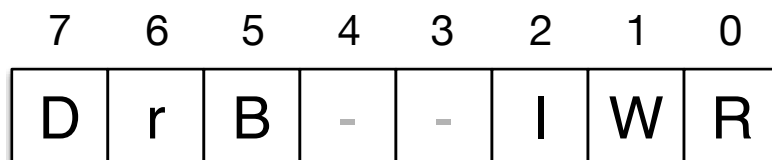
5.1.16 Code d'accès Fichier

Pour chaque entrée de fichier au catalogue, l'octet relatif \$1E permet de définir les opérations possibles pour ce fichier. Le bit 0 de l'octet identifie la lecture, le bit 1 l'écriture, le bit 6 la possibilité de renommer le fichier, et le bit 7 la possibilité de détruire ce fichier. Si la valeur d'un bit est à 1, ProDOS autorise l'opération associée à ce bit.

Le bit 2 indique si le fichier doit être considéré comme visible ou invisible. Si ce bit est à 1, les sous-routines de catalogue d'un disque ignorent ce fichier.

Le bit 5 indique si ce fichier a été modifié depuis sa dernière copie. C'est bien entendu au programme de copie de mettre ce bit à 0 quand il réalise une copie de ce fichier.

Les bits 3 et 4 ne sont pas utilisés et doivent toujours être à 0.



Si le bit correspondant est à 1 :

- | | |
|---------------------------------|-------------------------------------|
| D = Autorisation de destruction | I = Invisibilité du fichier |
| r = Autorisation de renommer | W = Autorisation d'écriture (Write) |
| B = Backup nécessaire | R = Autorisation de lecture (Read) |

Les commandes BASIC.SYSTEM LOCK et UNLOCK affectent l'état du code d'accès fichier:

- LOCK Interdit l'écriture, la fonction Rename et la fonction Destroy
- UNLOCK Autorise à nouveau les fonctions bloquées par LOCK.

Il n'y a pas de commandes BASIC.SYSTEM pour modifier individuellement chacun de ces bits. Par contre, nous verrons plus loin que l'on peut le faire avec la commande SetFileInfo de ProDOS 8 ou de GS/OS.

Si un bit est à 1, la fonction attribuée à ce bit est autorisée; s'il vaut 0 la fonction n'est pas autorisée. Les bits 4 et 3 (bits réservés) doivent toujours être à 0.

Si les bits D, r, et W sont tous à 1, on dit que le fichier n'est pas verrouillé ("Unlocked" en anglais); s'ils sont tous les trois à 0, alors ce fichier est verrouillé ("Locked" en anglais).

Toutes autres combinaisons signifient que ce fichier a des limitations restrictives d'accès.

Le bit d'invisibilité concerne les routines de catalogue d'un volume qui gèrent les fichiers cachés appelés aussi fichiers invisibles. Si ce bit est à 1, la routine de catalogue ne doit pas faire apparaître ce fichier dans le listing.

ProDOS 8 et GS/OS mettent automatiquement à 1 le bit de Backup (B) dès qu'il écrivent une data dans un fichier. Cela donne la possibilité d'écrire des programmes de copie qui ne recopient que les fichiers ayant été modifiés. C'est aux programmes de copie en question de remettre le bit B à 0 dès que la copie du fichier a été faite.

5.1.17 Organisation des Datas d'un Fichier : Structures de fichier

Le travail principal de ProDOS et de GS/OS est de connaître les numéros des blocs qui constituent un fichier. Le système d'exploitation a une technique d'index pour retrouver ces blocs. En général (pour les fichiers ayant un nombre de blocs de données compris entre 2 et 256), le pointeur sur le Bloc clé dans l'entrée du fichier au catalogue (octets relatifs \$11 et \$12) pointe sur un bloc d'index contenant une liste ordonnée des numéros de tous les blocs que le fichier utilise pour ranger ses données. L'avantage principal de cette technique d'index, c'est qu'un fichier peut occuper sur un volume une série de blocs qui ne sont pas nécessairement consécutifs (par exemple le système d'exploitation Pascal oblige l'utilisation de blocs consécutifs pour le rangement des données). Cela signifie que l'on ne perd pas d'espace sur le volume disque. L'inconvénient est que les opérations disque I/O sont plus lentes car il est nécessaire de positionner la tête de lecture/ écriture sur tous les blocs dispersés sur le volume d'un fichier fragmenté. Pour défragmenter un fichier il existe l'utilitaire Beach Comber de Glen Bredon (sur un disque dur cela peut faire gagner énormément de temps).

Il existe 3 types de structures qu'un fichier peut adopter, en fonction de sa taille:

- Fichier "Seedling": 1 à 512 octets (1 seul bloc de données, ce qui permet de stocker un maximum de 512 octets)

- Structure "Sapling": Pour les fichiers dont la taille est comprise entre 513 et 131072 octets (entre 2 et 256 blocs de données, soit un maximum de 128 Ko)
- Structure "Tree": Utilisé lorsque la taille du fichier est comprise entre 131073 et 16777215 octets (au maximum 32768 blocs de données, soit un maximum de 16 Mo)

Pourquoi trois structures différentes? Le but est de limiter au maximum la consommation inutile d'espace disque. Les fichiers qui utilisent une structure "Sapling" ou "Tree", comme nous le verrons plus tard, ont besoin d'utiliser un ou plusieurs blocs d'index pour que le système d'exploitation puisse trouver où sont enregistrées les données. En utilisant une structure plus simple pour les fichiers plus petits, on peut utiliser au mieux l'espace disponible.

On peut déterminer la technique d'index utilisée par un fichier (autre qu'un sous-catalogue) en examinant les 4 bits de poids fort du code de type d'enregistrement stockés dans l'octet relatif \$00 dans une entrée de fichier au catalogue. Ce nombre est égal à \$1 pour un fichier "Seedling", \$2 pour un fichier "Sapling", et \$3 pour un fichier "Tree". Si ce nombre est égal à \$0 le fichier a été effacé. Les fichiers sous-catalogues utilisent les codes \$D, \$E, ou \$F; \$D identifie une entrée au catalogue pour un fichier sous-catalogue, \$E un sous-catalogue, et \$F le catalogue du volume. Le code \$4 identifie une partition Pascal et le code \$5 identifie un fichier de type étendu.

Le pointeur clé d'un fichier (octets relatifs \$11 et \$12) pointe sur un bloc d'index (aussi appelé Bloc clé). Nous allons maintenant examiner comment le système d'exploitation utilise le bloc d'index pour chacun de ces trois types de fichier.

Fichier "Seedling" - fichier de 1 à 512 octets (1 bloc). Ce type de fichier utilise un bloc unique pour ses datas; c'est le numéro de bloc indiqué par le pointeur clé (octets relatifs \$11 et \$12).

Par exemple, créons le programme Basic Applesoft suivant:

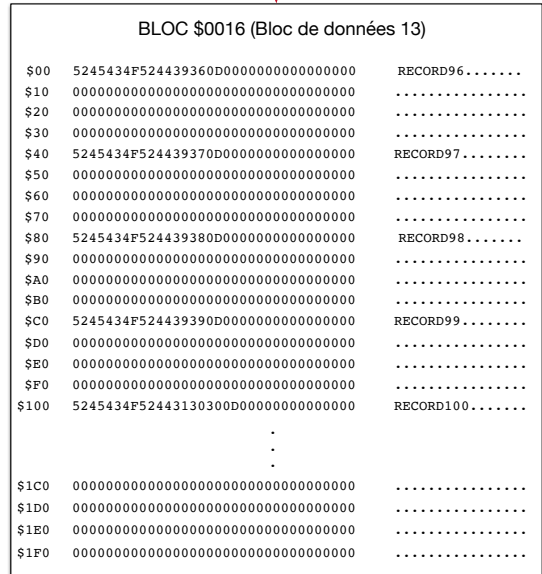
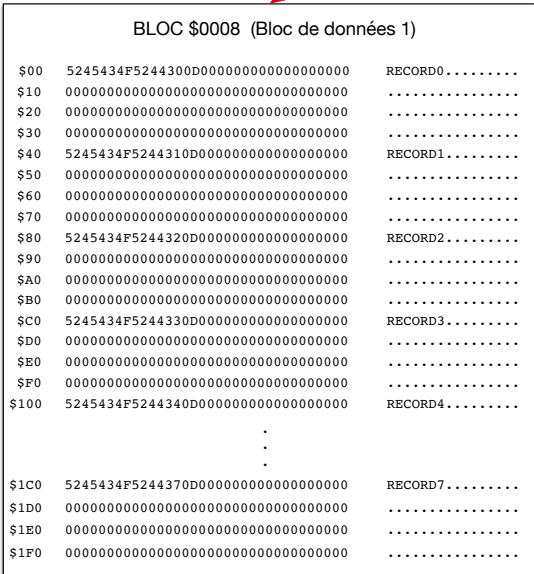
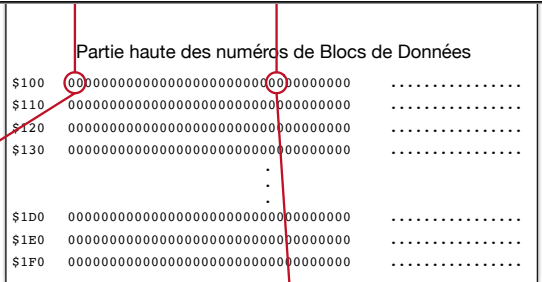
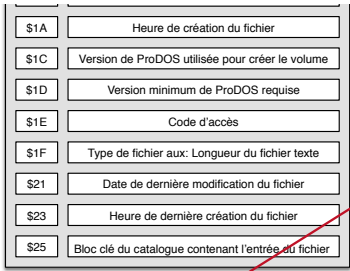
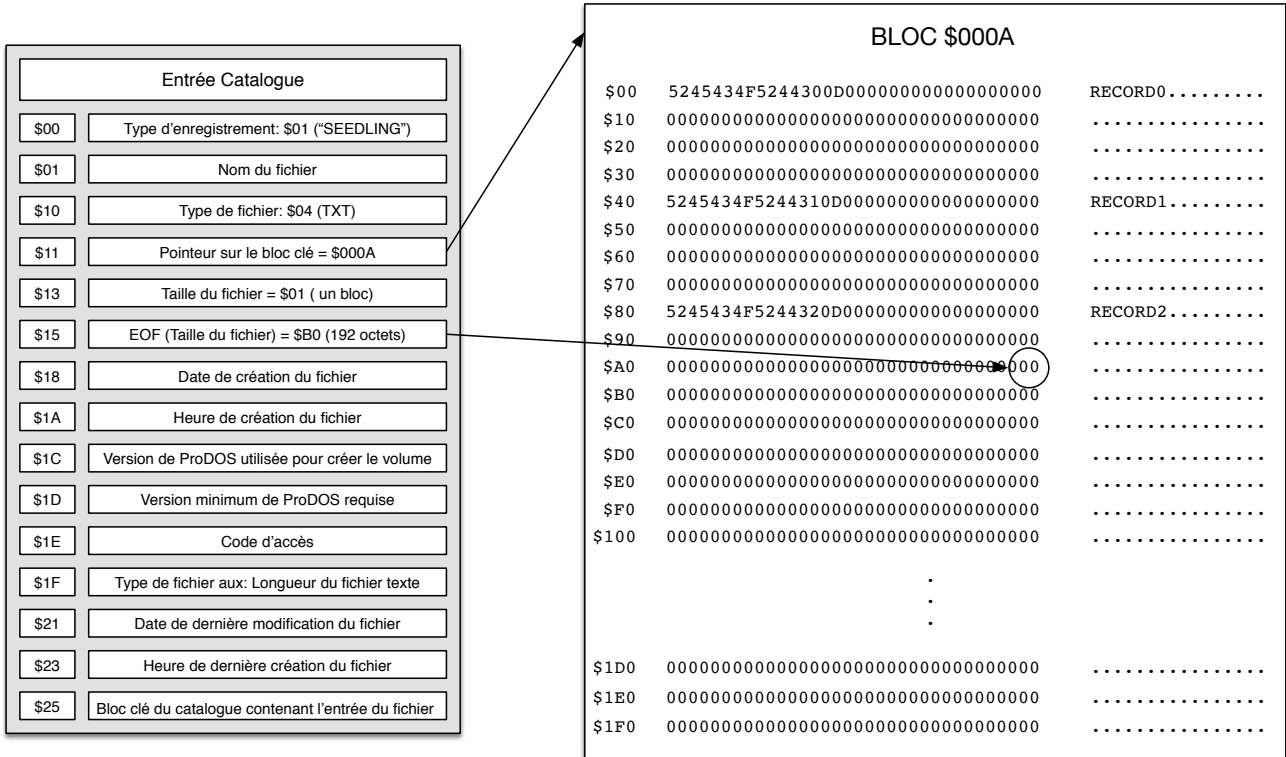
```

10 PRINT CHR$(4);"OPEN TEXTILE,L64"
20 FOR I=0 TO 2
30 PRINT CHR$(4);"WRITE TEXTILE,R";I
40 PRINT "RECORD";I
50 NEXT I
60 PRINT CHR$(4);"CLOSE TXTFILE"
70 END

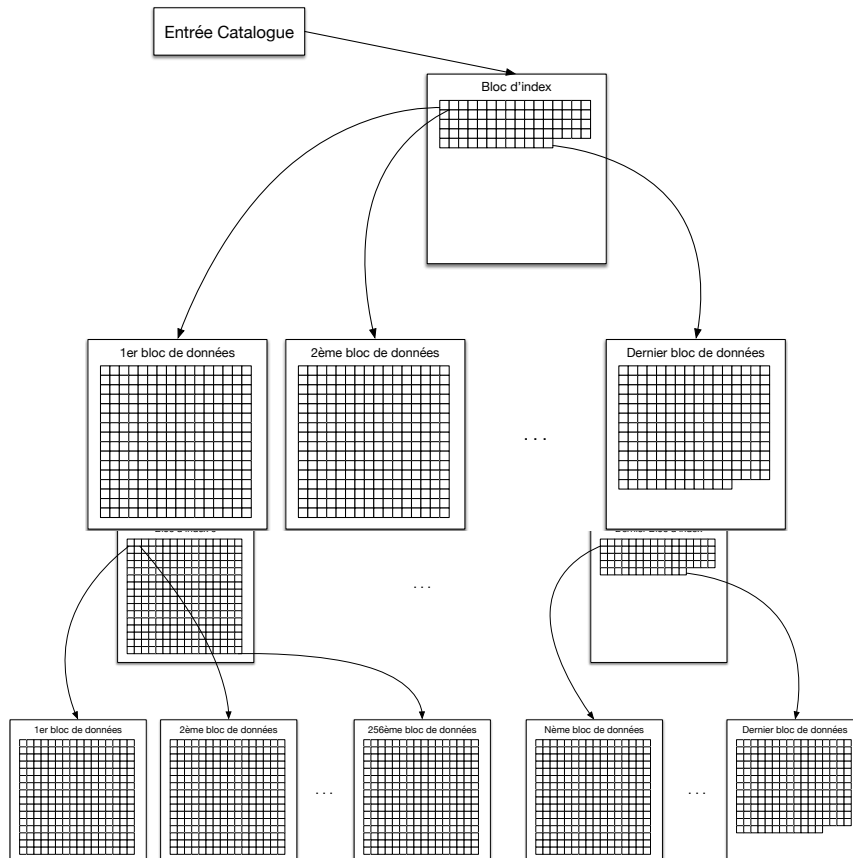
```

Ce petit programme crée un fichier texte appelé «TXTFILE» avec des enregistrements d'une longueur de 64 octets. Il écrit ensuite dans ce fichier 3 enregistrements contenant

les chaînes de caractères "RECORD0", "RECORD1", et "RECORD2". La taille totale de ce fichier est de 3 fois 64 octets soit 192 octets. Ce nombre étant inférieur à 512 octets, ce fichier sera sauvegardé sous forme de fichier "Seedling".



Fichier "Sapling" - Le pointeur clé d'un fichier "Sapling" pointe sur un numéro de bloc qu'on appelle le bloc d'index. Ce bloc d'index contient une liste ordonnée des numéros de blocs utilisés pour le rangement des données de ce fichier. Deux octets sont nécessaires pour stocker chaque numéro de bloc (un volume pouvant contenir au maximum 65536 blocs, soit 32Mo). La partie basse du numéro de bloc est toujours stockée dans la première partie de ce bloc d'index; la partie haute du numéro de bloc est stockée dans la deuxième partie du bloc d'index. La taille maximum d'un fichier "Sapling" est de 128 Ko; il ne peut pas être plus long car un bloc d'index ne peut référencer au maximum que 256 blocs.



Reprenons notre exemple de tout à l'heure, et modifions la ligne 20 par:

```
20 FOR I=0 TO 100
```

Relançons le programme par l'instruction RUN. Le fichier créé contient maintenant 101 enregistrements de 64 octets chacun; la taille totale du fichier "TXTFILE" est donc de 6464 octets. Quand le neuvième enregistrement est écrit ("RECORD8"), le système d'exploitation (ProDOS ou GS/OS) se rend compte que le bloc "Seedling" original est plein. Le système d'exploitation va alors créer ce que l'on appelle un bloc d'index. Ce bloc contient tous les numéros de blocs de chaque blocs de données pour le fichier "TXTFILE" dans l'ordre dans lequel le système d'exploitation va y accéder. En utilisant un bloc d'index, le système d'exploitation peut accéder à un fichier dans un ordre séquentiel même si ses blocs de datas ne sont pas physiquement contigus (l'un après l'autre sur le disque).

Ainsi, dans notre exemple, un nouveau bloc est attribué pour le bloc d'index (par exemple le bloc \$000A; on suppose que les datas du fichier de type "Seedling" étaient stockées en bloc \$0008). Les blocs \$000B à \$0016 sont utilisés pour la sauvegarde des datas suivantes du fichier "TXTFILE". Ce sont ces numéros de blocs qui sont placés dans le bloc d'index (\$000A). Bien entendu, l'entrée au catalogue pour le fichier "TXTFILE" est mise à jour pour que le bloc de pointe sur le bloc d'Index (octets relatifs \$11-\$12); le fichier est maintenant identifié comme étant "Sapling" (4 bits de poids fort de l'octet relatif \$00). Dans le bloc d'Index, les numéros de blocs pointés sont stockés en deux parties : le poids faible des numéros de blocs sont dans la première partie du bloc d'index (de l'octet relatif \$000 à \$0FF); le poids fort de ces mêmes numéros sont dans la deuxième partie du bloc d'index (de l'octet relatif \$100 à \$1FF).

Fichier "Tree" - Pour un fichier de type "Tree", le Pointeur Clé pointe sur le numéro de bloc d'un bloc d'index maître, qui contient une liste ordonnée de tous les numéros de blocs d'index. La taille maximum d'un fichier "Tree" est de 16 MO.

Supposons maintenant que nous modifions encore une fois notre programme Basic pour que 2144 enregistrements soient créés (ligne 20 du programme). Cela va donner une taille de fichier de 137216 octets, bien plus que ce que peut décrire un bloc d'index unique. Le système d'exploitation doit donc "promouvoir" notre fichier "TXTFILE" au niveau de hiérarchie suivant, c'est à dire un fichier de type "Tree". Un fichier "Tree" consiste en un bloc d'index maître unique référencé dans l'Entrée au Catalogue par le Pointeur Clé. Ce bloc d'index maître contient les numéros de blocs des blocs d'index. Ces blocs d'index décrivent comme auparavant les numéros des blocs de datas qui constituent notre fichier. Un bloc d'index maître peut décrire 256 blocs d'index, chacun d'entre eux pouvant décrire 256 blocs de données.

En principe, la taille maximum d'un fichier devrait être de 256×256 blocs de données, soit un maximum de 65536 blocs de données par fichier ou 32MO. Cependant, Apple a décidé que les numéros de blocs devaient être des entiers positifs et les a limité au rang 0-32767. Un fichier ne peut donc pas avoir une taille supérieure à 16MO. Autre conséquence de cette décision assez arbitraire, un bloc d'index maître ne sera jamais complètement plein.

Il faut remarquer ici que le bloc d'index original du fichier "Sapling" (bloc \$000A) est devenu le premier bloc d'index du fichier "Tree". Au moment du changement de type de fichier, le bloc d'Index maître a été attribué en premier (\$010A), puis le second bloc d'index (\$010B), et enfin le premier bloc de données du second bloc d'index (\$010C). Le dernier bloc utilisé par notre fichier "TXTFILE" est pour les enregistrements "RECORD2136" à "RECORD2143" (pour un total de 2144 enregistrements).

Finalement, il faut aussi signaler que le programme d'exemple a été exécuté sur un volume qui était à l'origine totalement vide. C'est la raison pour laquelle dans le bloc

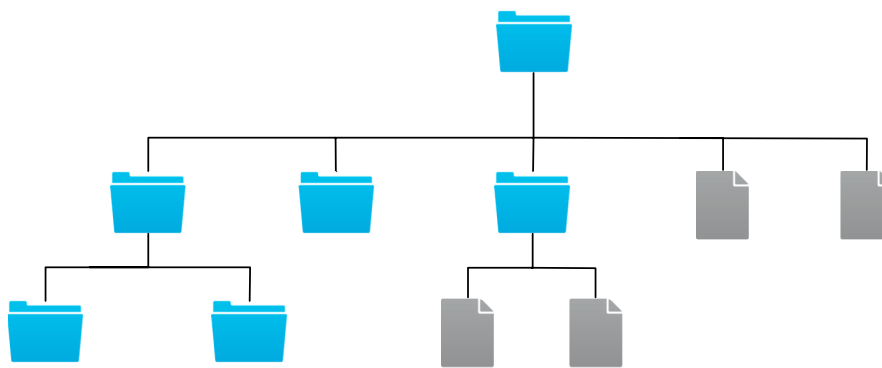
BLOC \$0008 - Bloc de données 0			BLOC \$0117 - Bloc de données 267		
\$00	5245434F5244300D0000000000000000	RECORD0.....	\$00	5245434F5244323133360D0000000000	RECORD2136.....
\$10	00000000000000000000000000000000	\$10	00000000000000000000000000000000
\$20	00000000000000000000000000000000	\$20	00000000000000000000000000000000
\$30	00000000000000000000000000000000	\$30	00000000000000000000000000000000
\$40	5245434F5244310D0000000000000000	RECORD1.....	\$40	5245434F5244323133370D0000000000	RECORD2137.....
\$50	00000000000000000000000000000000	\$50	00000000000000000000000000000000
\$60	00000000000000000000000000000000	\$60	00000000000000000000000000000000
\$70	00000000000000000000000000000000	\$70	00000000000000000000000000000000
\$80	5245434F5244320D0000000000000000	RECORD2.....	\$80	5245434F5244323133380D0000000000	RECORD2138.....
\$90	00000000000000000000000000000000	\$90	00000000000000000000000000000000
\$A0	00000000000000000000000000000000	\$A0	00000000000000000000000000000000
\$B0	00000000000000000000000000000000	\$B0	00000000000000000000000000000000
\$C0	5245434F5244330D0000000000000000	RECORD3.....	\$C0	5245434F5244323133390D0000000000	RECORD2139.....
\$D0	00000000000000000000000000000000	\$D0	00000000000000000000000000000000
\$E0	00000000000000000000000000000000	\$E0	00000000000000000000000000000000
\$F0	00000000000000000000000000000000	\$F0	00000000000000000000000000000000
\$100	5245434F5244340D0000000000000000	RECORD4.....	\$100	5245434F524432313330D00000000000	RECORD2140.....
\$110	00000000000000000000000000000000	\$110	00000000000000000000000000000000
	.			.	
	.			.	
\$1D0	00000000000000000000000000000000	\$1D0	00000000000000000000000000000000
\$1E0	00000000000000000000000000000000	\$1E0	00000000000000000000000000000000
\$1F0	00000000000000000000000000000000	\$1F0	00000000000000000000000000000000
	.			.	
	.			.	
\$1F0	00000000000000000000000000000000			

d'index 0, tous les numéros (ou presque) sont consécutifs. Dans un disque que l'on a déjà utilisé et où l'on a créé et effacé des fichiers (c'est à dire que le volume est fragmenté), il est rare de voir qu'un fichier soit placé sur des blocs consécutifs. Sur un disque fragmenté, le bras du disque doit sauter souvent d'une piste à l'autre pour lire un fichier et cela cause que les opérations d'E/S soient plus lentes. C'est pourquoi, si vous disposez d'un disque dur, il est important de le défragmenter de temps en temps pour conserver une vitesse optimale.

5.1.18 Les Sous-Catalogues ou fichiers de type DIR

Nous avons vu que le catalogue principal d'un volume ProDOS ou GS/OS ne peut pas contenir plus de 51 entrées. Sans l'utilisation de sous-catalogues cette limite ne pourrait pas être dépassée. Il faut considérer un sous-catalogue comme une extension du catalogue principal du volume. Dans le cas le plus simple, un sous-catalogue est créé et est considéré comme une entrée placée dans le catalogue principal du volume. Le sous-catalogue a une structure très similaire au catalogue principal: il a une entête placée au début, ses blocs sont doublement référencés par des pointeurs sur les 4 premiers octets de chaque bloc qui le constitue, et il peut contenir des entrées pour d'autres fichiers (ainsi que des sous-catalogues). Contrairement au catalogue principal du volume, il peut avoir une longueur quelconque (au moment de la création il est constitué par un bloc unique auquel on ajoute d'autres blocs quand la taille du sous-catalogue augmente), son entête a une structure légèrement différente que l'entête du catalogue principal du volume, il peut être situé n'importe où sur le Volume, et enfin ses blocs ne sont pas nécessairement contigus.

Voici un diagramme d'une structure classique de sous-catalogues:



On peut voir qu'avec un seul sous-catalogue on peut créer autant de fichiers que l'on a de blocs disponibles sur le Volume. En principe, il est préférable de créer une arborescence de sous-catalogues, chaque sous-catalogue renfermant un certain genre de fichier; fichiers tableur, dessins, programmes personnels, etc

On peut considérer ces sous-catalogues comme de mini volumes l'intérieur d'un volume d'une taille beaucoup plus importante.

Voici la description de la structure de l'entête d'un sous catalogue; les 4 premiers octets de chaque bloc de sous-catalogue pointent pour les deux premiers octets (octets \$00-\$01) sur le bloc précédent de ce sous- catalogue, les octets \$02-\$03 pointent sur le bloc suivant de ce sous- catalogue.

Entête d'un sous-catalogue

Index	Description
\$04	4 Bits de poids fort: Type d'enregistrement \$E = Sous-catalogue \$F = Catalogue volume 4 Bits de poids faible: Longueur du nom du volume.
\$05-\$13	Nom du sous-catalogue: 15 octets (ASCII positif). Les octets après la fin de la chaîne de caractères sont ignorés
\$14	Doit contenir \$75
\$15-\$1B	Réservé pour une utilisation future. En général cet espace est rempli avec des \$00.
\$1C-\$1D	Date de création du catalogue (formatage) YYYYYYYMMMMDDDD (Y=Année, M=Mois, D=Jour) en forme binaire
\$1E-\$1F	Minute et heure de création du catalogue 000HHHHH00MMMMMM (H=Heure, M=Minute) en forme binaire
\$20	Version ProDOS lors de la création de ce sous-catalogue

Index	Description
\$21	Version minimum de ProDOS pour ce fichier
\$22	Code d'accès pour ce fichier \$80 Le fichier peut être détruit \$40 Le fichier peut être renommé \$20 Fichier modifié depuis de dernier backup \$02 Autorisation d'écrire dans le fichier \$01 Autorisation de lecture du contenu du fichier En général le code d'accès d'un fichier non verrouillé est \$C3 (\$80 + \$40 +\$02 +\$01) et celui d'un fichier verrouillé vaut \$01
\$23	Le nombre d'octets utilisés par chaque entrée dans le sous-catalogue (\$27)
\$24	Le nombre d'entrées sur chaque bloc d'un sous-catalogue (\$0D). L'entête du sous-catalogue est considérée comme une entrée.
\$25-\$26	Nombre d'entrées actives dans les sous-catalogue, sans compter l'entête du sous-catalogue. Une entrée active décrit un fichier ou un sous-catalogue non-effacé.
\$27-\$28	Dans un sous-catalogue, c'est le pointeur indiquant le bloc définissant l'entrée de ce sous-catalogue dans le volume principal ou un sous-catalogue.
\$29	Numéro de l'entrée fichier dans le numéro de bloc pointé par les octets \$27-\$28. Ce nombre est compris entre \$01 et \$0D.
\$2A	Longueur des entrées dans le catalogue père. En général \$27 (nombre d'octets)

5.1.19 Fichiers Étendus

GS/OS (mais pas ProDOS 8) peut créer des fichiers de type étendu. Ces fichiers ont un code pour le type d'enregistrement de \$5. Un fichier étendu comprend deux segments de données, le segment de ressource et le segment de données. Le segment de données contient en général les données spécifiques de l'application elle même. Le segment de ressource contient en général les structures de données définissables; ces structures de données définissent des éléments comme par exemple les définitions de menu, les boites de dialogue, etc...

Le bloc clé pour un fichier étendu n'en est pas vraiment un; c'est simplement une extension de l'entrée du fichier au catalogue. La première moitié du bloc contient les informations du segment de données; la deuxième moitié contient les informations relatives au segment de ressource. GS/OS utilise seulement les huit premiers octets de chaque moitié de bloc. Voici la signification de chacun de ces octets:

- \$00 Code du type d'enregistrement pour le segment
- \$01-\$02 Numéro du bloc clé pour le segment
- \$03-\$04 Taille du segment (nombre de blocs)
- \$05-\$07 Taille du segment (nombre d'octets)

Le code du type d'enregistrement pour le segment est soit \$01 (Seedling), \$02 (Sapling), ou \$03 (Tree). Le bloc clé pour le segment d'un fichier étendu (octets \$01-\$02) est organisé de la même façon que un fichier de type classique.

5.1.20 Les fichiers Sparse

Il est possible de créer des fichiers qui ne sont pas séquentiels. Cela signifie qu'ils sont constitués d'une série d'enregistrements (dont on peut choisir la taille); on peut lire ou écrire sur n'importe lequel de ces enregistrements sans avoir à se positionner sur les précédents. Cela signifie que l'on peut écrire dans un enregistrement même si celui-ci est séparé de l'enregistrement écrit précédemment par un grand nombre d'enregistrements vides (remplis de 00). Bien entendu, le système d'exploitation ne sauvegarde pas les enregistrements vides car cela prendrait énormément de place sur le volume. En fait, dans le bloc d'index il place des \$0000 pour indiquer les blocs de datas qui ne sont pas utilisés; il le seront quand ces enregistrements seront écrits.

Ce genre de fichier est appelé Sparse; il n'occupe pas sur le Volume autant de place que sa taille l'indique.

Créons un fichier Sparse avec le programme en BASIC AppleSoft qui suit. On suppose que la longueur de chaque enregistrement est de 128 octets, et que l'on écrit seulement dans les enregistrements 2 et 64.

```
10 F$ = "RANDOM"
30 PRINT CHR$(4);"OPEN ";F$;" ,L128"
40 PRINT CHR$(4);"WRITE ";F$;" ,R2"
50 PRINT "RECORD 2"
60 PRINT CHR$(4);"WRITE ";F$;" ,R64"
70 PRINT "RECORD 64"
80 PRINT CHR$(4);"CLOSE"
```

Le bloc d'index pour le fichier créé par ce programme est le suivant:

0000 8C 00 00 00 00 00 00 00	Cela indique que les Blocs 0 et 16 de
0008 00 00 00 00 00 00 00 00	données sont rangés dans les bloc \$008C
0010 8E 00 00 00 00 00 00 00	et \$008E sur le Volume
...	
0100 00 00 00 00 00 00 00 00	
0108 00 00 00 00 00 00 00 00	
0110 00 00 00 00 00 00 00 00	

...

01F8 00 00 00 00 00 00 00 00

Bloc de données 0 (bloc \$008C du Volume)

0000 00 00 00 00 00 00 00 00

...

0100 52 45 43 4F 52 44 20 32 RECORD 2

0108 0D

0109 00 00 00 00 00 00 00 00

...

01F8 00 00 00 00 00 00 00 00

L'enregistrement 2 (RECORD 2) est stocké à la position \$100 (2 X 128) dans le fichier; cela correspond à la position \$100 dans le premier bloc attribué au fichier (entrée 0 du bloc d'index pointant sur le bloc \$008C)

Bloc de données 16 (bloc \$008E du Volume)

0000 52 45 43 4F 52 44 20 36 RECORD 6

0008 34 4

0009 0D

000A 00 00 00 00 00 00 00 00

...

01F8 00 00 00 00 00 00 00 00

L'enregistrement 64 ("RECORD 64") commence à la position \$2000 dans le fichier (64 X 128); cela correspond à la position \$0000 de la 16 ème entrée du bloc d'Index qui pointe sur le bloc de datas \$008E. Les 15 blocs inutilisés entre ces deux enregistrements sont indiqués par des \$0000 dans les entrées du bloc d'index.

Ainsi, même si logiquement ce fichier a une longueur de 17 blocs, le système d'exploitation en utilise seulement 3 pour le sauvegarder sur le volume (1 pour le bloc d'index, et 2 pour les blocs de datas).

5.2 Les commandes de GS/OS et de ProDOS 8

GS/OS et ProDOS 8 ont tous les deux un interpréteur de commandes de bas niveau.

L'interpréteur de commandes de ProDOS 8 s'appelle le MLI (Machine Language Interface). Le MLI reconnaît 26 commandes.

GS/OS reconnaît 47 commandes.

On peut appeler ces différentes commandes à partir d'un programme en langage machine à l'aide d'une technique générale utilisant des protocoles d'appel définis par Apple.

Les protocoles pour ProDOS 8 et GS/OS sont de structures similaires mais pas identiques.

Dans cette partie, nous allons examiner de très près les commandes de GS/OS et de ProDOS 8, et voir comment on les utilise dans les programmes en langage machine.

5.2.1 Utilisation des commandes MLI de ProDOS 8

Il est très facile d'exécuter une commande MLI sous de ProDOS 8. Une séquence typique d'appel ressemble à quelque chose comme cela:

```
...                ; Code pour initialiser le contenu de la
...                ; table PARMTNL
JSR  $BF00         ; Point d'entrée MLI
DFB  CMDNUM        ; Numéro de commande MLI
DA   PARMTBL       ; Adresse de la table des paramètres
BCS  Error         ; Retenue = 1 en cas d'erreur
...
RTS
Error ...          ; Routine de traitement d'erreurs
RTS
PARMTBL           DFB  NPARMS      ; NPARMS représente le nombre d'octets utilisés
                                   ; par PARMTBL. Le nombre varie en fonction de la
                                   ; commande ProDOS que l'on va invoquer.
```

On place dans cette table (PARMTBL) les paramètres dans l'ordre attendu par la commande MLI.

L'instruction clé est le JSR \$BF00.

\$BF00 est l'adresse du point d'entrée du MLI en mémoire principale. L'interpréteur MLI détermine à quelle commande l'application a fait appel et passe alors le contrôle à la sous routine ProDOS 8 correspondante pour traiter cette commande.

Dès que le MLI prend le contrôle, il modifie 4 variables importantes dans la page globale ProDOS 8 : MLIACTV (\$BF9B), CMDADR (\$BF9C-\$BF9D), SAVEX (\$BF9E), et SAVEY (\$BF9F).

Le MLI modifie le bit 7 de MLIACT; il passe de 0 à 1 afin qu'une sous routine de gestion d'interruption puisse déterminer si une condition d'interruption survient pendant le déroulement d'une opération MLI.

Le MLI sauvegarde ensuite la valeurs des registres d'index X et Y dans SAVEX et SAVEY.

Enfin, le MLI range l'adresse de l'instruction suivant immédiatement les 3 octets après l'instruction JSR \$BF00 en CMDADR. Le contrôle passera à cette adresse après l'exécution de la commande MLI.

Le MLI détermine quelle commande il doit traiter en examinant la valeur rangée dans l'octet qui suit immédiatement l'instruction JSR \$BF00. Cette valeur est un code unique identifiant un numéro de commande.

Si le MLI rencontre un numéro de commande inconnu, une erreur système survient.

Les deux octets suivants le numéro de commande contiennent l'adresse (poids faible poids fort) de la table des paramètres que la commande MLI va utiliser.

Cette table débute par un octet qui indique le nombre de paramètres contenus dans cette table. Le reste de la table contient des données que le MLI va utiliser pour exécuter la commande.

Après l'exécution de la commande MLI, cette table de paramètres contient les résultats retournés par le MLI.

Nous décrivons plus loin les contenus de la table des paramètres pour chacune des commandes MLI.

Les paramètres qu'une application passe au MLI sont de deux types: pointeurs ou valeurs.

- Un pointeur est une valeur sur deux octets qui pointe sur l'adresse d'une structure de données, par exemple une table de datas (poids faible - poids fort).

- Une valeur sur 1, 2, ou 3 octets est une quantité numérique ayant une signification binaire (octet de poids faible en premier).

Les paramètres retournés par le MLI sont appelés résultats.

Un résultat est en général une quantité numérique sur 1, 2, ou 3 octets (octet de poids faible en premier), mais il peut s'agir aussi d'un pointeur sur 2 octets; tout dépend de la commande MLI appelée.

Si le nombre au début de la table de paramètres ne correspond pas au nombre de paramètres attendus par la commande MLI, une erreur système survient. Autrement, le MLI exécute la commande.

Pendant l'exécution d'une commande, une erreur critique peut survenir.

Les erreurs critiques sont très rares et surviennent seulement quand des zones de données ProDOS ont été effacées par un programme, ou si une interruption est requise et qu'il n'y a pas de programme d'interruption pour la gérer.

Avec ce genre d'erreur, on est obligé de rebooter tout le système. Dans ce cas, le MLI exécute l'instruction JSR \$BF0C.

La routine en \$BF0C (SYSDEATH) fait apparaître le message suivant:

```
INSERT SYSTEM DISK AND RESTART - ERR xx
```

où xx est un code d'erreur hexadécimal ayant 4 valeurs possibles :

\$01 = Erreur Interruption

\$0A = Bloc Contrôle volume endommagé

\$0B = Bloc Contrôle Fichier endommagé

\$0C = Bloc d'Allocation endommagé

Il s'agit de structures de datas internes que ProDOS 8 utilise pour manipuler les volumes disque et pour ouvrir les fichiers.

Le MLI, une fois la commande traitée récupère les valeurs des registres X et Y (l'application n'a donc pas besoin de les sauvegarder). En cas d'erreur, le MLI exécute l'instruction \$BF09. Cette sous routine (SYSERR) range un code d'erreur en \$BF0F (SERR).

Finalement, le contrôle est passé à l'instruction qui suit immédiatement le pointeur sur la Table des Paramètres (BCS ERROR dans notre exemple).

Cette adresse a été rangé par le MLI en \$BF9C - \$BF9D juste au début du traitement.

5.2.2 Utilisation des commandes GS/OS

La procédure générale d'appel d'une commande GS/OS est similaire à l'appel d'une commande MLI ProDOS 8.

	JSL	\$E100A8	; Point d'entrée GS/OS
	DA	COMMAND	; Numéro de commande GS/OS
NUM	ADRL	PARMTABLE	; Adresse de la table des paramètres
	BCS	Error	; Retenue = 1 en cas d'erreur

\$E100A8 est l'adresse du point d'entrée de l'interpréteur de commandes de GS/OS. Vous pouvez appeler ce point d'entrée soit en mode natif, soit en mode émulation (voir la partie consacrée au micro-processeur).

Immédiatement après l'instruction JSL \$E100A8, il y a le Mot (2 octets, en anglais Word) contenant le numéro d'indentification de la commande GS/OS que vous voulez utiliser.

A la suite du numéro de commande, on trouve l'adresse longue (4 octets; octets de poids faible en premier) de la Table des Paramètres nécessaires au fonctionnement de la commande et au rangement des résultats éventuels.

Les paramètres peuvent être des valeurs numériques sur 1 ou 2 Mots, ou des pointeurs longs (4 octets). Ils sont rangés dans l'ordre poids faible - poids fort.

La structure exacte de la Table des Paramètres varie d'une commande à une autre, mais elle débute toujours par un nombre indiquant le nombre de paramètres dans la table; ce paramètre s'appelle PCOUNT.

Lorsqu'il a terminé avec le traitement d'une commande, GS/OS ajoute 6 à l'adresse de retour placée dans la pile par l'instruction JSL puis revient avec une instruction RTL. Cela permet de passer le contrôle au code se trouvant après le pointeur sur la Table des Paramètres (BCS ERROR dans notre exemple).

Au retour, le contenu de tous les registres reste inchangé sauf l'Accumulateur (qui contient un code d'erreur), le Program Counter (PC), et le registre d'état P (les indicateurs m, x, D, I et e sont inchangés, N et V sont indéfinis; C et Z indiquent s'il y a eu une erreur).

A partir de là, on peut vérifier l'état de C pour déterminer s'il y a une erreur :

Si C=0, il n'y a pas d'erreur

Si C = 1, il y a eu une erreur

On peut aussi examiner l'état de Z. En cas d'erreur, Z = 0.

Le code d'erreur indiquant la nature de cette erreur se trouve dans l'Accumulateur.

S'il n'y a pas d'erreur, l'Accumulateur contient \$00.

5.2.3 Les erreurs sous GS/OS et ProDOS 8

Une erreur qui n'est pas une erreur critique est appelée une erreur système.

Ces erreurs peuvent avoir diverses origines :

Un pathname non valide, une tentative d'écriture sur un disque protégé, une ouverture d'un fichier inexistant, etc...

Si aucune erreur n'est survenue pendant l'exécution d'une commande, $A = 0$, $C = 0$, et $Z = 1$.

En cas d'erreur, l'Accumulateur contient le code de cette erreur, $C = 1$ et $Z = 0$.

Cela signifie que l'on peut utiliser un BCS ou un BNE pour se brancher sur une sous routine de traitement des erreurs.

On doit toujours vérifier si une erreur est survenue après le traitement d'une commande ProDOS 8 ou GS/OS. Si on ne le fait pas, on obtient dans la plupart des cas un programme qui ne fonctionne pas toujours parfaitement. Pensez par exemple aux conséquences d'une commande d'écriture dans un fichier qui n'a pas pu être ouvert parce qu'il n'existe pas.

Voici la liste des codes d'erreur :

Code	Message d'erreur
\$00	Pas d'erreur
\$01	Numéro de commande non valide.
\$04	Nombre de Paramètres spécifié dans la Table des Paramètres non valide.
\$07	Une commande GS/OS a été demandé par une routine d'interruption.
\$10	Un dispositif (Device) spécifié n'a pu être trouvé. GS/OS retourne cette erreur après que la commande GetDevNum n'ait pu localiser un Device.
\$11	Le numéro de référence du dispositif (Device) n'est pas valide. GS/OS donne cette erreur si le numéro de Device n'est pas dans la liste des Devices actifs.
\$22	Paramètre invalide pour un Driver GS/OS.
\$23	Le Driver "CONSOLE DRIVER" n'est pas ouvert.
\$25	La Table des vecteurs d'Interruption de ProDOS 8 est pleine.

Code	Message d'erreur
\$27	Une erreur I/O (Entrée/Sortie) sur disque 5.25" est survenue empêchant le transfert correct des données. Le volume disque est sans aucun doute abîmé. On obtient aussi cette erreur s'il n'y a pas de disque dans le lecteur.
\$28	Le dispositif (Device) disque spécifié n'est pas présent. Cette erreur arrive quand on tente d'accéder à un second lecteur quand il n'y a qu'un lecteur de connecté.
\$2B	Une opération d'écriture a échoué parce que le Volume disque est protégé contre l'écriture.
\$2E	Une commande a échoué parce que le Volume disque contenant un fichier ouvert a été enlevé de son lecteur.
\$2F	Le dispositif (Device) spécifié n'est pas en ligne. Cette erreur se produit s'il n'y a pas de disque dans le lecteur 3.5".
\$40	La syntaxe du pathname n'est pas valide.
\$42	Tentative d'ouverture d'un 9ème fichier. ProDOS 8 n'autorise l'ouverture que de 8 fichiers au maximum.
\$43	Le numéro de référence d'un fichier n'est pas valide. Cela arrive quand un mauvais numéro de référence est spécifié lors de l'ouverture d'un fichier ou lorsqu'un mauvais numéro de référence est spécifié lors de la fermeture d'un fichier.
\$44	Le Pathname spécifié n'a pas été trouvé. Cela signifie que l'un des noms de sous-catalogue, dans un autre pathname valide, n'existe pas.
\$45	Le Volume spécifié n'a pas été trouvé. Cela arrive quand, par exemple, on change le disque du lecteur.
\$46	Le fichier spécifié n'a pas été trouvé. Cela signifie que le nom de fichier dans un Pathname correct n'existe pas.
\$47	Le nom de fichier spécifié existe déjà. Cette erreur arrive quand on tente de renommer ou de créer un fichier, et que le nom utilisé existe déjà.
\$48	Le Volume disque est plein. Il n'y a plus de blocs disponibles sur le Volume disque pour la sauvegarde des données.
\$49	Le catalogue principal du Volume est plein. Seulement 51 fichiers peuvent être rangés dans le Catalogue principal.
\$4A	Le format du fichier spécifié est inconnu ou incompatible avec la version du système d'exploitation utilisée.
\$4B	Le type de fichier n'est pas valide ou n'est pas reconnu.
\$4C	La fin du fichier (EOF) a été atteinte durant une opération de lecture.
\$4D	La valeur MARK spécifiée est supérieure à EOF.

Code	Message d'erreur
\$4E	Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier contrôle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.
\$4F	La taille du buffer de classe 1 de sortie de GS/OS est trop petite.
\$50	Commande non valide parce que le fichier est ouvert. Les commandes Open, Rename, et Destroy ne s'appliquent qu'aux fichiers fermés.
\$51	Le nombre de fichiers au catalogue est différent du nombre de fichiers indiqué dans l'entête du catalogue.
\$52	Volume Disque non ProDOS. La structure de catalogue est inconsistante avec ProDOS.
\$53	Paramètre non valide. Un paramètre est invalide quand il est hors des limites fixées par le système d'exploitation.
\$54	Plus de mémoire disponible.
\$55	Cette erreur survient si 8 fichiers sur 8 Devices différents ont été ouverts, et que la commande ONLINE est demandée sur un Device n'ayant pas de fichiers ouverts.
\$56	L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map de ProDOS 8, ou parce que le buffer ne débute par sur un saut de page mémoire.
\$57	Volume disque en ligne ayant le même nom de catalogue principal.
\$58	Le dispositif (Device) spécifié n'est pas organisé en blocs.
\$59	Le niveau de priorité passée par la commande GS/OS SetLevel est hors des limites acceptées par le système.
\$5A	La Bit Map du Volume indique qu'un bloc ayant un numéro supérieur au nombre de blocs disponibles sur le Volume est déclaré libre. La Bit Map du volume a été endommagée.
\$5B	Changement illégal de Pathname. Cette erreur survient si les Pathnames spécifiés dans la commande ChangePath de GS/OS font référence à 2 Volumes différents. On ne peut déplacer des fichiers seulement entre les catalogues d'un même volume.
\$5C	Le fichier spécifié n'est pas un fichier système exécutable. GS/OS donne cette erreur si on tente d'utiliser la commande Quit pour passer le contrôle à un fichier qui n'est pas un fichier système GS/OS (S16 ou \$B3; EXE ou \$B5), ou un fichier système ProDOS 8 (SYS ou \$FF).
\$5D	Le système d'exploitation spécifié n'est pas disponible. GS/OS retourne cette erreur quand on tente d'exécuter un programme système ProDOS 8 quand le fichier /SYSTEM/P8 ne se trouve pas sur le volume système.
\$5E	Le Volume /RAM ne peut pas être supprimé.
\$5F	La pile des adresses de retour des Quit est saturée. GS/OS retourne cette erreur quand on tente de pousser un autre ID d'un programme sur cette pile quand celle-ci est pleine.

Code	Message d'erreur
\$61	Fin de Catalogue. Cette erreur ne peut être retournée que par la commande GS/OS GetDirEntry.
\$62	Numéro de classe non valide.
\$64	ID invalide pour un fichier Système.
\$65	Opération FST invalide.

5.2.4 Buffers de sortie de classe 0 et de classe 1

Même si un pointeur sur une chaîne ou sur un buffer est placé dans une Table de Paramètres, ce n'est jamais ProDOS 8 ou GS/OS qui a placé ce pointeur. Le système d'exploitation se contente de retourner des données dans le buffer pointé par ce pointeur.

Sous ProDos 8, c'est l'application qui doit s'allouer un buffer d'une taille convenable et qui doit passer le pointeur sur l'adresse de début de ce buffer dans la Table des Paramètres. Si vous ne créez pas un buffer d'une taille suffisante, les données qui suivent ce buffer seront écrasées. Un buffer de ce genre est appelé un buffer de sortie de classe 0.

GS/OS utilise des buffers de sortie de classe 1 pour éviter l'écrasement de données pour le cas où le buffer de sortie n'aurait pas une taille suffisante. Un buffer de classe 1 débute par un mot (2 octets) indiquant le nombre d'octets de ce buffer (en y incluant ce mot). Quand on appelle une commande qui utilise un buffer de sortie de classe 1, GS/OS vérifie le mot de longueur placé en début du buffer pour voir si la taille du buffer est suffisamment importante. Si cette taille est trop petite, la commande retourne un code d'erreur \$4F (buffer trop petit) et retourne la taille du buffer nécessaire dans le mot suivant le mot de longueur du buffer. Si la taille du buffer est suffisante, la commande envoie les données dans le buffer juste après le mot contenant la longueur du buffer.

Les commandes de ProDOS 8 par ordre alphabétique

ALLOC_INTERRUPT

Code de fonction ProDOS 8: \$40

Fonction

Placer l'adresse d'une sous-routine de gestion d'interruption dans la table des vecteurs d'interruption de ProDOS 8. La table des vecteurs d'interruption peut contenir jusqu'à 4 sous-routines.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (2)
1	int_num	1	S	Numéro de référence du Handler d'interruption
2	int_code	2	E	Pointeur sur le Handler d'interruption

Description des paramètres

num_parms Nombre de paramètres dans la Table des Paramètres ProDOS 8. Cette valeur vaut toujours 2 pour cette fonction.

int_num C'est le numéro de référence que ProDOS 8 attribue ,à la sous-routine de gestion d'interruption. Il faut utiliser ce nombre quand on utilise la commande DEALLOC_INTERRUPT pour retirer cette sous-routine.

int_code Pointeur sur la sous-routine de prise en charge de l'interruption. ProDOS 8 passe le contrôle à cette sous-routine quand une interruption est demandée. La sous-routine de prise en charge de l'interruption doit débiter par une instruction CLD.

Il est important d'installer la sous-routine de prise en charge de l'interruption avant de permettre au périphérique de générer une interruption. Dans le cas contraire, le système plantera si une interruption est demandée avant l'installation de la sous-routine destinée à la prendre en charge.

Codes d'erreur que peut retourner la fonction

Code	Description
\$25	La Table des Vecteurs d'Interruption est pleine. La solution consiste ,à enlever une sous-routine de prise en charge d'interruption en utilisant la commande DEALLOC_INTERRUPT, puis de re-essayer.

Exemple de programme

Voici comment installer une sous-routine ProDOS 8 de prise en charge d'interruption qui a été chargée en mémoire à l'adresse \$300.

```
                JSR    MLI
                DFB    $40          ; ALLOC_INTERRUPT
                DA     PARMTBL      ; Adresse de la table des paramètres
                BCS    Error        ; Retenue = 1 en cas d'erreur
                RTS
                ;
PARMTBL         DFB    2            ; Nombre de paramètres
                DS     1            ; Espace où ALLOC_INTERRUPT retournera le
                ; numéro de référence
                DA     $300        ; Adresse de la sous-routine d'interruption
```

Votre application devra sauvegarder *int_num* pour pouvoir utiliser par la suite la commande DEALLOC_INTERRUPT.

Commentaires

Sous GS/OS, la fonction équivalente s'appelle BindInt.

CLOSE

Code de fonction ProDOS 8: \$CC

Fonction

Permet de fermer un fichier qui a été ouvert au préalable. Le système d'exploitation écrit le contenu du buffer attribué à ce fichier sur le Volume et met à jour l'entrée au Catalogue de ce fichier. Ensuite, le système d'exploitation libère la mémoire utilisée par le buffer de ce fichier.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (1)
1	ref_num	1	E	Numéro de référence du fichier que l'on veut fermer

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 1 pour cette fonction.

ref_num Numéro de Référence que le système d'exploitation a attribué au fichier quand celui-ci a été ouvert. Si on fixe *ref_num* à 0, tous les fichiers ouverts ayant le même niveau de priorité que le fichier système, ou ayant un niveau de priorité supérieur au fichier système sont fermés. Pour fixer la valeur du niveau de priorité d'un fichier, on stocke cette valeur à l'adresse LEVEL (\$BF94).

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le Volume est protégé en écriture.
\$43	Le Numéro de Référence du fichier n'est pas valide. Vous utilisez très certainement le numéro de Référence d'un fichier ayant déjà été fermé.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$5A

CREATE

Code de fonction ProDOS 8: \$C0

Fonction

Utilisée pour créer un nouveau fichier. Le système d'exploitation réalise cette fonction en plaçant une nouvelle Entrée dans le Catalogue spécifié.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (7)
1	pathname	2	E	Pointeur sur l'adresse du Pathname
3	access	1	E	Code d'Accès
4	file_type	1	E	Code du Type de Fichier
5	aux_type	2	E	Code du Type Auxiliaire de Fichier
7	storage_type	1	E	Code du Type d'Enregistrement
8	create_date	2	E	Date de création
10	create_time	2	E	Heure de création

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 7 pour cette fonction.

pathname Un pointeur sur un buffer de classe 1 sur l'adresse du Pathname du fichier qui va être créé. Si le Pathname spécifié n'est pas précédé par un séparateur ("/" ou ":"), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut (sous GS/OS c'est le préfixe "0/"), pour créer un Pathname complet.

access Voir la partie consacrée à l'Organisation des Fichiers sur Disque pour une explication détaillée du Code d'Accès.

file_type Code indiquant le Type du Fichier à créer. Voir la table avec la liste des codes existants, dans ce même chapitre.

aux_type Code indiquant le Type Auxiliaire du Fichier à créer. La signification de ce code dépend du Type de Fichier. Pour les fichiers SYS, BIN, BAS, et

VAR, le Type Auxiliaire de Fichier contient l'adresse de chargement par défaut de ce fichier; pour les fichiers TXT, ce code est la longueur de chaque enregistrement du fichier texte.

storage_type Ce code indique la façon dont le système d'exploitation a sauvegardé le fichier sur le volume disque; il s'agit du Type d'Enregistrement du fichier:

Type	Description
\$00-\$03	Fichier standard
\$05	Fichier de type étendu
\$0D	Fichier Sous-Catalogue

On ne peut pas changer ce code une fois que le fichier est créé.

create_date Date de création du fichier (Année, mois, jour). Si ces octets sont à 0 la Date actuelle est utilisée.

create_time Heure de création (Heure, Minute). Si ces octets sont à 0 l'Heure actuelle est utilisée.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le Volume est protégé en écriture.
\$40	Le Pathname contient des caractères non valides ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.
\$47	Le nouveau Pathname indiqué existe déjà.
\$48	Le Volume Disque est plein.
\$49	Le Catalogue principal est plein. Seuls 51 fichiers peuvent être sauvegardés dans le catalogue principal.
\$4B	Code du Type d'Enregistrement non valide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$10, \$27, \$52, \$53, \$58

DEALLOC_INTERRUPT

Code de fonction ProDOS 8: \$41

Fonction

Permet d'enlever l'adresse d'une sous-routine de gestion d'interruption dans la table des vecteurs d'interruption de ProDOS 8.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (1)
1	int_num	1	E	Numéro de référence du Handler d'interruption

Description des paramètres

num_parms Nombre de paramètres dans la Table des Paramètres ProDOS 8. Cette valeur vaut toujours 1 pour cette fonction.

int_num Numéro d'identification du Handler d'interruption. ProDOS 8 attribue ce numéro lorsque le Handler est installé avec la commande ALLOC_INTERRUPT.

Il est important de pas enlever la sous-routine de gestion d'interruption avant que l'application ait indiqué au périphérique d'arrêter de générer des interruptions.

Codes d'erreur que peut retourner la fonction

Code	Description
\$53	Le paramètre int_num n'est pas valide. Il faut utiliser le numéro que la commande ALLOC_INTERRUPT a retourné au moment où le Handler d'Interruption a été installé.

Autres codes d'erreur que peut retourner la fonction

\$04

Exemple de programme

Voici comment enlever de la Table des Vecteurs d'interruption une sous-routine de gestion d'interruption dont le numéro de Référence est 1.

```
JSR   MLI
DFB   $41           ; DEALLOC_INTERRUPT
DA    PARMTBL      ; Adresse de la table des paramètres
BCS   Error       ; Retenue = 1 en cas d'erreur
```

RTS

PARMTBL	DFB	1	; Nombre de paramètres
---------	-----	---	------------------------

	DFB	1	; Numéro de code de l'interruption
--	-----	---	------------------------------------

Commentaires

Sous GS/OS, on utilise la commande UnbindInt.

DESTROY

Code de fonction ProDOS 8: \$C1

Fonction

Permet d'éliminer un fichier d'un volume disque. Quand on détruit un fichier, le système d'exploitation libère tous les blocs que le fichier utilisait et met à zéro l'octet de longueur dans l'entrée catalogue qui correspond à ce fichier. On ne peut pas effacer les fichiers dont le bit de code d'accès de destruction est à 1; les fichiers sous-catalogues doivent être vides avant de pouvoir être éliminés.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (1)
1	pathname	2	E	Pointeur sur l'adresse du Pathname

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 1 pour cette fonction.

patname Un pointeur de classe 0 sur l'adresse du Pathname du fichier qui va être détruit. Si le Pathname spécifié n'est pas précédé par un séparateur ("/"), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut pour créer le Pathname complet. Si le Pathname décrit un fichier étendu, les deux segments sont détruits.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le Volume est protégé en écriture.
\$40	Le Pathname contient des caractères non valides, ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.
\$46	Le fichier n'a pas été trouvé.
\$4E	Le système ne peut pas accéder au fichier.
\$50	Le fichier est ouvert; on ne peut détruire que les fichiers fermés.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$10, \$27, \$4A, \$52, \$58

FLUSH

Code de fonction ProDOS 8: \$CD

Fonction

Force le système d'exploitation à écrire le contenu du buffer fichier sur le Volume Disque; le fichier n'est pas fermé.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (1)
1	ref_num	1	E	Numéro de référence du fichier

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 1 pour cette fonction.

ref_num Numéro de Référence attribué au fichier quand celui-ci a été ouvert.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le disque est protégé en écriture.
\$43	Le Numéro de Référence du fichier n'est pas valide. Vous devez utiliser un Numéro de Référence d'un fichier qui a déjà été fermé.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$48

GET_BUF

Code de fonction ProDOS 8: \$D3

Fonction

Permet de déterminer l'adresse de départ d'un buffer de 1024 octets utilisé par un fichier ouvert.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (2)
1	ref_num	1	E	Numéro de référence du fichier
2	io_buffer	2	S	Pointeur sur le buffer

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

ref_num Numéro de Référence attribué au fichier quand celui-ci a été ouvert.

io_buffer Pointeur sur un buffer de 1024 octets utilisé par le fichier ouvert. L'octet de poids faible de ce pointeur est toujours \$00, car un buffer commence toujours sur un saut de page mémoire.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$43	Le Numéro de Référence du fichier n'est pas valide. Vous devez utiliser un Numéro de Référence d'un fichier qui a déjà été fermé.

Autres codes d'erreur que peut retourner la fonction

\$04

Exemple de programme

Vous pouvez utiliser ce programme pour déterminer l'adresse d'un buffer utilisé par un fichier ayant le Numéro de Référence 2. Après l'exécution de la commande GET_BUF, l'adresse du buffer se trouve en BUFFPTR.

```
JSR   MLI
```

```
DFB   $D3           ; GET_BUF
```

```
DA    PARMTBL       ; Adresse de la table des paramètres
```

BCS Error ; Retenue = 1 en cas d'erreur

RTS

PARMTBL DFB 2 ; Nombre de paramètres

DFB 2 ; Numéro de Référence du fichier

DS 2 ; L'adresse du buffer est retournée ici

GET_EOF

Code de fonction ProDOS 8: \$D3

Fonction

Permet de déterminer dans un fichier ouvert la valeur courante du pointeur EOF. Cette valeur représente la taille du fichier.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (2)
1	ref_num	1	E	Numéro de référence du fichier
2	eof	4	S	Position EOF

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 2 pour cette fonction.

ref_num Numéro de Référence attribué au fichier quand celui ci a été ouvert.

eof Valeur représentant la position courante EOF. Cette valeur est égale à la taille du fichier en nombre d'octets.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$43	Le numéro de Référence du fichier n'est pas valide. Vous devez très certainement utiliser un numéro d'un fichier qui a été fermé.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07

GET_FILE_INFO

Code de fonction ProDOS 8: \$C4

Fonction

Permet de retrouver l'information relative à un fichier stockée dans l'Entrée au catalogue de ce fichier. Cela comprend le code d'Accès, le code du Type de fichier, le code du Type Auxiliaire de fichier, le code du Type d'Enregistrement, le nombre de blocs utilisés par le fichier, et la Date et l'Heure de création et de dernière modification de ce fichier.

Quand un Pathname pointe sur le nom d'un Volume plutôt que sur un nom de fichier, la taille du Volume (en blocs) est retournée dans le champ `aux_type`; le nombre de blocs utilisés par tous les fichiers de ce volume est retourné dans le champ `blocks_used`.

Paramètres

Offset	Nom	Octets	E/S	Description
0	<code>num_parms</code>	1	E	Nombre de paramètres (10)
1	<code>pathname</code>	2	E	Pointeur sur le pathname
3	<code>access</code>	1	S	Code d'Accès
4	<code>file_type</code>	1	S	Code Type de fichier
5	<code>aux_type</code>	2	S	Code Type Auxiliaire de fichier
7	<code>storage_type</code>	1	S	Code Type d'Enregistrement
8	<code>blocks_used</code>	2	S	Nombre de blocs utilisés par le fichier
10	<code>modify_date</code>	2	S	Date de dernière Modification
12	<code>modify_time</code>	2	S	Heure de dernière Modification
14	<code>create_date</code>	2	S	Date de Création
16	<code>create_time</code>	2	S	Heure de Création

Description des paramètres

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. La valeur minimum pour cette fonction est 1 et la maximum 10.

pathname Pointeur de classe 0 pointant sur l'adresse de la chaîne décrivant le Pathname du fichier à utiliser. Si le Pathname spécifié n'est pas précédé par un séparateur ("/"), le système d'exploitation ajoute

automatiquement le nom du préfixe par défaut pour créer le Pathname complet.

access Code d'Accès au fichier. Voir les explications correspondantes dans la partie consacrée à l'organisation des fichiers sur un Volume.

file_type Code du Type de fichier.

aux_type Code du Type auxiliaire de fichier.

storage_type Code décrivant l'organisation physique du fichier sur disque:

Type	Description
\$01	Fichier "seedling"
\$02	Fichier "sapling"
\$03	Fichier "tree"
\$04	Partition Pascal
\$05	Fichier de type étendu
\$0D	Fichier Sous-Catalogue
\$0F	Fichier catalogue du Volume

blocks_used Ce champ contient le nombre total de blocs utilisés par le fichier pour le rangement des données et des blocs d'index. Si le fichier est de type étendu, alors il représente seulement le nombre de blocs utilisés par le segment de données.

modify_date Ce champ contient la date (heure, mois, année) de dernière modification du fichier. Le format est celui utilisé par ProDOS 8 pour le codage de la date.

modify_time Ce champ contient l'heure (heure, minute) de dernière modification du fichier. Le format est celui utilisé par ProDOS 8 pour le codage de l'heure.

create_date Ce champ contient la date (heure, mois, année) de création du fichier. Le format est celui utilisé par ProDOS 8 pour le codage de la date.

create_time Ce champ contient l'heure (heure, minute) de création du fichier. Le format est celui utilisé par ProDOS 8 pour le codage de l'heure.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$40	Le Pathname contient des caractères non valides, ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.
\$46	Le fichier n'a pas été trouvé.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$4A, \$4B, \$52, \$53, \$58.

GET_MARK

Code de fonction ProDOS 8: \$CF

Fonction

Permet de déterminer la valeur courante du pointeur MARK d'un fichier ouvert.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (1)
1	ref_num	1	E	Numéro de Référence du fichier
2	position	4	S	Position courante du Pointeur MARK

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 2 pour cette fonction.

ref_num Valeur du Niveau du fichier système. Les valeurs retournées peuvent être comprises entre \$0000 et \$00FF.

position Position courante de MARK en octets.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$43	Le Numéro de Référence du fichier n'est pas valide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07.

GET_PREFIX

Code de fonction ProDOS 8: \$C7

Fonction

Permet de déterminer le nom du Préfixe par défaut.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (1)
1	prefix	2	S	Pointeur sur le Préfixe

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 1 pour cette fonction.

prefix Ce buffer doit avoir 67 octets de longueur car un nom de Préfixe peut être constitué de 64 caractères de long; plus l'octet de longueur, plus les deux délimiteurs /.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$56	L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map, ou parce que le buffer ne débute pas sur un saut de page mémoire.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$4F, 53

GET_TIME

Code de fonction ProDOS 8: \$82

Fonction

Permet de lire et de placer la date et l'heure dans la Page globale ProDos 8 aux adresses DATE (\$BF90 - \$BF91) et TIME (\$BF92 - \$BF93).

Paramètres

Il n'y a pas de Table des Paramètres, la routine appelante doit pointer sur une adresse fictive.

Exemple de programme

Quand on utilise cette commande, la date courante (année, mois, jour) et l'heure courante (heure, minute) sont rangées dans la Page globale ProDOS 8 aux adresses DATE (\$BF90 - \$BF91) et TIME (\$BF92 - \$BF93), dans le format utilisé par ProDOS 8. Il faut bien entendu posséder une carte horloge compatible ProDOS (ce qui est bien entendu le cas de l'horloge de l'Apple IIGS).

```
JSR  MLI
DFB  $82          ; GET_TIME
DA   $0000       ; Table des Paramètres fictive
BCS  Error       ; Retenue = 1 en cas d'erreur
RTS
```

NEWLINE

Code de fonction ProDOS 8: \$C9

Fonction

Permet d'activer ou de désactiver le mode de lecture "newline". Quand le mode de lecture "newline" est activé, toutes les opérations de lecture successives se terminent à chaque fois qu'un caractère déterminé est lu; c'est le caractère "newline". Quand le mode de lecture "newline" est désactivé, toutes les opérations de lecture se terminent quand la position EOF (fin de fichier) est atteinte, ou quand le nombre indiqué de caractères a été lu.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (3)
1	ref_num	1	E	Numéro de Référence du fichier
2	enable_mask	1	E	Masque d'activation "newline"
3	newline_char	1	E	Caractère "newline"

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 3 pour cette fonction.

ref_num Numéro de Référence que le système d'exploitation a attribué au fichier quand celui-ci a été ouvert.

enable_mask Une opération mathématique AND est faite avec cette valeur et chaque octet lu consécutivement dans le fichier. Si le résultat de l'opération AND est identique à *newline_char*, l'opération de lecture s'arrête; autrement, la lecture continue normalement.
Exception: Si *enable_mask* est égal à 0, le mode de lecture "newline" est désactivé et les opérations de lecture ne sont pas affectées.

num_chars Nombre de caractères se trouvant dans la table "newline" des caractères. Si *enable_mask* est différent de 0, *num_chars* ne peut pas valoir 0.

new_line_char Valeur du caractère *new_line*. Les opérations de lecture s'arrêtent automatiquement si le résultat de l'opération AND entre *enable_mask* et le caractère qui vient d'être lu est égal à *newline_char*.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$43	Le Numéro de Référence du fichier n'est pas valide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07

ON_LINE

Code de fonction ProDOS 8: \$C5

Fonction

Permet de déterminer soit le nom de Volume d'un disque spécifique ou soit tous les noms de tous les Volumes ProDOS 8 actifs.

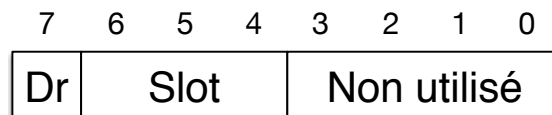
Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (2)
1	unit_num	1	E	Numéro d'Unité
2	data_buffer	2	E	Pointeur sur le buffer de données

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 2 pour cette fonction.

unit_num Numéro de slot et de drive auquel on veut accéder.



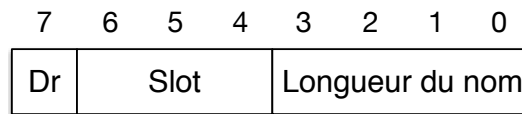
Slot peut en fait être le numéro réel ou logique d'un slot si le système contient des périphériques disque comme par exemple un RAM disque. Par exemple, le numéro d'unité d'un volume /RAM connecté en slot 3, Drive 2 sur un Ile, lic ou IIGS est \$B0, soit 1 011 0000 en binaire.

Dr indique le numéro de Drive (lecteur): 0 pour le lecteur 1 et 1 pour le lecteur 2. Plus de deux lecteurs peuvent être connectés au port 5 du SmartPort. Dans ce cas, ProDOS 8 assigne logiquement les deux prochains lecteurs en Slot 2, Drive 1 et en Slot 2, Drive 2. ProDOS 8 ignore tous les lecteurs connectés au SmartPort après le quatrième.

Exception : Si *unit_num* est égal à 0, les noms de Volume de tous les drives sont retournés.

data_buffer Pointeur sur un buffer contenant le nom de Volume du Drive spécifié. Si *unit_number* = 0, les noms de Volume de tous les drives sont retournés. Chaque nom de Volume est constitué d'un champ de 16 octets.

Le premier octet de chaque champ de 16 octets contient le numéro de Drive et de slot de ce Volume disque puis la longueur du nom de ce Volume. Le format utilisé est le suivant:



Dr et Slot ont le même format que unit_num. On trouve la longueur du nom de Volume dans les 4 bits de poids faible. Les 15 octets suivants contiennent le nom de Volume; ce nom n'est pas précédé par un "/".

Si unit_num est égal à 0, l'enregistrement après le dernier champ de 16 octets commence par \$00. Vous devez réserver un buffer de 256 octets si vous appelez ON_LINE avec unit_num = 0.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$27	Le disque est illisible. Il est très certainement endommagé. Cette erreur peut aussi survenir si la porte du lecteur est ouverte, ou s'il n'y a pas de disquette dans le lecteur.
\$28	Pas de périphériques de connectés. ProDOS 8 retourne cette erreur quand on essaye d'accéder à un second lecteur 5.25 qui n'existe pas.
\$2E	Un disque a été extrait de son lecteur, alors qu'un fichier était encore ouvert.
\$2F	Le périphérique n'est pas prêt. Il n'y a pas de disque dans le lecteur 3.5.
\$52	Le disque dans le lecteur indiqué par unit_num n'est pas un disque formaté sous ProDOS.
\$56	L'adresse du buffer pour le Pathname n'est pas valide; elle est déclarée occupée dans la Bit Map système de ProDOS 8.

Autres codes d'erreur que peut retourner la fonction

\$04, \$55

Commentaire

ON LINE traite les erreurs d'une manière différente des autres commandes MLI. En cas d'erreur, "LONGUEUR DU NOM" prend la valeur 0, et le code d'erreur est stockée dans le deuxième octet de l'enregistrement de 16 octets correspondant. Le code d'erreur n'est pas rangé dans l'accumulateur (A), l'indicateur de carry (C) n'est pas mis à 1.

Exemple de programme

Ce programme lit le nom de Volume d'un disque placé en Slot 6, Drive 2.

```
JSR   MLI
DFB   $C5           ; ON_LINE
```

```

        DA    PARM_TBL    ; Adresse de la table des paramètres
LDA     BUFFER    ; Vérifions que la longueur du nom ne vaut pas 0.
AND     #$0F      ; Gardons les quatre derniers bits
BEQ     ERROR     ; Si la longueur vaut 0, il s'est produit une
                ; erreur
...
ERROR   RTS

PARM_TBL    DFB    2      ; Nombre de paramètres
          DFB    $E0     ; unit_num = slot6, drive2
          DA     BUFFER   ; Pointeur sur le buffer du pathname

BUFFER     DS     1      ; Slot/Drive (bits7-4) et Longueur du nom
                ; de Volume (bits 3-0)
          DS     15     ; Nom du Volume (ASCII)

```

Si par exemple, le nom de Volume est ASM.FILES, l'octet stocké en BUFFER est \$E9, et les octets stockés à partir de BUFFER+1 sont 41 53 4D 2E 46 49 4C 45 53 (les codes ASCII des caractères qui composent la chaîne "ASM.FILES").

OPEN

Code de fonction ProDOS 8: \$C8

Fonction

Permet de préparer un fichier pour des opérations de lecture et d'écriture à venir. Quand on ouvre un fichier, le pointeur MARK pointe sur le début du fichier (MARK = 0), et le niveau du fichier est égal au niveau du fichier système. La commande renvoie également tous les attributs du fichier.

En général on ne peut ouvrir qu'un fichier qui soit fermé, mais si un fichier est déjà ouvert et que son code d'accès d'autorisation d'écriture n'est pas à 1 (c'est à dire que l'on ne peut pas le modifier), ce fichier peut être ouvert plus d'une fois.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (3)
1	pathname	2	E	Pointeur sur le Pathname
3	io_buffer	2	S	Pointeur sur le buffer E/S
5	ref_num	1	S	Numéro de Référence du fichier

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 3 pour cette fonction.

pathname Pointeur de classe 0 pointant sur l'adresse de la chaîne décrivant le Pathname du fichier à utiliser. Si le Pathname spécifié n'est pas précédé par un séparateur ("/"), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut pour créer le Pathname entier.

io_buffer Pointeur sur un buffer de 1024 octets. L'octet de poids faible de ce pointeur doit être à \$00; cela signifie que ce buffer doit débuter sur un saut de page. La première moitié de ce buffer contient une copie des blocs de données auxquels on vient d'accéder; la deuxième moitié contient les blocs d'index.

ref_num Numéro de Référence attribué au fichier par le système d'exploitation. De très nombreuses commandes utilisent ce Numéro de Référence pour identifier le fichier. Le niveau du fichier est le même que pour le fichier système.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$40	Le Pathname contient des caractères non valides, ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$42	Tentative d'ouverture d'un 9ème fichier. Sous ProDOS 8, on ne peut ouvrir que 8 fichiers simultanément.
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.
\$46	Le fichier n'a pas été trouvé.
\$50	Le fichier est ouvert. Cette commande ne fonctionne qu'avec des fichiers fermés.
\$56	L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map de ProDOS 8, ou parce que le buffer ne débute pas sur un saut de page mémoire.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$4A, \$4B, \$52.

QUIT

Code de fonction ProDOS 8: \$65

Fonction

Permet de quitter l'application courante. Le contrôle est repassé au sélecteur de programme.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (4)
1	quit_type	1	E	Type de Quit
2	pathname	2	E	Pointeur sur le prochain Pathname
4	réservé	1	E	Zone réservée
5	réservé	1	E	Zone réservée

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres PRODOS 8. Ce nombre est toujours 4 pour cette fonction.

quit_type Code du type de Quit. Il y a deux types de Quit sous ProDOS 8:

Type	Description
\$00	Quit "standard"
\$EE	Quit vers un programme système

Le code \$EE peut seulement être utilisé si le fichier système a été lancé depuis GS/OS.

pathname Pointeur sur un buffer de classe 0 contenant le pathname du fichier système à lancer après avoir quitté l'application en cours d'exécution. Le type de fichier doit être \$FF (ProDOS 8) ou \$B3 (GS/OS). Le pathname ne peut pas résider en page 2 car la commande Quit utilise cette zone mémoire. Ce champ doit être à 0 si quit_type est \$00.

flags Indicateurs du type de Quit; seuls les bits 15 et 14 sont significatifs. Si le bit 15 est à 1, le UserID du programme est placé dans la pile de Quit, afin que le programme puisse être relancé. Si le bit 14 est à 1, le programme est relancé à partir de la mémoire.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$46	Le fichier n'a pas été trouvé.
\$5C	Le fichier indiqué par le Pathname n'est pas un programme exécutable. Ce Pathname doit pointer sur un programme système ProDOS 8 (type de fichier \$FF) ou un programme système GS/OS (type de fichier \$B3).
\$5D	Le Pathname pointe sur un programme système ProDOS 8, mais le fichier P8 (qui contient le système d'exploitation ProDOS 8) n'est pas présent dans le sous-catalogue /SYSTEM du disque de boot de GS/OS.
\$5F	La pile des adresses de retour des Quit est saturée. GS/OS retourne cette erreur quand on tente de pousser un autre ID d'un programme sur cette pile quand celle ci est pleine.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$40, \$5E.

READ

Code de fonction ProDOS 8: \$CA

Fonction

Permet de lire dans un fichier ouvert des octets à partir de la position courante du pointeur MARK. Après l'opération de lecture, le système d'exploitation incrémente MARK du nombre d'octets qui ont été lus. L'opération de lecture se termine quand le nombre indiqué d'octets a été transféré, quand un caractère "newline" est rencontré ou quand la fin du fichier est atteinte.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_params	1	E	Nombre de paramètres (4)
1	ref_num	1	E	Numéro de Référence du fichier
2	data_buffer	2	E	Pointeur sur le début du buffer de données
4	request_count	2	E	Nombre d'octets à lire
6	transfer_count	2	S	Nombre d'octets lus

Description des paramètres

- num_parms* Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 4 pour cette fonction.
- ref_num* Numéro de Référence attribué au fichier par le système d'exploitation quand celui-ci a été ouvert.
- data_buffer* Pointeur sur le début du buffer dans lequel les données du fichier vont être placées. La taille du buffer doit être égale à *request_count*.
- request_count* Nombre de caractères à lire dans le fichier. Ils sont lus dans le fichier et placés dans le buffer pointé par *data_buffer*.
- transfer_count* Nombre de caractères qui ont été lus dans le fichier. Il est en général égal à *request_count*, mais peut être inférieur si le système d'exploitation atteint la fin du fichier ou si le mode de lecture est actif et qu'un caractère "newline" est lu (voir la commande NewLine).

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$43	Le numéro de référence d'un fichier n'est pas valide.
\$4C	La fin du fichier (EOF) a été atteinte durant une opération de lecture.
\$4E	Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier contrôle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.
\$56	L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map de ProDOS 8, ou parce que le buffer ne débute par sur un saut de page mémoire.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27.

READ_BLOCK

Code de fonction ProDOS 8: \$80

Fonction

Permet de transférer un block de datas (512 octets) d'un périphérique disque vers un buffer en mémoire.

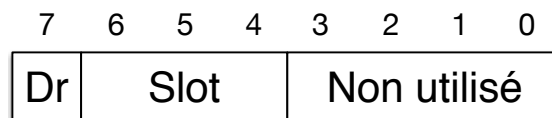
Paramètres

Offset	Nom	Octets	E/S	Description
0	num_params	1	E	Nombre de paramètres (3)
1	unit_num	1	E	Unit number
2	data_buffer	2	S	Pointeur sur le début du buffer de données
4	block_num	2	E	Numéro du bloc à lire

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 3 pour cette fonction.

unit_num Numéro de slot et de drive auquel on veut accéder.



Slot peut en fait être le numéro réel ou logique d'un slot si le système contient des périphériques disque comme par exemple un RAM disque. Par exemple, le numéro d'unité d'un volume /RAM connecté en slot 3, Drive 2 sur un IIe, lic ou IIGS est \$B0, soit 1 011 0000 en binaire.

Dr indique le numéro de Drive (lecteur): 0 pour le lecteur 1 et 1 pour le lecteur 2. Plus de deux lecteurs peuvent être connectés au port 5 du SmartPort. Dans ce cas, ProDOS 8 assigne logiquement les deux prochains lecteurs en Slot 2, Drive 1 et en Slot 2, Drive 2. ProDOS 8 ignore tous les lecteurs connectés au SmartPort après le quatrième.

data_buffer Pointeur sur le début d'un bloc mémoire de 512 octets qui contient le bloc lu par READ_BLOCK.

block_num Numéro du bloc à lire. Les valeurs autorisées dépendent du périphérique disque.

Valeurs	Type de drive
0-279	Lecteur de disquettes 5.25"
0-1599	Lecteur de disquettes 3.5"
0-127	Volume /RAM

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$27	Le disque est illisible. Il est très certainement endommagé. Cette erreur peut aussi survenir si la porte du lecteur est ouverte, ou s'il n'y a pas de disquette dans le lecteur.
\$28	Pas de périphériques de connectés. ProDOS 8 retourne cette erreur quand on essaye d'accéder à un second lecteur 5.25" qui n'existe pas.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$11, \$2F, \$53, \$56.

Commentaires

Sous GS/OS, on utilise la commande DRead.

RENAME

Code de fonction ProDOS 8: \$C2

Fonction

Permet de modifier le nom d'un fichier.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_params	1	E	Nombre de paramètres (2)
1	curr_name	2	E	Pointeur sur le Pathname actuel du fichier
3	new_name	2	S	Pointeur sur le nouveau Pathname que l'on veut donner au fichier

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 2 pour cette fonction.

curr_name Pointeur vers un buffer de classe 0 contenant le pathname du fichier à renommer.

new_name Pointeur vers un buffer de classe 0 contenant le nouveau pathname pour le fichier. Le fichier doit se trouver dans le même sous-catalogue.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Une opération d'écriture a échoué parce que le volume disque est protégé contre l'écriture.
\$40	La syntaxe du pathname n'est pas valide.
\$44	Le Pathname spécifié n'a pas été trouvé. Cela signifie que l'un des noms de sous-catalogue, dans un autre pathname valide, n'existe pas.
\$45	Le Volume spécifié n'a pas été trouvé. Cela arrive quand par exemple on change le disque du lecteur.
\$46	Le fichier n'a pas été trouvé.
\$47	Le nouveau nom de fichier existe déjà.
\$4E	Impossible d'accéder au fichier.
\$50	Le fichier est ouvert. On ne peut renommer que les fichiers fermés.

Autres codes d'erreur que peut retourner la fonction

\$04, \$27, \$4A.

Exemple de programme

Voici une sous-routine permettant de changer le nom d'un fichier appelé FILE.1 dans un sous-catalogue /PROG; le nouveau nom de fichier sera FILE.2 dans le même sous-catalogue.

	JSR	MLI	
	DFB	\$C2	; RENAME
	DA	PARMTBL	; Adresse de la table des paramètres
	BCS	ERROR	; Retenue = 1 en cas d'erreur
		...	
ERROR	RTS		
PARMTBL	DFB	2	; Nombre de paramètres
	DA	PATH1	; Pointeur sur le pathname actuel
	DA	PATH2	; Pointeur sur le nouveau nom
BUFFER	ASC	"/PROG/FILE.1"	
	ASC	"/PROG/FILE.2"	

SET_BUF

Code de fonction ProDOS 8: \$D2

Fonction

Permet de déplacer un buffer d'un fichier de sa position courante vers une autre zone de 1024 octets en mémoire.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_params	1	E	Nombre de paramètres (2)
1	ref_num	1	E	Numéro de Référence du fichier
2	io_buffer	2	S	Pointeur sur le buffer

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 2 pour cette fonction.

ref_num Numéro de Référence attribué au fichier par le système d'exploitation.

io_buffer Pointeur sur un buffer de 1024 octets dans lequel le buffer courant va être transféré.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$43	Le Numéro de Référence du fichier n'est pas valide.
\$56	L'adresse du buffer pour le pathname n'est pas valide.

Autres codes d'erreur que peut retourner la fonction

\$04.

Exemple de programme

Le programme suivant déplace le buffer d'un fichier ayant pour Numéro de Référence 1 de sa position courante à l'adresse \$2000. Vous devez vous assurer que la zone \$2000-\$23FF n'est pas utilisée.

```
JSR   MLI  
DFB   $D2           ; SET_BUF  
DA    PARMTBL       ; Adresse de la table des paramètres
```

BCS	ERROR		; Retenue = 1 en cas d'erreur
-----	-------	--	-------------------------------

...

ERROR	RTS		
-------	-----	--	--

PARMTBL	DFB	2	; Nombre de paramètres
---------	-----	---	------------------------

	DFB	1	; Numéro de Référence du fichier
--	-----	---	----------------------------------

	DA	\$2000	; Pointeur sur le nouveau buffer
--	----	--------	----------------------------------

SET_EOF

Code de fonction ProDOS 8: \$D0

Fonction

Permet de modifier le pointeur EOF d'un fichier ouvert. Si on diminue EOF, tous les blocks de données se trouvant après la nouvelle valeur EOF sont libérés. Si on augmente EOF, le système d'exploitation n'attribue pas de nouveaux blocks tant que d'autres données ne sont pas écrites dans ce fichier. Si la nouvelle valeur EOF est inférieure au pointeur MARK, ce dernier prend la valeur de EOF. On peut modifier la valeur EOF de tout fichier dont le bit d'accès d'écriture est à 1.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (2)
1	ref_num	1	E	Numéro de Référence du fichier
2	eof	2	E	Nouvelle valeur de EOF

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 2 pour cette fonction.

ref_num Numéro de Référence attribué au fichier par le système d'exploitation.

eof Nouvelle valeur du pointeur EOF.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le disque est protégé contre l'écriture.
\$43	Le Numéro de Référence du fichier n'est pas valide.
\$4D	La valeur MARK spécifiée est supérieure à EOF.
\$4E	Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier contrôle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$4E.

SET_FILE_INFO

Code de fonction ProDOS 8: \$C3

Fonction

Permet de modifier l'information concernant un fichier dans l'Entrée au catalogue.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (10)
1	pathname	2	E	Pointeur sur le Pathname
3	access	1	E	Code d'Accès
4	file_type	1	E	Type de fichier
5	aux_type	2	E	Type Auxiliaire de fichier
7	non utilisé	1	E	
8	non utilisé	2	E	
10	modify_date	2	E	Date de Modification
12	modify_time	2	E	Heure de Modification

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 10 pour cette fonction.

pathname Pointeur de classe 0 pointant sur l'adresse de la chaîne décrivant le Pathname du fichier à utiliser. Si le pathname spécifié n'est pas précédé par un séparateur ("/"), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut pour créer le Pathname complet.

access Code d'Accès au fichier. Voir explications dans la partie consacrée à l'organisation des fichiers sur un Volume.

file_type Code du Type de fichier

aux_type Code du Type auxiliaire de fichier.

non utilisé Ces octets ne sont pas utilisés. Ils permettent de conserver une symétrie avec la Table des Paramètres de la commande GET_FILE_INFO.

modify_date Ce champ contient la date (heure, mois, année) de dernière modification du fichier. Le format est celui utilisé par ProDOS 8 pour le codage de la date.

modify_time Ce champ contient l'heure (heure, minute) de dernière modification du fichier. Le format est celui utilisé par ProDOS 8 pour le codage de l'heure.

Note

Les paramètres non utilisés doivent être mis à 0.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le disque est protégé contre l'écriture.
\$40	Le Pathname contient des caractères non valides, Ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.
\$46	Le fichier n'a pas été trouvé.
\$4E	Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier contrôle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$4A, \$4B, \$52, \$53, \$58.

SET_PREFIX

Code de fonction ProDOS: \$C6

Fonction

Permet de fixer le Préfixe par défaut. Quand on appelle une commande, le système d'exploitation convertit le Pathname demandé en y ajoutant le Préfixe par défaut pour obtenir le Pathname complet.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (1)
1	prefix	2	E	Pointeur sur le Préfixe

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 1 pour cette fonction.

prefix Pointeur de classe 0 pointant sur l'adresse de la chaîne décrivant le Préfixe à utiliser.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$40	La syntaxe du pathname n'est pas valide.
\$44	Le Pathname spécifié n'a pas été trouvé.
\$45	Le Volume spécifié n'a pas été trouvé.
\$46	Fichier non trouvé.
\$4B	Le type de fichier n'est pas valide ou n'est pas reconnu.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$53.

WRITE

Code de fonction ProDOS 8: \$CB

Fonction

Permet d'écrire des datas dans un fichier ouvert. L'écriture débute à la position courante de MARK. Le pointeur MARK est incrémenté au fur et à mesure de l'écriture.

Paramètres

Offset	Nom	Octets	E/S	Description
0	num_parms	1	E	Nombre de paramètres (4)
1	ref_num	1	E	Numéro de Référence du fichier
2	data_buffer	2	E	Pointeur sur le buffer de données
4	request_count	2	E	Nombre d'octets à écrire
6	transfer_count	2	S	Nombre d'octets écrits

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 4 pour cette fonction.

ref_num Numéro de Référence du fichier attribué par le système d'exploitation.

data_buffer Pointeur sur le début du bloc mémoire contenant les données à écrire sur le disque.

request_count Nombre de caractères à écrire sur disque.

transfer_count Nombre de caractères écrits sur disque.

cache_priority Ce code indique comment GS/OS va utiliser le cache pour y placer les blocks lors de la lecture du fichier.

Code	Description
\$0000	Pas de cache pour ce fichier
\$0001	Le cache est utilisé pour ce fichier

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le disque est protégé contre l'écriture.
\$43	Le Numéro de Référence du fichier n'est pas valide.

Code	Description
\$48	Le Volume est plein.
\$4E	Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier contrôle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.
\$56	L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map, ou parce que le buffer ne débute par sur un saut de page mémoire.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27.

WRITE_BLOCK

Code de fonction ProDOS 8: \$81

Fonction

Permet de transférer le contenu du buffer mémoire de 512 octets de la mémoire sur le Volume disque.

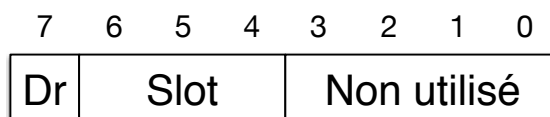
Paramètres

Offset	Nom	Octets	E/S	Description
0	num_params	1	E	Nombre de paramètres (3)
1	unit_num	1	E	Unit number
2	data_buffer	2	S	Pointeur sur le début du buffer de données
4	block_num	2	E	Numéro du bloc à lire

Description des paramètres

num_parms Nombre de paramètres dans la Table de Paramètres ProDOS 8. Ce nombre est toujours 3 pour cette fonction.

unit_num Numéro de slot et de drive auquel on veut accéder.



Slot peut en fait être le numéro réel ou logique d'un slot si le système contient des périphériques disque comme par exemple un RAM disque. Par exemple, le numéro d'unité d'un volume /RAM connecté en slot 3, Drive 2 sur un Ile, lic ou IIGS est \$B0, soit 1 011 0000 en binaire.

Dr indique le numéro de Drive (lecteur): 0 pour le lecteur 1 et 1 pour le lecteur 2. Plus de deux lecteurs peuvent être connectés au port 5 du SmartPort. Dans ce cas, ProDOS 8 assigne logiquement les deux prochains lecteurs en Slot 2, Drive 1 et en Slot 2, Drive 2. ProDOS 8 ignore tous les lecteurs connectés au SmartPort après le quatrième.

data_buffer Pointeur sur le début d'un bloc mémoire de 512 octets qui contient le bloc à écrire par WRITE_BLOCK.

block_num Numéro du bloc à écrire. Les valeurs autorisées dépendent du périphérique disque.

Valeurs	Type de drive
0-279	Lecteur de disquettes 5.25"
0-1599	Lecteur de disquettes 3.5"
0-127	Volume /RAM

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$27	Le disque est illisible. Il est très certainement endommagé. Cette erreur peut aussi survenir si la porte du lecteur est ouverte, ou s'il n'y a pas de disquette dans le lecteur.
\$28	Pas de périphériques connectés. ProDOS 8 retourne cette erreur quand on essaye d'accéder à un second lecteur 5.25" qui n'existe pas.
\$2B	Le disque est protégé contre l'écriture.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$11, \$2F, \$53, \$56.

Commentaires

Sous GS/OS, on utilise la commande DWrite.

Les commandes de GS/OS par ordre alphabétique

BeginSession

Code de fonction GS/OS: \$201D

Fonction

Indique à GS/OS de différer toutes les opérations d'écriture qui portent sur la mise à jour de la Bit Map et des blocs de catalogue.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	1	E	Nombre de paramètres (0)

Description des paramètres

pcount C'est le nombre de paramètres dans la Table des paramètres. Cette valeur vaut toujours 0 pour cette fonction.

Codes d'erreur que peut retourner la fonction

Cette routine ne retourne jamais un code d'erreur.

Commentaires

Les sessions d'écriture différée sont utiles quand une application doit transférer rapidement un groupe de fichiers d'un disque sur un autre. Si on n'utilise pas une session d'écriture différée, les opérations de copie seront ralenties parce que la tête de Lecture / Ecriture doit balayer toute la surface du disque pour accéder à la Bit Map et aux blocs de Catalogue avant chaque transfert de fichier. A la fin d'une opération de copie de ce type, il faut utiliser la commande EndSession pour écrire sur le disque les blocs restants. A chaque commande BeginSession doit correspondre une commande EndSession.

BindInt

Code de fonction GS/OS: \$2031

Fonction

Permet d'attribuer une sous-routine de gestion d'interruption sous GS/OS vers une source particulière d'interruption.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (3)
2	int_num	2	S	Numéro de référence de l'interruption
4	vrn	2	E	Numéro de référence du vecteur
6	int_code	4	E	Pointeur sur le Handler d'interruption

Description des paramètres

pcount C'est le nombre de paramètres dans la Table de Paramètres GS/OS. La valeur est toujours 3 pour cette fonction.

int_num C'est le numéro de référence que GS/OS attribue à la sous-routine de gestion d'interruption. On utilise ce numéro de référence quand on veut enlever la sous-routine avec la commande UnBindInt.

vrn C'est le numéro de référence qui identifie le type d'interruption système auquel est attribué le Handler d'interruption.

Code	Description
\$0008	AppleTalk (SCC ou Serial Communications Controller)
\$0009	Ports Série(SCC ou Serial Communications Controller)
\$000A	Scan Line
\$000B	Fin de Waveform (Ensoniq)
\$000C	VBL (Vertical Blanking Signal)
\$000D	Souris (Mouvement ou click)
\$000E	Timer 1/4 de seconde
\$000F	Clavier
\$0010	Octet de réponse ADB (Apple Desktop Bus) prêt
\$0011	SRQ (ADB Service Request)

Code	Description
\$0012	Acessoire de bureau demandé
\$0013	Interruption Buffer clavier
\$0014	Interruption Micro Clavier
\$0015	Timer 1 seconde
\$0016	VGC (Video Graphics Controller) (Externe)
\$0017	Autre source d'interruption

int_code Pointeur sur le début de la routine de gestion d'interruption.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$25	La table des vecteurs d'interruption est remplie. Utilisez la commande UnbindInt puis recommencez.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$53

Commentaires

Sous ProDOS 8, la fonction équivalente s'appelle ALLOC_INTERRUPT.

ChangePath

Code de fonction GS/OS: \$2004

Fonction

Permet de renommer un fichier ou un Volume disque ou de déplacer un fichier d'un sous-catalogue à un autre sur le même Volume Disque. Vous pouvez modifier le path de tout fichier fermé dont le bit de code d'accès est à 1. Sous ProDOS 8, utilisez la commande RENAME pour renommer un fichier ou un Volume Disque. Il n'y a pas de commande ProDOS 8 pour déplacer un fichier d'un sous-catalogue à un autre.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (2)
2	pathname	4	S	Pointeur sur le pathname
6	new_pathname	4	E	Pointeur sur le nouveau pathname

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 2 pour cette fonction.

pathname Pointeur de classe 1 pointant sur le Pathname courant du fichier qui doit être modifié. Si ce Pathname n'est pas précédé par un séparateur (/ ou :), le système d'exploitation ajoute automatiquement le préfixe par défaut (le préfixe 0/) pour créer le pathname complet.

new_ pathname Pointeur de classe 1 pointant sur le nouveau Pathname du fichier qui va être modifié. Si ce Pathname n'est pas précédé par un séparateur (/ ou :), le système d'exploitation ajoute automatiquement le/préfixe par défaut (le préfixe 0/) pour créer le pathname complet.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le Volume Disque est protégé en écriture.
\$40	Le Pathname contient des caractères non valides, ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.

Code	Description
\$46	Le fichier n'a pas été trouvé.
\$47	Le nouveau Pathname indiqué existe déjà.
\$4E	Le système ne peut pas accéder au fichier. Son code d'accès ne lui permet pas d'être renommé. Utilisez la commande SetFileInfo pour modifier ce code d'accès.
\$50	Le fichier est ouvert. Cette commande ne fonctionne qu'avec des fichiers fermés.
\$5B	Les deux Pathnames indiquent des Volumes différents.

Autres codes d'erreur que peut retourner la fonction

\$07, \$27, \$4A, \$4B, \$52, \$57, \$58

Exemple de programme

Supposez que vous vouliez déplacer un fichier appelé ACCESSOIRE d'un sous-catalogue appelé CDA.NDA du disque de Boot vers le Catalogue DESK.ACCESS du même disque.

```
JSL $E100A8 ; Point d'entrée GS/OS
```

```
DA $2004 ; Numéro de commande GS/OS
```

```
ADRL CP_Parms ; Adresse de la table des paramètres
```

```
BCS Error ; Retenue = 1 en cas d'erreur
```

```
CP_Parms DA $2
```

```
ADRL NomCourant
```

```
ADRL NouveauNom
```

```
NomCourant ASC "*" : CDA . NDA : ACCESSOIRE "
```

```
NouveauNom ASC "*" : SYSTEM : DESK . ACCESS : ACCESSOIRE "
```

Commentaire

Quand les deux Pathnames spécifiés décrivent le même fichier dans le même sous-catalogue, la commande ChangePath est équivalente à la commande ProDOS 8 RENAME.

ClearBackup

Code de fonction GS/OS: \$200B

Fonction

Met à 0 le bit Backup dans le code d'accès d'un fichier.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	pathname	4	E	Pointeur sur le pathname

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

pathname Pointeur de classe 1 pointant sur le Pathname courant du fichier qui doit être modifié. Si ce Pathname n'est pas précédé par un séparateur (/ ou :), le système d'exploitation ajoute automatiquement le préfixe par défaut (le préfixe 0/) pour créer le pathname complet.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$40	Le Pathname contient des caractères non valides, ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.
\$46	Le fichier n'a pas été trouvé.

Autres codes d'erreur que peut retourner la fonction

\$07, \$4A, \$52, \$58

Exemple de programme

Un programme de copie de fichier doit seulement copier les fichiers qui ont été modifié depuis la dernière opération de copie. Le programme de copie doit vérifier le bit de Backup de chacun, fichiers pour déterminer s'ils doivent être copiés ou non; la copie se fait si le bit de Backup est à 1. GS /OS et ProDos 8 mettent automatiquement ce bit à 1 après chaque opération d'écriture concernant un fichier. Une fois la copie terminée, le programme de

copie est responsable de remettre à 0 le bit de Backup en appelant la commande ClearBackup.

JSL \$E100A8 ; Point d'entrée GS/OS

DA \$200B ; Code de la commande ClearBackup

ADRL ParmTbl ; Adresse de la table de paramètres

BCS Error ; Retenue = 1 en cas d'erreur

ParmTbl DA \$1 ; Nombre de paramètres

ADRL Pathname ; Adresse du pathaname du fichier cible

Pathname ASC "/DISK/NEW.FILE"

Close

Code de fonction GS/OS: \$2014

Fonction

Permet de fermer un fichier qui a été ouvert au préalable. Le système d'exploitation écrit le contenu du buffer attribué à ce fichier sur le Volume et met à jour l'entrée au Catalogue de ce fichier. Ensuite, le système d'exploitation libère la mémoire utilisée par le buffer de ce fichier.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	ref_num	2	E	Numéro de référence du fichier que l'on veut fermer

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

ref_num Numéro de Référence que le système d'exploitation a attribué au fichier quand celui-ci a été ouvert. Si on fixe *ref_num* à 0, tous les fichiers ouverts ayant le même niveau de priorité que le fichier système, ou ayant un niveau de priorité supérieur au fichier système sont fermés. Pour fixer la valeur du niveau de priorité d'un fichier, on utilise la commande GS/OS SetLevel.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le Volume est protégé en écriture.
\$43	Le Numéro de Référence du fichier n'est pas valide. Vous utilisez très certainement le numéro de Référence d'un fichier ayant déjà été fermé.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$5A

Exemple de programme

Pour fermer tous les fichiers ouverts ayant un niveau de priorité supérieur ou égal à 1, on

utilise la commande SetLevel et la commande Close avec ref_num = 0. Voici comment procéder:

```
JSL $E100A8 ; Point d'entrée GS/OS
```

```
DA $201A ; Code de la commande SetLevel
```

```
ADRL ParmTbl1 ; Adresse de la table de paramètres
```

```
BCS Error ; Retenue = 1 en cas d'erreur
```

```
JSL $E100A8 ; Point d'entrée GS/OS
```

```
DA $2014 ; Code de la commande Close
```

```
ADRL ParmTbl2 ; Adresse de la table de paramètres
```

```
BCS Error ; Retenue = 1 en cas d'erreur
```

```
ParmTbl1 DA $1 ; Nombre de paramètres
```

```
DA $1 ; Nouveau niveau de priorité pour le fichier
```

```
ParmTbl2 DA $1 ; Nombre de paramètres
```

```
DA $0 ; Num_ref=0, donc on ferme tous les fichiers
```

Create

Code de fonction GS/OS: \$2001

Fonction

Utilisée pour créer un nouveau fichier. Le système d'exploitation réalise cette fonction en plaçant une nouvelle Entrée dans le Catalogue spécifié.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (7)
2	pathname	4	E	Pointeur sur l'adresse du Pathname
6	access	2	E	Code d'Accès
8	file_type	2	E	Code du Type de Fichier
10	aux_type	4	E	Code du Type Auxiliaire de Fichier
14	storage_type	2	E	Code du Type d'Enregistrement
16	eof	4	E	Taille prévue du segment de données
20	resource_eof	4	E	Taille prévue du segment de ressources

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum de ce paramètre, pour cette fonction est 1, la maximum 7.

pathname Un pointeur sur un buffer de classe 1 sur l'adresse du Pathname du fichier qui va être créé. Si le Pathname spécifié n'est pas précédé par un séparateur ("/" ou ":"), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut (sous GS/OS c'est le préfixe "0/"), pour créer un Pathname complet.

access Voir la partie consacrée à l'Organisation des Fichiers sur Disque pour une explication détaillée du Code d'Accès.

file_type Code indiquant le Type du Fichier à créer. Voir la table avec la liste des codes existants, dans ce même chapitre.

aux_type Code indiquant le Type Auxiliaire du Fichier à créer. La signification de ce code dépend du Type de Fichier. Pour les fichiers SYS, BIN, BAS, et VAR, le Type Auxiliaire de Fichier contient l'adresse de chargement par défaut de ce fichier; pour les fichiers TXT, ce code est la longueur de chaque enregistrement du fichier texte.

storage_type Ce code indique la façon dont le système d'exploitation a sauvegardé le fichier sur le volume disque; il s'agit du Type d'Enregistrement du fichier:

Type	Description
\$00-\$03	Fichier standard
\$05	Fichier de type étendu
\$0D	Fichier Sous-Catalogue

On ne peut pas changer ce code une fois que le fichier est créé.

eof Si le fichier crée est un fichier standard, ce champ indique la taille prévue pour ce fichier (en nombre d'octets). Le système d'exploitation prévoit un nombre suffisant de blocs pour sauvegarder ce fichier.

Si le fichier crée est un fichier de type étendu, ce champ indique la taille prévue du segment de datas (en nombre d'octets).

Si le fichier crée est un fichier Sous-Catalogue, ce champ indique le nombre d'Entrées prévues dans ce Sous-Catalogue.

resource_eof Si le fichier crée est de type étendu, ce champ indique la taille prévue du segment de ressource (en nombre d'octets).

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le Volume est protégé en écriture.
\$40	Le Pathname contient des caractères non valides ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.
\$47	Le nouveau Pathname indiqué existe déjà.

Code	Description
\$48	Le Volume Disque est plein.
\$49	Le Catalogue principal est plein. Seuls 51 fichiers peuvent être sauvegardés dans le catalogue principal.
\$4B	Code du Type d'Enregistrement non valide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$10, \$27, \$52, \$53, \$58

Exemple de programme

Voici une petite routine sous GS/OS permettant de créer un fichier Texte (TXT); le nom du fichier TXT est ALLIANCE; le Pathname complet est: "GS:ALLIANCE".

```

                JSL    $E100A8      ; Point d'entrée GS/OS
                DA    $2001        ; Code de la commande Create
                ADRL  ParmTbl      ; Adresse de la table de paramètres
                BCS   Error        ; Retenue = 1 en cas d'erreur

ParmTbl        DA    $5           ; Nombre de paramètres
                ADRL  Pathname     ; Adresse du pathaname du fichier à créer
                DA    $E3         ; Code d'Accès standard (non verrouillé)
                DA    $04
                ADRL  $0           ; Type auxiliaire (0 = séquentiel)
                DA    $01         ; Type d'enregistrement (1 = standard)

Pathname       ASC    ":GS:ALLIANCE"

```

DControl

Code de fonction GS/OS: \$202E

Fonction

Permet de passer les commandes à un Device GS/OS.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (5)
2	dev_num	2	E	Numéro de Référence du Device
4	control_code	2	E	Code de contrôle demandé
6	control_list	4	E	Pointeur sur la liste de Contrôle
10	request_count	4	E	Taille de la liste de Contrôle
14	transfer_count	4	S	Nombre d'octets transférés

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 5 pour cette fonction.

dev_num Numéro de Référence du Device.

control_code C'est un code qui indique l'opération de contrôle à réaliser.

Code	Opération
\$0000	Reset du dispositif (Device)
\$0001	Formatage disque device
\$0002	Éjection disque device
\$0003	Réglages paramètres de configuration
\$0004	Réglage mode attente / pas d'attente
\$0005	Réglage des options de formatage
\$0006	Attribution de la partition
\$0007	Amorçage du signal
\$0008	Désamorçage du signal
\$0009	Réglage carte de partition
\$000A-\$7FFF	Réservé
\$8000-\$FFFF	Opérations spécifiques sur le dispositif (Device)

control_list Pointeur sur un buffer contenant des données supplémentaires dont GS/OS peut avoir besoin pour réaliser une opération de contrôle.

request_count Taille du buffer de la Liste de Contrôle.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$11	Le numéro de Référence du Device n'est pas valide.
\$53	Le paramètre est hors des limites admises.

Autres codes d'erreur que peut retourner la fonction

\$07

Exemple de programme

La seule commande de contrôle que vous serez très vraisemblablement amené à utiliser est la commande d'éjection d'un disque. Voici comment le faire.

	JSL	\$E100A8	; Point d'entrée GS/OS
	DA	\$202E	; Code de la commande DControl
	ADRL	ParmTbl	; Adresse de la table de paramètres
	BCS	Error	; Retenue = 1 en cas d'erreur
ParmTbl	DA	\$5	; Nombre de paramètres
	DA	\$2	; Numéro du dispositif (device)
	DA	\$2	; Code de contrôle (2 = Eject)
	ADRL	CtrlList	; Adresse de la liste de contrôle
	ADRL	\$0	
	DS	\$04	; Type d'enregistrement (1 = standard)
CtrlList	DS	\$04	; Liste de contrôle vide

Commentaire

On peut déterminer si un disque est éjectable en utilisant la commande DInfo et en examinant le bit 2 du mot des caractéristiques; si le bit est à 1, le disque est éjectable. Pour avoir des informations complètes sur les commandes de contrôle, consultez GS/OS Reference Volume 2.

Destroy

Code de fonction GS/OS: \$2002

Fonction

Permet d'éliminer un fichier d'un volume disque. Quand on détruit un fichier, le système d'exploitation libère tous les blocs que le fichier utilisait et met à zéro l'octet de longueur dans l'entrée catalogue qui correspond à ce fichier. On ne peut pas effacer les fichiers dont le bit de code d'accès de destruction est à 1; les fichiers sous-catalogues doivent être vides avant de pouvoir être éliminés.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	pathname	4	E	Pointeur sur l'adresse du Pathname

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

pathname Un pointeur de classe 1 sur l'adresse du Pathname du fichier qui va être détruit. Si le Pathname spécifié n'est pas précédé par un séparateur ("/" ou ":"), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut (sous GS/OS c'est le préfixe "0/") pour créer le Pathname complet. Si le Pathname décrit un fichier étendu, les deux segments sont détruits.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le Volume est protégé en écriture.
\$40	Le Pathname contient des caractères non valides, ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.
\$46	Le fichier n'a pas été trouvé.
\$4E	Le système ne peut pas accéder au fichier.
\$50	Le fichier est ouvert; on ne peut détruire que les fichiers fermés.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$10, \$27, \$4A, \$52, \$58

Exemple de programme

Prenons le cas dans lequel le Préfixe 0/ est /DEMO/NUCLEUS. Pour détruire un fichier ayant le pathname entier /DEMO/NUCLEUS/PROG, on peut utiliser la sous-routine GS/OS suivante.

```
JSL $E100A8 ; Point d'entrée GS/OS
```

```
DA $2002 ; Code de la commande Destroy
```

```
ADRL ParmTbl ; Adresse de la table de paramètres
```

```
BCS Error ; Retenue = 1 en cas d'erreur
```

```
RTS
```

```
ParmTbl DA $1 ; Nombre de paramètres
```

```
ADRL Pathname ; Adresse du pathname
```

```
Pathname ASC "PROG"
```

Commentaire

Un fichier de type étendu ne peut être détruit que par la commande Destroy de GS/OS.

DInfo

Code de fonction GS/OS: \$202C

Fonction

Permet d'avoir des informations sur le dispositif (Device) connecté au système.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (10)
2	dev_num	2	E	Numéro de Référence du dispositif (Device)
4	dev_name	2	E	Pointeur sur le nom du dispositif (Device)
8	characteristics	4	S	Carastéristiques du dispositif (Device)
10	total_blocks	4	S	Capacité du volume en nombre de blocs
14	slot_num	4	S	Numéro de Slot du dispositif (Device)
16	unit_num	2	S	Numéro d'unité du dispositif (Device)
18	version	2	S	Numéro de version du driver du dispositif (Device)
20	device_ID_num	2	S	Numéro ID du dispositif (Device)
22	head_link	2	S	Premier dispositif (Device) en relation
24	forward_link	2	S	Prochain dispositif (Device) en relation
26	extendedDIBptr	4	E	extendedDIBptr

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum de ce paramètre, pour cette fonction est 2, le maximum 11.

dev_num Numéro de Référence du Device.

dev_name Pointeur de classe 1 sur un buffer de sortie dans lequel GS/OS a retourné le nom du Device. Un nom de Device peut avoir jusqu'à 31 caractères de longueur; il faut donc fixer la taille de ce buffer à 35 octets.

characteristics Chacun des bits de ce mot donne les caractéristiques du Device.

Bit	Description
15	Si le bit vaut 1 alors le dispositif est un Ramdisk ou un Romdisk
14	Si le bit vaut 1 alors le driver du dispositif a été généré

Bit	Description
13	Réservé
12	Si le bit vaut 1 alors le dispositif est actif
11	Réservé
10	Réservé
9	Vitesse du dispositif (poids fort)
8	Vitesse du dispositif (poids faible)
7	Si le bit vaut 1 alors le dispositif est organisé en blocs
6	Si le bit vaut 1 alors l'écriture est autorisée
5	Si le bit vaut 1 alors la lecture est autorisée
4	Réservé
3	Si le bit vaut 1 alors le formatage du dispositif est possible
2	Si le bit vaut 1 alors le dispositif permet d'éjecter son contenu
1	Réservé
0	Réservé

Les bits 9 et 8 indiquent la vitesse à laquelle le dispositif peut fonctionner:

Valeur	Description
00	Le dispositif fonctionne à 1MHz (Mode émulation Apple II du IIGS)
01	Le dispositif fonctionne à 2.6MHz (Mode Fast du IIGS)
10	Le dispositif supporte des vitesses supérieures à 2.6MHz
11	La vitesse du processeur n'a pas d'influence sur le dispositif

total_blocks

Pour un dispositif organisé en blocs, c'est la capacité du volume en nombre de blocs. Pour un dispositif de type Character, ce champ reste à 0.

slot_num

Numéro du slot où se trouve le dispositif.

unit_num

Numéro d'unité du dispositif utilisé par le SmartPort.

version

Numéro de version du driver du dispositif.

Bit	Description
15-12	Numéro de version principale

Bit	Description
11-8	Premier numéro de version accessoire
7-4	Deuxième numéro de version accessoire
3-0	Type de version

device_ID_num Numéro de code qui identifie un type de dispositif.

Code	Type de dispositif
\$0000	Lecteur de disques 5.25"
\$0001	Disque dur Profile (5 Mo)
\$0002	Disque dur Profile (10 Mo)
\$0003	Lecteur de disques 3.5"
\$0004	Dispositif SCSI générique
\$0005	Disque dur SCSI
\$0006	Lecteur de bandes magnétiques SCSI
\$0007	Lecteur CD-ROM SCSI
\$0008	Imprimante SCSI
\$0009	Modem série
\$000A	Console
\$000B	Imprimante série
\$000C	LaserWriter série
\$000D	LaserWriter AppleTalk
\$000E	Disque RAM
\$000F	Disque ROM
\$0010	Fichier serveur
\$0011	Téléphone IBX
\$0012	Dispositif ADB
\$0013	Disque dur générique
\$0014	Lecteur de disques générique
\$0015	Lecteur de bandes magnétiques générique
\$0016	Dispositif de type Caractère générique

Code	Type de dispositif
\$0017	Lecteur de disques de type MFM
\$0018	Dispositif générique réseau AppleTalk
\$0019	Dispositif SCSI à accès séquentiel
\$001A	Scanner SCSI
\$001B	Scanner non SCSI
\$001C	LaserWriter SCSI
\$001D	Driver principal AppleTalk
\$001E	Driver fichier de service AppleTalk
\$001F	Driver AppleTalk RPM

head_link Numéro de dispositif indiquant quelle est la première entrée dans une liste de numéros de devices. Les dispositifs de la liste sont ceux qui pour un volume ont une partition distincte. Si *head_link* = 0, il n'y a pas de lien.

forward_link Numéro de device indiquant quelle est la prochaine entrée dans une liste de numéros de device. Les devices de la liste sont ceux qui pour un volume ont une partition distincte. Si *forward_link* = 0, il n'y a pas de lien.

extendedDIBptr Pointeur sur un buffer dans lequel GS/OS nous retourne des informations au sujet du bloc d'information du dispositif étendu.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$11	Le numéro de Référence du Device n'est pas valide.

Autres codes d'erreur que peut retourner la fonction

\$07

Exemple de programme

On peut utiliser la commande DInfo pour connaître les noms de tous les dispositifs connectés au système. Pour cela, il faut faire une série d'appels à DInfo en incrémentant à chaque appel *dev_num* de 1, jusqu'à ce que DInfo retourne un code d'erreur de \$11 (Numéro de Référence du Device non valide). Le premier *dev_num* à passer DInfo doit être 1 puisque c'est le numéro de Device que GS/OS attribue au premier Device qu'il trouve au moment du chargement. Il ne faut pas oublier que le nombre de dispositifs actifs connectés au système peut changer pendant l'exécution du programme. Par exemple,

dans un réseau, des Volumes peuvent se connecter ou se déconnecter ,à tout moment. Ainsi, quand on veut constituer une liste de dispositifs connectés au système, il est préférable de constituer cette liste à chaque fois que l'on en a besoin plutôt que de la constituer une fois pour toute au tout début du programme.

```

                LDA  #$01
                STA  DevNum

Loop           JSL  $E100A8      ; Point d'entrée GS/OS
                DA   $202C      ; Code de la commande DInfo
                ADRL ParmTbl    ; Adresse de la table de paramètres
                BCS  Exit       ; Retenue = 1 en cas d'erreur

                ...            ; Ici on mettrait le code pour afficher le nom
                                ; des dispositifs

                INC  DevNum
                BRA  Loop

Exit          RTS

ParmTbl       DA   $0A          ; Nombre de paramètres (10)
DevNum        DA   $01         ; Numéro du dispositif
                ADRL Dev_Space  ; Pointeur sur le buffer qui contient le nom
Dev_Space     DA   35          ; Taille du buffer
Dev_Name      DS   33          ; Le nom du dispositif est stocké ici

```

DRead

Code de fonction GS/OS: \$202F

Fonction

Permet de réaliser des opérations de lecture de bas niveau sur un dispositif (Device) GS/OS.

Paramètres

Offset	Nom	Octets	E/S	Description
0	<i>pcount</i>	2	E	Nombre de paramètres (6)
2	<i>dev_num</i>	2	E	Numéro de référence du dispositif (Device)
4	<i>buffer</i>	4	E	Adresse du buffer dans lequel on rangera les octets lus
8	<i>request_count</i>	4	E	Nombre d'octets à lire
12	<i>starting_block</i>	4	E	Premier numéro de bloc à lire
16	<i>block_size</i>	2	E	Nombre d'octets par bloc
18	<i>transfer_count</i>	4	S	Nombre d'octets lus

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 6 pour cette fonction.

dev_num Numéro de Référence du Device.

buffer Pointeur vers un buffer de sortie de classe 0 dans lequel les données vont être lues.

request_count Nombre d'octets à lire.

starting_block Pour un dispositif organisé en blocs, c'est le numéro de bloc où va commencer la lecture. Pour les autres devices, ce champ n'est pas utilisé.

block_size Taille d'un bloc en nombre d'octets.

transfer_count Nombre d'octets qui ont été lus sur le Device.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$11	Le numéro de Référence du Device n'est pas valide.

Autres codes d'erreur que peut retourner la fonction

\$07

Exemple de programme

DRead est surtout utilisée pour lire le contenu des blocs d'un volume disque. Voici une sous-routine GS/OS permettant de lire les blocs 6 et 7 sur un volume disque organisé en blocs de 512 octets.

	JSL	\$E100A8	; Point d'entrée GS/OS
	DA	\$202F	; Code de la commande DRead
	ADRL	ParmTbl	; Adresse de la table de paramètres
	RTS		
ParmTbl	DA	\$06	; Nombre de paramètres (6)
	DA	\$02	; Numéro du dispositif
	ADRL	Buffer	; Pointeur sur le buffer
	ADRL	1024	; Lecture de 1024 octets
	ADRL	100	; En commençant à partir du bloc 100
	DA	512	; 512 octets par bloc
	DS	4	; Espace pour le résultat transfer_count
Buffer	DS	1024	; Les données lues du disque sont stockées ici

Commentaire

Sous ProDOS 8, on utilise la commande READ_BLOCK.

DStatus

Code de fonction GS/OS: \$202D

Fonction

Permet de déterminer les caractéristiques d'un dispositif (Device) GS/OS.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (5)
2	dev_num	2	E	Numéro de référence du dispositif (Device)
4	status_code	2	E	Code de contrôle
6	status_list	4	E	Pointeur sur la liste de contrôle
10	request_count	4	E	Taille de la liste de contrôle
14	transfer_count	2	S	Nombre d'octets transférés

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 5 pour cette fonction.

dev_num Numéro de Référence du dispositif (Device).

status_code Code indiquant quelle caractéristique est demandée.

Code	Caractéristique demandée
\$0000	Caractéristiques du dispositif (Device)
\$0001	Paramètres de configuration
\$0002	Caractéristique attente / pas d'attente
\$0003	Options de formatage
\$0004	Caractéristiques de la partition
\$0005-\$7FFF	Réservé
\$8000-\$FFFF	Caractéristiques d'appels spécifiques

status_list Pointeur de classe 0 sur un buffer contenant les caractéristiques que la commande retourne.

request_count Nombre d'octets de caractéristiques à retourner dans la Liste de Contrôle.

transfer_count Nombre d'octets retournés dans la Liste de Contrôle.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$11	Le numéro de Référence du Device n'est pas valide.
\$53	Paramètre hors des limites autorisées par GS/OS.

Autres codes d'erreur que peut retourner la fonction

\$07

DWrite

Code de fonction GS/OS: \$2030

Fonction

Permet de déterminer les caractéristiques d'un dispositif (Device) GS/OS.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (6)
2	dev_num	2	E	Numéro de référence du dispositif (Device)
4	buffer	4	E	Buffer de données
8	request_count	4	E	Nombre d'octets à écrire
12	starting_block	4	E	Premier numéro de bloc à écrire
16	block_size	2	E	Nombre d'octets par bloc
18	transfer_count	4	S	Nombre d'octets écrits

Description des paramètres

<i>pcount</i>	Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 6 pour cette fonction.
<i>dev_num</i>	Numéro de Référence du dispositif (Device).
<i>buffer</i>	Pointeur vers un buffer contenant les données qui vont être écrites.
<i>request_count</i>	Nombre d'octets à écrire.
<i>starting_block</i>	Pour un dispositif organisé en blocs, c'est le numéro de bloc où va commencer l'écriture. Pour les autres Devices, ce champ n'est pas utilisé.
<i>block_size</i>	Taille d'un bloc en nombre d'octets.
<i>transfer_count</i>	Nombre d'octets qui ont été écrits sur le dispositif (Device).

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$11	Le numéro de Référence du Device n'est pas valide.
\$53	Paramètre hors des limites autorisées par GS/OS.

Autres codes d'erreur que peut retourner la fonction
\$07

EndSession

Code de fonction GS/OS: \$201E

Fonction

Permet de réaliser toutes les opérations d'écriture de blocs disque qui n'ont pas été faites parce qu'une commande BeginSession est active.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (0)

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 0 pour cette fonction.

Principaux codes d'erreur que peut retourner la fonction

Aucun

EraseDisk

Code de fonction GS/OS: \$2025

Fonction

Permet d'écrire sur un disque le programme de boot, la Bit Map, et la structure du Catalogue principal vide. Cette commande efface donc le volume disque. Contrairement à la commande Format, EraseDisk n'initialise pas le disque; cela signifie qu'on ne peut l'utiliser qu'avec des disques ayant déjà été formatés.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (4)
2	dev_name	4	E	Pointeur sur le nom du Device
6	vol_name	4	E	Pointeur sur le nom du Volume
10	file_sys_id	2	S	Code ID pour le fichier système utilisé
12	requested_fsyz	2	E	Code ID pour le fichier système utilisé

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum pour cette fonction est 3 et la maximum 4.

dev_name Pointeur de classe 1 sur l'adresse de la chaîne décrivant le nom du Device à utiliser.

vol_name Pointeur de classe 1 sur l'adresse de la chaîne décrivant le nom du Volume à utiliser. Ce nom doit être précédé par un "/".

file_sys_id Si le champ requested_fsyz est à 0, GS/OS affiche une boîte de dialogue qui permet à l'utilisateur de choisir le fichier système utilisé sur le volume disque. Au retour, le champ file_sys_id indique quel type de file system a été sélectionné.

Valeur	Type de file system
\$01	ProDOS / SOS
\$02	DOS 3.3
\$03	DOS 3.2 / 3.1
\$04	Apple II Pascal
\$05	Macintosh MFS

Valeur	Type de file system
\$06	Macintosh HFS
\$07	Macintosh XL (Lisa)
\$08	Apple CP/M
\$09	Non utilisé
\$0A	MS-DOS
\$0B	High Sierra (CD-ROM)
\$0C	ISO 9660 (CD-ROM)

Si GS/OS retourne 0 dans ce champ, cela signifie que l'utilisateur a annulé l'opération.

requested_fsys Ce champ contient le code ID du fichier système écrire sur le Volume Disque. Ce code est le même que pour *file_sys_id*. Si ce champ est à 0, GS/OS affiche une boîte de dialogue permettant de choisir le type de fichier système ; GS/OS retourne le code ID choisi dans le champ *file_sys id*.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$10	Le nom de Device indiqué n'existe pas.
\$40	Le nom de Volume indiqué contient des caractères non valides, ou ne commence par un séparateur valide ("/" ou ".").
\$5D	Le fichier système indiqué n'est pas reconnu.

Autres codes d'erreur que peut retourner la fonction

\$07, \$11, \$27

Exemple de programme

Supposez que nous voulions effacer la disquette placée dans un device appelé .APPLEDISK3.5A et que nous allons lui donner comme nom :BLANK. Voici le code à utiliser.

```
JSL  $E100A8      ; Point d'entrée GS/OS
DA   $2025        ; Code de la commande EraseDisk
ADRL ParmTbl      ; Adresse de la table de paramètres
RTS
```

```
ParmTbl  DA  $04      ; Nombre de paramètres (4)
```

	ADRL	DevName	; Pointeur sur le nom du dispositif (Device)
	ADRL	VolName	; Pointeur sur le nom du Volume
	DS	2	; file_sys_id
	DA	0	; Signifie que l'utilisateur peut choisir
DevName	ASC	“.APPLEDISK3.5A”	
VolName	ASC	“:BLANK”	

ExpandPath

Code de fonction GS/OS: \$200E

Fonction

Permet de convertir un nom de fichier, un pathname partiel, ou un pathname complet, en un pathname complet utilisant le caractère “:” comme séparateur.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (3)
2	input_path	4	E	Pathname devant être converti
6	output_path	4	E	Pointeur sur le pathname converti
10	flags	2	E	Indicateur de conversion en majuscules

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum pour cette fonction est 2 et la maximum 3.

input_path Pointeur sur un buffer de classe 1 contenant la chaîne de caractères qui représente le pathname à convertir.

output_path Pointeur sur un buffer de classe 1 où GS/OS va retourner le pathname converti.

flags Le bit 15 de cet indicateur définit si les caractères en minuscules doivent être convertis en majuscules. Si ce bit est à 1, la conversion doit se faire.

Les bits 0-14 doivent toujours être à zéro.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$40	Le nom de Volume indiqué contient des caractères non valides, ou ne commence par un séparateur valide (“/” ou “.”).
\$4F	Le buffer de sortie de classe 1 est trop petit pour contenir le résultat.

Flush

Code de fonction GS/OS: \$2015

Fonction

Force le système d'exploitation à écrire le contenu du buffer fichier sur le Volume Disque; le fichier n'est pas fermé.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	ref_num	2	E	Numéro de référence du fichier

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

ref_num Numéro de Référence attribué au fichier quand celui-ci a été ouvert.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le disque est protégé en écriture.
\$43	Le Numéro de Référence du fichier n'est pas valide. Vous devez utiliser un Numéro de Référence d'un fichier qui a déjà été fermé.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$48

Format

Code de fonction GS/OS: \$2024

Fonction

Permet de formater un disque, d'y écrire le programme de boot, la Bit Map, et une structure de Catalogue vide pour le système d'exploitation indiqué.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (4)
2	dev_name	4	E	Pointeur sur le nom du dispositif (Device)
6	vol_name	4	E	Pointeur sur le nom du Volume
10	file_sys_id	2	S	Code ID pour le file system utilisé
12	requested_fsys	2	E	Code ID pour le file system utilisé

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum pour cette fonction est 3 et la maximum 4.

dev_name Pointeur de classe 1 sur l'adresse de la chaîne décrivant le nom du dispositif (Device) à utiliser.

vol_name Pointeur de classe 1 sur l'adresse de la chaîne décrivant le nom du Volume à utiliser. Ce nom doit être précédé par un "/".

file_sys_id Si le champ *requested_fsys* est à 0, GS/OS affiche une boîte de dialogue qui permet à l'utilisateur de choisir le fichier système utilisé sur le Volume Disque. A u retour, le champ *file_sys_id* indique quel type de file system a été choisi.

Valeur	Type de file system
\$01	ProDOS / SOS
\$02	DOS 3.3
\$03	DOS 3.2 / 3.1
\$04	Apple II Pascal
\$05	Macintosh MFS
\$06	Macintosh HFS

Valeur	Type de file system
\$07	Macintosh XL (Lisa)
\$08	Apple CP/M
\$09	Non utilisé
\$0A	MS-DOS
\$0B	High Sierra (CD-ROM)
\$0C	ISO 9660 (CD-ROM)

Si GS/OS retourne 0 dans ce champ, cela signifie que l'utilisateur a annulé l'opération.

requested_fsys Ce champ contient le code ID du file system à écrire sur le Volume Disque. Ce code est le même que pour file_sys_id. Si ce champ est à 0, GS/OS affiche une boîte de dialogue permettant de choisir le type de fichier système. ; GS/OS retourne le code ID choisi dans le champ file_sys_id.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$10	Le nom de Device indiqué n'existe pas.
\$40	Le nom de Volume indiqué contient des caractères non valides, ou ne commence pas par un séparateur valide ("/" ou ":").
\$5D	Le file system indiqué n'est pas reconnu.

Autres codes d'erreur que peut retourner la fonction

\$07, \$11, \$27

FSTSpecific

Code de fonction GS/OS: \$2033

Fonction

Permet de réaliser des opérations propres à un FST (File System Translator). Cette commande permet de communiquer avec les FST. La nature de ces opérations varie d'un FST à un autre.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (3)
2	file_sys_id	2	E	Code ID du fichier système
4	command_num	2	E	Numéro de la commande du FST
6	command_parm	2	E/S	Paramètre de la commande ou Résultat

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 3 pour cette fonction.

file_sys_id Ce champ indique le file system que le FST implémente:

Valeur	Type de file system
\$01	ProDOS / SOS
\$02	DOS 3.3
\$03	DOS 3.2 / 3.1
\$04	Apple II Pascal
\$05	Macintosh MFS
\$06	Macintosh HFS
\$07	Macintosh XL (Lisa)
\$08	Apple CP/M
\$09	Non utilisé
\$0A	MS-DOS
\$0B	High Sierra (CD-ROM)
\$0C	ISO 9660 (CD-ROM)

command_num Ce champ contient un numéro de commande spécifique au FST.

command_parm Ceci peut être une entrée ou un champ de résultat; cela dépend de *command_num*. Sa signification dépend du FST avec lequel vous communiquez.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$53	Paramètre non valide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$54

GetBootVol

Code de fonction GS/OS: \$2028

Fonction

Permet de déterminer le nom du Volume Disque à partir duquel GS/OS a été lancé.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	vol_name	4	S	Pointeur sur le nom du volume

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 3 pour cette fonction.

vol_name Pointeur sur un buffer de classe 1 où GS/OS retourne le nom du Volume Disque; ce nom est précédé et suivi par le séparateur ":". Le buffer doit avoir 35 octets de longueur. Le nom de Volume que GetBootVol retourne est le même nom que GS/OS attribue au préfixe */ lors du boot.

Codes d'erreur que peut retourner la fonction

\$07

Exemple de programme

Ce programme montre comment utiliser la commande GetBootVol. En sortie, le nom de Volume est rangé à BootName, il est précédé par un mot indiquant la longueur du nom de Volume.

```
JSL $E100A8 ; Point d'entrée GS/OS
```

```
DA $2028 ; Code de la commande GetBootVol
```

```
ADRL ParmTbl ; Adresse de la table de paramètres
```

```
RTS
```

```
ParmTbl DA $01 ; Nombre de paramètres (1)
```

```
ADRL BootSpace ; Pointeur sur le buffer de sortie
```

```
BootSpace DA 35
```

```
BootName DS 33 ; Emplacement pour le nom
```

GetDevNumber

Code de fonction GS/OS: \$2020

Fonction

Permet de déterminer le numéro de référence d'un device en indiquant un nom de device ou un nom de volume.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (2)
2	dev_name	4	E	Pointeur sur le nom du volume
6	dev_num	2	S	Numéro de référence du dispositif (Device)

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 2 pour cette fonction.

dev_name Pointeur de classe 1 pointant sur l'adresse du buffer contenant le nom du Device ou le nom du Volume. Un nom de Volume doit être précédé par "/" ou "."

dev_num Numéro de Référence du dispositif (Device) retourné par la commande.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$10	Aucun dispositif ne possède le nom indiqué
\$40	Le nom indiqué contient des caractères non valides
\$45	Le Volume indiqué ne peut être trouvé

Autres codes d'erreur que peut retourner la fonction

\$07, \$11

Exemple de programme

Voici un exemple de programme pour déterminer le Numéro de Référence d'un Device contenant un disque appelé /PIMP :

```
JSL $E100A8 ; Point d'entrée GS/OS
```

```
DA $2020 ; Code de la commande GetDevNumber
```

ADRL ParmTbl ; Adresse de la table de paramètres

RTS

ParmTbl DA \$02 ; Nombre de paramètres (2)

Adr1 VolName

DS 2 ; Le numéro de référence du dispositif est
; retourné ici

VolName ASC "/PIMP"

GetDirEntry

Code de fonction GS/OS: \$201C

Fonction

Permet de lire les caractéristiques d'un fichier du Catalogue. GS/OS retourne les informations contenues au catalogue pour un fichier particulier.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (17)
2	ref_num	2	E	Numéro de référence du fichier
4	flags	2	S	Indicateur pour fichier étendu
6	base	2	E	Code de Base
8	displacement	2	E	Code de Déplacement
10	name_buffer	4	E	Pointeur sur le Nom de fichier
14	entry_num	2	S	Numéro absolu d'Entrée au Catalogue
16	file_type	2	S	Type de fichier
18	eof	4	S	Taille du fichier
22	block_count	4	S	Nombre de blocs utilisés par le fichier
26	create_td	8	S	Heure et Date de création
34	modify_td	8	S	Heure et Date de dernière modification
42	access	2	S	Code d'accès
44	aux_type	4	S	Type Auxiliaire de fichier
48	file_sys_id	2	S	Code ID du système d'exploitation
50	option_list	4	S	Pointeur sur la Liste Option
54	res_eof	4	S	Taille du segment Ressource
58	res_block_count	4	S	Nombre de blocs utilisés par le segment Ressource

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum de ce nombre est 5 pour cette fonction.

ref_num Numéro de Référence attribué au fichier quand celui ci a été ouvert.

<i>flags</i>	Le Bit 15 de ce mot indique si le fichier est de fichier étendu (bit 15 = 1), ou non (bit 15 = 0).
<i>base</i>	Ce code indique à GS/OS comment calculer le numéro de la prochaine entrée au catalogue à lire. Si base = 0, le déplacement est une entrée au catalogue absolue; si base = 1, GS/OS ajoute le déplacement au numéro courant de l'entrée pour calculer le numéro d'entrée suivant; si base = 2, GS/OS soustrait le déplacement au numéro courant de l'entrée pour calculer le numéro d'entrée suivant. Notez que GS/OS fixe le numéro courant d'entrée à 0 quand il ouvre un fichier pour la première fois, et qu'il le met à jour chaque fois que l'application appelle GetDirEntry.
<i>displacement</i>	Si base = 0, cela représente le numéro absolu de l'entrée au catalogue à retourner. Autrement, il représente le déplacement sur la prochaine entrée au catalogue à retourner, qui peut être positive ou négative selon la valeur de la base. Notez que si base et displacement sont tous les deux à 0, GS/OS retourne dans entry_num le nombre total d'entrées actives dans le sous-catalogue. Pour lire dans le catalogue une entrée après l'autre, fixez base et displacement à 1 et appelez GetDirEntry jusqu'à obtenir l'erreur \$61 (fin de catalogue).
<i>name_buffer</i>	Pointeur sur un buffer de sortie de classe 1 dans lequel GS/OS va ranger le nom de fichier qu'il a trouvé dans l'entrée au catalogue. Sous les systèmes ProDOS et GS/OS la taille du buffer doit être de 19 octets (15 pour le nom, 2 pour le mot de longueur, et 2 pour le mot indiquant la taille du buffer). GetDirEntry pouvant être utilisée pour lire les catalogues d'autres systèmes d'exploitation utilisant des noms de fichiers plus longs, il faudra ajuster en conséquence la taille de ce buffer. Si le buffer de sortie est trop petit, GetDirEntry retourne la longueur nécessaire au nom de fichier.
<i>entry_num</i>	Numéro absolu de l'entrée au catalogue de l'entrée courante.
<i>file_type</i>	Code indiquant le type de fichier.
<i>eof</i>	Valeur contenant la position EOF courante. Cette valeur est égale à la taille du fichier en octets. Si le fichier est de type étendu, ce champ ne s'applique qu'au segment de données.

block_count Ce champ contient le nombre total de blocs utilisés par le fichier pour le rangement de ses datas et de ses blocs d'index. Si le fichier est de type étendu, ce champ ne s'applique qu'au segment de données.

create_td Heure et date de création. Les 8 octets sont rangés dans l'ordre suivant:

Octet	Contenu	Commentaire
1	Secondes	
2	Minutes	
3	Heure	
4	Année	Année - 1990
5	Jour	Jour du mois - 1
6	Mois	0 = Janvier, 1 = Février, etc.
7	Non utilisé	
8	Jour de la semaine	1 = Dimanche, 2 = Lundi, etc.

modify_td Heure et date de dernière modification. Le format est le même que pour *create_td*.

access Code d'accès au fichier. Voir partie consacrée à l'organisation sur disque.

aux_type Type Auxiliaire de fichier.

file_sys_id Code d'identification du file system.

Valeur	Type de file system
\$01	ProDOS / SOS
\$02	DOS 3.3
\$03	DOS 3.2 / 3.1
\$04	Apple II Pascal
\$05	Macintosh MFS
\$06	Macintosh HFS
\$07	Macintosh XL (Lisa)
\$08	Apple CP/M
\$09	Non utilisé
\$0A	MS-DOS

Valeur	Type de file system
\$0B	High Sierra (CD-ROM)
\$0C	ISO 9660 (CD-ROM)

Toutes les autres valeurs sont réservées.

option_list Pointeur sur un buffer de sortie de classe 1 dans lequel GS/OS retourne des informations spécifiques utilisées par le FST accédant au fichier.

res_eof Valeur contenant la position EOF courante du segment de ressource pour un fichier de type étendu. Cette valeur est égale à la taille du segment de ressource en octets.

res_block_count Ce champ contient le nombre total de blocs utilisés par le segment ressource d'un fichier de type étendu pour les blocs de datas et les blocs d'index.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$4F	Le buffer est trop petit pour contenir le nom du fichier.
\$61	Fin de catalogue. Après avoir obtenu cette erreur, fermez le fichier sous-catalogue que vous avez ouvert avant d'appeler GetDirEntry.

Autres codes d'erreur que peut retourner la fonction

\$07, \$27, \$43, \$4A, \$4B, \$52, \$53, \$58

Exemple de programme

Voici une sous-routine GS/OS qui permet d'afficher tous les noms de fichiers se trouvant dans un sous-catalogue donné; on appelle continuellement la commande GetDirEntry. En entrée, le pointeur sur le pathname du sous-catalogue doit être dans les registres A (mot fort) et X (mot faible).

```
STX  Name_Ptr      ; Pointeur sur le pointeur
```

```
STA  Name_Ptr+2
```

```
JSL  $E100A8      ; Point d'entrée GS/OS
```

```
DA   $2010        ; Code de la commande Open
```

```
Adr1 ParmTbl1     ; Adresse de la table de paramètres
```

```
LDA  ref_num
```

STA ref_num1

STA ref_num2

Lecture JSL \$E100A8 ; Point d'entrée GS/OS

DA \$201C ; Code de la commande GetDirEntry

Adr1 ParmTbl2 ; Adresse de la table de paramètres

BCS Exit

...

Code que l'on écrirait pour afficher les résultats

...

BRA Lecture

Exit JSL \$E100A8 ; Point d'entrée GS/OS

DA \$2014 ; Close

Adr1 ParmTbl3 ; Adresse de la table de paramètres

RTS

ParmTbl1 DA \$02 ; Nombre de paramètres (2) pour Open

ref_num DS 2 ; Numéro de référence

Name_Ptr DS 4 ; Pointeur sur le pathname

ParmTbl2 DA \$05 ; Nombre de paramètres (5) pour GetDirEntry

ref_num2 DS 2 ; Numéro de Référence

DS 2 ; Flags

DA 1 ; Base = incrémente

DA 1 ; Displacement = +1

Adr1 NameBuff ; Pointeur sur le buffer contenant le nom

ParmTbl3 DA 1 ; Nombre de paramètres (1) pour Close

ref_num1 DS 2

NameBuff DA 19 ; Taille du buffer

DS 2 ; Longueur

DS 15 ; Nom de fichier

GetEOF

Code de fonction GS/OS: \$2019

Fonction

Permet de déterminer dans un fichier ouvert la valeur courante du pointeur EOF. Cette valeur représente la taille du fichier.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (2)
2	ref_num	2	E	Numéro de référence du fichier
4	eof	4	S	Position EOF

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 2 pour cette fonction.

ref_num Numéro de Référence attribué au fichier quand celui ci a été ouvert.

eof Valeur représentant la position courante EOF. Cette valeur est égale à la taille du fichier en nombre d'octets.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$43	Le numéro de Référence du fichier n'est pas valide. Vous devez très certainement utiliser un numéro d'un fichier qui a été fermé.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07

Exemple de programme

Ce programme permet de connaître la taille d'un fichier ayant 1 comme Numéro de Référence.

```
JSL $E100A8 ; Point d'entrée GS/OS
```

```
DA $2019 ; Code de la commande GetEOF
```

```
Adr1 ParmTbl ; Adresse de la table de paramètres
```

```
BCS Error
```

```
RTS
```

ParmTbl	DA	2	; Nombre de paramètres (2) pour GetEOF
	DA	1	; Numéro de Référence du fichier
Position	DS	4	; Position EOF courante

GetFileInfo

Code de fonction GS/OS: \$2006

Fonction

Permet de retrouver l'information relative à un fichier stockée dans l'Entrée au catalogue de ce fichier. Cela comprend le code d'Accès, le code du Type de fichier, le code du Type Auxiliaire de fichier, le code du Type d'Enregistrement, le nombre de blocs utilisés par le fichier, et la Date et l'Heure de création et de dernière modification de ce fichier.

Quand un Pathname pointe sur le nom d'un Volume plutôt que sur un nom de fichier, la taille du Volume (en blocs) est retournée dans le champ `aux_type`; le nombre de blocs utilisés par tous les fichiers de ce volume est retourné dans le champ `blocks_used`.

Paramètres

Offset	Nom	Octets	E/S	Description
0	<code>pcount</code>	2	E	Nombre de paramètres (12)
2	<code>pathname</code>	4	E	Pointeur sur le pathname
6	<code>access</code>	2	S	Code d'Accès
8	<code>file_type</code>	2	S	Code Type de fichier
10	<code>aux_type</code>	4	S	Code Type Auxiliaire de fichier
14	<code>storage_type</code>	2	S	Code Type d'Enregistrement
16	<code>create_td</code>	8	S	Heure et Date de Création
24	<code>modify_td</code>	8	S	Heure et Date de dernière Modification
32	<code>option_list</code>	4	E	Pointeur sur la Liste Option
36	<code>eof</code>	4	S	Taille du fichier
40	<code>blocks_used</code>	4	S	Nombre de blocs utilisés par le fichier
44	<code>resource_eof</code>	4	S	Taille du segment de ressource
48	<code>resource_blocks</code>	4	S	Nombre de blocs du segment de ressource

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum pour cette fonction est 2 et la maximum 12.

pathname Pointeur de classe 1 pointant sur l'adresse de la chaîne décrivant le Pathname du fichier à utiliser. Si le Pathname spécifié n'est pas précédé par un séparateur (“/” ou “:”), le système d'exploitation ajoute

automatiquement le nom du préfixe par défaut (le préfixe "0/") pour créer le Pathname complet.

access Code d'Accès au fichier. Voir les explications correspondantes dans la partie consacrée à l'organisation des fichiers sur un Volume.

file_type Code du Type de fichier.

aux_type Code du Type auxiliaire de fichier.

storage_type Code décrivant l'organisation physique du fichier sur disque:

Type	Description
\$01	Fichier "seedling"
\$02	Fichier "sapling"
\$03	Fichier "tree"
\$04	Partition Pascal
\$05	Fichier de type étendu
\$0D	Fichier Sous-Catalogue
\$0F	Fichier catalogue du Volume

create_td *Heure et date de création. Les 8 octets sont rangés dans l'ordre suivant:*

Octet	Contenu	Commentaire
1	Secondes	
2	Minutes	
3	Heure	
4	Année	Année - 1990
5	Jour	Jour du mois - 1
6	Mois	0 = Janvier, 1 = Février, etc.
7	Non utilisé	
8	Jour de la semaine	1 = Dimanche, 2 = Lundi, etc.

modify_td Heure et date de dernière modification. Le format est le même que pour *create_td*.

option_list Pointeur sur un buffer de sortie de classe 1 où GS/OS retourne des informations spécifiques utilisées par le FST pour accéder au fichier.

- eof* Taille du fichier en nombre d'octets. Si le fichier est de type étendu, cette valeur ne concerne que le segment de datas. Ce champ n'a pas de signification pour un fichier sous-catalogue.
- blocks_used* Ce champ contient le nombre total de blocs utilisés par le fichier pour le rangement des données et des blocs d'index. Si le fichier est de type étendu, alors il représente seulement le nombre de blocs utilisés par le segment de données. Ce champ n'est pas défini pour un fichier sous-catalogue.
- resource_eof* Si le fichier est de type étendu, cette valeur est la taille du segment de ressource.
- resource_blocks* Si le fichier est de type étendu, cette valeur est le nombre de blocs utilisés par le segment de ressource.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$40	Le Pathname contient des caractères non valides, ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.
\$46	Le fichier n'a pas été trouvé.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$4A, \$4B, \$52, \$53, \$58.

Exemple de programme

Voir l'exemple donné pour la commande SetFileinfo.

GetFSTInfo

Code de fonction GS/OS: \$202B

Fonction

Permet d'obtenir des informations générales sur les caractéristiques d'un FST (File System Translator).

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (8)
2	FST_Num	2	E	Numéro de référence du FST
4	file_sys_id	2	S	ID du fichier système
6	FST_Name	4	E	Pointeur sur le nom du FST
10	version	2	S	Numéro de version du FST
12	attributes	2	S	Attributs du FST
14	block_size	2	S	Taille en blocs
16	max_vol_size	4	S	Taille maximum d'un volume
20	max_file_size	4	S	Taille maximum d'un fichier

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum pour cette fonction est 2 et la maximum 8.

FST_num Numéro de Référence du FST. GS/OS attribue des numéros de référence consécutifs aux différents FST qu'il trouve dans le système.

file_sys_id Code d'identification pour le fichier système reconnu par le FST:

Valeur	Type de file system
\$01	ProDOS / SOS
\$02	DOS 3.3
\$03	DOS 3.2 / 3.1
\$04	Apple II Pascal
\$05	Macintosh MFS
\$06	Macintosh HFS

Valeur	Type de file system
\$07	Macintosh XL (Lisa)
\$08	Apple CP/M
\$09	Non utilisé
\$0A	MS-DOS
\$0B	High Sierra (CD-ROM)
\$0C	ISO 9660 (CD-ROM)

FST_Name Pointeur sur un buffer de sortie de classe 1 où GS/OS retourne le nom du FST.

version Numéro de version du FST:

Bit	Description
15	1=Version prototype, 0=Version finale
14-8	Numéro de version principale
7-0	Numéro de version secondaire

attributes Attributs du FST:

Bit	Description
15	1 = les noms de fichiers doivent être en Majuscules
14	FST Caractères; 0 = PST blocks
12	Numéro de version secondaire

block_size Taille en octets du nombre de blocks que le FST peut gérer.

max_vol_size Taille maximum en nombre de blocs des Volumes Disque que le PST peut gérer.

max_file_size Taille maximum en nombre d'octets du nombre de fichiers que le FST peut gérer.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$53	Paramètres hors des limites admises par GS/OS

Autres codes d'erreur que peut retourner la fonction

\$07.

Commentaire

GS/OS n'offre pas un moyen facile pour déterminer le nombre de FST actifs. Pour avoir des informations sur tous les FST, appelez continuellement GetFSTInfo jusqu'à obtenir une erreur \$53.

GetLevel

Code de fonction GS/OS: \$201B

Fonction

Permet de déterminer la valeur du niveau d'un fichier système.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	level	2	S	Niveau du fichier système

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

level Valeur du Niveau du fichier système. Les valeurs retournées peuvent être comprises entre \$0000 et \$00FF.

Codes d'erreur que peut retourner la fonction

\$07.

Exemple de programme

Voici une sous-routine sous GS/OS permettant de retourner le Niveau du fichier système:

```
JSL    $E100A8    ; Point d'entrée GS/OS
DA     $201B      ; Code de la commande GetLevel
Adr1   ParmTbl    ; Adresse de la table de paramètres
BCS    Error
...
Error  RTS

ParmTbl  DA    1    ; Nombre de paramètres (1) pour GetLevel
level    DS    2    ; Niveau du fichier Système retourné ici
```

GetMark

Code de fonction GS/OS: \$2017

Fonction

Permet de déterminer la valeur courante du pointeur MARK d'un fichier ouvert.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (2)
2	ref_num	2	E	Numéro de Référence du fichier
4	position	4	S	Position courante du Pointeur MARK

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 2 pour cette fonction.

ref_num Valeur du Niveau du fichier système. Les valeurs retournées peuvent être comprises entre \$0000 et \$00FF.

position Position courante de MARK en octets.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$43	Le Numéro de Référence du fichier n'est pas valide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07.

GetName

Code de fonction GS/OS: \$2027

Fonction

Permet d'obtenir le nom de l'application en cours d'exécution.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	data_buffer	4	E	Pointeur sur le nom de l'application

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

data_buffer Pointeur sur un buffer de classe 1 dans lequel GS/OS place le nom de l'application courante. Le nom est une chaîne ASCII précédée par un mot de longueur. Ce buffer doit avoir 35 octets de longueur pour pouvoir y placer le nom de l'application.

Codes d'erreur que peut retourner la fonction

\$07, \$4F

Exemple de programme

Une application en exécution doit parfois déterminer son propre nom:

JSL	\$E100A8		; Point d'entrée GS/OS
DA	\$2027		; Code de la commande GetLevel
Adr1	ParmTbl		; Adresse de la table de paramètres
BCS	Error		
...			
Error	RTS		
ParmTbl	DA	2	; Nombre de paramètres (2) pour GetName
	Adr1	NameSpace	; Pointeur sur le buffer
NameSpace	DA	35	; Taille du buffer
Name	DS	33	; Espace pour ranger le nom

GetPrefix

Code de fonction GS/OS: \$200A

Fonction

Permet de déterminer le nom d' un Préfixe GS/OS (0/ à 31/).

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (2)
2	prefix_num	2	E	Numéro du Préfixe (0 à 31)
4	prefix	4	E	Pointeur sur le Préfixe

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 2 pour cette fonction.

prefix_num Numéro de Préfixe GS/OS (0 à 31). C'est un nombre binaire et pas une valeur ASCII suivie d'un /.

prefix Pointeur sur un buffer de sortie de classe 1 dans lequel le système d'exploitation va retourner le Préfixe.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$56	L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map, ou parce que le buffer ne débute pas sur un saut de page mémoire.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$4F, 53

Exemple de programme

Cette sous routine GS/OS va lire le Préfixe 7/ et le range dans un buffer débutant en PathName, précédé par un mot de longueur. Notez que si le Préfixe 7/ n'a pas été défini avec la commande SetPrefix, le mot de longueur du Préfixe retourné par GetPrefix est alors égal à 0.

```
JSL    $E100A8    ; Point d'entrée GS/OS
```

```
DA    $200A    ; Code de la commande GetPrefix
```

Adr1 ParmTbl ; Adresse de la table de paramètres

BCS Error

...

Error RTS

ParmTbl DA 2 ; Nombre de paramètres (2) pour GetName

DA 7 ; Numéro du Préfixe (7)

Adr1 PathBuf ; Pointeur sur le buffer

PathBuf DA 69 ; Taille du buffer

PathName DS 67

GetSysPrefs

Code de fonction GS/OS: \$200F

Fonction

Permet de déterminer les Préférences système.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	preference	2	S	Préférences système

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

preference Mot indiquant les Préférences système:

Bit	Description
15	1 = afficher la boite de dialogue Changement de Volume 0 = ne pas afficher cette boite de dialogue

Commentaires

Les commandes GS/OS qui utilisent des Pathnames comme paramètres d'entrée, affichent normalement une boite de dialogue demandant à l'utilisateur d'insérer le Volume Disque demandé si celui ci n'est pas en ligne. Si l'application est capable de gérer les erreurs "Volume non trouvé", il faut utiliser SetSysPrefs pour mettre à 0 le bit 15 du mot de Préférences système.

GetVersion

Code de fonction GS/OS: \$202A

Fonction

Permet de déterminer le numéro de version de GS/OS.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	version	2	S	Numéro de version GS/OS

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

version Numéro de la version courante de GS/OS. L'octet de poids fort contient le numéro de version secondaire; l'octet de poids faible contient le numéro de version principale. Par exemple, la version 2.1 serait codée par \$0201. Le bit 7 de l'octet de poids fort s'il est 1 indique que la version de GS/OS est un prototype (version beta).

Codes d'erreur que peut retourner la fonction

\$07

NewLine

Code de fonction GS/OS: \$2011

Fonction

Permet d'activer ou de désactiver le mode de lecture "newline". Quand le mode de lecture "newline" est activé, toutes les opérations de lecture successives se terminent à chaque fois qu'un caractère déterminé est lu; c'est le caractère "newline". Quand le mode de lecture "newline" est désactivé, toutes les opérations de lecture se terminent quand la position EOF (fin de fichier) est atteinte, ou quand le nombre indiqué de caractères a été lu.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (4)
2	ref_num	2	E	Numéro de Référence du fichier
4	enable_mask	2	E	Masque d'activation "newline"
6	num_chars	2	E	Nombre de caractères dans la table
8	newline_table	4	E	Pointeur sur la table

Description des paramètres

- pcount* Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 4 pour cette fonction.
- ref_num* Numéro de Référence que le système d'exploitation a attribué au fichier quand celui-ci a été ouvert.
- enable_mask* Une opération mathématique AND est faite avec cette valeur et chaque octet lu consécutivement dans le fichier. Si le résultat de l'opération AND est identique à l'un des caractères se trouvant dans la table newline table, l'opération de lecture s'arrête; autrement, la lecture continue normalement.
Exception: Si enable_mask est égal à 0, le mode de lecture "newline" est désactivé et les opérations de lecture ne sont pas affectées.
- num_chars* Nombre de caractères se trouvant dans la table "newline" des caractères. Si enable_mask est différent de 0, num_chars ne peut pas valoir 0.

new_line_table Pointeur sur une table des caractères "newline". Chaque caractère occupe un octet dans la table qui peut être constitué au maximum de 256 octets de long.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$43	Le Numéro de Référence du fichier n'est pas valide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07

Exemple de Programme

En général, on utilise la commande NewLine lorsque l'on veut lire une ligne à la fois dans un fichier texte. Chaque ligne dans un fichier texte est terminée par le code ASCII \$0D qui est le code du retour charriot. Pour cela, il suffit de mettre *enable_mask* égal à \$FF et *newline_char* à \$0D avant d'exécuter la commande NewLine. Certaines applications travaillent en ASCII négatif et utiliseront le code ASCII \$8D pour le retour charriot. Si vous voulez terminer une opération de lecture pour \$0D ou \$8D, utilisez un *newline_char* de \$0D et un *enable_mask* de \$7F.

JSL	\$E100A8		; Point d'entrée GS/OS
DA	\$2011		; Code de la commande NewLine
Adr1	ParmTbl		; Adresse de la table de paramètres
BCS	Error		
	...		
Error	RTS		
ParmTbl	DA	4	; Nombre de paramètres (4) pour NewLine
	DA	1	; Numéro de référence du fichier (1)
	DA	\$7F	; <i>enable_mask</i>
	DA	1	; Nb de caractères <i>new_line</i> dans la table
	Adr1	NL_Table	; Pointeur sur la table New_Line
NL_Table	DA	\$0D	; Retour charriot

Null

Code de fonction GS/OS: \$200D

Fonction

Permet d'exécuter des événements en attente dans la liste d'attente GS/OS et du Scheduler.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (0)

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 0 pour cette fonction.

Open

Code de fonction GS/OS: \$2010

Fonction

Permet de préparer un fichier pour des opérations de lecture et d'écriture à venir. Quand on ouvre un fichier, le pointeur MARK pointe sur le début du fichier (MARK = 0), et le niveau du fichier est égal au niveau du fichier système. La commande renvoie également tous les attributs du fichier.

En général on ne peut ouvrir qu'un fichier qui soit fermé, mais si un fichier est déjà ouvert et que son code d'accès d'autorisation d'écriture n'est pas à 1 (c'est à dire que l'on ne peut pas le modifier), ce fichier peut être ouvert plus d'une fois.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (15)
2	ref_num	2	S	Numéro de Référence du fichier
4	pathname	2	E	Pointeur sur le Pathname
8	request_access	2	E	Type d'accès demandé
10	ressource_num	2	E	Type de Segment
12	access_code	2	S	Code d'Accès
14	file_type	2	S	Type de Fichier
16	aux_type	4	S	Type Auxiliaire de Fichier
20	storage_type	2	S	Type d'Enregistrement
22	create_td	8	S	Heure et Date de Création
30	modify_td	8	S	Heure et Date de Modification
38	option_list	4	E	Pointeur sur la liste Option
42	eof	4	S	Taille du fichier
46	blocks_used	4	S	Blocs utilisés par le fichier
50	resource_eof	4	S	Taille du segment de Ressource
54	resource_blocks	4	S	Blocs utilisés par le segment Ressource

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum pour cette fonction est 2 et la maximum 15.

ref_num Numéro de Référence attribué au fichier par le système d'exploitation. De très nombreuses commandes utilisent ce Numéro de Référence pour identifier le fichier. Le niveau du fichier est le même que pour le fichier système.

pathname Pointeur de classe 1 pointant sur l'adresse de la chaîne décrivant le Pathname du fichier à utiliser. Si le Pathname spécifié n'est pas précédé par un séparateur ("/" ou ":"), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut ("0/") pour créer le Pathname complet.

request_access Ce mot décrit le mode d'accès:

Bit	Description
0	1 = Écriture demandée
1	1 = Lecture demandée

Par exemple, on ne peut pas demander une écriture sur un lecteur de CD-ROM. Si ce mot est égal à \$0000, l'accès demandé est le même que celui permis par *access_code*.

resource_num Si le fichier est de type étendu, ce mot indique à GS/OS quel segment il faut ouvrir:

Type	Description
\$0000	Ouvrir le segment de données
\$0001	Ouvrir le segment de ressources

access Code d'Accès au fichier. Voir les explications correspondantes dans la partie consacrée à l'organisation des fichiers sur un Volume.

file_type Code du Type de fichier.

aux_type Code du Type auxiliaire de fichier.

storage_type Code décrivant l'organisation physique du fichier sur disque:

Type	Description
\$01	Fichier "seedling"
\$02	Fichier "sapling"
\$03	Fichier "tree"

Type	Description
\$04	Partition Pascal
\$05	Fichier de type étendu
\$0D	Fichier Sous-Catalogue
\$0F	Fichier catalogue du Volume

create_td

Heure et date de création. Les 8 octets sont rangés dans l'ordre suivant:

Octet	Contenu	Commentaire
1	Secondes	
2	Minutes	
3	Heure	
4	Année	Année - 1990
5	Jour	Jour du mois - 1
6	Mois	0 = Janvier, 1 = Février, etc.
7	Non utilisé	
8	Jour de la semaine	1 = Dimanche, 2 = Lundi, etc.

modify_td

Heure et date de dernière modification. Le format est le même que pour *create_td*.

option_list

Pointeur sur un buffer de sortie de classe 1 où GS/OS retourne des informations spécifiques utilisées par le FST pour accéder au fichier.

eof

Taille du fichier en nombre d'octets. Si le fichier est de type étendu, cette valeur ne concerne que le segment de datas. Ce champ n'a pas de signification pour un fichier sous-catalogue.

blocks_used

Ce champ contient le nombre total de blocs utilisés par le fichier pour le rangement des données et des blocs d'index. Si le fichier est de type étendu, alors il représente seulement le nombre de blocs utilisés par le segment de données. Ce champ n'est pas défini pour un fichier sous-catalogue.

resource_eof

Si le fichier est de type étendu, cette valeur est la taille du segment de ressource.

resource_blocks

Si le fichier est de type étendu, cette valeur est le nombre de blocs utilisés par le segment de ressource.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$40	Le Pathname contient des caractères non valides, ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.
\$46	Le fichier n'a pas été trouvé.
\$50	Le fichier est ouvert. Cette commande ne fonctionne qu'avec des fichiers fermés.
\$56	L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map, ou parce que le buffer ne débute pas sur un saut de page mémoire.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$4A, \$4B, \$52.

Exemple de Programme

La sous-routine suivante ouvre un fichier appelé SESAME qui réside dans un sous catalogue ayant pour préfixe "0/".

JSL	\$E100A8		; Point d'entrée GS/OS
DA	\$2010		; Code de la commande Open
Adr1	ParmTbl		; Adresse de la table de paramètres
BCS	Error		
...			
Error	RTS		
ParmTbl	DA	2	; Nombre de paramètres (2) pour Open dans ce cas
	DS	2	; ref_num est retourné ici
	Adr1	Pathname	; Pointeur sur le Pathname
	DA	1	; Nb de caractères new_line dans la table
Pathname	ASC	"SESAME"	; Nom du fichier

GS/OS retourne un code d'erreur de \$46 si vous tentez d'ouvrir un fichier qui n'existe pas. Une fois que le fichier est ouvert, vous devez récupérer le Numéro de Référence du fichier pour pouvoir l'utiliser avec les autres commandes.

OSShutdown

Code de fonction GS/OS: \$2003

Fonction

Permet de déconnecter GS/OS avant de rebooter ou d'éteindre le système.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	shutdown_flag	4	E	Pointeur sur le Pathname suivant

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

shutdown_flag Les deux bits de poids faible de cet indicateur contrôlent le mécanisme de déconnection de GS/OS:

Bit	Description
0	1 = GS/OS est déconnecté et le système est rebooté 0 = GS /OS est déconnecté et l'utilisateur a le choix entre rebooter ou éteindre le système.
1	1 = Le Ram Disque est laissé intact 0 = Le Ram Disque est initialisé

Commentaires

Quand GS/OS est déconnecté, il écrit tous les blocs qui se trouvent dans la mémoire cache, il ferme tous les NDA, etc. Cette commande doit seulement être utilisée par les sélecteurs de programmes comme le Finder ou ProSel 16, et pas par les applications.

Quit

Code de fonction GS/OS: \$2029

Fonction

Permet de quitter l'application courante. Le contrôle est repassé au sélecteur de programme.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (2)
2	pathname	4	E	Pointeur sur le Pathname suivant
6	flags	2	E	Indicateurs Retour / Reboot

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum pour cette fonction est 0 et la maximum 2.

pathname Pointeur sur un buffer de classe 1 contenant le pathname du prochain fichier système à lancer. Le type de fichier doit être \$FF ou \$B3. Le pathname ne peut pas résider en page 2 car la commande Quit utilise cette zone mémoire.

flags Indicateurs du type de Quit; seuls les bits 15 et 14 sont significatifs. Si le bit 15 est à 1, le UserID du programme est placé dans la pile de Quit, afin que le programme puisse être relancé. Si le bit 14 est à 1, le programme est relancé à partir de la mémoire.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$46	Le fichier n'a pas été trouvé.
\$5C	Le fichier indiqué par le Pathname n'est pas un programme exécutable. Ce Pathname doit pointer sur un programme système ProDOS 8 (type de fichier \$FF) ou un programme système GS/OS (type de fichier \$B3).
\$5D	Le Pathname pointe sur un programme système ProDOS 8, mais le fichier P8 (qui contient le système d'exploitation ProDOS 8) n'est pas présent dans le sous-catalogue /SYSTEM du disque de boot de GS/OS.
\$5F	La pile des adresses de retour des Quit est saturée. GS/OS retourne cette erreur quand on tente de pousser un autre ID d'un programme sur cette pile quand celle ci est pleine.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$40, \$5E.

Read

Code de fonction GS/OS: \$2012

Fonction

Permet de lire dans un fichier ouvert des octets à partir de la position courante du pointeur MARK. Après l'opération de lecture, le système d'exploitation incrémente MARK du nombre d'octets qui ont été lus. L'opération de lecture se termine quand le nombre indiqué d'octets a été transféré, quand un caractère "newline" est rencontré ou quand la fin du fichier est atteinte.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (5)
2	ref_num	2	E	Numéro de Référence du fichier
4	data_buffer	4	E	Pointeur sur le début du buffer de données
8	request_count	4	E	Nombre d'octets à lire
12	transfer_count	4	S	Nombre d'octets lus
16	cache_priority	2	E	Niveau de priorité du cache

Description des paramètres

<i>pcount</i>	Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum pour cette fonction est 4 et la maximum 5.
<i>ref_num</i>	Numéro de Référence attribué au fichier par le système d'exploitation quand celui-ci a été ouvert.
<i>data_buffer</i>	Pointeur sur le début du buffer dans lequel les données du fichier vont être placées. La taille du buffer doit être égale à <i>request_count</i> .
<i>request_count</i>	Nombre de caractères à lire dans le fichier. Ils sont lus dans le fichier et placés dans le buffer pointé par <i>data_buffer</i> .
<i>transfer_count</i>	Nombre de caractères qui ont été lus dans le fichier. Il est en général égal à <i>request_count</i> , mais peut être inférieur si le système d'exploitation atteint la fin du fichier ou si le mode de lecture est actif et qu'un caractère "newline" est lu (voir la commande NewLine).

cache_priority Ce code indique comment GS/OS va utiliser le cache pour y placer les blocks lors de la lecture du fichier.

Code	Description
\$0000	Pas de cache pour ce fichier
\$0001	Le cache est utilisé pour ce fichier

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$43	Le numéro de référence d'un fichier n'est pas valide.
\$4C	La fin du fichier (EOF) a été atteinte durant une opération de lecture.
\$4E	Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier contrôle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.
\$56	L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map, ou parce que le buffer ne débute pas sur un saut de page mémoire.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27.

Exemple de Programme

La sous-routine suivante lit \$1000 octets dans un fichier ayant comme Numéro de Référence 1 à l'adresse mémoire Buffer.

	JSL	\$E100A8		; Point d'entrée GS/OS
	DA	\$2012		; Code de la commande Read
	Adr1	ParmTbl		; Adresse de la table de paramètres
	BCS	Error		
	...			
Error	RTS			
ParmTbl	DA	4		; Nombre de paramètres (4) pour Read dans ce cas
	DA	1		; Numéro de référence du fichier
	Adr1	Buffer		; Pointeur sur le Pathname
	Adel	\$1000		; Nombre d'octets à lire
TransCnt	DS	4		; Nb d'octets lus

Buffer DS \$1000 ; Buffer pour les données (4096 octets)

ResetCache

Code de fonction GS/OS: \$2026

Fonction

Permet de forcer la taille du cache utilisé par GS/OS; cette taille est stockée dans la BRAM.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (0)

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 0 pour cette fonction.

SessionStatus

Code de fonction GS/OS: \$201F

Fonction

Permet de déterminer si une session d'écriture est différée; c'est à dire si une commande BeginSession est active.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	status	2	S	Code de retour

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

status Ce code indique si une session d'écriture est différée.

Code	Description
\$0000	Session d'écriture différée active
\$0001	Session d'écriture différée non active

SetEOF

Code de fonction GS/OS: \$2018

Fonction

Permet de modifier le pointeur EOF d'un fichier ouvert. Si on diminue EOF, tous les blocks de données se trouvant après la nouvelle valeur EOF sont libérés. Si on augmente EOF, le système d'exploitation n'attribue pas de nouveaux blocks tant que d'autres données ne sont pas écrites dans ce fichier. Si la nouvelle valeur EOF est inférieure au pointeur MARK, ce dernier prend la valeur de EOF. On peut modifier la valeur EOF de tout fichier dont le bit d'accès d'écriture est à 1.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (3)
2	ref_num	2	E	Numéro de Référence du fichier
4	base	2	E	Code pour déterminer la nouvelle valeur EOF
6	displacement	2	E	Nouvelle valeur de EOF

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 3 pour cette fonction.

ref_num Numéro de Référence attribué au fichier par le système d'exploitation.

base Ce code indique à GS/OS comment déterminer la nouvelle valeur de EOF:

Code	Description
\$0000	Nouveau EOF = displacement
\$0001	Nouveau EOF = ancien EOF + displacement
\$0002	Nouveau EOF = MARK + displacement
\$0003	Nouveau EOF = MARK - displacement

displacement GS/OS utilise cette valeur en conjonction avec base pour déterminer la nouvelle valeur du pointeur EOF.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le disque est protégé contre l'écriture.
\$43	Le Numéro de Référence du fichier n'est pas valide.
\$4D	La valeur MARK spécifiée est supérieure à EOF.
\$4E	Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier contrôle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$4E.

SetFileInfo

Code de fonction GS/OS: \$2005

Fonction

Permet de modifier l'information concernant un fichier dans l'Entrée au catalogue.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (12)
2	pathname	4	E	Pointeur sur le Pathname
6	access	2	E	Code d'Accès
8	file_type	2	E	Type de fichier
10	aux_type	4	E	Type Auxiliaire de fichier
14	non utilisé	2	E	
16	create_dt	8	E	Date et Heure de Création
24	modify_dt	8	E	Date et Heure de Modification
32	option_list	4	E	Pointeur sur la liste Option
36	non utilisé	4	E	
40	non utilisé	4	E	
44	non utilisé	4	E	
48	non utilisé	4	E	

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum pour cette fonction est 2 et la maximum 12.

pathname Pointeur de classe 1 pointant sur l'adresse de la chaîne décrivant le Pathname du fichier à utiliser. Si le pathname spécifié n'est pas précédé par un séparateur ("/" ou ":"), le système d'exploitation ajoute automatiquement le nom du préfixe par défaut (sous GS/OS c'est le préfixe "0/") pour créer le Pathname complet.

access Code d'Accès au fichier. Voir explications dans la partie consacrée à l'organisation des fichiers sur un Volume.

file_type Code du Type de fichier

aux_type Code du Type auxiliaire de fichier.

non utilisé Ces octets ne sont pas utilisés. Ils permettent de conserver une symétrie avec la Table des Paramètres de la commande GetFileInfo.

create_dt Heure et date de création du fichier sous GS/OS. Ces 8 octets représentent les paramètres suivants:

Octet	Contenu	Commentaire
1	Secondes	
2	Minutes	
3	Heure	
4	Année	Année - 1990
5	Jour	Jour du mois - 1
6	Mois	0 = Janvier, 1 = Février, etc.
7	Non utilisé	
8	Jour de la semaine	1 = Dimanche, 2 = Lundi, etc.

modify_dt Heure et date de dernière modification du fichier sous GS/OS. Ces 8 octets sont rangés dans le même ordre que dans le champ *create_td*.

option_list Pointeur sur un buffer de sortie de classe 1 où GS/OS retourne des informations spécifiques utilisées par le FST pour accéder au fichier.

Note

Les paramètres non utilisés doivent être mis à 0.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le disque est protégé contre l'écriture.
\$40	Le Pathname contient des caractères non valides, Ou un Pathname complet n'est pas spécifié (et il n'y a pas de préfixe par défaut défini).
\$44	Un Catalogue dans un Pathname n'a pas été trouvé.
\$45	Le Catalogue principal n'a pas été trouvé.
\$46	Le fichier n'a pas été trouvé.
\$4E	Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier contrôle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$4A, \$4B, \$52, \$53, \$58.

SetLevel

Code de fonction GS/OS: \$201A

Fonction

Permet de fixer le niveau du fichier système. Le niveau de fichier système joue sur les commandes Open, Close, et Flush.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	level	2	E	Nouveau niveau du fichier système

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

level Valeur du niveau du fichier système. La valeur peut être comprise entre \$0000 et \$00FF.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$59	Niveau de fichier non valide.

Autres codes d'erreur que peut retourner la fonction

\$07.

Exemple de Programme

Voici comment mettre le niveau du fichier système à la valeur 2.

```
JSL $E100A8 ; Point d'entrée GS/OS
DA $201A ; Code de la commande SetLevel
Adr1 ParmTbl ; Adresse de la table de paramètres
BCS Error
...
Error RTS

ParmTbl DA 1 ; Nombre de paramètres (2) pour SetLevel
DA 2 ; Nouveau niveau du fichier système
```

SetMark

Code de fonction GS/OS: \$2016

Fonction

Permet de modifier la valeur du pointeur MARK dans un fichier ouvert. On peut utiliser cette commande pour se positionner n'importe où dans le fichier; les opérations de lecture et d'écriture prennent effet à partir de cette position.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (3)
2	ref_num	2	E	Numéro de Référence du fichier
4	base	2	E	Code pour déterminer MARK
6	displacement	4	E	Nouvelle valeur de MARK

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 3 pour cette fonction.

ref_num Numéro de Référence du fichier attribué par la système d'exploitation.

base Ce code indique à GS/OS comment déterminer la nouvelle valeur de MARK.

Code	Description
\$0000	Nouveau MARK = déplacement
\$0001	Nouveau MARK = EOF - déplacement
\$0002	Nouveau MARK = MARK + déplacement
\$0003	Nouveau MARK = MARK - déplacement

displacement GS/OS utilise cette valeur en conjonction avec base pour déterminer la nouvelle valeur du pointeur MARK.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$43	Le Numéro de Référence du fichier n'est pas valide.
\$4D	Le pointeur MARK est supérieur à EOF.

Autres codes d'erreur que peut retourner la fonction
\$04, \$07, \$27.

SetPrefix

Code de fonction GS/OS: \$2009

Fonction

Permet de fixer le Préfixe par défaut. Quand on appelle une commande, le système d'exploitation convertit le Pathname demandé en y ajoutant le Préfixe par défaut pour obtenir le Pathname complet.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (2)
2	prefix_num	2	E	Numéro de Préfixe (0 à 31)
4	prefix	4	E	Pointeur sur le Préfixe

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 2 pour cette fonction.

prefix_num Numéro de Préfixe (0 à 31).

prefix Pointeur de classe 1 pointant sur l'adresse de la chaîne décrivant le Préfixe à utiliser.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$40	La syntaxe du pathname n'est pas valide.
\$44	Le Pathname spécifié n'a pas été trouvé.
\$45	Le Volume spécifié n'a pas été trouvé.
\$46	Fichier non trouvé.
\$4B	Le type de fichier n'est pas valide ou n'est pas reconnu.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27, \$53.

SetSysPrefs

Code de fonction GS/OS: \$200C

Fonction

Permet de régler les Préférences système.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	preference	2	E	Préférences système

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

preference Mot indiquant les Préférences système:

Bit	Description
15	1 = afficher la boite de dialogue Changement de Volume 0 = ne pas afficher cette boite de dialogue

Commentaires

Les commandes GS/OS qui utilisent des Pathnames comme paramètres d'entrée, affichent normalement une boite de dialogue demandant à l'utilisateur d'insérer le Volume Disque demandé si celui ci n'est pas en ligne. Si l'application est capable de gérer les erreurs "Volume non trouvé", il faut utiliser SetSysPrefs pour mettre à 0 le bit 15 du mot de Préférences système.

UnbindInt

Code de fonction GS/OS: \$2032

Fonction

Permet d'enlever une sous-routine de gestion d'interruption.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (1)
2	int_num	2	E	Numéro de Référence de l'interruption

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. Ce nombre est toujours 1 pour cette fonction.

int_num Numéro de Référence que GS/OS a attribué à la sous-routine de gestion d'interruption.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$53	Le Numéro de Référence int_num n'est pas valide.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07.

Volume

Code de fonction GS/OS: \$2008

Fonction

Retourne les caractéristiques du disque volume.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (6)
2	dev_name	4	E	Pointeur sur le nom de Device
6	vol_name	4	E	Pointeur sur le nom de Volume
10	total_blocks	4	S	Taille du volume en Blocs
14	free_blocks	4	S	Nombre de Blocks inutilisés
18	file_sys_id	2	S	Code ID du système d'exploitation
20	block_size	2	S	Nombre d'octets par Blocs

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum pour cette fonction est 2 et la maximum 6.

dev_name Pointeur sur un buffer de classe 1 contenant le nom du dispositif (Device).

vol_name Pointeur sur un buffer de classe 1 où GS/OS retourne le nom du volume disque. Ce buffer doit avoir 35 octets.

total_blocks Nombre total de blocs sur le Volume Disque.

free_blocks Nombre de blocs inutilisés sur le Volume Disque. Pour le format FST cette valeur est toujours à 0.

file_sys_id Code d'identification du système d'exploitation utilisé sur le volume disque.

Valeur	Type de file system
\$01	ProDOS / SOS
\$02	DOS 3.3
\$03	DOS 3.2 / 3.1

Valeur	Type de file system
\$04	Apple II Pascal
\$05	Macintosh MFS
\$06	Macintosh HFS
\$07	Macintosh XL (Lisa)
\$08	Apple CP/M
\$09	Non utilisé
\$0A	MS-DOS
\$0B	High Sierra (CD-ROM)
\$0C	ISO 9660 (CD-ROM)

block_size Taille d'un bloc sur disque en nombre d'octets.

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$10	Le nom de Device indiqué n'existe pas.
\$27	Une erreur I / O sur disque 5.25" est survenue empêchant le transfert correct des données. Le Volume disque est sans aucun doute abimé. On obtient aussi cette erreur s'il n'y a pas de disque dans le lecteur.
\$28	Pas de Device connecté.
\$2F	Le Device spécifié n'est pas en ligne. Cette erreur arrive s'il n'y a pas de disque dans le lecteur 3.5".

Autres codes d'erreur que peut retourner la fonction

\$07, \$11, \$2E, \$40, \$45, \$4A, \$52, \$55, \$57, \$58.

Write

Code de fonction GS/OS: \$2013

Fonction

Permet d'écrire des datas dans un fichier ouvert. L'écriture débute à la position courante de MARK. Le pointeur MARK est incrémenté au fur et à mesure de l'écriture.

Paramètres

Offset	Nom	Octets	E/S	Description
0	pcount	2	E	Nombre de paramètres (5)
2	ref_num	2	E	Numéro de Référence du fichier
4	data_buffer	4	E	Pointeur sur le buffer de données
8	request_count	4	E	Nombre d'octets à écrire
12	transfer_count	4	S	Nombre d'octets écrits
16	cache_priority	2	E	Niveau de priorité du cache

Description des paramètres

pcount Nombre de paramètres dans la Table de Paramètres GS/OS. La valeur minimum pour cette fonction est 4 et la maximum 5.

ref_num Numéro de Référence du fichier attribué par le système d'exploitation.

data_buffer Pointeur sur le début du bloc mémoire contenant les données à écrire sur le disque.

request_count Nombre de caractères à écrire sur disque.

transfer_count Nombre de caractères écrits sur disque.

cache_priority Ce code indique comment GS/OS va utiliser le cache pour y placer les blocks lors de la lecture du fichier.

Code	Description
\$0000	Pas de cache pour ce fichier
\$0001	Le cache est utilisé pour ce fichier

Principaux codes d'erreur que peut retourner la fonction

Code	Description
\$2B	Le disque est protégé contre l'écriture.

Code	Description
\$43	Le Numéro de Référence du fichier n'est pas valide.
\$48	Le Volume est plein.
\$4E	Le système d'exploitation ne peut pas accéder au fichier. Cette erreur arrive quand une opération interdite par le code d'accès fichier a été demandée. Le code d'accès fichier contrôle les commandes Rename, Destroy, Read, et Write. L'erreur peut aussi survenir si on tente de détruire un sous-catalogue qui n'est pas vide.
\$56	L'adresse d'un buffer n'est pas valide car il y a un conflit en mémoire dans les zones déclarées utilisées dans la Bit Map, ou parce que le buffer ne débute pas sur un saut de page mémoire.

Autres codes d'erreur que peut retourner la fonction

\$04, \$07, \$27.

Exemple de Programme

Cette sous-routine GS/OS écrit 256 octets dans un fichier dont le Numéro de Référence est 2; les données sont stockées dans l'espace mémoire Buffer.

	JSL	\$E100A8		; Point d'entrée GS/OS
	DA	\$2013		; Code de la commande Write
	Adr1	ParmTbl		; Adresse de la table de paramètres
	BCS	Error		
	...			
Error	RTS			
ParmTbl	DA	4		; Nombre de paramètres (4) pour Write
	DA	2		; Numéro de Référence du fichier
	Adr1	Buffer		; Pointeur sur les données à écrire
	Adr1	256		; Nombre d'octets à écrire
TransCnt	DS	4		; Espace où GS/OS ne retournera le nombre ; d'octets réellement écrits par le S0.
Buffer	DS	256		; Espace mémoire qui contient les données à ; écrire



VI

Son & Musique

Qu'ils s'agissent de programmes commerciaux (notamment dans les jeux) ou de ceux réalisés par des amateurs plus ou moins éclairés, vous avez pu vous rendre compte des capacités sonores et musicales de votre Apple IIGS. Je suis certain que vous n'avez pas été déçu par ses fantastiques possibilités dues en grande partie à son propre microprocesseur sonore: l'Ensoniq. Cependant il est curieux que peu de personnes se soient attaquées à la sonorisation, qui n'est pas aussi difficile qu'elle pourrait paraître. Une fois la méthode acquise, programmer le son sur Apple IIGS n'est plus que de la routine, sans pour autant devenir quelque chose de lassant.

Cependant, pour pouvoir profiter au maximum du potentiel sonore de votre Apple IIGS favori, vous ne devriez pas vous contenter du haut-parleur interne. Le mieux est de brancher le IIGS sur un amplificateur HIFI ou d'acheter spécialement des enceintes amplifiées. Et, si vous succomez à cet achat, pendant que vous y êtes achetez aussi une carte stéréo qui possède de nombreux avantages dont nous reparlerons.

6.1 Le son

Avant de vous lancer dans de fabuleuses musiques, vous devrez tout d'abord apprendre à programmer l'Ensoniq pour jouer des sons, puis créer vous même vos propres sons. A ce moment là, vous pourrez déjà apprécier toutes les possibilités de bruitages.

Heureusement, pour vous aider, Apple a doté le IIGS du fameux microprocesseur Ensoniq 5503 Digital Oscillator Chip, que nous appellerons désormais plus communément le DOC. Outre le fait d'être une puce utilisée dans plusieurs synthétiseurs, le DOC a l'avantage de pouvoir jouer

plusieurs sons en même temps, sans pour autant prendre de temps machine. C'est à dire que le microprocesseur 65C816 reste entièrement disponible pour d'autres tâches, une fois les commandes exécutées.

6.1.1 Description du système

L'Ensoniq DOC dispose d'un très bon potentiel sonore puisqu'il possède 32 oscillateurs indépendants les uns des autres. Chacun des oscillateurs pouvant produire un son, ceci signifie que vous pouvez avoir jusqu'à 32 sons différents joués en même temps ! Bien entendu si vous écoutez autant de sons, vous risquez de ne plus rien entendre du tout pour cause de non harmonisation. Par ailleurs, chaque oscillateur peut fonctionner avec un volume réglable et indépendant et selon quatre modes distincts.

Il faut remarquer que tous les registres du Sound GLU et du DOC sont sur 8 bits, donc tous les exemples de routines sont à considérer avec l'accumulateur et les registres d'index XY sur 8 bits, sauf changements précisés.

6.1.1.1 La RAM son

Le DOC possède sa propre mémoire RAM de 64K . Cependant cette RAM est assez particulière, car d'une part elle n'est pas adressable par le microprocesseur 65C816 et d'autre part elle n'est pas exécutable, c'est à dire que même si vous y stockez des programmes, ceux-ci ne pourront en aucun cas fonctionner. Il est également important de savoir que lors d'un démarrage à chaud (de type Ctrl-Reset) cette RAM Son n'est pas effacée. Par contre elle est initialisée avec des valeurs \$80 lors d'un démarrage à froid (de type Ctrl-Pomme-Reset ou lors de la mise sous tension de l'Apple). Etant donné que cette

mémoire RAM n'est pas adressable par le microprocesseur, il vous faudra passer par l'intermédiaire du Sound GLU et de ses commutateurs.

En ce qui concerne les valeurs à mettre en RAM Son, toute valeur de \$01 à \$FF est une constituante de son. Ainsi une onde sonore est une suite de valeurs qui sera ensuite convertie en un signal audible. La valeur \$00 quant à elle est réservée pour indiquer, si besoin, la fin du son.

Comme vous pouvez vous en douter, la RAM Son contiendra des sons, mais ceux-ci ne peuvent pas avoir n'importe quelle taille, ainsi les longueurs possibles sont: \$100 ; \$200; \$400 ; \$800 ; \$1000 ; \$2000 ; \$4000 ou \$8000 octets par son. Comme vous pouvez le voir, il n'y a pas de taille de 64 K possible normalement; cependant nous verrons que cela est tout de même réalisable avec une petite astuce. De plus les sons ne peuvent pas commencer n'importe où en RAM Son. Ainsi ils doivent obligatoirement commencer en début de page mémoire de la RAM Son.

Voici un petit tableau indiquant où peut commencer un son selon sa taille :

Par exemple un son de \$4000 de long (16384 octets) pourra commencer en RAM Son aux adresses \$0000 ; \$4000 ; \$8000 ; \$C000. Mais il ne pourra en aucun cas commencer en \$0003, ni \$1000, ni \$C500.

Ainsi vous remarquerez que plus un son est long, moins il y a de possibilités d'emplacements. Mais qu'arrive t'il si la taille d'un son ne correspond pas exactement aux longueurs standards ? Dans ce cas, il faut prendre la taille par excès et terminer le sons par des zéros.

Par exemple, si votre son occupe une taille de 13600 octets de long (\$3520), on considèrera que sa taille est de 16384 octets (\$4000) ; mais à partir du 13600 ème octet jusqu'au 16383 ème, vous devrez mettre des zéros. Evidemment les 2784 octets à \$00 sont gaspillés, sauf si vous arrivez à y loger un autre petit son de 2048 octets. Ainsi dans cet exemple, les deux sons pourraient occuper la RAM Son comme suit :

\$0000 : début du premier son de 13600 octets déclaré \$4000 de long

\$351F : fin du premier son

\$3520 : début de la zone des \$00 indiquant la fin du premier son.

\$37FF : fin de la zone des \$00

\$3800 : début du deuxième son de 2048 octets déclaré \$800 de long

\$3FFF : fin du deuxième son

Quoi qu'il en soit, la zone des \$00 représente encore 736 octets perdus où l'on pourrait éventuellement installer un son de 512 octets. La mise en place organisée des sons peut vraiment devenir un art! Aussi est- il préférable de connaître exactement la taille de tous les sons que l'on désire utiliser avant de commencer la programmation.

Malgré sa petite taille et sa relative mauvaise organisation, la RAM Son reste souvent suffisante pour la plupart des bruitages ou des musiques, pour peu que l'on choisisse bien ses sons.

6.1.1.2 Les commutateurs de contrôles (Sound GLU)

Le DOC possède ses propres registres de commandes permettant de le programmer complètement. En tout il y a 227 registres. Mais pour pouvoir accéder à ces registres, il vous faut passer par les quatre commutateurs du Sound GLU (General Logic Unit). Ce circuit est donc un intermédiaire entre l'Ensoniq DOC et le microprocesseur 65C816. Voici les quatre registres de commande du microprocesseur Ensoniq DOC.

Registres GLU	Adresse	Type
Registre de contrôle du son	\$C03C	lecture/écriture
Register de donnée	\$C03D	lecture/écriture
Registre pointeur d'adresse, octet bas	\$C03E	lecture/écriture
Registre pointeur d'adresse, octet haut	\$C03F	lecture/écriture

A noter que les commutateurs peuvent également être accédés en adressage long (exemple : \$E1C03C ou \$E0C03C), il n'est cependant pas conseillé de les accéder par un adressage indexé ou autre. L'accès à la Ram Son se fait également via ces registres.

6.1.1.2/1 Le commutateur de contrôle du son \$C03C

Ce commutateur contrôle d'une part le volume général du haut-parleur interne, mais contient également quelques informations relatives aux accès du DOC. Voici la description de ses bits :

Bit	Valeur	Description
7	1	Le DOC est occupé. Attendez jusqu'à ce qu'il soit libre.
	0	Le DOC est libre
6	1	Tous les accès se font en direction de la Ram Son
	0	Tous les accès se font en direction des registres du DOC
5	1	Auto-incrémentation des registres pointeurs activés
	0	Auto-incrémentation inactivée

Bit	Valeur	Description
4	-	Réservé, ne pas modifier
3-0	\$0-\$F	Contrôle du volume : \$0 faible, \$F fort

Le test du bit 7 de ce registre n'est normalement pas indispensable, car l'Ensoniq est assez rapide et rarement utilisé en rendement maximum. Ce bit est à lecture seulement, mais si vous écrivez dedans, rien ne se produira.

En ce qui concerne les bits 5 et 6, si vous voulez accéder aux registres du DOC, je vous conseille de les mettre tous deux à zéro. Par contre,

lors d'un transfert de sons de la mémoire principale vers la Ram Son, les mettre tous deux à un.

Le bit 4 est réservé.

Enfin, les bits 0 à 3 contiennent donc le volume général du haut-parleur interne du GS, et de la prise jack située à l'arrière. Cependant, il faut noter que ce volume n'a aucun effet dans le cas d'une carte stéréo, car les connecteurs reliés directement au DOC (sur la fiche interne à 7 pattes, de type connecteur Molex) ont un volume indépendant de celui du petit haut-parleur.

Si vous tenez à mettre comme volume général celui du réglage du tableau de bord, sachez que la mémoire \$E100CA contient dans ses bits 0 à 3 le volume.

Exemple d'accès aux registres du DOC :

```
LDAL $E100CA ; Lecture du volume du tableau de bord
AND #$0F ; On ne garde que les bits 0 à 3
STA $C03C ; Les bits 5 et 6 étant forcés à zéro, on
; accède aux registres du DOC
```

Exemple d'accès à la Ram Son :

```
LDAL $E100CA ; Volume...
AND #%00001111 ; On ne garde que les bits 0 à 3
ORA #%01100000 ; On force à 1 les bits 5 et 6
STA $C03C ; donc accède aux registres du DOC
```

6.1.1.2/2 Le commutateur de transfert des données \$C03D

Qu'ils s'agissent d'accès au DOC ou à la Ram Son, vous devrez passer des données, et c'est justement le rôle du commutateur \$C03D. Cependant ce commutateur ne fonctionne pas de la même façon selon que vous faites une lecture ou une écriture.

Dans le cas d'une écriture, il suffit simplement d'un STA \$C03D.

Exemple:

```
LDA  #$xx          ; xx étant une valeur hexadécimale quelconque
STA  $C03D         ; que l'on transfère dans un registre du DOC
                        ; ou dans une mémoire de la RAM Son
```

Mais lors d'une lecture, le comportement de \$C03D est différent, ainsi si vous voulez lire une donnée, le premier octet ne doit pas être pris en compte si vous venez de programmer les commutateurs de pointage (\$C03E ou \$C03F). Nous verrons des exemples dans le paragraphe suivant.

De plus dans le cas d'une lecture d'un registre du DOC, le problème est un peu plus complexe, car vous serez amené à modifier très souvent le commutateur \$C03E, je vous conseille donc très vivement d'effectuer systématiquement deux lectures pour lire une valeur !

6.1.1.2/3 Les commutateurs de pointages \$C03E et \$C03F

Les deux commutateurs \$C03E et \$C03F sont utilisés pour indiquer où seront transférées les données. Dans le cas d'un accès au DOC, seul le commutateur \$C03E est utilisé. Il suffit alors d'écrire dans le commutateur quel registre du DOC vous voulez accéder. Il est inutile de s'occuper du commutateur \$C03F, lors d'un accès au DOC puisque sa valeur n'est pas prise en compte.

Exemple d'accès à un registre du DOC :

```
LDAL $E100CA      ; Tout d'abord, indiquer que l'on veut accéder
AND  #$0F         ; aux registres du DOC
STA  $C03C
LDA  #$xx         ; xx étant un nombre hexadécimal de $00 à $E2
                        ; inclus, indiquant le numéro du registre du DOC
STA  $C03E        ; que l'on stocke dans $C03E
```

A ce stade là, le registre xx du DOC est prêt pour être lu ou écrit.

Voici également une routine de transfert de données entre la mémoire et le DOC :

*=====

* Ecriture des 224 premiers registres du DOC

*=====

XC ; Directive d'assemblage

XC ;

TAMPON =\$2000 ; Adresse de la mémoire tampon (ou ailleurs)

ORG \$1000 ; Le programme commence à cette adresse

; (ou à une autre adresse du banc \$00)

CLC ; Mode natif

XCE ;

SEP #\$30 ; A et X,Y sur 8 bits

LDAL \$E100CA ; Lecture du volume du tableau de bord

AND #\$0F ; Mise à zéro des bits 4 à 7 pour choisir le mode

STA \$C03C ; d'accès au DOC et sans auto-incrémentation

LDX #\$00 ; On veut écrire depuis le début du tampon

ENCORE STX \$C03E ; \$C03E contient le numéro du registre à écrire

LDA TAMPON,X ; Lecture d'une donnée du tampon (ici \$2000)

STA \$C03D ; Transfert de la donnée vers le DOC

INX ; Valeur suivante

CPX #\$E0 ; Jusqu'à ce que l'on ait transféré 224 valeurs

BCC ENCORE

RTS ; Fin de la routine

Cette routine permet de programmer les registres du DOC en un seul transfert, mais n'est vraiment utile que pour faire des essais. Vous remarquerez que seuls les 224 premiers registres du DOC sont concernés, les trois derniers registres étant spéciaux.

De la même façon, voici la routine inverse qui permet de lire les registres du DOC :

*=====

* Lecture des 224 premiers registres du DOC

*=====

XC ; Directive d'assemblage

XC ;

TAMPON	=\$2000	; Adresse de la mémoire tampon (ou ailleurs)
	ORG \$1000	; Le programme commence à cette adresse
		; (ou à une autre adresse du banc \$00)
	CLC	; Mode natif
	XCE	;
	SEP #\$30	; A et X,Y sur 8 bits
	LDAL \$E100CA	; Lecture du volume du tableau de bord
	AND #\$0F	; Mise à zéro des bits 4 à 7 pour choisir le mode
	STA \$C03C	; d'accès au DOC et sans auto-incrémentation
	LDX #\$00	; À partir du premier registre
ENCORE	STX \$C03E	; \$C03E contient le numéro du registre à lire
	LDA \$C03D	; Attention, il faut faire deux lectures pour ne
	LDA \$C03D	; pas avoir de problèmes!
	STA TAMPON,X	; Stockage de la valeur lue dans le tampon
	INX	; Valeur suivante
	CPX #\$E0	; 224 valeurs lues?
	BCC ENCORE	; Si oui, on a fini
	RTS	; Fin de la routine

Par contre, lors d'un accès a la Ram Son, l'emploi des commutateurs \$C03E et \$C03F constituent ensemble l'adresse 16 bits où doit être transférée la donnée. Ainsi \$C03E contient la partie basse de l'adresse tandis que \$C03F contient la partie haute.

De plus, ces deux commutateurs ont la possibilité d'être auto-incrémentés à chaque fois qu'une donnée est lue ou écrite par \$C03D. Ceci est particulièrement utile lorsque vous voulez transférer des données en Ram Son.

Voici l'exemple d'une routine de transfert de 64 K de sons du banc \$03 vers la Ram Son. La plupart du temps, c'est une routine de ce genre qui est utilisée pour installer d'un coup tous les sons. Bien entendu, il faut préalablement charger soit même les sons voulus dans le banc \$03.

```
*=====
* Ecriture en RAM Son de 64K de données
*=====
```

```

XC          ; Directive d'assemblage
XC          ;

MEMOIRE     =$030000      ; Banc où se trouvent les données que nous allons
                ; écrire (ou ailleurs)
ORG $1000   ; Le programme commence à cette adresse
                ; (ou à une autre adresse du banc $00)
CLC         ; Mode natif
XCE         ;
REP #30     ; Registres X,Y sur 16 bits
SEP #20     ; et A sur 8 bits.

LDAL $E100CA ; On choisit d'accéder à la RAM Son avec
AND #00001111 ; auto-incrémentation des pointeurs...
ORA #01100000
STA $C03C

STZ $C03E   ; Mise à zéro des deux pointeurs.
STZ $C03F

LDX #0000   ; On veut écrire depuis le début du banc.
ENCORE LDAL MEMOIRE,X ; Lecture d'une valeur du banc (ici le banc $03).
STA $C03D   ; Transfert en RAM Son. À noter qu'en même temps
                ; les pointeurs sont augmentés d'une unité, donc
                ; il est inutile de s'occuper de ces pointeurs
INX         ; Valeur suivante
BNE ENCORE  ; Jusqu'à ce que les 64K soient transférés
RTS        ; Sortie du sous-programme.

```

De même voici une routine de lecture de 64 K depuis la RAM Son vers le banc \$03. Il est plus rare de lire la Ram Son que d'y écrire, mais cela peut parfois servir et la routine est légèrement différente de celle d'écriture.

*=====

* Lecture des 64 K de la Ram Son

*=====

```

XC          ; Directive d'assemblage
XC          ;

```

```

MEMOIRE      =\$030000      ; Banc où se trouvent les données que nous allons
; écrire (ou ailleurs)
      ORG    \$1000      ; Le programme commence à cette adresse
; (ou à une autre adresse du banc $00)
      CLC      ; Mode natif
      XCE      ;
      REP    #\$30      ; Registres X,Y sur 16 bits
      SEP    #\$20      ; et A sur 8 bits.

      LDAL   \$E100CA    ; On choisit d'accéder à la RAM Son avec
      AND   #%00001111  ; auto-incrémentation des pointeurs...
      ORA   #%01100000
      STA   \$C03C      ; Accès Ram Son avec auto-incrémentation

      STZ   \$C03E      ; Mise à zéro des deux pointeurs.
      STZ   \$C03F
      LDA   \$C03D      ; Attention! Ici on fait une première
; lecture pour rien, comme indiqué. Mais
; surtout, on l'a fait après la déclaration des
; deux pointeurs et non pas avant!
      LDX   #\$0000

ENCORE      LDA   \$C03D      ; Lecture d'une valeur et incrémentation
; automatique des pointeurs.
      STAL  MEMOIRE,X    ; Stockage...
      INX      ; Valeur suivante
      BNE   ENCORE

      RTS      ; Fin de la routine.

```

La programmation des 32 oscillateurs du DOC se fait par l'intermédiaire de ses 227 registres. Trois de ces registres concernent le DOC en général, mais les autres sont divisés en sept séries de 32 registres. Chaque oscillateur peut produire un et un seul son à la fois, mais Apple recommande de ne pas utiliser les oscillateurs 31 et 32, car ils sont réservés. Cependant, si vous n'utilisez pas les outils sonores (Sound Tools), vous pourrez les programmer sans aucun problème. Vous pouvez donc avoir jusqu'à 32 sons différents à la fois.

Pour effectivement produire un son, il faut choisir un oscillateur libre, et lui indiquer dans ses registres les éléments suivants :

- La fréquence à laquelle le son sera joué.
- Le volume de l'oscillateur.
- Le début du son en Ram Son.
- La longueur du son.
- Le mode de fonctionnement de l'oscillateur et son ordre de mise en action.

Dans les exemples des paragraphes suivants, il faudra considérer que le registre de contrôle du son (\$CO3C) est programmé pour accéder aux registres du DOC. Je vous rappelle les instructions qui permettent d'accéder au DOC :

CLC			
XCE			
REP	#\$30		; Registres X,Y sur 16 bits
SEP	#\$20		; et A sur 8 bits
LDAL	\$E100CA		; Lecture du volume général du tableau de bord
AND	#\$0F		; Mise à zéro des bits 5 et 6 pour accéder au DOC
STA	\$C03C		; sans auto-incrémentation

6.1.1.3/1 Les registres de fréquence (bas et haut) \$00-\$3F

Ce qui va déterminer la hauteur du son produit est la fréquence laquelle le son est joué. A basse fréquence, le son sera plutôt grave alors qu'à fréquence élevée il aura tendance à être aigu. Cependant, la fréquence du son que l'on indique à l'oscillateur ne correspond pas à la fréquence réelle (en hertz) à laquelle le son est entendu. La fréquence de programmation correspond à la vitesse de lecture du son qui se trouve en Ram Son. En effet, l'oscillateur en lui-même ne produit pas de son, mais il converti chaque valeur en une tension qui a pour effet de faire plus ou moins vibrer la membrane du haut-parleur. Ainsi le fait de convertir à la suite différentes valeurs va produire un son. Bien entendu, la réalité est plus compliquée que cela, mais nous allons nous limiter à la programmation du son, sans entrer dans les concepts scientifiques.

La fréquence du son doit être donnée sur 16 bits, ce qui permet d'avoir 65536 fréquences différentes possibles. Evidemment, à partir d'une certaine fréquence élevée, les

différences ne sont plus tellement audible. Cependant il ne semble pas possible d'atteindre les ultra- sons.

Les registres \$00 à \$1F contiennent la valeur basse de la fréquence de chacun des 32 oscillateurs tandis que les registres \$20 à \$3F la valeur haute de la fréquence.

Un petit détail : si vous programmez un oscillateur avec une fréquence très basse (\$0001 par exemple), le son sera lu très lentement et sera à peine audible ; par contre, si vous choisissez une fréquence nulle (\$0000), le son ne sera pas lu du tout, mais l'oscillateur ne sera pas arrêté. Ceci peut être utilisé pour interrompre momentanément un son, qui ensuite pourra continuer à être joué depuis l'endroit où il avait été coupé.

Exemple : pour mettre la fréquence \$156 (342) dans l'oscillateur n \$07 :

```
LDA  #$07          ; Le registre n $07 est celui de fréquence (bas)
STA  $C03E        ; de l'oscillateur n $07.
LDA  #$56          ; Partie basse de la fréquence voulue.
STA  $C03D        ; On stocke par l'intermédiaire du registre data.
LDA  #$27          ; Le registre n $27 est quand à lui celui de
STA  $C03E        ; fréquence (haut) de l'oscillateur n $07.
LDA  #$01          ; Partie haute de la fréquence.
STA  $C03D        ; Stockage dans le DOC.
```

Il est également possible de lire, si besoin, les registres de fréquence:

```
LDA  #$07          ; Fréquence (bas) de l'oscillateur n $07.
STA  $C03E        ; Que l'on indique dans le pointeur bas.
LDA  $C03D        ; Deux lectures sont nécessaires pour lire une
LDA  $C03D        ; valeur valide, ne l'oubliez pas!
```

6.1.1.3/2 Les registres de volume \$40-\$5F

Il y a 256 volumes possibles; de plus, le fait que chaque oscillateur possède son propre volume devient très intéressant, surtout pour faire de la musique. Le volume est croissant : \$00 équivaut à un son blanc, puisqu'il est inaudible et \$FF est le volume maximum.

Exemple : mettre le volume de l'oscillateur n \$00 au maximum :

```
LDA  #$40          ; Registre du volume de l'oscillateur n $00.
STA  $C03E
```

```
LDA #$$FF ; $FF est la valeur maximum pour le volume
STA $C03D
```

6.1.1.3/3 Les registres de données \$60-\$7F

Ces registres sont à lecture seulement. Ils contiennent la dernière valeur de la Ram Son lue par tel ou tel oscillateur et leur emploi n'est pas d'une grande utilité. Cependant, nous verrons dans une autre partie un emploi détourné de ces registres.

Exemple : Quelle est la dernière valeur lue par l'oscillateur numéro \$1F?

```
LDA #$7F ; Registre de données de l'oscillateur n $1F.
```

```
STA $C03E
```

```
LDA $C03D ; $FF est la valeur maximum pour le volume
```

```
LDA $C03D ; L'accumulateur contient la valeur recherchée
```

6.1.1.3/4 Les registres pointeurs en Ram Son \$80-\$9F

Pour indiquer le début du son utilisé en Ram Son par tel ou tel oscillateur, il suffit d'écrire dans le registre correspondant la valeur de début de page. Reportez vous au paragraphe traitant de la Ram Son pour savoir où mettre les sons.

Exemple : Nous voulons indiquer l'emplacement du son qui sera joué par l'oscillateur n \$10. Soit un son de \$4000 octets de longs (16384 octets). Il est possible de l'installer en Ram Son aux adresses suivantes :\$0000 ; \$4000 ; \$8000 ; \$C000. Nous choisirons pour l'exemple l'emplacement en \$C000 :

```
LDA #$90 ; Pointeur RAM Son de l'oscillateur n $10.
```

```
STA $C03E
```

```
LDA #$$FF ; Page $C0 de la RAM Son (pour l'adresse $C000)
```

```
LDA $C03D
```

Ce sont les registres qui contiennent le plus d'informations. Voici la description des bits d'un de ces registres :

Les bits 7 à 4 contiennent donc le numéro du canal de sortie du son joué par l'oscillateur en question, ce qui permet avec une carte stéréo d'entendre les sons sur les enceintes gauche ou droite. Pour l'instant seules les cartes stéréo existent, mais il serait tout à fait possible de fabriquer des cartes avec 4,6 ou 8 haut-parleurs branchés, permettant ainsi

d'extraordinaires effets... Cependant, bien que l'on puisse programmer jusqu'à 16 canaux de sortie, seulement 8 sont réellement accessibles étant donné qu'il n'y a que trois des quatre broches «Channel address» qui sont effectivement reliées au connecteur J25.

Mais de toute façon, comme la carte stéréo est le standard, il suffira de commuter le bit 4, les trois autres restant à zéro. On obtient donc une sortie à droite si le bit 4 est à 0 et à gauche s'il est à 1. Si l'oscillateur arrive à la fin d'un son et que le bit 3 de son registre de contrôle est à 1, il enverra un signal d'interruption au microprocesseur. Bien entendu ce signal est traitable et l'on peut même savoir quel oscillateur l'a envoyé.

Les modes de fonctionnement des oscillateurs déterminent la façon dont le son va être joué. Les bits 2 et 1 commutent les différents modes. Ainsi le mode «Free-run» permet de produire un son qui une fois fini se répétera indéfiniment depuis le début. A l'opposé le mode «One-shot» arrêtera l'oscillateur dès la fin du son. Les deux autres modes ont une utilisation plus particulière dont nous reparlerons dans le paragraphe spécialement consacré aux quatre modes.

Enfin, le bit 0 est l'indicateur de marche de l'oscillateur. S'il est à 0 cela signifie que l'oscillateur est en train de produire un son, tandis qu'à 1 il est arrêté et ne produit donc aucun son. Ainsi, il suffit de programmer ce bit pour mettre l'oscillateur en action, mais il faut également savoir que la commutation de ce bit peut se faire automatiquement selon les modes de fonctionnement. Ainsi avec le mode «One-shot» lorsque le son a été joué en entier, le DOC arrête automatiquement l'oscillateur et le bit 0 de son registre de contrôle est donc mis à 1.

Exemple : nous voulons que l'oscillateur n \$05 joue un son unique, sur le haut-parleur de gauche, sans produire d'interruption :

- 1) Pour jouer sur le haut-parleur de gauche, il faut mettre \$1 dans les bits 7 à 4 du registre \$A5 (0001 en binaire).
- 2) Le bit 3 doit être à 0 puisqu'aucune interruption ne doit être produite.
- 3) Pour un son unique, il faut le mode «One-shot», donc les bits 2 et 1 doivent contenir 0 et 1.
- 4) Pour la mise en marche de l'oscillateur, il faut mettre 0 dans le bit 0.

Ainsi il faut mettre la valeur binaire 00010010 (\$12) dans le registre \$A5.

```
LDA    #$A5          ; Registre de contrôle de l'oscillateur n $05
STA    $C03E
LDA    #$FF          ; Son unique à gauche, sans interruption.
```


Il nous reste encore à indiquer la taille du son utilisée par l'oscillateur désiré. C'est entre autre le rôle des registres de la série \$C0 dont voici la description :

Le bit 7 est réservé et le mettre à zéro conviendra sans problème. Par contre, le bit 6 doit absolument être mis à zéro, car il est réservé pour une éventuelle extension de la Ram Son. Il se pourrait qu'un futur IIGS utilise 128 K de Ram Son et dans ce cas le bit 6 aurait le rôle de commuter entre les deux bancs. Donc, afin de ne pas avoir de mauvaise surprise de compatibilité avec les «prochains GS», restez d'office dans le premier banc en mettant ce bit à zéro.

La taille du son est donc codée avec les bits 5 à 3, pas de problème particulier si ce n'est de bien choisir la taille des sons et leurs emplacements.

La résolution d'adressage qui est codée avec les bits 2 à 0 est d'un usage peu évident. Elle détermine selon son codage et selon la taille du son le nombre d'octet effectivement pris en compte lors de la lecture de la Ram Son par un oscillateur. Ainsi à fréquence et résolution égales un son de différente taille n'aura pas la même hauteur.

Heureusement, le fait de choisir la résolution proportionnellement à la taille du son annule cet effet le plus souvent indésirable. Afin de vous aider dans le calcul des valeurs à mettre dans le registre de taille des sons, voici un petit tableau indiquant les bonnes valeurs selon les différentes tailles de sons:

Taille du son		Valeur à mettre dans le registre de la série \$C0	
256	\$100	\$00	%00000000
512	\$200	\$09	%00001001
1024	\$400	\$12	%00010010
2048	\$800	\$1B	%00011011
4096	\$1000	\$24	%00100100
8192	\$2000	\$2D	%00101101
16384	\$4000	\$36	%00110110
32768	\$8000	\$3F	%00111111

Exemple : Soit un son de \$4000 octets de long (16384 octets). Nous voulons l'indiquer pour l'oscillateur n \$15.

```
LDA #D5 ; Registre de de taille du son de l'oscillateur
STA $C03E ; n $05
```

LDA #36 ; Selon le tableau, la valeur à utiliser est 36

LDA \$C03D

6.1.1.3.17 Le registre des interruptions \$E0

Ce registre, comme les deux suivants ne concerne pas un oscillateur en particulier, mais le DOC en général. Le registre n \$E0 s'occupe des interruptions sonores en indiquant tout d'abord si une interruption a été provoquée, et si tel est le cas lequel des 32 oscillateurs en est l'origine. A noter que ce registre est à lecture seulement.

Voici la description des différents bits:

Bits	Valeur	Description
7	1	Aucun oscillateur n'a produit d'interruption.
	0	Un des 32 oscillateurs est, à l'origine de l'interruption.
6	-	Réservé
5-1		Contient le numéro de l'oscillateur ayant généré l'interruption
0	-	Réservé

L'indicateur d'interruption (bit 7) n'a d'intérêt que si vous gérez vous même les interruptions. Dans ce cas, il permet de savoir que c'est le DOC qui a provoqué l'interruption et non autre chose (VGC, Horloge, souris, etc...). Bien entendu, si vous décidez de gérer vous même les interruptions, vous devrez faire une routine de sondage qui aura pour but de repérer la source de l'interruption. Mais le CS possède déjà un excellent système de gestion des interruptions employant différents vecteurs de branchement. Dans le cas du son, il se trouve en \$E1002C. Par contre, il peut être très utile de savoir lequel des 32 oscillateurs a généré l'interruption, sauf s'il est prévu qu'un seul oscillateur puisse le faire. Donc, en cas d'interruption, les bits 5 à 1 du registre SEO contiennent le numéro de l'oscillateur, qu'il faudra ensuite traiter.

Exemple : Extrait d'une routine de gestion d'interruption :

On supposera que le microprocesseur 65C816 vient de produire une interruption et qu'il se soit branché sur votre routine de gestion d'interruption. Cette routine a pour but de connaître l'origine de l'interruption...

... ; D'autres instructions auraient ici pour but

... ; de savoir si c'est le VGC, la souris ou autre

```

... ; qui vient de provoquer l'interruption

LDA #$E0 ; Etant donné qu'ici ni le VGC, ni la souris
STA $C03E ; n'ont produit l'interruption, peut-être que
LDA $C03D ; que c'est le DOC?
LDA $C03D ; On va donc lire le registre $E0, mais n'oubliez
; pas que l'on doit faire deux lectures!

BPL INTERSON

...

*=====
* Gestion de l'interruption sonore *
*=====

INTERSON AND #%00111110 ; On ne garde que les bits 5 à 1.
LSR ; Un petit décalage à droite et
; l'accumulateur contient maintenant le
; numéro de l'oscillateur qui a provoqué
; l'interruption.

... ; Suite de la routine...

```

6.1.1.3/8 Le registre de fonctionnement des oscillateurs \$E1

Normalement les 32 oscillateurs sont en fonction et chacun d'eux prend 1,2 micro seconde en temps machine, ce qui fait environ 38 micro secondes pour les 32. Ce temps est négligeable, mais si pour une raison ou une autre vous voudriez aller plus vite, il est possible de ne sélectionner qu'une partie des oscillateurs. Toutefois il faut au minimum qu'un oscillateur soit en fonction, et il n'est possible de les mettre en action que dans un ordre séquentiel. Pour choisir le nombre d'oscillateurs que vous voulez, il suffit de multiplier ce nombre par deux et de l'indiquer via le registre \$E1

Mais le fait d'avoir moins d'oscillateurs en même temps provoque donc une accélération du traitement, et les fréquences réelles des sons s'en trouvent également modifiées. Donc je recommande absolument de garder 32 oscillateurs actifs étant donné que le gain de temps n'en vaut pas la peine.

Les 32 oscillateurs sont d'office sélectionnés lors du démarrage de la machine ou après un Reset, mais il est plus prudent de l'indiquer avant de commencer une utilisation du son

(il faut toujours penser qu'un jour cela pourrait ne plus être compatible). Voici donc les quelques instructions nécessaires :

```
LDA  #$E1          ; Registre $E1
```

```
STA  $C03E
```

```
LDA  #$40          ; On veut 32 oscillateurs, mais il faut
```

```
LDA  $C03D          ; multiplier ce nombre par 2, d'où $40 (64)
```

6.1.1.3/9 Le registre de numérisation \$E2

L'Ensoniq est doté d'un système de numérisation qui permet d'acquérir des données de type analogique et de les convertir en nombres de 0 à 255. Bien entendu ce système est utilisé en priorité pour la numérisation du son depuis un magnétophone par exemple, mais rien n'empêche de l'utiliser pour d'autres genres de numérisations. Pour numériser des données, il suffit de relier deux fils au connecteur du DOC, l'un à la masse (broche n 2) et l'autre à l'entrée du convertisseur (broche n 1), le tout branché à la source. Cependant la tension maximale admise lors de l'acquisition des données est de 2,5 volts avec une impédance maximale de 3000 ohms. Il vous est vivement conseillé DE NE PAS DEPASSER CES VALEURS pour ne pas mettre l'Ensoniq, voire votre IIGS en danger. De toute façon si vous décidez d'utiliser la numérisation en bricolant vous même, ce sera à vos risques et périls. De plus, si vous souhaitez numériser du son et travailler avec soin et facilité, mieux vaut vous acheter une carte d'extension stéréo-numérisante.

En ce qui concerne la routine de numérisation, sachez que le DOC prend 31 micro secondes pour convertir une valeur et qu'il vous faudra donc attendre suffisamment de temps entre chaque cycle d'acquisition, sous peine de manquer des valeurs. Si vous souhaitez travailler sur des temps précis, je vous conseille de commuter la cadence du IIGS en 1 Mhz, ce qui facilitera le calcul des temps.

Dernier point : ce registre est à lecture seulement, mais vous vous en seriez douté !

Voici une routine exemple qui a pour but de numériser 64 K de données.

```
*=====
```

```
* Exemple d'une routine de numérisation
```

```
*=====
```

```
XC          ; Directive d'assemblage
```

```
XC          ;
```

```
MEMOIRE    =$030000          ; Banc de stockage des données (vous pouvez la
```

			; changer)
	ORG	\$1000	; Le programme commence à cette adresse
			; (ou à une autre adresse si vous préférez)
	CLC		; Mode natif
	XCE		;
	REP	#\$30	; Registres X,Y sur 16 bits
	SEP	#\$20	; et A sur 8 bits.
	LDA	#\$80	; Mise en vitesse 1Mhz du GS en
			; mettant le bit 7 du commutateur \$C036 à
	TRB	\$C036	; zéro
	LDAL	#\$100CA	; Volume selon celui du beep.
	AND	#\$0F	
	STA	\$C03C	; Accès aux registres du DOC
	LDA	#\$E2	; Choix du registre de numérisation
	STA	\$C03E	
	LDA	\$C03D	; Première lecture pour rien. Remarquez qu'ici
			; nous n'appliquons pas la règle de deux lectures
			; pour une valeur, étant donné que nous ne
			; changeons pas de registre
	LDX	#\$0000	; Début du banc.
BOUCLE	LDA	\$C03D	; Lecture d'une valeur venant d'être convertie.
	BNE	ADMIS	; Si c'est un \$00 alors on ne l'admet pas
	LDA	#\$01	; et on le remplace par un \$01
ADMIS	STAL	MEMOIRE,X	; Stockage
	INX		; Valeur suivante
	BEQ	FINI	; Si on est arrivé à 64K de données, alors fin
	LDY	#\$20	; Boucle de délai à régler selon la qualité
DELAY	DEY		; de la numérisation que l'on veut obtenir
	BNE	DELAY	
	BRA	BOUCLE	; Retour pour un nouveau cycle
FINI	RTS		; Fin de la routine.

6.1.1.4 Le connecteur du DOC

A titre indicatif, voici les caractéristiques du connecteur 7 broches de type «Molex» qui se trouve placé sur la carte mère juste à droite du DOC et à gauche du connecteur d'extension mémoire. La patte n 1 se trouve vers le voyant vert du IIGS.

«Canal strobe» est au niveau bas lorsque l'adresse du canal est valide. Le fait que seulement trois des quatre canaux de sortie du DOC soient reliés au connecteur ne permet que 8 combinaisons possibles. Donc, bien qu'il soit possible de programmer les oscillateurs sur 16 canaux, vous ne pourrez réellement en entendre que jusqu'à huit ! Mais de toute façon, la plupart des personnes se contenteront de la stéréo qui n'utilise que deux canaux.

6.1.2 Création des sons

Vous voilà presque capable de commencer les premiers essais, si ce n'est un petit détail : vous ne possédez encore aucun son et il va falloir les créer. Il existe différentes possibilités en matière de création sonore. Cependant, n'oubliez pas que vous ne disposez que de 64 K de Ram Son et que la valeur \$00 est réservée pour indiquer si besoin la fin d'un son. Voici donc trois méthodes pour constituer une bibliothèque de sons :

6.1.2.1 Sons numérisés

Le principe de la numérisation consiste à convertir en données numériques des informations de type analogiques. Dans le cas du son, une source sonore (micro, radio, lecteur CD, etc...) est reliée au IIGS et les impulsions sonores sont transformées en valeurs. Bien entendu pour réaliser ces conversions, il faut disposer du matériel nécessaire et il vous faudra acheter une carte d'extension, de numérisation. Cependant la plupart des cartes stéréo possèdent également la numérisation et votre investissement se révélera vite rentable. De plus un logiciel de numérisation est normalement fourni avec ces cartes.

Les sons ainsi obtenus ont une origine réelle mais leurs qualités à la restitution dépendent de plusieurs paramètres : le matériel utilisé bien sûr, mais aussi la fréquence d'échantillonnage. L'échantillonnage consiste à convertir en un certain temps un certain nombre de valeurs. Ainsi, si le son que vous souhaitez numériser à une durée réelle de 3 secondes, vous pouvez choisir par exemple de prendre des échantillons toute les 40 micro secondes ou toutes les 80 micro secondes. Dans le premier cas, le son aura un meilleur rendu que dans le deuxième cas, mais il sera également deux fois plus long en mémoire ! Hélas la numérisation est couteuse en place mémoire, et les 64 K de Ram Son peuvent parfois être un peu justes.

La plupart du temps, les sons numérisés sont joués en mode «One-shot» pour un seul bruit, ou en mode «Free-run» pour des répétitions. Mais, étant donné que la numérisation suppose l'achat de matériel, il me sera impossible de donner ici un exemple.

6.1.2.2 Sons synthétisés

La synthèse sonore est complètement différente par rapport à la numérisation :

- Les sons créés n'ont pas d'origine réelle
- La taille des sons est très courte.
- La plupart du temps 256 octets.

Le principe de la synthèse consiste à créer une forme d'onde qui représentera une période et qui sera jouée en mode «Free-run», c'est dire en continu. Toute forme d'onde est imaginable, mais certaines formes sont plus harmonieuses que d'autres et seule l'écoute finale pourra être un critère de choix.

Pour notre exemple, nous allons créer une onde de type sinusoïdale l'aide d'un petit programme en Basic :

```
10 FOX X=0 TO 255
20 Y=INT (128-127*SIN(X/ 40.6))
30 POKE 16384-1-X,Y
40 NEXT X
```

Ligne 10: On veut créer une onde de 256 octets.

Ligne 20: Il faut des valeurs entières. Le 128 est utilisé car il correspond au milieu de l'axe des ordonnées, si l'on imagine un repère représentant la courbe. La formule a été étudiée pour ne pas fournir de valeur nulle. Rappelez vous qu'un zéro indiquerait la fin du son, or nous voulons jouer le son en continu. Rappelez vous également que le Basic Applesoft calcule en radians.

Ligne 30: L'onde sera stockée à partir de \$4000 jusqu'à \$40FF.

Puis nous allons transférer l'onde en Ram Son aux adresses \$0000- \$00FF. Ensuite, il faudra programmer les registres du DOC pour enfin pouvoir entendre quelque chose. Nous allons utiliser l'oscillateur n \$00 pour notre exemple, avec une fréquence de \$65B et son volume au maximum. Mais avant de commencer la programmation, voici un petit tableau récapitulatif des valeurs exactes à mettre dans les registres :

Registre	Valeur
\$00	\$5B (partie basse de la fréquence)
\$20	\$06 (partie haute de la fréquence)
\$40	\$FF (volume maximum)
\$80	\$00 (l'onde sonore débute en \$0000 de la Ram Son)
\$A0	\$00 (canal de sortie à droite ; pas d'interruption ; mode «Free-run» ; oscillateur en action)
\$C0	\$00 (l'onde occupe 256 octets)

Il est judicieux de programmer les différents registres dans leur ordre, mais il faut absolument programmer le registre de contrôle en dernier, car c'est lui qui enclenchera le son. Par contre, s'il n'était pas déclaré en dernier, il y aurait certainement un problème, car le son serait joué avant que ses paramètres fussent indiqués.

Voici la routine en assembleur, qui jouera cette onde après l'avoir transféré en Ram Son :

6.1.2.3 Sons synthétisés utilisés comme des sons numérisés

Ce type de son est un compromis entre les sons numériques et synthétiques. En effet, ils n'ont pas d'existence réelle tout comme les sons de synthèse, mais ils sont utilisées de la même façon que les sons numérisés. Les avantages sont d'une part la pureté des sons (contrairement à la numérisation qui produit toujours quelques parasites), d'autre part le fait de n'avoir besoin d'aucun appareil relatif à la numérisation. Cependant le désavantage est de ne pas pouvoir obtenir les sons voulus et ici encore il faudra faire beaucoup d'essais pour trouver des sons plaisants. De plus comme la taille de ce type de sons est voisine de celle des sons numérisés et qu'ils sont construits par calculs, un certain temps d'attente est nécessaire.

Pour notre exemple, nous allons créer un son de 16 K (\$4000 octets) l'aide d'un petit programme Basic. Attention, le programme prend environ six minutes pour tout calculer.

```

10 FOR X=0 TO 16383
20 Y=INT ((128-1-(121-(X/140))*SIN(.12*X)))
30 POKE 8192-4-X,Y 40 NEXT X

```

Ligne 20 : La formule est prévue pour donner un son dont le volume diminuera dans le temps. Bien entendu vous êtes libre de créer n'importe quel son à partir du moment où il ne comporte pas de valeur nulle.

Ligne 30 : Le son est stocké de \$2000 à \$5FFF.

Un petit «BSAVE SOUND,A\$2000,L\$4000» serait le bienvenu pour sauver le son sur un disque au format Prodos 8 afin de ne pas avoir patienter six minutes à chaque fois.

Une fois le son créé et sauvé, tapez les commandes suivantes :

```
]CALL -151
0<3/0.FFEEZ          (mise à zéro du banc $03)
3 /0<0/ 2000.5FFFM   (transfert du son au début du banc $03)
```

Le programme exemple jouant un son numérisé serait le même que pour ce type de son. Ainsi nous voulons que l'oscillateur n \$01 (pour changer) joue ce son une seule fois (mode «One-shot»), à fréquence \$130, volume \$C0, et su le canal de gauche. Voici donc le petit tableau récapitulatif:

Registre	Valeur
\$01	\$30 (partie basse de la fréquence)
\$21	\$01 (partie haute de la fréquence)
\$41	\$C0 (volume maximum)
\$81	\$00 (l'onde sonore débute en \$0000 de la Ram Son)
\$A1	\$12 (canal de sortie à gauche ; pas d'interruption ; mode «One-shot» ; oscillateur en action)
\$C1	\$36 (l'onde occupe 16384 octets). Référez vous au tableau du paragraphe 6.1.1.3.6 traitant des registres de taille des sons, pour trouver la bonne valeur à mettre dans ce registre.

Comme vous pouvez le constater la programmation d'un son est en elle même extrêmement simple, à partir du moment où l'on connaît les bases de fonctionnement de l'Ensoniq DOC.

6.1.3 Les 4 modes de fonctionnement des oscillateurs

Comme nous avons pu le voir les oscillateurs du DOC peuvent fonctionner selon quatre modes, choisis en programmant les bits 1 et 2 des registres de la série \$A0. Certains de ces modes n'utilisent qu'un seul oscillateur, d'autres doivent être groupés par deux. Dans le cas où ils sont groupés par deux, il s'agit toujours d'un oscillateur pair couplé avec son voisin impair, par exemple le n \$00 avec le n \$01, le n \$08 avec le n \$09, le n \$1E avec le n \$1F, etc...

6.1.3.1 Mode «One-Shot»

C'est le mode le plus simple, que nous avons déjà employé. Principalement utilisé pour des sons longs (de type son numérisé), il suffit d'un seul oscillateur pour qu'il fonctionne.

Le fait de programmer l'oscillateur avec le mode «One-shot» remet à zéro son accumulateur, c'est à dire que lorsque l'on va mettre en route l'oscillateur, le son sera joué obligatoirement depuis le début. L'oscillateur jouera une seule fois le son puis s'arrêtera de lui même à la fin du son.

Dernier détail sur le mode «One-shot» : il permet de jouer sans problème un son finissant par des zéros.

6.1.3.2 Mode «Free-run»

Egalement utilisé avec un seul oscillateur ce mode permet de jouer plusieurs fois le même son. Pour les sons de synthèses, généralement de courtes tailles (256 octets le plus souvent), c'est le mode par excellence.

Cependant, pour rejouer plusieurs fois le même son, il faut obligatoirement que celui-ci ait une taille exacte (256, 512, 1024, 2048, 4096, 8192, 16384 ou 32768 octets) en ne contenant aucun zéro. Si tel était le cas, l'oscillateur s'arrêterait de lui même.

Contrairement au mode «One-shot», le mode «Free-run» ne remet pas l'accumulateur de l'oscillateur à zéro, ce qui signifie que si vous interrompez un son en train d'être joué en arrêtant l'oscillateur, puis que vous le remettiez en marche, le son reprendra la où il en était et non depuis le début.

6.1.3.3 Mode «Swap»

Ce mode qui fonctionne obligatoirement avec une paire d'oscillateurs offre des possibilités intéressantes. Le principe de fonctionnement est le suivant: un oscillateur programmé en mode «Swap» va jouer le son comme s'il était en mode «One-shot», puis va s'arrêter et mettre en action l'oscillateur voisin. Mais il y a différentes possibilités de programmer les oscillateurs pairs et impairs en combinaison avec le mode «Swap» :

Oscillateurs		Effet recherché
Pair	Impair	
Swap	Swap	Possibilité de jouer les 64K de la Ram Son ou un son finissant par des zéros, en émulation du mode Free-run
Swap	One-shot	Possibilité de jouer les 64K de la Ram Son en émulation du mode «One-shot» ou faire jouer deux fois seulement le même son.
One-shot	Swap	Possibilité de faire jouer l'oscillateur pair en mode émulation «Free-run». Pour cela il suffit de déclarer l'oscillateur impair en mode «Swap» sans le mettre en marche ni programmer ses autres registres à part celui de contrôle.

Les autres combinaisons qui seraient possibles avec le mode «Swap» et les modes «Free-run» et «Sync» ne donnent rien d'intéressant.

Exemple de deux oscillateurs programmés en mode «Swap» et «Swap» afin de jouer un son finissant par des zéros. Le son sera joué par les oscillateurs n \$06 et n \$07 en alternance, avec une fréquence de \$120 en volume maximum et sur le canal droit.

Effectuez d'abord les opérations suivantes pour charger votre son :

```
]BLOAD SOUND           (chargement du son en $2000)
]CALL-151              (passage sous moniteur)
3 /4000<0/2000.5FFFM   (transfert du son en $034000-$037FFF)
0<3/7500.7FFFM        (ici on met une suite de zéros pour que notre
                       son n'occupe plus que $3500 octets de long)
```

Ainsi notre son à une taille réelle de 13568 octets. Cependant, d'après la structure de la Ram Son, il faut déclarer sa taille par excès, d'où une taille de \$4000 octets et la valeur \$36 à mettre dans les registres de la serie \$C0. Voici maintenant le tableau récapitulatif des valeurs mettre dans les registres des oscillateurs n \$06 et n \$07 :

n	Valeur
\$06	\$20 (partie basse de la fréquence de l'oscillateur pair)
\$07	\$20 (partie basse de la fréquence de l'oscillateur impair)
\$26	\$01 (partie haute de la fréquence de l'oscillateur pair)
\$27	\$01 (partie haute de la fréquence de l'oscillateur impair)
\$46	\$FF (volume de l'oscillateur pair)
\$47	\$FF (volume de l'oscillateur impair)
\$86	\$40 (le son commence en \$4000 de la Ram Son)
\$87	\$40 (de même pour l'oscillateur impair)
\$A6	\$06 (canal de sortie à droite ; pas d'Interruption ; mode «Swap» ; oscillateur en action)
\$A7	\$07 (canal de sortie à droite ; pas d'interruption ; mode «Swap» ; oscillateur à l'arrêt)
\$C6	\$36 (Notre son de 13568 octets de long est tout de même déclaré \$4000, d'où la valeur \$36 selon le tableau des tailles).
\$C7	\$36 (de même pour l'oscillateur impair)

Notez que l'oscillateur n \$06 va être le premier à se mettre en action, tandis que le numéro \$07 est au départ à l'arrêt. Par la suite, le n \$06 passera à l'arrêt dès qu'il rencontrera un \$00 dans la Ram Son, enclenchant immédiatement le n \$07, et ainsi de suite...

Voici enfin le programme, avec la routine de transfert des sons, maintenant bien connue :

*=====

* Routine jouant un son terminé par des zéros (première méthode)

*=====

XC ; Directive d'assemblage

XC ;

MEMOIRE =\$030000 ; Banc de stockage des sons

ORG \$1000 ; Le programme commence à cette adresse

; (ou ailleurs si vous préférez)

CLC ; Mode natif

XCE

*=====

* Écriture en RAM son de 64K de données

*=====

REP #\$30 ; Registres X,Y sur 16 bits

SEP #\$20 ; et A sur 8 bits.

LDAL \$E100CA ; On choisit d'accéder à la

AND #%00001111 ; RAM Son avec auto-incrémentation

ORA #%01100000 ; des pointeurs...

STA \$C03C

STZ \$C03E ; Mise à zéro des deux pointeurs

STZ \$C03F

LDX ; On veut lire depuis le début du banc.

ENCORE LDAL MEMOIRE,X ; Lecture d'une valeur du banc (ici le banc \$03).

STA \$C03D ; Transfert en Ram Son. A noter qu'en même temps

; les pointeurs sont augmentés d'une unité, donc

; il est inutile de s'occuper des pointeurs.

INX ; Valeur suivante.

BNE ENCORE ; Jusqu'à ce que les 64K soient transférés.

*=====

* Programmation du DOC

*=====

LDAL \$E100CA ; Volume selon celui du beep

AND #\$0F

STA \$C03C ; Accès aux registres du DOC.

LDA #\$06 ; Registre fréquence (bas) de

STA \$C03E ; l'oscillateur \$06

LDA #\$20

STA \$C03D

INC \$C03E ; Registre de l'oscillateur n \$07

STA \$C03D ; Notez que l'accumulateur contient toujours \$20

LDA \$26 ; Registre fréquence (haut).

STA \$C03E

LDA #\$01

STA \$C03D

INC \$C03E

STA \$C03D

LDA #\$46 ; Registre volume

STA \$C03E

LDA #\$FF

STA \$C03D

INC \$C03E

STA \$C03D

LDA #\$86 ; Registre d'adresse en RAM son.

STA \$C03E

LDA #\$40

STA \$C03D

INC \$C03E

STA \$C03D

LDA #\$C6 ; Registre de taille du son

STA \$C03E

LDA #\$36

STA \$C03D

INC \$C03E

STA \$C03D

LDA #\$A6 ; Registre de contrôle de l'oscillateur.

STA \$C03E ; Attention, il est toujours préférable

LDA #\$06 ; d'écrire le registre de contrôle en

```

STA  $C03D      ; dernier, car c'est lui qui va
INC  $C03E      ; effectivement déclencher le son.
LDA  #$07       ; Remarquez qu'au début seul
STA  $C03D      ; l'oscillateur pair va être en action.

RTS             ; Sortie. Le son est en train d'être joué

```

Cette technique pour jouer un son se terminant par des zéros est tout à fait valable, cependant elle a l'inconvénient de demander deux fois les mêmes valeurs pour les registres pair et impair. Ainsi cela peut être gênant si vous avez besoin d'intervenir en court de traitement du son, car vous devrez le faire pour l'oscillateur pair, mais aussi pour l'oscillateur impair. Voici donc la seconde méthode, utilisant deux oscillateurs programmés en mode «One-shot» et «Swap». Nous garderons le même son et les mêmes paramètres. Le tableau des valeurs est le suivant:

n	Valeur
\$06	\$20 (partie basse de la fréquence de l'oscillateur pair)
\$26	\$01 (partie haute de la fréquence de l'oscillateur pair)
\$46	\$FF (volume de l'oscillateur pair)
\$86	\$40 (le son commence en \$4000 de la Ram Son)
\$A6	\$02 (canal de sortie à droite ; pas d'Interruption ; mode «One-shot» ; oscillateur en action)
\$A7	\$07 (canal de sortie à droite ; pas d'interruption ; mode «Swap» ; oscillateur à l'arrêt)
\$C6	\$36 (Notre son de 13568 octets de long est tout de même déclaré \$4000, d'où la valeur \$36 selon le tableau des tailles).

Notez que seul le fait que l'oscillateur impair soit en mode «Swap» arrêté est important. On se moque des autres paramètres de l'oscillateur impair.

Voici maintenant la routine :

```
*=====
```

```
* Routine jouant un son terminé par des zéros (seconde méthode)
```

```
*=====
```

```

XC          ; Directive d'assemblage
XC          ;

```

```
MEMOIRE    =$030000      ; Banc de stockage des sons
```

ORG \$1000 ; Le programme commence à cette adresse

; (ou ailleurs)

CLC ; Mode natif

XCE

*=====

* Écriture en RAM son de 64K de données

*=====

REP #\$30 ; Registres X,Y sur 16 bits

SEP #\$20 ; et A sur 8 bits.

LDAL \$E100CA ; On choisit d'accéder à la

AND #%00001111 ; RAM Son avec auto-incrémentation

ORA #%01100000 ; des pointeurs...

STA \$C03C

STZ \$C03E ; Mise à zéro des deux pointeurs

STZ \$C03F

LDX #\$0000 ; On veut lire depuis le début du banc.

ENCORE LDAL MEMOIRE,X ; Lecture d'une valeur du banc (ici le banc \$03).

STA \$C03D ; Transfert en Ram Son. A noter qu'en même temps

; les pointeurs sont augmentés d'une unité, donc

; il est inutile de s'occuper des pointeurs.

INX ; Valeur suivante.

BNE ENCORE ; Jusqu'à ce que les 64K soient transférés.

*=====

* Programmation du DOC

*=====

LDAL \$E100CA ; Volume selon celui du beep

AND #\$0F

STA \$C03C ; Accès aux registres du DOC.

LDA #\$06 ; Registre fréquence (bas) de

STA \$C03E ; l'oscillateur \$06

LDA #\$20

STA \$C03D

```
LDA #$26 ; Registre fréquence (haut)
```

```
STA $C03E
```

```
LDA #$01
```

```
STA $C03D
```

```
LDA #$46 ; Registre volume
```

```
STA $C03E
```

```
LDA #$FF
```

```
STA $C03D
```

```
LDA #$86 ; Registre d'adress en RAM son.
```

```
STA $C03E
```

```
LDA #$40
```

```
STA $C03D
```

```
LDA #$C6 ; Registre de taille du son
```

```
STA $C03E
```

```
LDA #$36
```

```
STA $C03D
```

```
LDA #$A6 ; Registre de contrôle de l'oscillateur.
```

```
STA $C03E ; Attention, il est toujours préférable
```

```
LDA #$02 ; d'écrire le registre de contrôle en
```

```
STA $C03D ; dernier, car c'est lui qui va
```

```
INC $C03E ; effectivement déclencher le son.
```

```
LDA #$07 ; Attention l'oscillateur impair doit absolument
```

```
STA $C03D ; être en mode «Swap» et à l'arrêt.
```

```
RTS ; Sortie. Le son est en train d'être joué
```

6.1.3.4 Mode «Sync»

Le mode «Sync» utilise aussi une paire d'oscillateurs, et générale-

ment seul l'oscillateur impair sera programmé avec ce mode. L'effet provoqué est la synchronisation de l'oscillateur impair sur le pair, en produisant un meilleur niveau sonore. Il n'est pas besoin de programmer les autres registres de l'oscillateur impair, seul son registre de contrôle doit être en mode «Sync» et peut même être arrêté.

Il faut avouer que ce mode n'est pas d'une grande utilité, bien qu'il soit possible d'obtenir certains effets selon la façon de programmer les deux oscillateurs. Par exemple mettre les deux oscillateurs en mode «Sync», ou seulement le premier, ou encore choisir une

fréquence différente pour chacun d'eux... Les possibilités sont trop nombreuses pour donner un exemple, et il vous sera facile de les tester vous même.

6.1.4 Les interruptions sonores

La facilité du traitement des interruptions du IIGS est telle qu'il serait dommage de s'en priver dans le cas du son. En effet chaque oscillateur du DOC a la possibilité d'envoyer un signal d'interruption s'il a été programmé pour. Le signal est déclenché dès la fin du son. Si vous utilisez le système normal de gestion des interruptions, le vecteur de connexion du son se trouve en \$E1002C et il vous suffit d'y mettre un JMP long vers votre routine de traitement. Par la suite votre routine devra déterminer, si nécessaire, lequel des 32 oscillateurs a produit l'interruption, en utilisant le registre \$E0 du DOC (voir le paragraphe 6.1.1.3.7 pour l'utilisation de ce registre). Bien entendu, s'il n'y a qu'un seul oscillateur programmé pour produire des interruptions, il sera inutile de se servir du registre \$E0.

Notre exemple aura pour but de changer la couleur du bord d'écran chaque fois que le son sera joué.

```
]BLOAD SOUND          (chargement du son en $2000)
]CALL-151              (passage sous moniteur)
3/0<0/ 2000.5FFFM      (transfert du son en $030000-$033FFF)
```

Nous utiliserons l'oscillateur n \$1F, programmé pour jouer le son en continu sur la canal gauche, avec une fréquence de \$160, un volume de moitié et l'autorisation d'interruption.

Voici le tableau des valeurs:

n	Valeur
\$1F	\$60 (partie basse de la fréquence)
\$3F	\$01 (partie haute de la fréquence)
\$5F	\$80 (volume de moitié)
\$9F	\$00 (le son commence en \$0000 de la Ram Son)
\$BF	\$18 (canal de sortie à gauche ; Interruption autorisée ; mode «Free-run» ; oscillateur en action)
\$DF	\$36 (Notre son occupe \$4000 octets de long).

*=====

* Routine jouant un son produisant des interruptions

*=====

```
XC          ; Directive d'assemblage.
```

```
XC          ;
```

```

MEMOIRE      $3000      ; Banc de stockage des sons.
              ORG $1000      ; Ou ailleurs..
CLC          ; Mode natif
XCE

*=====
* Écriture en RAM son de 64K de données
*=====

              REP  #$30      ; Registres X,Y sur 16 bits
              SEP  #$20      ; et A sur 8 bits.
LDAL $E100CA  ; On choisit d'accéder à la
AND  #%00001111 ; Ram Son avec auto-incrémentation
ORA  #%01100000 ; des pointeurs...
STA  $C03C
STZ  $C03E      ; Mise à zéro des deux pointeurs
STZ  $C03F
LDX  #$0000      ; On veut lire depuis le début du banc.
ENCORE      LDAL MEMOIRE,X  ; Lecture d'une valeur du banc (ici le banc $03).
              STA  $C03D      ; Transfert en Ram Son. A noter qu'en même temps
              ; les pointeurs sont augmentés d'une
              ; unité, donc il est inutile de
              ; s'occuper des pointeurs.
              INX          ; Valeur suivante.
              BNE  ENCORE    ; Jusqu'à ce que les 64K soient
              ; transférés.

*=====
* Détournement du vecteur son
* vers notre routine de traitement
*=====

              LDA  #$5C      ; Code du JMP long.
              STAL $E1002C    ; $E1002C est le vecteur son.
LDAL #<INTER  ; Partie basse de l'adresse de notre
              ; routine
              STAL $E1002D    ; de traitement des interruptions
              ; sonores.
LDAL #>INTER  ; Partie haute de l'adresse.

```

STAL \$E1002E

LDA #\$00 ; Nous sommes dans le banc \$00.

STAL \$E1002F

*=====

* Programmation du DOC

*=====

LDAL #\$EC

AND #\$0F

STA \$C03C ; Accès aux registres du DOC.

LDA #\$1F ; Registre fréquence (bas) de
; l'oscillateur n \$06.

STA \$C03E

LDA #\$60

STA \$003D

LDA #\$3F ; Registre fréquence (haut)

STA \$C03E

LDA #\$01

STA \$C03D

LDA #\$5F ; Registre volume.

STA \$C03E

LDA #\$80

STA \$C03D

LDA #\$9F ; Registre d'adresse en Ram Son.

STA \$C03E

LDA #\$00

STA \$C03D

LDA #\$DF ; Registre de taille du son.

STA \$C03E

LDA #\$36

STA \$C03D

LDA #\$BF ; Registre de contrôle de l'oscillateur.

STA \$C03E ; Attention, il est toujours préférable

LDA #\$18 ; d'écrire le registre de contrôle en dernier,

STA \$C03D ; car c'est lui qui va effectivement déclencher
; le son

RTS ; Sortie. Le son est en train d'être joué et le

```
; bord d'écran change à chaque fin du son.
```

```
INTER      SEP    #$30      ; A et X,Y sur 8 bits
```

```
          INC    $C034    ; Change la couleur du bord de l'écran
```

```
          CLC
```

```
          RTL      ; Fin de la routine d'interruption
```

Cette routine, qui une fois lancée fonctionne automatiquement n'est qu'un simple exemple, mais vous pouvez déjà vous rendre compte des possibilités. Le fait de changer la couleur du bord d'écran n'est guère intéressant, mais vous remarquerez qu'en changeant la fréquence de l'oscillateur, vous changerez également la vitesse de variation de la couleur. Effectivement, cette routine n'est autre qu'un tempo, qui nous sera par la suite très utile pour la musique.

6.1.5 La stéréo

Si en dehors de l'extension mémoire il n'y avait qu'une seule carte acheter, ce serait une carte stéréo ! En effet, le IIGS et ses capacités sonores le méritent bien et vous ne regretterez pas non plus d'y ajouter des enceintes. Ainsi une telle carte permet pour chaque oscillateur de choisir si le son sortira à droite ou à gauche. Du point de vue programmation, chaque oscillateur programmé avec un canal pair jouera à droite, tandis qu'avec un canal impair ce sera à gauche.

Cependant il est préférable de choisir le canal zéro pour la droite et le un pour la gauche.

Outre la stéréo cette extension offre d'autres avantages. Déjà, le fait de connecter des enceintes sur la sortie de la carte et non sur la sortie du IIGS évite certains parasites, car le son ne transite plus que par l'interface de l'Ensoniq. De part ce fait, le beep du contrôle G ne sera pas reproduit par les enceintes connectées à la carte stéréo. Ceci est très important car vous pourrez baisser le niveau du volume de beep pour ne pas être gêné, sans pour autant perdre le volume des sons du GS, étant donné que le réglage du volume général par le tableau de bord ne concerne que le haut-parleur interne et la prise «jack» du IIGS.

La plupart des cartes stéréo comporte également un système de numérisation, et vous auriez tort de vous en priver. De plus, certaines cartes possèdent leur propre amplificateur réglable. Mais si vous désirez donc acheter une carte stéréo, prévoyez également l'achat d'enceintes, auto amplifiées de préférence, à moins que vous ne souhaitiez brancher votre chaîne HIFI via la carte.

6.1.6 Effets spéciaux et divers

Cette partie contient quelques exemples de ce que l'on peut faire avec le IGS et le son, mais elle n'est pas exhaustive.

6.1.6.1 Echo simple

Le but pour produire un écho est de faire jouer un son une première fois à volume élevé, puis une seconde fois moins fort. Mais tout l'intérêt est de le faire de façon automatique, sans intervention d'un programme en cours de traitement. Pour cela, il suffit d'utiliser deux oscillateurs, le premier programmé en mode «Swap» en action, volume maximum et le second en mode «One-Shot» arrêté, volume plus faible. Bien entendu, il faut que la fréquence des deux oscillateurs soit la même, à moins que vous ne désiriez un autre effet. Vu la simplicité de la routine, il me paraît inutile de vous la donner.

6.1.6.2 Passage droite-gauche

L'effet de passage d'un son d'un haut-parleur à l'autre pendant qu'il est en train d'être joué est assez intéressant, mais il est nécessaire qu'un programme intervienne en cours de traitement, et vous ne pourrez donc rien faire d'autre pendant ce temps.

Le principe consiste à utiliser deux oscillateurs programmés en mode «One-shot» qui joueront le son en même temps, l'un sur la droite volume élevé, l'autre sur la gauche à volume nul. Mais pendant que le son sera joué, une routine diminuera le volume du premier oscillateur tout en augmentant celui du second. L'effet procure une agréable sensation, pour peu que les deux haut-parleurs soient bien disposés et à condition bien entendu de posséder une carte stéréo, sans laquelle il n'y aurait aucun effet.

Pour notre exemple, nous allons charger notre son comme d'habitude, et nous le ferons jouer à fréquence \$120.

```
]BLOAD SOUND                (chargement du son en $2000)
]CALL-151                    (passage sous moniteur)
3/0<0/2000.5FFFM            (transfert du son en $030000-$033FFF)
```

Nous utiliserons les oscillateurs n \$00 et n \$01, voici le tableau des valeurs :

n	Valeur
\$00	\$20 (partie basse de la fréquence de l'oscillateur pair)
\$01	\$20 (partie basse de la fréquence de l'oscillateur impair)
\$20	\$01 (partie haute de la fréquence de l'oscillateur pair)
\$21	\$01 (partie haute de la fréquence de l'oscillateur impair)
\$40	\$FF (volume de l'oscillateur pair)

n	Valeur
\$41	\$FF (volume de l'oscillateur impair)
\$80	\$00 (le son commence en \$0000 de la Ram Son)
\$81	\$00 (de même pour l'oscillateur impair)
\$A0	\$02 (canal de sortie à droite ; pas d'interruption; mode «One-shot» ; oscillateur en action)
\$A1	\$12 (canal de sortie à gauche ; pas d'interruption; mode «One-shot» ; oscillateur en action)
\$C0	\$36 (Notre son occupe \$4000 octets de long).
\$C1	\$36 (de même pour l'oscillateur impair)

*=====

* Routine jouant un son de droite à gauche

*=====

	XC		; Directive d'assemblage.
	XC		;
MEMOIRE	\$3000		; Banc de stockage des sons.
	ORG	\$1000	; Ou ailleurs..
	CLC		; Mode natif
	XCE		

*=====

* Écriture en RAM son de 64K de données

*=====

	REP	#\$30	; Registres X,Y sur 16 bits
	SEP	#\$20	; et A sur 8 bits.
	LDAL	\$E100CA	; On choisit d'accéder à la
	AND	##00001111	; Ram Son avec auto-incrémentation
	ORA	##01100000	; des pointeurs...
	STA	\$C03C	
	STZ	\$C03E	; Mise à zéro des deux pointeurs
	STZ	\$C03F	
	LDX	#\$0000	; On veut lire depuis le début du banc.
ENCORE	LDAL	MEMOIRE,X	; Lecture d'une valeur du banc (ici le
			; banc \$03).
	STA	\$C03D	; Transfert en Ram Son. A noter qu'en
			; même temps

```

; les pointeurs sont augmentés d'une
; unité, donc il est inutile de
; s'occuper des pointeurs.
INX ; Valeur suivante.
BNE ENCORE ; Jusqu'à ce que les 64K soient
; transférés

```

```
*=====
```

```
* Programmation du DOC
```

```
*=====
```

```

LDAL $E100CA ; Volume selon celui du deep
AND #$0F
STA $C03C ; Accès aux registres du DOC.
LDA #$00 ; Registre fréquence (bas)
STA $C03E
LDA #$20
STA $C03D
INC $C03E
STA $C03D
LDA #$21 ; Registre fréquence (haut)
STA $C03E
LDA #$01
STA $C03D
STA $C03E
STA $C03D
LDA #$40 ; Registre volume.
STA $C03E
LDA #$FF
STA $C03D
INC $C03E
LDA #$00
STA $C03D
LDA #$80 ; Registre d'adresse en Ram Son.
STA $C03E
LDA #$00
STA $C03D

```

```

INC    $C03E
STA    $C03D
LDA    #$C0          ; Registre de taille du son.
STA    $C03E
LDA    #$36
STA    $C03D
INC    $C03E
STA    $C03D
LDA    #$A0          ; Registre de controle de l'oscillateur
STA    $C03E
LDA    #$02
STA    $C03D
INC    $C03E
LDA    #$12          ; A partir d'ici, le son commence à
                        ; être joué, mais au début on ne
                        ; l'entend qu'à droite

```

```
*=====
```

```
* Modification du volume
```

```
*=====
```

```

LDY    #$FF          ; On va maintenant faire "glisser" le son
                        ; vers la gauche, progressivement 255
                        ; fois

```

```

BOUCLE LDA    #$40
STA    $C03E
LDA    $C03D
LDA    $C03D
DEC                    ;
STA    $C03D
LDA    #$41
STA    $C03E
LDA    $C03D
LDA    $C03D
INC                    ;
STA    $C03D
LDX    #$FF

```


BOUCLE2	JSR	ATTENTE	; il est nécessaire d'avoir une routine
	DEX		; de timing qui dépendra de la longueur du
	BNE	BOUCLE2	; son et de la valeur de la fréquence.
	DEY		
	BNE	BOUCLE	
	RTS		; Fin de routine.
ATTENTE	DS	18,\$EA	; Dans ce cas précis (longueur du son de
			; \$4000 octets et fréquence de \$120), il
			; faut 18 NOP (dont le code est \$EA).
	RTS		; Fin du sous-programme de timing.

Notez que le réglage à pour fonction de bien répartir l'effet de passage du volume de la droite vers la gauche, afin que le volume maximum arrive sur le haut-parleur de gauche exactement à la fin du son.

6.1.6.3 Fréquences décalées

L'effet appelé «Fréquences décalées» peut être parfois très surprenant lorsque l'on ne s'y attend pas. Le principe consiste à faire jouer un même son par plusieurs oscillateurs à la fois, mais pas avec la même fréquence. Par exemple, le premier oscillateur jouerait le son avec une fréquence de \$120, le deuxième avec \$121, le troisième avec \$122, etc... Le mode de fonctionnement de tous les oscillateurs serait soit le «One-shot», soit le «Free-run». Quoi qu'il en soit l'effet est spectaculaire car au début les sons joués commencent en même temps, mais très vite ils se décalent les uns des autres. Plus vous utiliserez un nombre important d'oscillateurs, plus le résultat sera saisissant, mais veillez tout de même à ne pas saturer les enceintes en baissant le volume des oscillateurs le cas échéant.

6.1.6.4 Ensoniq et le hasard

Il arrive assez souvent que les programmeurs aient besoin de nombres aléatoires dans leurs applications, notamment s'il s'agit de jeux où le hasard tient une place importante. Il existe divers moyens d'obtenir des nombres tirés au hasard, et je vais vous proposer ici une méthode originale basée sur une utilisation détournée des registres du DOC de la série \$60.

En effet, ces registres qui contiennent la dernière valeur lue par leur oscillateur correspondant (cf. paragraphe 6.1.1.3.3) n'ont pas d'utilisation pratique pour le son. Mais imaginons maintenant qu'un oscillateur joue un son de 256 octets en continu, à volume nul. Il suffirait de lire son registre de donnée correspondant pour obtenir une des 256 valeurs contenues dans l'onde de la Ram Son. Il suffirait donc de mettre des valeurs

quelconques dans cette onde (lire lorsque l'on en a besoin le registre de donnée pour obtenir un nombre aléatoire. Voici les bases d'un exemple :

Programmer les 256 premiers octets de la Ram Son avec une suite de \$01 et de \$02.

Programmer l'oscillateur n \$00 pour jouer les 256 premiers octets de la Ram Son en mode «Free-run», fréquence \$99 (choisir de préférence une fréquence impaire), volume \$00 (car il n'y a aucun intérêt d'entendre ce son).

Quand vous aurez besoin d'une valeur tirée au hasard (ici soit le \$01, soit le \$02), effectuer les instructions suivante :

```
LDA    #$60          ; Accès au registre de donnée
```

```
STA    $C03E
```

```
LDA    $C03D          ; Deux lectures nécessaires.
```

```
LDA    $C03D          ; L'accumulateur contiendra une valeur aléatoire.
```

Les avantages d'une telle routine sont : la facilité, le fait qu'elle soit automatique, la possibilité de configurer exactement les nombres que vous souhaitez. En effet, vous pouvez très bien, dans notre exemple, mettre un plus fort pourcentage de \$01 que de \$02 afin de favoriser tel ou tel nombre. Et oui, vous pourrez vous même configurer les probabilités !

Seule restriction : il est nécessaire de lire le registre de donnée d'une façon irrégulière. Mais dans le cas d'un grand programme, il est rare d'avoir des cycles de temps réguliers.

6.2 La Musique

Si l'Ensoniq est capable de gérer automatiquement les sons, il n'en va pas de même pour la musique. Ainsi vous devrez vous même élaborer votre routine. L'idéal est de créer une routine de musique indépendante de votre programme, en utilisant les interruptions bien entendu. Comme cela il n'y aura pas de problème de décalage temporel dépendant de votre programme et, une fois lancée, la musique s'exécutera en prenant un minimum de temps machine. Mais de toute façon, faire de la musique sur un Apple IIGS n'est que l'aboutissement logique du son et la programmation reste d'un niveau simple.

6.2.1 La méthode des 64 K

Si pour une raison ou une autre vous avez besoin de tout votre temps machine ou de toute la RAM du IIGS et que vous désiriez tout de même entendre une musique, voici une méthode très simple. Il s'agit de numériser 64 K de musique depuis un CD ou autre, puis de jouer les quelques secondes en continu de façon à ce que l'on ait l'impression d'entendre un morceau sans fin. Bien entendu, vous aurez tout intérêt de choisir un

passage bien rythmé et sans paroles. Hélas, au bout de quelques instants d'écoute, même si le résultat est tout à fait acceptable, la musique n'en devient pas moins très lassante.

Donc, si vous choisissez cette solution, il vous suffira de programmer deux oscillateurs en mode «Swap», même fréquence pour les deux. Le premier oscillateur jouant les 32 premiers Kilos octets de la Ram Son, tandis que le second les autres 32K.

6.2.2 Musique et interruptions

Par contre, si vous cherchez une véritable solution pour jouer de la musique dans vos programmes, vous devrez utiliser les interruptions. Il est certain qu'il existe plusieurs méthodes possibles avec les interruptions, mais celle consistant à s'en servir comme tempo me paraît la plus intéressante du point de vue facilité et rapidité d'exécution.

Ici, les sons représentant les instruments seront stockés en Ram Son, et les notes avec leurs durées seront inscrites quelque part en RAM normale. Les notes devront être converties en fréquences selon une table donnée, tandis que les durées seront en fait des délais d'attente entre chaque changement de note. Hélas il est très difficile de trouver les véritables fréquences correspondantes aux notes de la gamme, étant donné que les sons n'ont pas tous été numérisés à la même fréquence.

Pour fonctionner, la routine devra interrompre le programme en cours tout les x temps et effectuer les divers changements de notes et de durée. Bien entendu, le programmeur devra veiller à ce que sa routine de musique soit la plus rapide possible afin de ne pas trop perturber le programme principal.

Voici donc comme exemple une petite routine jouant une musique avec un seul son. La routine utilise alternativement l'oscillateur n°00 et n°01 pour éviter d'entendre la mise en action et l'arrêt des oscillateurs.

Pour exécuter cette routine, il suffit de charger notre son :

```
]BLOAD SOUND           (chargement du son en $2000)
]CALL-151              (passage sous moniteur)
3/0<0/2000.5FFFFM     (transfert du son en $030000-$033FFF)
```

Puis il suffit de lancer la routine en \$1000.

*=====

* Routine de musique

*=====

XC ; Directive d'assemblage.

XC

```
MEMOIRE    $3000    ; Banc de stockage des sons.
FREQBAS    =$FE    ; Deux octets de la page zéro
FREQHAUT   =$FF    ; utilisés pour la sauvegarde des
              ; fréquences.
ORG        $1000   ; Ou ailleurs..
CLC        ; Mode natif
XCE
```

*=====

* Écriture en RAM son de 64K de données

*=====

```
REP    #$30    ; Registres X,Y sur 16 bits
SEP    #$20    ; et A sur 8 bits.
LDAL   $E100CA ; On choisit d'accéder à la
AND    #%00001111 ; Ram Son avec auto-incrémentation
ORA    #%01100000 ; des pointeurs...
STA    $C03C
STZ    $C03E    ; Mise à zéro des deux pointeurs
STZ    $C03F
LDX    #$0000   ; On veut lire depuis le début du banc.
ENCORE LDAL MEMOIRE,X ; Lecture d'une valeur du banc (ici le
              ; banc $03).
STA    $C03D    ; Transfert en Ram Son. A noter qu'en
              ; même temps
              ; les pointeurs sont augmentés d'une
              ; unité, donc il est inutile de
              ; s'occuper des pointeurs.
INX    ; Valeur suivante.
BNE    ENCORE   ; Jusqu'à ce que les 64K soient
              ; transférés
```

*=====

* Détournement du vecteur son

*=====

```

SEP   #$30
LDA   #$5C           ; $5C = opcode du JMP long.
STAL  $E1002C       ; Le vecteur d'interruption du son
LDA   #<INTER       ; se trouve en $E1002C.
STAL  $E1002D
LDA   #>INTER
STAL  $E1002E
LDA   #$00
STAL  $E1002F

```

```
*=====
```

```
* Mise en place du tempo *
```

```
*=====
```

```

LDAL  $E100CA
AND   #$0F
STA   $C03C         ; Accès aux registres du DOC.
LDA   #$1F         ; On utilisera l'oscillateur n $1F
STA   $C03E         ; pour le tempo, avec une fréquence
                    ; de $100.
LDA   #$00         ; En augmentant la fréquence, vous
STA   $C03D         ; jouerez la musique plus rapidement
LDA   #$3F
STA   $C03E
LDA   #$01
STA   $C03D
LDA   #$5F         ; Le volume de cet oscillateur est à
STA   $C03E         ; zéro étant donné que nous ne
LDA   #$00         ; voulons pas entendre le son du
STA   $C03D         ; tempo
LDA   #$9F
STA   $C03E
LDA   #$00         ; Le «Son» joué commence en $0000 de
STA   $C03D         ; la Ram Son et n'a qu'une durée de
                    ; 256 octets
LDA   #$BF
STA   $C03E

```

```

LDA  #$00
STA  $C03D
LDA  #$BF
STA  $C03E
LDA  #$08      ; Le mode utilisé est le «Free-run»
STA  $C03D      ;

```

```
*=====
```

```
* Valeurs fixes pour les oscillateurs n $00 et n $01 *
```

```
*=====
```

```

LDA  #$40      ; Le volume des deux oscillateurs
STA  $C03E      ; sera à $50
LDA  #$50
STA  $C03D
INC  $C03E
STA  $C03D
LDA  #$80      ; Le son utilisé commencera en $0000
STA  $C03E      ; de la Ram Son.
LDA  #$00
STA  $C03D
INC  $C03E
STA  $C03D
LDA  #$C0      ; Le son aura une taille de $4000
STA  $C03E      ; octets.
LDA  #$36
STA  $C03D
INC  $C03E
STA  $C03D

```

```
*=====
```

```
* Initialisation *
```

```
*=====
```

```

REP  #$30      ;
STZ  NUMERO    ; Remise à zéro du compteur des
STZ  DELAI     ; notes et du délai.

```

```

RTS          ; Retour à l'appelant. A cet
              ; instant, la musique est lancée et
              ; fonctionne automatiquement.

```

```

*=====

```

```

* Routine d'interruption *

```

```

*=====

```

```

INTER      REP  #$30      ; A chaque interruption provoquée
              ; par l'oscillateur n $1F (tempo),
              ; le IIGS exécute cette routine.

```

```

          LDA  DELAI      ; Si le délai est arrivé à zéro,
          BEQ  LIBRE      ; alors c'est que la note jouée est
              ; terminée. Sinon elle est en train

```

```

          DEC  DELAI      ; d'être jouée et on décrémente le
          BRA  FIN        ; le délai, puis on sort

```

```

LIBRE      LDX  NUMERO    ; X contient le numéro (*2) de la
              ; note à jouer.

```

```

          LDA  MUSIQUE,X  ; A contient la note et sa durée.
          CMP  #$FFFF     ; Si A=$FFFF alors on est en fin
              ; de morceau

```

```

          BEQ  RESET     ; et on se branche à RESET pour
              ; recommencer.

```

```

          AND  #$00FF    ; On ne garde que la note.

```

```

          ASL                ; Multiplication par 2 car les

```

```

          TAY                ; fréquences sont codées sur deux
              ; octets. Puis transfert dans Y.

```

```

          LDA  NOTE,Y      ; A contient maintenant la fréquence
              ; sur 16 bits.

```

```

          SEP  #$20        ; Mise de A sur 8 bits. Mais les 8
              ; bits de poids fort de A restent
              ; cependant inchangés.

```

```

          STA  FREQBAS     ; Sauvegarde dans une mémoire de la
              ; fréquence (bas)

```

```

          XBA                ; Les 8 bits de poids forts de A
              ; sont échangés avec les 8 bits de
              ; poids faible.

```

```

STA  FREQHAUT      ; Sauvegarde de la fréquence (haut).
LDA  #$A0          ; Accès au registre de mode de
CLC                                     ; l'oscillateur n$00 ou $01 en
                                     ; fonction de celui qui est en train
ADC  UNCOUP        ; de jouer la note.
STA  $C03E
LDA  #$03          ; Mise à l'arrêt de cet oscillateur
STA  $C03D          ; au cas où il ne se serait pas
                                     ; encore arrêté de lui même
LDA  UNCOUP        ; Commutation de la variable UNCOUP
EOR  #$01          ; pour changer d'oscillateur
STA  UNCOUP
LDA  #$00          ; Programmation de la fréquence de
                                     ; l'oscillateur
CLC
ADC  UNCOUP
STA  $C03E
LDA  FREQBAS
STA  $C03D
LDA  #$20
CLC
ADC  UNCOUP
STA  $C03E
LDA  FREQHAUT
STA  $C03D
LDA  #$A0          ; Mise en action de l'oscillateur avec
CLC                                     ; le mode "One-Shot", canal de droite
ADC  UNCOUP
STA  $C03E
LDA  #$02
STA  $C03D
REP  #$30          ; Passage de A sur 8 bits
LDA  MUSIQUE,X     ; Le registre X qui est toujours en 16
                                     ; bits contient encore la valeur du
                                     ; numéro de la note.
XBA                                     ; On commute A, pour accéder à la
                                     ; durée

```



```

AND   #$00FF      ; et on ne garde que la durée.
STA   DELAI       ; Stockage de la durée dans la variable
                                ; DELAI
INX                                       ; Note suivante
INX
STX   NUMERO      ; On sauve dans NUMERO
BRA   FIN         ; Et c'est fini
RESET LDX   #$0000 ; Mise à zéro de la variable NUMERO
STX   NUMEROFIN
CLC                                       ; CLC avant de sortir d'une routine
                                ; d'interruption
RTL

```

```
*=====
```

```
* Table et sauvegarde des données *
```

```
*=====
```

```
; Table de conversion note --> fréquence (les fréquences sont codées sur 16 bits)
```

```
NOTE      HEX   60008000A000C000
          HEX   E000000120014001
```

```
NUMERO    HEX   0000      ; 2 octets réservés pour stocker le numéro de la
                                ; note. Les notes étant sur 2 octets, le numéro
                                ; est multiplié par 2.
```

```
UNCOUP    HEX   0000      ; la variable UNCOUP est utilisée pour
                                ; déterminer si c'est l'oscillateur n $00 ou
                                ; n $01 qui doit être utilisé.
```

```
; Table contenant le numéro de musique selon la structure suivante:
```

```
; Un octet pour la note, suivi d'un octet pour la durée.
```

```
; Une paire $FFFF signifie la fin du morceau
```

```
MUSIQUE   HEX   0110011002200330
          HEX   0110052005200340
          HEX   0610071006100510
          HEX   02200220FFFF
```

Certes cet exemple ne joue de la musique que sur une seule voix, alors que le IIGS possède 32 oscillateurs. En tenant compte qu'il est préférable de prendre deux oscillateurs

par instrument et que l'oscillateur n \$1F est utilisé pour le tempo, cela nous fait 15 voix utilisables plus l'oscillateur n \$1E de libre.

Création, mise en page, maquette, Impression

Sauveur Cacoub, Forgest

Imprimé en France

Dépot légal, mars 1990

"Le livre français qui vous fera comprendre le IIGS "

par :

**D.BÄR
D.DELAY
Y.DURANT
J.L. SCHMITT
E.ric WEYLAND**

Dans ce livre de 464 pages, vous trouverez des informations techniques détaillées sur :

le 65C816, ses registres, ses instructions, ses modes d'adressage ...
les drives 5,25" & 3,5", lecture, écriture, déplacement du bras ...
le SmartPort ...
l'organisation de la mémoire, les softswiches, le shadowing,
la BRAM ...
le graphisme, les animations, les interruptions, la structure SCB ...
le système d'exploitation GS/OS et Prodos, l'organisation des
données sur disque ...
le son, la Ram son, le GLU, le DOC, l'Ensoniq, la stéréo...

ISBN 2-9504508-0-6

Mars 1990, Edité par *Toolbox*
Dépôt légal, 3/90

Imprimé en France

Conception, mise en page, maquette, impression
Sauveur Cacoub, Forgest