le IIGS "épluché"

Tome 1

par :

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

édité par *Toolbox*

IIGS "épluché"

SOMMAIRE

le IIGS "épluché"

Tome 1

par :

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

édité par Toolbox

le IIGS "épluché" ou... le IIGS « *Au ras du métal* »

Tome 1

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 // "

A propos des auteurs :

Par ordre alphabétique :

Monsieur D. Bär,
Etudiant.

Il a notamment participé à la programmation de Nucleus, et de Photonix. Il développe actuellement un jeu de course de voitures, sur 2 GS.

Il est l'auteur de la partie son.

Monsieur D. Delay,
Actuellement étudiant en médecine.

Il a 8 ans d'expérience en programmation sur Apple.

Il est l'auteur de la partie graphisme et animation.

Monsieur Y. Durant,
Titulaire d'un DEA de système informatique.
Ingénieur informatique.

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

Monsieur J-L. Schmitt,
Diplomé 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.

Monsieur E. Weyland,
Maitrise en Econométrie

Possède un Apple II depuis 1980
Il est l'auteur de la partie sur le GS/OS.

Bibliographie :

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

ISBN 2-9504508-0-6

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

Imprimé en France
Conception, mise en page, maquette, impression
Sauvear Cacoab, Forgest

Préface

L'Apple // , encore et toujours !

Eplucher 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'ére 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 anar, 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 // .

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 mûs 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 profession-

nalisé 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 //) 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 // .

L'Apple // lui-même a changé: il s'appelle désormais Apple // GS. 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 // ?

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 // 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 // GS est un ordinateur d'aujourd'hui (16 bits, graphisme, son, etc...), c'est toujours un Apple // : 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 supermoniteur, 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 // est bien vivant.

Ceux qui ont écrit cet ouvrage ne sont pas des développeurs professionnels sur //Gs. 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'aprés 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 // GS, c'est qu'on ne dispose, en Super Haute Résolution, que d'une seule page graphique, à la différence du graphisme HGR Apple // . Ce qui est génant pour les animations. Par ailleurs, pour être compatible avec l'Apple //, l'Apple // GS dispose d'un systéme 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'aurai-je pas l'équivalent d'une pseudo-page 2?

Toujours cette mentalité Apple // :
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 //, c'est déjà comme cela qu'on avait inventé la Double Haute Résolution.

C'est cela l'esprit Apple // : les raisons qui font que nous avons choisi un Apple // GS comme ordinateur personnel ne sont pas d'abord techniques (bien que le GS 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 GS.

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 // GS,

mais pas seulement: en mettant du Macintosh dans le GS, Apple a introduit le loup (les bidouilleurs sur Apple //) dans la bergerie du Mac. L'Apple // GS 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 débuguer entiérement 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 // GS, 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, suprémement 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'empécher de faire certaines choses: il y a par exemple, dans la Rom de l'Apple // e, 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 // +, 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 particuliérement 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 // GS. 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 // GS: elle fait de lui un petit Mac //, 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 // 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 // GS est aussi un Apple //, et un Apple // 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 quatriéme couche (le compilateur et ses librairies). 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 gére 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 // , 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 // GS 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 // 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 régilière sur l'Apple // appelée «Apple // for ever»

Le 5 Février 1990.

Introduction

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 IIGS de Madame Nicole Breaud-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 taches à des outils, qui prennent énormement 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 2 GS, et il est d'ailleurs amusant de constater que ces pseudos-informaticiens ont disparu avec l'évolution technique qu'a représenté le 2 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, notament sur les périphériques, les extentions, si ce livre connait dans le milieu du IIGS, un réel succès. Nous espérons simplement que vous l'apprecierez, et que vous comprendrez mieux le fonctionnement de l'Apple IIGS. Ceci afin que le IIGS ai la même durée de vie que le 2e. **C'est notre seul souhait.**

Pour toutes questions, remarques, ou suggestion 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

I

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 boitier.

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 à 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'éxécution 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 accéssibles par les programmeurs.
Ces registres se classent eux-mêmes en trois sortes :

1.0.1.3/1 *les registres de données,*

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 connait 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 selectionner une instruction parmi 256 positions de mémoire.

1.1. Le microprocesseur du GS.

1.1.1 Introduction

Le microprocesseur de l'apple 2 Gs est le 65C816 concu 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 fabiqué 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 à l'endroit effectif de la mémoire. Les 8 bits de poids fort sont stockés à l'endroit effectif de la mémoire plus un.

1.1.2 les registres

Les registres peuvent être considérés comme des mémoires, mais à 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 évidement les 3 nouveaux seront vus dans le détail ci après.

Shéma des registres du 65C816

8 bits	8 bits (poids fort)	8 bits (poids faible)
Registre DBR	Registre d'index X	Registre d'index X
Registre DBR	Registre d'index Y	Registre d'index Y
00	Pointeur de pile (S)	Pointeur de pile (S)
	Accumulateur (B)	Accumulateur (A)
Registre PBR	Compteur (PC) (PCH)	Ordinal (PCL)
00	Registre D	Registre D

1.1.2.1 *L'accumulateur*

L'accumulateur est un registre de 16 bits, où toutes les valeurs sont gardées pendant que des opérations arithmétiques où logiques sont effectuées. Les 16 bits du registre sont disponibles dans les 2 modes du 65C816, aussi bien en mode émulation qu'en mode natif. 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.

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 appellé 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 0.
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.
Voyons maintenant bit par bit le registre P :

bit 0 ou C —> (carry generated) c'est le bit de report ou de retenu. Ce bit a un rôle double. Il indique si une opération de soustraction ou d'addition a engendré une retenue, et sert de neuvième bit dans les opérations de rotation ou de décalage dans le registre d'état du processeur (instructions ASL, LSR, ROR et ROL)

ou E —> (emulation mode) si ce bit est à 1 le 65C816 fonctionnera en mode émulation (65C02). Les instructions sur la pile et les opérations arithmétiques seront effectuées en 8 bits. Les autres opérations restent effectuées en 16 bits. Les registres X et Y seront en mode 8 bits.Par contre si ce bit est à 0 le 65C816 est en mode natif, toutes les opérations sous ce mode sont des opérations en 16 bits.

bit 1 ou Z —> (zero result) ou bit zéro : Ce bit sera à zéro si la dernière opération n'a pas engendrée 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 (6502) ce bit est toujours à 1, et les registres X et Y fonctionnent en 8 bits.

ou B —> (interrupt source flag) : En mode émulation 65C02, ce bit est à 1,lorque la dernière instruction exécutée était BRK.

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.

bit 6 ou V —> (accumator oVerflov) : 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 définis une fois pour toute 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.

Légende :	.	=	bit non modifié
	/	=	bit modifié
	0	=	bit mis à zéro
	1	=	bit mis à un
	Acc	=	Accumulateur
	X	=	Registre d'index X
	Y	=	Registre d'index Y
	Mem	=	Mémoire

N V M X D I Z et C sont les bits du registre d'état du processeur.

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 <— A + M + C

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.	.	.	.	/	/
Registre	**Acc**	**X**	**Y**	**Mem**				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu	ADC addr	6D	3	4 *(1,4)
Absolu long	ADC long	6F	4	5 *(1,4)
Absolu long indéxé par X	ADC long,X	7F	4	5 *(1,4)
Absolu indéxé par X	ADC addr,X	7D	3	4 *(1,3,4)
Absolu indéxé par Y	ADC addr,Y	79	3	4 *(1,3,4)
Direct	ADC dp	65	2	3 *(1,2,4)
Direct indirect	ADC (dp)	72	2	5 *(1,2,4)
Direct indirect indéxé par Y	ADC (dp),Y	71	2	5*(1,2,3,4)
Direct indirect long	ADC <dp>	67	2	6 *(1,2,4)
Direct indirect long indéxé par Y	ADC <dp>,Y	77	2	6 *(1,2,4)
Direct indéxé indirect	ADC (dp),X	61	2	6 *(1,2,4)
Direct indéxé par X	ADC dp,X	75	2	4 *(1,2,4)
Immédiat en émulation 65C02 (M=1)	ADC #const	69	2	2 *(1,4)
Immédiat en mode natif 65816 (M=0)	ADC#const	69	3	3 *(1,4)
Pile relative	ADC sr,S	63	2	4 *(1,4)
Pile relative indirecte indexée	ADC (sr,S),Y	73	2	7 *(1,4)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0

(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.

(4) = ajouter 1 au nombre de cycles si le bit D (Decimal flag) du registre d'état du processeur est à 1 et si le 65C816 est en mode émulation (6502).

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 :

Deuxième opérande		0	1
Première opérande	0	0	0
	1	0	1

Fonction : A <— A ^ M

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 indéxé par X	AND addr,X	3D	3	4 *(1,3)
Absolu indéxé par Y	AND addr,Y	39	3	4 *(1,3)
Absolu long indéxé 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 indéxé par Y	AND (dp),Y	31	2	5*(1,2,3)
Direct indirect long	AND <dp>	27	2	6 *(1,2)
Direct indéxé indirect	AND (dp,X)	21	2	6 *(1,2)
Direct indirect long indéxé par Y	AND <dp>,Y	37	2	6 *(1,2)
Direct indéxé 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)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0.

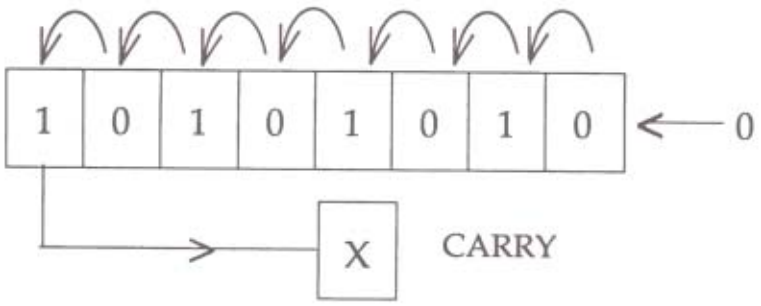
(2) = ajouter 1 au nombre de cycles si l'octet de poids faible du registre D (direct page register) est différent de zéro.

(3) = ajouter 1 au nombre de cycles si l'addition du registre d'index fait changer de page mémoire.

ASL

(Arithmetic Shift Left)

Fonction :



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) à gauche est transféré dans la retenue. En mode 16 bits le bit 0 est mis à zéro, tandis que le 15 ème bit est transféré dans la retenue.
Cette opération correspond en fait à une multiplication par 2.

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	/
Registre	**Acc**	**X**	**Y**	**Mem**				
	/	.	.	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Accumulateur	ASL A	0A	1	2
Absolu	ASL addr	0E	3	6 *(1)
Absolu indéxé par X	ASL addr,X	1E	2	7 *(1)
Direct	ASL dp	06	2	5 *(1,2)
Direct indéxé par X	ASL dp,X	16	2	6 *(1,2)

pour le calcul du nombre de cycles

(1) = ajouter 2 au nombre de cycles si M=0.

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

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 BCC qui est exécutée. Ce branchement permet

d'accéder à toute instruction située entre -128 et + 127 octets à partir de l'instruction suivante 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).

Fontion : Si C=0 alors PC <— PC+N

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Relatif	BCC adr	90	2	2 *(1,2)

pour le calcul du nombre de cycles :

(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 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 à 1 dans ce cas un branchement relatif au registre PC (program counter) est alors exécuté. Dans le cas contraire c'est l'instruction suivante BCS qui est exécutée. Ce branchement permet d'accéder à toute instruction située entre -128 et + 127 octets à partir de l'instruction suivante BCS. 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 grand que» ou «égal à» dans ce cas le branchement est effectué, ou bien «plus petit que» dans ce cas le branchement n'est pas effectué.

Fonction : Si C=1 alors PC <— PC + N

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Relatif	BCS adr	B0	2	2 *(7,8)

pour le calcul du nombre de cycles :

(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 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 executée.
Ce branchement permet d'accéder à toute instruction située entre -128 et + 127 octets à partir de l'instruction suivante BEQ.
BEQ est surtout utilisé pour déterminer si le résultat d'une comparaison est zéro (les deux valeurs comparées sont égales).

Fonction : Si Z=1 alors PC <— PC + N

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Relatif	BEQ adr	F0	2	2 *(1,2)
Relatif	BEQ adr	F0	2	2 *(1,2)

pour le calcul du nombre de cycles :

(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 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 < — A ^ M

Modification des registres :

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

Registre	Acc	X	Y	Mem
	.	.	.	.

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Immédiat Emulation 65C02 (M=1)	BIT #const	89	2	2
Immediat Natif 65816 (M=0)	BIT #const	89	3	3
Absolu	BIT addr	2C	3	4 *(1)
Absolu indéxé par X	BIT addr,X	3C	3	4 *(1,3)
Direct	BIT dp	24	2	3 *(1,2)
Direct indéxé par X	BIT dp,X	34	2	4 *(1,2)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0

(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 <— PC + N

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Relatif	BMI adr	30	2	2 *(1,2)

pour le calcul du nombre de cycles :

(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 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 BMI.
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 <— PC + N

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Relatif	BNE adr	D0	2	2 *(1,2)

pour le calcul du nombre de cycles :

(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 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 N=0, alors PC <— PC + N

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Relatif	BPL adr	10	2	2 *(1,2)

pour le calcul du nombre de cycles :

(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 en mode émulation (65C02/ E=1)

BRA

(BRanch Always)

Le branchement est toujours effectué (branchement inconditionnel). Cette instruction est équivalente à JMP 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 <— PC + N

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Relatif	BRA adr	80	2	3 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si le branchement conduit à changer de page mémoire 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 00FFE6,00FFE7.
En mode 8 bits le registre compteur ordinal est forcé à l'adresse mémoire contenue dans le vecteur d'interruption en 00FFFE,00FFFF.

Fonction : en mode émulation (65C02 / E=1)

PCL <— ($00FFFE); PCH <— ($00FFFF)

en mode natif (65816 / e=0)

PCL <— ($00FFE7); PCH <— ($00FFE6)

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
Emulation 65c02	.	.	/	.	0	1	.	.
Natif 65816	.	.	.	.	0	1	.	.

Registre	Acc	X	Y	Mem
	.	.	.	.

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	BRK	00	2	7 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si le 65C816 est en mode natif (E=0)

BRL

(BRanch always Long)

Branchement inconditionnel long. Cette instruction est similaire à 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 JMP est qu'elle est entièrement relogeable. Cependant on gagne un cycle en exécutant un JMP en mode absolu.

Fonction : PC <— PC + NN

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 <— PC + N

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Relatif	BVC	50	2	2 *(1,2)

pour le calcul du nombre de cycles :

(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 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 à 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 <— PC + N

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Relatif	BVS adr	70	2	2 *(1,2)

pour le calcul du nombre de cycles :

(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 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 but de transformer cette 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 <— 0

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	0
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	CLC	18	1	2

CLD

(CLear Decimal mode 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 <— 0

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	0	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 hardwares sont ignorées.

Fonction : I <— 0

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	0	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 zéro.

Fonction : V <— 0

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	0	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 existe 2 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 ==>A > M
A < M
A = M

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	/
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 indéxé par X	CMP addr,X	DD	3	4 *(1,3)
Absolu indéxé par Y	CMP addr,Y	D9	3	4 *(1,3)
Absolu long indéxé par X	CMP long,X	DF	4	5 *(1)
Direct	CMP dp	C5	2	3 *(1,2)
Direct indirect	CMP (dp)	D2	2	5 *(1,2)
Direct indirect indéxé par Y	CMP (dp),Y	D1	2	5 *(1,2,3)
Direct indirect long	CMP <dp>	C7	2	6 *(1,2)
Direct indirect long indéxé par Y	CMP <dp>,Y	D7	2	6 *(1,2)
Direct indéxé indirect	CMP (dp,X)	C1	2	6 *(1,2)
Direct indéxé 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)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0.

(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 chaine entre $80 et $FF ont été réservées par le concepteur du 65C816 (Western Design Center).

Fonction :

en mode emulation (65C02 / e=1)

PCL <— ($FFF4) ; PCH <— ($FFF5) ; PB <— 00

en mode natif (65C816 / e=0)

PCL <— ($FFE4) ; PCH <— ($FFE5) ; PB <— 00

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	0	/	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	COP const	02	2	7 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycle si le 65C816 est en mode natif (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 existe 2 possibilités :

Le registre d'index X est en mode 8 bits : la comparaison sur fait sur 8 bits

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 ==> **X > M**
X < M
X = M

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	/
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si X = 0.

(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 Y avec le contenu d'une mémoire. Cette comparaison correspond en fait à une soustraction virtuelle 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 existe 2 possibilités :

Le registre d'index Y est en mode 8 bits : la comparaison sur fait sur 8 bits

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 ==> **Y > M**
Y < M
Y = M

Modification des registres :

Registre d'état du processeur	N /	V .	M .	X .	D .	I .	Z /	C /
Registre	Acc .	X .	Y .	Mem .				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si X = 0.

(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 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 #$00, le résultat sera alors de #$FF

Fonction : M <— M - 1 ou A <— A - 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu	DEC addr	CE	3	6 *(1)
Absolu indexé par X	DEC addr,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)

pour le calcul du nombre de cycles :

(1) = ajouter 2 au nombre de cycles si M=0.

(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' 1. Si le contenu original du registre X est de #$00, le résultat sera alors de #$FF

Fonction : X <— X - 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	DEX	CA	1	2

DEY

(DEcrement index register Y)

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

Fonction : Y <— Y - 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	.	.	/	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 :

Deuxième opérande / Première opérande	0	1
0	0	1
1	1	0

Fonction : A <— A EOR M

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	/	.	/					

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 indéxé par X	EOR addr,X	5D	3	4 *(1,3)
Absolu indéxé par Y	EOR addr,Y	59	3	4 *(1,3)
Absolu long indéxé	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 indéxé par Y	EOR (dp),Y	51	2	5*(1,2,3)
Direct indirect long	EOR <dp>	47	2	6 *(1,2)
Direct indirect long indéxé par Y	EOR <dp>,Y	57	2	6 *(1,2)
Direct indéxé indirect	EOR (dp,X)	41	2	6 *(1,2)
Direct indéxé par X	EOR dp,X	55	2	4 *(1,2)
Pile relative	EOR sr,S	43	2	4 *(1)
Pile relative indirecte indéxée	EOR (sr,S),Y	53	2	7 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0.

(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.

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 <— M + 1 ou A <— A + 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu	INCaddr	EE	3	6 *(1)
Absolu indexé par X	INC addr,X	FE	3	7 *(1)
Accumulateur	INC A	1A	1	2
Direct	INC dp	E6	2	5 *(1,2)
Direct indexé par X	INC dp,X	F6	2	6 *(1,2)

pour le calcul du nombre de cycles :

(1) = ajouter 2 au nombre de cycles si M=0

(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)

Une incrémentation de 1 est effectuée sur le registre d'index X. Si le contenu original du registre X est de #$FF, le résultat sera alors de #$00.

Fonction : X <— X + 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	INX	E8	1	2

INY

(INcrement index register Y)

Une incrémentation de 1 est effectuée sur le registre d'index Y. Si le contenu original du registre Y est de #$FF, le résultat sera alors de #$00.

Fonction : Y <— Y + 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	.	.	/	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 <— addr
PB <— addr + 2

Modification des registres :

Registre d'état du registre	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu indirect	JML addr	DC	3	6

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 est entièrement relogeable.

Fonction : PC <— addr

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu	JMP addr	4C	3	3
Absolu long	JMP long	5C	4	4
Absolu indirect	JMP(addr)	6C	3	5
Absolu indirect indéxé par X	JMP(addr,X)	7C	3	6

JSL

(Jump to Sub routine Long / Inter-bank)

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

Fonction : **(S) <— PBR, S <— S - 1**
(S) <— PCH, S <— S - 1
(S) <— PCL, S <— S - 1
PC <— ADDR, PBR <— ADDR + 2

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu long	JSL long	22	4	8

JSR

(Jump to SubRoutine)

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

Fonction : (S) <— PCH, S <— S - 1
(S) <— PCL, S <— S - 1
PC <— ADDR

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu	JSR addr	20	3	6
Absolu indexé indirect	JSR (addr,X)	FC	3	6

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 <— M

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 indéxé par X	LDA addr,X	BD	3	4 *(1,3)
Absolu indéxé par Y	LDA addr,Y	B9	3	4 *(1,3)
Absolu long indéxé 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 indéxé par Y	LDA (dp),Y	B1	2	5*(1,2,3)
Direct indirect long	LDA <dp>	A7	2	6 *(1,2)
Direct indirect long indéxé par Y	LDA <dp>,Y	B7	2	6 *(1,2)
Direct indéxé indirect	LDA (dp,X)	A1	2	6 *(1,2)
Direct indéxé par X	LDA dp,X	B5	2	4 *(1,2)
Pile relative	LDA sr,S	A3	2	4 *(1)
Pile relative indirecte indéxée	LDA (sr,S),Y	B3	2	7 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0

(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 <— M

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 indexé par Y	LDX dp,Y	B6	2	4 *(1,2)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si X=0

(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 <— M

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	/	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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	LDY 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)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si X=0.

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

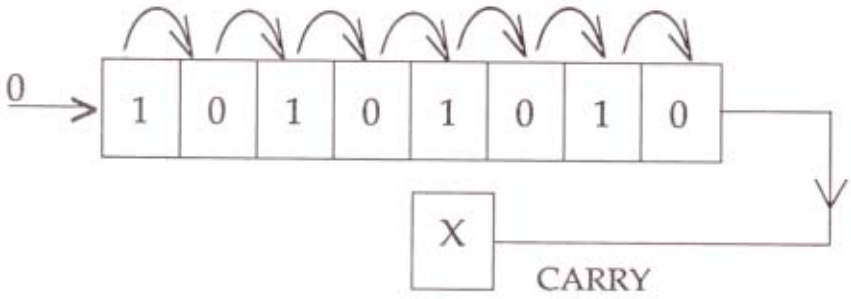
(3) = ajouter 1 au nombre de cycles si l'addition du registre d'index fait changer de page mémoire.

LSR

(Logical Shift memory or accumulator Right)

On décale à droite l'accumulateur ou le contenu d'une mémoire. En mode 8 bits (émulation 65C02) 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 consiste en fait à une division par 2.

Fonction :



Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	0	.	.	.	.	.	/	/
Registre	Acc	X	Y	Mem				
	/	.	.	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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)

pour le calcul du nombre de cycles :

(1) = ajouter 2 au nombre de cycles si M=0.

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

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 <— M, X <— X + 1, Y <— Y + 1, A <— A - 1, DBR <— N

Modification des registres :

Registre d'état du processeur	N .	V .	M .	X .	D .	I .	Z .	C .
Registre	Acc .	X .	Y .	Mem .				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Bloc de départ -> destination (source,destination)	MVN *	54	3	**

** MVN srcblk,destblk*

*** 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 <— M, X <— X - 1, Y <— Y - 1, A <— A - 1, DBR <— N

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	/	/	/	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Bloc d'origine -> destination (source,destination)	MVP *	44	3	**

** MVP srcbk,destbk*

*** 7 cycles par octet déplacé*

NOP

(No OPeration)

Le microprocesseur n'effectue aucune opération pendant 2 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 : /

Modification des registres :

Registre d'état: du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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.
Ou inclusif suit la table de vérité suivante :

Deuxième opérande		0	1
Première opérande	0	0	1
	1	1	1

Fonction : A <— A V M

Modification des registres :

Registre d'état du processeur	N /	V .	M .	X .	D .	I .	Z /	C .
Registre	Acc /	X .	Y .	Mem /				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Immédiat en émulation 65C02 (M=1)	ORA #const	09	2	2
Immédiat en mode natif 65816 (M=0)	ORA #const	09	3	3
Absolu	ORA addr	0D	3	4 *(1)
Absolu long	ORA long	0F	4	5 *(1)
Absolu indéxé par X	ORA addr,X	1D	3	4 *(1,3)
Absolu indéxé par Y	ORA addr,Y	19	3	4 *(1,3)
Absolu long indéxé	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 indéxé par Y	ORA (dp),Y	11	2	5 *(1,2,3)
Direct indirect long	ORA <dp>	07	2	6 *(1,2)
Direct indirect long indéxé par Y	ORA <dp>,Y	17	2	6 *(1,2)
Direct indéxé indirect	ORA (dp,X)	01	2	6 *(1,2)
Direct indéxé par X	ORA dp,X	15	2	4 *(1,2)
Pile relative	ORA sr,S	03	2	4 *(1)
Pile relative indirecte indéxée	ORA (sr,S),Y	13	2	7 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0.

(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.

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) <— PC + 1, S <— S - 1**

(S) <— PC + 2, S <— S - 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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) <— (D + (PC + 1)), S <— S - 1

(S) <— (D + (PC + 1) + 1), S <— S - 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile (direct indirect)	PEI (dp)	D4	2	6 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si les bits de poids faible du registre D sont différents 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) <— PC + NN + 2, S <— S - 2

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile (relatif long)	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) <— A, S <— S - 1 ou S <— S - 2

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	PHA	48	1	3 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0

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) <— DBR, S <— S - 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	PHB	8D	1	3

PHD

(PusH Direct page register on the stack)

Cette instruction place le contenu du registre D (registre direct - 16 bits) au sommet de la pile, le pointeur de pile S est mis à jour.

Fonction : (S) <— D, S <— S - 2

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	PHD	0B	1	4

PHK

(PusH program banK register on the stack)

Cette instruction place le contenu du registre PBR (Program bank register - 8 bits) au sommet de la pile, le pointeur de pile est mis à jour.

Fonction : (S) <— PBR, S <— S - 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	PHK	4B	1	3

PHP

(PusH Processor status register)

Cette instruction place le contenu du registre d'état du processeur 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.

Fonction : (S) <— P, S <— S - 1

Modification des registres :

Registre d'état du processeur	N .	V .	M .	X .	D .	I .	Z .	C .
Registre	Acc .	X .	Y .	Mem .				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	PHP	08	1	3

PHX

(PusH index register X)

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) <— X, S <— S - 1 ou S <— S - 2

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	PHX	DA	1	3 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si X=0

PHY

(PusH index register Y)

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) <— Y, S <— S - 1 ou S <— S - 2

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	PHY	5A	1	3 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si X=0

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 : S <— S + 1 ou S <— S + 2, A <— (S)

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	PLA	68	1	4 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0

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.

Fonction : S <— S + 1, DBR <— (S)

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 <— S + 2, D <— (S)

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	PLD	2B	1	5

PLP

(PulL status flag)

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 <— S + 1, P <— (S)

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	/	/	/	/	/	/
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 <— S + 1 ou S <— S + 2, X <— (S)

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	PLX	FA	1	4 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si X=0

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 <— S + 1 ou S <— S + 2, Y <— (S)

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	/	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	PLY	7A	1	4 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si X =0

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 <— P ^ N

Modification des registres :

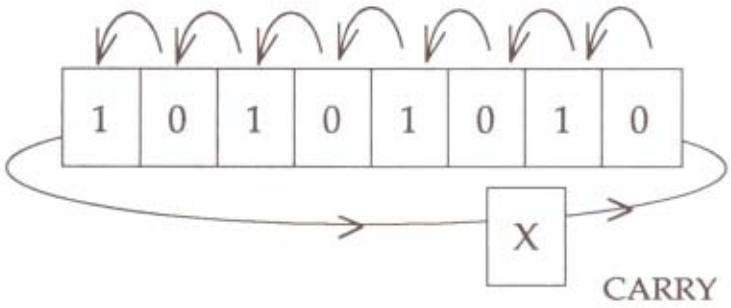
Registre d'état du processeur	N	V	M	X	D	I	Z	C
(mode émulation)	/	/	.	.	/	/	/	/
(mode natif)	/	/	/	/	/	/	/	/
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Immédiat	REP #const	C2	2	3

ROL

(ROtate memory or accumulator Left)

Fonction :



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

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	/
Registre	**Acc**	**X**	**Y**	**Mem**				
	/	.	.	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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)

pour le calcul du nombre de cycles :

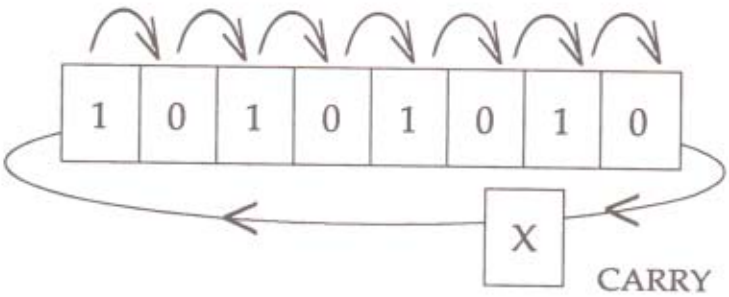
(1) = ajouter 2 au nombre de cycles si M=0.

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

ROR

(ROtate memory or accumulator Right)

Fonction :



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.

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	/
Registre	Acc	X	Y	Mem				
	/	.	.	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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)

pour le calcul du nombre de cycles :

(1) = ajouter 2 au nombre de cycles si M=0.

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

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 <— S + 1, P <— (S)
S <— S + 1, PCL <— (S)
S <— S + 1, PCH <— (S)
S <— S + 1, PBR <— (S)

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
(Mode émulation)	/	/	.	.	/	/	/	/
(Mode natif)	/	/	/	/	/	/	/	/
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	RTI	40	1	6 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si E=0 (mode natif du 65C816)

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 <— S + 1, PCL <— (S)**
S <— S + 1, PCH <— (S)
S <— S + 1, PBR <— (S)

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 <— S + 1, PCL <— (S)
S <— S + 1, PCH <— (S)

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Pile	RTS	60	1	6

SBC

(Substract with Borrow from aCcumulator)

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'ai pas d'influence sur la soustraction, il faut utiliser l'instruction SEC (SEt Carry Flag) avant d'utiliser SBC.

Fonction : A <— A - M - C

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	/	.	.	.	.	/	/
Registre	Acc	X	Y	Mem				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 indéxé par X	SBC addr,X	FD	3	4 *(1,3)
Absolu indéxé par Y	SBC addr,Y	F9	3	4 *(1,3)
Absolu long indéxé	SBC long,X	FF	4	5 *(1)
Direct page zéro	SBC dp	E5	2	3 *(1,2)
Direct indirect	SBC (dp)	F2	2	5 *(1,2)
Direct indirect indéxé par Y	SBC (dp),Y	F1	2	5*(1,2,3)
Direct indirect long	SBC <dp>	E7	2	6 *(1,2)
Direct indirect long indéxé par Y	SBC <DP>,Y	F7	2	6 *(1,2)
Direct indéxé indirect	SBC (dp),Y	E1	2	6 *(1,2)
Direct indéxé 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)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0.

(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 <— 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	1

Registre	Acc	X	Y	Mem
	.	.	.	.

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	SEC	38	1	2

SED

(SEt Decimal mode 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 <— 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	1	.	.	.

Registre	Acc	X	Y	Mem
	.	.	.	.

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 <— 1

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	1	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 <— P V N

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
(Mode émulation)	/	/	.	.	/	/	/	/
(Mode natif)	/	/	/	/	/	/	/	/
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Immédiat	SEP#const	E2	2	3

STA

(Store Accumulator to memory)

Cette instruction stocke l'accumulateur en mémoire.

2 possibilités :

L'accumulateur est en mode 8 bits : La valeur dans l'accumulateur est mise à l'adresse mémoire spécifiée par l'opérande.

L'accumulateur est en mode 16 bits : les 8 bits de poids faible de l'accumulateur sont 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 <— A

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu	STA addr	8D	3	4
Absolu long	STA long	8F	4	5
Absolu indéxé par X	STA addr,X	9D	3	5 *(1)
Absolu indéxé par Y	STA addr,Y	99	3	5 *(1)
Absolu long indéxé par X	STA long,X	9F	4	5 *(1)
Direct page zéro	STA dp	85	2	3 *(1,2)
Direct indirect	STA (dp)	92	2	5 *(1,2)
Direct indirect indéxé par Y	STA (dp),Y	91	2	6 *(1,2)
Direct indirect long	STA <dp>	87	2	6 *(1,2)
Direct indirect long indéxé par Y	STA <dp>,Y	97	2	6 *(1,2)
Direct indéxé indirect	STA (dp,X)	81	2	6 *(1,2)
Direct indéxé par X	STA dp,X	95	2	4 *(1,2)
Pile relative	STA sr,S	83	2	4 *(1)
Pile relative indirecte indexée	STA (sr,S),Y	93	2	7 *(1)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0.

(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.

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	STP	DB	1	3

STX

(STore index register X to memory)

Cette instruction stocke le registre d'index X en mémoire.

Il y a 2 possibilités :

Le registre d'index X est en mode 8 bits : La valeur dans le registre d'index X est mise à l'adresse mémoire spécifiée par l'opérande.
Le registre d'index X est en mode 16 bits : les 8 bits de poids faible du registre d'index X sont mis à l'adresse mémoire spécifiée par l'opérande, et les 8 bits de poids fort sont mis à l'adresse mémoire spécifiée par l'opérande plus une.

Fonction : M <— X

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu	STX addr	8E	3	4 *(1)
Direct	STX dp	86	2	3 *(1,2)
Direct indexé par Y	STX dp,Y	96	2	4 *(1,2)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si X=0.

(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 index register Y to memory)

Cette instruction stocke le registre d'index Y en mémoire.
2 possibilités :
Le registre d'index Y est en mode 8 bits : La valeur dans le registre d'index Y est mise à l'adresse mémoire spécifiée par l'opérande.
Le registre d'index Y est en mode 16 bits : les 8 bits de poids faible du registre d'index Y sont mis à l'adresse mémoire spécifiée par l'opérande, et les 8 bits de poids fort sont mis à l'adresse mémoire spécifiée par l'opérande plus une.

Fonction : M <— Y

Modification des registres

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu	STY addr	8C	3	4 *(1)
Direct	STY dp	84	2	3 *(1,2)
Direct indexé par Y	STY dp,X	94	2	4 *(1,2)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si X=0.

(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 to memory)

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 à l'adresse mémoire spécifiée par l'opérande plus une.
Fonction : M <— 0

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu	STZ addr	9C	3	4 *(1)
Absolu indexé par X	STZ addr,X	9E	3	5 *(1)
Direct page zéro	STZ dp	64	2	3 *(1,2)
Direct indexé par X	STZ dp,X	74	2	4 *(1,2)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0.

(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 Accumulator to index register X)

Cette instruction transfère les bits de 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 & 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 & 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 & 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 en 16 bits & 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 <— A

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	TAX	AA	1	2

TAY

(Transfer Accumulator to index register Y)

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

1) Registre d'index Y en 8 bits & 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 & 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 & 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 en 16 bits & 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 <— A

Modification des registres :

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

Registre	Acc	X	Y	Mem
	.	.	/	.

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	TAY	A8	1	2

TCD

(Transfer 16 bit aCcumulator to Direct page register)

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 <— C

Modification des registres :

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

Registre	Acc	X	Y	Mem
	.	.	.	.

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	TCD	5B	1	2

TCS

(Transfer C accumulator to Stack pointer)

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 <— C

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	TCS	1B	1	2

TDC

(Transfer Direct page register to 16 bit aCcumulator)

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 <— D

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	TDC	7B	1	2

TRB

(Test and Reset memory Bits against 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 <— M ^ A

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu	TRB addr	1C	3	6 *(1)
Direct	TRB dp	14	2	5 *(1,2)

pour le calcul du nombre de cycles :

1 = ajouter 1 au nombre de cycles si M=0.

2 = ajouter 1 au nombre de cycles si les bits de poids faible du registre D sont différents de zéro.

TSB

(Test and Set memory Bits against accumulator)

Cette instruction n'agit que sur le banc 0 et opère a 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 <— M V A

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	.	.	.	/				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Absolu	TB addr	0C	3	6 *(1)
Direct	TSB dp	04	2	5 *(1,2)

pour le calcul du nombre de cycles :

(1) = ajouter 1 au nombre de cycles si M=0

(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 Stack pointer to 16 bit aCcumulator)

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 <— S

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	TSC	3B	1	2

TSX

(Transfer Stack pointer to index register 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=O) ; la totalité des 16 bits du pointeur de pile sont transférés dans le registre d'index X

Fonction : X <— S

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code Octet	Cycle	
Implicite	TSX	BA	1	2

TXA

(Transfer index register X to Accumulator)

Transfert de X dans A
Cette instruction transfère le registre d'index X dans l'accumulateur. Le 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 & 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 & 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 & 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 en 16 bits & Accumulateur en 16 bits (X=0 ; M=0):
Les 16 bits du registre X sont transférés dans l'accumulateur.
Fonction : A <— X

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	TXA	8A	1	2

TXS

(Transfer index register X to Stack pointer)

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 <— X

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	TXS	9A	1	2

TXY

(Transfer index register 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 <— X

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	/	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	TXY	9B	1	2

TYA

(Transfer index register Y to Accumulator)

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

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

1) Registre d'index Y - 8 bits & Accumulateur - 8 bits :
Les 8 bits du registre d'index Y sont transférés dans l'accumulateur.

2) Registre d'index Y en 16 bits & 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 & 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 en 16 bits & Accumulateur en 16 bits (X=0 ; M=0):
Les 16 bits du registre Y sont transférés dans l'accumulateur.

Fonction : A <— Y

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	TYA	98	1	2

TYX

(Transfer index register Y to X)

Cette instruction transfère le 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 <— Y

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	Acc	X	Y	Mem				
	.	/	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
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 <— 0

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	Acc	X	Y	Mem				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	WAI	CB	1	3

WDM

(William D. Mensch, jr)

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

Fonction : /

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	.	.	.	.	.	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	WDM	42	?	?

XBA

(eXchange B and A accumalators)

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 <—> B

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	/	.	.	.	.	.	/	.
Registre	**Acc**	**X**	**Y**	**Mem**				
	/	.	.	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	XBA	EB	1	3

XCE

(eXchange Carry and Emulation 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 65C02 ou en 65C816, suivant que l'on est dans un mode ou dans l'autre.

Fonction : X <—> E

Modification des registres :

Registre d'état du processeur	N	V	M	X	D	I	Z	C
	.	.	/	/	.	.	.	/
Registre	**Acc**	**X**	**Y**	**Mem**				
	.	/	/	.				

Modes d'adressage	Syntaxe	Code	Octet	Cycle
Implicite	XCE	FB	1	2

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

Code Hexa	Mnémonique	Mode d'adressage
00	BRK	Pile
01	ORA	Direct indirect indéxé 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	Immédiat
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 indéxé par Y
12	ORA	Direct indirect
13	ORA	Relatif à la pile indirect indéxé par Y
14	TRB	Direct
15	ORA	Direct indéxé par X
16	ASL	Direct indéxé par X
17	ORA	Direct indirect long indéxé par Y
18	CLC	Implicite
19	ORA	Absolu indéxé par Y
1A	INC	Accumulateur
1B	TCS	Implicite
1C	TRB	Absolu
1D	ORA	Absolu indéxé par X
1E	ASL	Absolu indéxé par X
1F	ORA	Absolu long indéxé par X
20	JSR	Absolu
21	AND	Direct Indéxé 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

Code Hexa	Mnémonique	Mode d'adressage
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	Drect indexé par X
37	AND	Drect indirect long indexé par X
38	SEC	Implicit
39	AND	Absolu indexé par Y
3A	DEC	Accumulateur
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
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
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

Code Hexa	Mnémonique	Mode d'adressage
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	Drect indexé par X
37	AND	Drect indirect long indexé par X
38	SEC	Implicit
39	AND	Absolu indexé par Y
3A	DEC	Accumulateur
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
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
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

Code Hexa	Mnémonique	Mode d'adressage
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
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

Code Hexa	Mnémonique	Mode d'adressage
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
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
BD	LDA	Absolu indexé par X
BE	LDX	Absolu indexé par Y

Code Hexa	Mnémonique	Mode d'adressage
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
DD	CMP	Absolu indexé par XP
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

Code Hexa	Mnémonique	Mode d'adressage
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	SR	Absolu indexé indirect
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 faite 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'incrementation 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 poid 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 appellé «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'additionner 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 appellé «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 le 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 obtenu 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'effectué 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 à 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 incrémentations à 2, ainsi qu'une auto incrementation du compteur ordinal entre deux bornes, notament pour les consultations de tables.

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

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

II

la RAM du GS

2.0 Organisation

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

•la mémoire principale accessible directement au micro-processesseur.

•la mémoire du DOC (Ensoniq) Ram de 64k accessible directement par l'ensoniq et via une série de Softswitches (commutateurs logiciels) par le micro-processesseur.

•la mémoire Cmos (BRAM) sauvegardée par batterie et contenant la configuration du GS. (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 mére. 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 Gs implique qu'il est nécessaire de tranférer 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 page 1
2000-3FFF	Page Hires 1
4000-5FFF	Page Hires 2
2000-9FFF	Super Hires
2000-3FFF	Page Hires mémoire auxiliaire
C000-FFFF	E/S et carte langage

On remarque que la page Texte 2 ne peut pas bénéficier de l'effet Miroir, on est donc obligé de le simuler de facon logiciel 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 controlé 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 mémoires affectées par l'effet Miroir.
Il faut noter que l'utilisation de l'effet Miroir ralenti 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 phénoméme 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 trouve la plupart des variables utilisées en mode émulation, par l'applesoft, Prodos 8 et le moniteur.
En mode GS/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 a 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 chaine
$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 dernière ligne de la fenêtre
$24	CHColonne 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 tem poraire (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 Checksum de l'entête du secteur (DOS 3.3 RWTS)
$2D	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 Masque couleur pour graphismes basse résolution FORMAT Zone temporaire décodage de l'opcode Mini assembleur. Numéro de piste contenu dans l'entête (DOS 3.3 RWTS)
$2F	LASTIN Utilisé lors de la lecture cassette sur Apple IIe et II+. LENGTH 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 Caractére 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
$38-$39	CSWH caracteres. KSW Adresse de la routine d'entrée de KSWH caractéres.
$3A-$4F	—— Zone utilisée par PRODOS
$3A-$3B	PCL Sauvegarde du compteur programme PCH
$3C-$3D	Adresse du DCT (DOS 3.3 RWTS) 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 (DOS 3.3 RWTS)
$42-$43	Adresse d'un buffer de travail (DOS 3.3)
$44-$45	OPérande numérique (DOS 3.3)
$46-$47	Zone de travail (DOS 3.3 RWTS)
$48-$49	Adresse de l'IOB (DOS 3.3 RWTS)
$3C	XQT Zone pour le pas a pas (Step) et la trace
$3C-$45	A1L Zone des paramétres 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 aprés un BRK

$46-$47 Zone de travail (DOS 3.3 RWTS)
$48-$49 Adresse de l'IOB (DOS 3.3 RWTS)
$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 dy 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
D numéro du drive
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 chaines 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'execution
$ffff si mode direct

$77-$78 OLDLIN Numéro de la derniere ligne executée ou interompue

$79-$7A OLDTXPTR Adresse utilisée par CONT pour continuer l'éxécution d'un programme basic intérompu.

$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 chaine 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 5octets
$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'un chaine 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 autoéxécution 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'éxécution
$DA-$DB ERRLIN numéro de la ligne où l'erreur a eu lieu
$DC-$DD ERRPOS adresse du caractère ou 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 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 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 $3F2    ; SOFTEV
LDA #$F     ; 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 CTRL-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.

$3FB-$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 $EO - $E1

Dans ces bancs, composés de mémoire accédée à 1Mhz, se trouve 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-$0003 DISPATCH1 :
Saut au «TOOL LOCATOR» localisateur d'Outils
Normalement sous la forme JML$xx/xxxx 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 gestionaire 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 0 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 initulisée	3,S		
Adresse de retour inutilisée	2,S		
Adresse de retour inutilisée	1,S		

$0020-$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'interuption de fin de ligne «scan line interrupt»
$002C-$002F	IRQ.SOUND : Saut vers le gestionnaire d'interruption processeur sonore
$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 Miscellanous Tool set en installant une tache «hearbeat» (fonction SetHeartBeat) générant une interruption lors de la scrutation du clavier par le VBL. Si une touche est enfoncée la tache «heart beat» appelera 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 (Requette de service) de l'ADB (Apple Desk 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 gestion naire 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 inter ruption 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 appellée 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 BREAKVECTOR :
Saut vers une routine «personnelle» de traitement de l'opcode BRK. Cette routine doit obéir au caractéristiques suivantes :
•Elle est appellée 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 preservée.
•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 le programme interrompu par le BRK est poursuivi 2 octets après le BRK.
Ce vecteur est prévu a l'usage des logiciels de déboguage.

$0074-$0077 TRACEVECTOR :
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 interompu est continué, sinon le message Trace est affiché sur l'écran avant continuation du programme. Ce vecteur est prévu à l'usage des logiciels de déboguage.

$0078-$007B STEPVECTOR :
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 déboguage.

$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 II GS. 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 $E1BATTERYRAM dans la ram du circuit horloge avec la bonne somme de contrôle. Cette routine est appelée par le Miscellanous 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 défauts 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 inialisation 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 à jour que grâce au Miscellanous Tool Set.

$0098-$009B TOPRINTMSG8 :
Ce vecteur pointe vers une routine affichant la chaine ASCII pointée en multipliant le contenu de l'accumulateur par 2 et en s'en servant comme index dans la table pointé 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 TOPRINTMSG16 :
Ce vecteur pointe vers une routine affichant la chaine 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 CTRLYVECTOR :
Saut inconditionel 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 préser ver 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 éxécutera la routine pointée en bank $00 /USRADR. Ce vecteur est utilisé par les debuggueurs.

$00A4-$00A7 TOTEXTPG2DA :
Ce vecteur pointe vers l'accessoire «Aternate 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.

$00A8-$00AB 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

$00AC-$00AF

$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 OS_KIND $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'éxécution d'un appel

$00C0-00C2 MSGPOINTER :
Pointeur vers la tables des adresses des chaines utilisées par le Panneau de controle le moniteur ..., la 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
$1DD8-$1FFF	Buffer de la sortie série
$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

Octet Fonction Normalement Limites

Port 1 :

$00 Fonction
$00 $00-$01
$00 imprimante
$01 modem

$01 Longueur de ligne
$00 $00-$04
$00 nonlimite
$01 40 caractères
$02 72 caractères
$03 80 caractères
$04 132 caractères
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 $00-$01
$00 Absent
$01 Présent affiche les caracteres transmis sur l'écran.

$05 Buffer $00 $00-$01
$00 Non
$01 Oui

$06 Vitesse en bauds
$0D $00-$0E

$00	50	$07	1200
$01	75	$08	1800
$02	110	$09	2400
$03	134.5	$0A	3600
$04	150	$0B	4800
$05	300	$0C	7200
$06	600	$0D	9600
$0E	19200		

$07 Nombre de bits de données
$06 $00-$07
Nombre de bits 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

$08 Parité $02 $00-$02
$00 Impaire
$01 Paire
$02 Aucunne

$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

Port 2 : Idem port 1

$0C Fonction
$01 $00-$01

$0D Longueur de ligne
$00 $00-$04

$0E Supprimer le premier Saut de lignes $00 $00-$01
après le retour chariot.

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

$10 Echo $00 $00-$01

$11 Buffer $00 $00-$01

$12 Vitesse en bauds
$07 $00-$0E

$13 Nombre de bits de donnée
$06 $00-$07
Nombre de bits de stop

$14 Parité $02 $00-$02
$15 DCD Handshake
$01 $00-$01

$16 DSR/DTR Handshake
$01 $00-$01

$17 XON/XOFF Handshake
$00 $00-$01

Divers :

$18 Affichage couleur/monochrome $00 $00-$01
$00 Couleur
$01 Monochrone

$19 Affichage 40/80 colonnes
$00 $00-$01
$00 40 colonnes
$01 80 colonnes

$1A Couleur pour le 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 foncée	$0C Vert pale
$05	Gris sombre	$0D Jaune
$06	Bleu	$0E Aquamarine
$07	Bleu clair	$0F Blanc

$1B Couleur pour le fond
$06 $00-$0F

$1C Couleur pour le bord
$06 $00-$0F

$1D 50 ou 60 Hertz
$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 $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 colonne 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 démarage
$00 $00-$09
$00 Recherche des slots pour le démarrage commence la recherche à partir du slot 7
$01-$07 slot de démarrage
$08 Démarre à partir du disque RAM
$09 Démarre à partir du disque ROM

$29 Langue de l'affichage.
$00 $00-$1F

$00	Anglais (USA)	*	x	
$01	Anglais (U K)	*	x	
$02	Français	*	x	
$03	Danois		*	x
$04	Espagnol		*	x
$05	Italien		*	x
$06	Allemand		*	x
$07	Suédois		*	x
$08	«Dvorak»			x
$09	Canadien (français)			x
$0A	Flamand			
$0B	Hebreux			
$0C	Japonnais			
$0D	Arabe			
$0E	Grec			
$0F	Turc			

$10 Finnois
$11 Portuguais
$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 ci-dessus)

x *Valeurs correspondant à un clavier valide.*

$2B Bufferisation 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 tics = 1/ 60 ème de seconde
$01 40 tics
$02 30 tics
$03 20 tics
$04 10 tics

Durée entre deux clics souris au bout de laquelle on consi - dère 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 simultanéement.*

—> 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 systeme 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 Miscellanous Tool Set.

Quelques Macros.

```
        Tool MAC

LDX #]1         ; charge le numéro  d'appel du tool
JSL $E10000     ; Appel au dispatcher
<<<

WriteBParam MAC
Tool $B03
<<<

_ReadBParam MAC
Tool $C03
<<<

                ; Lecture d'un paramètre en BRAM
                ; 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 : Lect : lecture
Ecr : écriture
L/E : lect Ecr

Description

$C000 Lect Données du clavier bit 7 a 1 si une touche à été pressée :

```
Ex : loop LDA $C000 ; attend la
                    ; pression
                    ; d'une
          BPL loop  ; touche
                    ; résultat
                    ; dans A
```

$C010 * Ecr CLR80COL utilise la mémoire principale

$C001 * Ecr SET80COL permet l'utilisation de la mémoire auxiliaire, on sélectione la mémoire auxiliare ou la mémoire principale en écrivant en TXTPAGE1 ou TXTPAGE2 (mémoire auxiliare) 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 * Ecr RDMAINRAM lecture à partir de la mémoire principale

$C003 * Ecr RDCARDRAM lecture à partir de la mémoire auxiliare

$C004 * Ecr WRMAINRAM écriture en mémoire principale.

$C005 * Ecr WRMAINRAM écriture en mémoire auxiliaire.

$C006 Ecr SETSLOTCxROM valide la rom se trouvant sur les cartes d'interfaces. ($C100-$CFFF espace mémoire des entrées sorties) Fonctionne conjointe ment a RDCxROM

$C007 Ecr SETINTCxROM valide la rom interne $C100-$CFFF

$C008 * Ecr SETSTDZP Page Zéro et Pile en mémoire principale

$C009 * Ecr SETALTZP Page Zéro et Pile en mémoire auxiliaire

$C00A Ecr SETINTC3ROM valide la rom interne à $C300. (rom de la carte 80 colonnes)

$C00B Ecr SETSLOTC3ROM valide la rom de la carte 80 colonnes.

$C00C Ecr 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 colon nes consiste à demander l'affichage d'un CTRL-Q).

$C00D Ecr SET80VID active l'affichage 80 colonnes. (même remarque)

$C00E Ecr CLRALTCHAR lettres minuscules normales Lettres majuscules clignotantes

$C00F Ecr SETALTCHAR Jeu de caractéres Normaux et inversés pas de lettres clignotantes.

$C00F Lect idem $C010

$C010 L/E KBDSTRB Remet à zéro le flag Touche enfoncée
le contenu de $C000 devient < 128
Ex : LDA $C000 A = $C1 code ascii de A

```
BIT $C010
LDA $C000      A =  $41
```


$C011 * Lect RDLCBNK2 le bit 7 de cette adresse indique le banc mémoire actif de la carte language

1 $D000 bank 2
0 $D000 bank 1

```
Ex :  LDA $C011
      BPL bank1
                  ; Le banc 1 est actif
                  ; dans la carte
                  ; language
bank1
                  ; le banc 2 est actif
                  ; dans la carte
                  ; language
```

$C012 * Lect RDLCRAM le bit 7 de cette adresse indique si on lit sur la carte language ou sur la ROM
1 Carte language
0 Rom.

$C013 * Lect RDRAMRD le bit 7 indique si on lit dans les 48kde la mémoire auxiliaire.
1 Mémoire auxiliaire
0 Mémoire principale

$C014 * Lect RDRAMWRT le bit 7 indique si on écrit dans les 48k de la mémoire auxiliaire.
1 Mémoire auxiliaire
0 Mémoire principale

$C015 Lect RDCxROM le bit 7 indique l'état du commutateur SLOTCXROM 1 la mémoire des cartes est utilisée 0 la mémoire interne est utilisée. Cet indicateur est utilisé conjointement à SETSLOTCxROM et SETINTCxROM

```
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
```

$C016 *	Lect RDALTZP Le bit 7 est à 1 si la page zéro et la pile de la mémoire auxiliaire sont activée. (Commutateur SETALTZP)
$C017	Lect RDC3ROM le bit 7 indique l'état du commutateur SLOTC3ROM 1 la mémoire des cartes est utilisée 0 la mémoire interne est utilisée. Cette indicateur est utilisé conjointement à SETSLOTC3ROM et SETINTC3ROM
$C018	Lect RD80COL le bit 7 est à 1 si utilisation de la mémoire auxiliaire pour l'affichage. (Commutateur SET80COL/CLR80COL)
$C019	Lect RDVBLBAR le bit 7 est à 1 si pas en VBL
$C01A	Lect RDTEXT le bit 7 est à 1 si on se trouve en mode texte. (Commutateurs TXTCLR/TXTSET)
$C01B	Lect RDMIX le bit 7 est à 1 si on se trouve en mode mixte texte et graphique. (Commutateurs MIXCLR/MIXSET)
$C01C	Lect RDPAGE2 le bit 7 est à 1 si on se trouve sur la page 2. (Commutateurs TXTPAGE1/TXTPAGE2)
$C01D	Lect RDHIRES le bit 7 est à 1 si on se trouve en mode haute résolution. (Commutateurs HIRES)
$C01E	Lect ALTCHARSET le bit 7 est à 1 si on affiche dans le jeu de caractères alternatifs, (Commutateurs CLRALTCHAR/SETALTCHAR)
$C01F	Lect RD80VID le bit 7 est a 1 si l'affichage est en 80 colonnes. (Commutateurs CLR80VID/SET80VID)
$C020	Réservé

$C021 L/E MONOCOLOR :

Bit 0-6 : Réservé
Bit 7 : Si bit à 1 on affiche des nuances de gris sinon l'affichage est en couleurs.

$C022 L/E TBCOLOR :
Registre de couleur du texte.

Bit 0-3 : couleur du fond
Bit 4-7 : couleur du texte

$0 Noir	$8 Marron
$1 Rouge sombre	$9 Orange
$2 Bleu foncé	$A Gris clair
$3 Pourpre	$B Rose
$4 Vert foncé	$C Vert
$5 Gris foncé	$D Jaune
$6 Bleu	$E Aquamarine
$7 Bleu clair	$F Blanc

$C023 L/E VGCINT registre d'interruption du contrôleur vidéo

Bit 0 : Réservé
Bit 1 : Autorise interruption de fin de ligne
Bit 2 : Autorise les interruptions de 1 seconde
Bit 3 : Réservé
Bit 4 : Réservé
Bit 5 : Status de l'interruption de fin de ligne
Bit 6 : Status de l'interruption de 1 seconde
Bit 7 : Status de l'interruption VGC

Bits de status à 1 si une interruption de ce type à eu lieu.
Bit à 1 pour autoriser une interruption.

$C024 Lect 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
Bit 7 : Etat du bouton 0 si bouton enfoncé
Bit 6 : Variation de mouvement si 1 négatif
Bit 5-0 : Mouvement de la souris

$C025 Lect KEYMODREG:
registre de status des touches mortes du clavier. (Shift Controle Pomme CapsLock)

Bit 7 : à 1 si la touche pomme a été enfoncée.
Bit 6 : à 1 si la touche Option a été enfoncée.
Bit 5 : à 1 si ce registre a été modifié.
Bit 4 : à 1 si une touche du pavé numérique a été enfoncée.
Bit 3 : à 1 si une touche est enfoncée.
Bit 2 : à 1 si la touche Caps Lock a été enfoncée.
Bit 1 : à 1 si la touche Contrôle a été enfoncée.
Bit à 1 si la touche Shift a été enfoncée.

$C026 L/E DATAREG :
registre de donnée/commande du clavier
Lors d'une interruption ce registre est défini de la façon suivante.

Bit 7 : à 1 si un octet de réponse, sinon octet de donnée.
Bit 6 : à 1 ABORT valide tous les autres bits de ce registre à 0.
Bit 5 : à 1 si on à appuyé sur CTRL-POMME-RESET
Bit 4 : à 1 si on à appuyé sur une séquence de vidange clavier.
CTRL-POMME-DEL
Bit 3 : à 1 si SRQ valide
Bit 2-0 : nombre d'octets de données reçu -1 si 0 pas de données valide.

$C027 L/E KMSTATUS registre de status de l'ADB

Bit 7 : à 1 si le registre de donnée de la souris est plein.
Bit 6 : à 1 si les interruptions de la souris sont autorisées
Bit 5 : à 1 si le registre DATAREG contient des données valides.

Bit 4 : à 1 si une interruption est générée lorsque le contenu du registre DATAREG est valide.
Bit 3 : à 1 si le registre de données du clavier est plein.
Bit 2 : à 1 si une interruption est générée lorsque le contenu du registre de données du clavier est valide
Bit 1: à 1 si coordonnée X de a souris dans MOUSEDATA à 0 si coordonnée Y
Bit 0 : à 1 si le registre de DATAREG est plein

$C028 xxxx ROMBANK :
bascule de sélection de banc de ROM (inutilisé sur le GS)

$C029 L/E NEWVIDEO :
Ce registre contrôle les capacités vidéo supplémentaire de l'Apple IIGs.

Bit 0-4 : Réservé ne pas modifier
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)
Bit 6 : Si ce bit est à 0 la carte mémoire des 128k est la même que sur l'Apple IIe (nécessaire pour utilisé la Double Haute Résolution) Si ce bit est à zéro. Le buffer vidéo devient une seule et même zone mémoire contigue linéaire de $2000 à $9D00 dans le bank $E1
Bit 7 : Si ce bit est à 0 tous les modes graphiques Apple II sont validé. 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.

$C02A xxxx Réservé futures extensions.
$C02B L/E LANGSEL :
Registre contrôlant le générateur de caractéres.
Bit7-5 : Sélection de la langue du générateur de caractéres
0 : anglais (USA)
1 : anglais (Uk)

2 : Français
3 : Danois
4 : Espagnol
5 : Italien
6 : Allemand
7 : Suédois

Bit 4 : à 1 si mode PAL sinon sortie NTSC
Bit 3 : à 0 si jeu de caractére primaire sélectionné.
Bit 2-0 : Réservé à 0 .

$C02C L/E CHARROM adresse pour les tests de lecture de la rom caractère.

$C02D L/E SLTROMSEL

Bit 0-4 : Réservé ne pas modifier
Bit 7 : à 1 valide la carte du slot 7. 0 valide la ROM appletalk
Bit 6 : à 1 valide la carte du slot 6. 0 valide la Rom des lecteurs 5.25
Bit 5 : à 1 valide la carte du slot 5. 0 valide la Rom des lecteurs 3.5
Bit 4 : à 1 valide la carte du slot 4.0 valide la Rom de la souris
Bit 3 : Réservé ne pas modifier
Bit 2 : à 1 valide la carte du slot 2.0 valide la Rom de la sortie série 2
Bit 1 : à 1 valide la carte du slot 1. 0 valide la Rom de la sortie série 1
Bit 0 : Réservé ne pas modifier

$C02E Lect VERTCNT :
Adresse pour la lecture des bits du contrôleur vidéo V5-VB

$C02F Lect HORIZCNT :
Adresse pour la lecture des bits contrôleur vidéo VA-H0

$C030 L/E SPKR :
l'accès à ce registre produit un clic du haut parleur.

$C031 L/E DISKREG :
registre d'interface disque
Bit 7 : à 1 sélectionne la tête 1 sur le disque 3.5 à 0 sélectionne la tête 0
Bit 6 : à 1 on utilise un : lecteur 3.5 à 0 un lecteur 5.25
Bit 5-0 : Réservé.

$C032 L/E SCANINT :
registre de remise à zéro des interruptions VGC.

Bit 5 : Remise à zéro des ITsfin de ligne
Bit 6 : Remise à zéro des ITs 1 secondes

Autres bits réservés.

Pour remettre à zéro les ITmettre à zéro le bit.

$C033 L/E CLOCKDATA :
registre de donnée de l'horloge.

$C034 L/E CLOCKCTL :
Permet de selectionner la couleur du bord et de contrôler l'horloge

Bit 0-3 : Couleur du bord

Bit 5 : Une fois le transfert du dernier octet, ou sa lecture, ce bit doit être positionner à 1. Ce bit doit être positionné à 0 avant tout transfert avec l'horloge.

Bit 6 : Positionner ce bit à 1 avant une lecture de l'horloge, à 0 avant une écriture

Bit 7 : Ce bit doit être positionner à 1 avant toute opération avec l'horloge, celle-ci le repositionne à 0 à la fin de l'opération.

$C035L/E SHADOW :
registre shadowing

Bit 0 : pas de shadowing page text 1 si bit à 1
Bit 1 : pas de shadowing Hires page 1 si bit à 1
Bit 2 : pas de shadowing Hires page 2 si bit à 1
Bit 3 : pas de shadowing Buffer superhires si à 1
Bit 4 : pas de shadowing bank auxiliare hires si à 1
Bit 5 : Réservé
Bit 6 : pas de shadowing pour la zone E/S si bit à 1
Bit 7 : Réservé

$C036 L/E CYAREG registre de contrôle de la vitesse

Bit 0 : slot 4 si bit à 1 vitesse lente sur motor-on
Bit 1 : slot 5
Bit 2 : slot 6
Bit 3 : slot 7
Bit 4 : si bit à 1 autorise les shadowing dans tous les bancs $00-$7F, à 0 seulement dans les bancs $00 et $01 (mode normal) Ne pas utiliser le mode tous bancs.
Bit 5 :
Bit 6 : Réservé
Bit 7 : si bit à 1 vitesse rapide

Bit 0-3 si une adresse motor-on ($C0F9, $C0E9 $C0D9, $C0C9) est accédée la vitesse passe automatiquement à 1Mhz (vitesse lente compatible avec les periphériques Apple 2) et sur l'accès à une adresse motor-off ($C0F8, $C0E8, $C0D8, $C0C8) la vitesse revient à sa valeur précédente Utilisé pour permettre l'utilisation des unités de disquettes 5 pouces ne fonctionnant qu' à 1Mhz

$C037 L/E DMAREG :
Utilisé lors des accès DMA comme adresse de banc.

$C038 L/E SCCBREG :
Registre de commande du port série numéro 2

$C039 L/E SCCAREG :
Registre de commande du port série numéro 1

$C03A L/E SCCBDATA :
Registre de donnée du port série numéro 2

$C03B L/E SCCADATA :
Registre de donnée du port série numéro 1

$C03C L/E 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 : Réservé ne pas modifier
Bit3-0 : Volume sonore de 0 à $F, $F le plus fort.

$C03D L/E 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 L/E SOUNDADRL :
Poids faible de l'adresse dans la ram du DOC

$C03F L/E SOUNDADRH :
Poids fort de cette même adresse.

$C040 xxxx : Réservé aux extensions futures.

$C041 L/E INTEN

Bit 7-5 : Réservés.
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
Bit0 : à 1 valide la souris en mode Apple II (mega 2)

$C042 xxxx : Réservé aux extensions futures.

$C043 xxxx : Réservé aux extensions futures.

$C044 Lect MMDELTAX variation du mouvement de la souris mode Apple 2 axe des X. (complément à 2)

$C045 Lect MMDELTAY variation du mouvement de la souris mode Apple 2 axe des Y. (complément à 2)

$C046 Ecr DIAGTYPE

Bit 7 : 1 si diagnostics ROM

$C047Lect 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 à 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

$C047Ecr CLRVBLINT : réinitialise l'interruption VBL

$C048Ecr CLRXYINT :
réinitialise les interruptions de la souris mode Apple 2.

$C049 xxxx : Réservé aux extensions futures.

$C04A xxxx : Réservé aux extensions futures.

$C04B xxxx : Réservé aux extensions futures.

$C04C xxxx : Réservé aux extensions futures.

$C04D xxxx : Réservé aux extensions futures.

$C04E xxxx : Réservé aux extensions futures.

$C04F xxxx : Réservé aux extensions futures.

$C050 L/E TXTCLR :
Sélectionne le mode graphique «standard» Apple II

$C051 L/E TXTSET :
Sélectionne le mode texte.

$C052 L/E MIXCLR :
Positionne en mode plein graphique.

$C053 L/E MIXSET :
Positionne en mode mixte graphisme plus 4 lignes de texte.

$C054 L/E TXTPAGE1 :
Sélectionne la page 1 texte ou graphique

$C055 L/E TXTPAGE2 :
Sélectionne la page 2 ou si on a utilisé SET80COL (mémoire auxiliaire pour l'affichage) la page 1 en mémoire auxiliaire.

$C056 L/E LORES :
Sélectionne le mode graphique basse résolution.

$C057 L/E HIRES :
Sélectionne le mode haute résolution ou si on a utilisé SETAN3 la Double Haute résolution

```
Ex:  STA SETAN3   ; passe
                  ; en double haute
     STA HIRES
```

$C058 Ecr SETAN0 : positionne l'annunciator 0

$C059 Ecr CLRAN0 : efface l'annunciator 0

$C05A Ecr SETAN1 : positionne l'annunciator 1

$C05B Ecr CLRAN1 efface l'annunciator 1

$C05C Ecr SETAN2 positionne l'annunciator 2

$C05D Ecr CLRAN2 efface l'annunciator 2

$C05E L/E SETAN3 autorise la double haute résolution.

$C05F L/E CLRAN3 inhibe le mode double haute résolution.

$C060 Lect BUTN3 lire le bouton 3 du joystick

$C061 Lect BUTN0 lire le bouton 0 du joystick

$C062 Lect BUTN1 lire le bouton 1 du joystick

$C063 Lect BUTN2 lire le bouton 2 du joystick

$C064 Lect PADDL0 Lecture paddle 0

$C065 Lect PADDL1 Lecture paddle 1

$C066 Lect PADDL2 Lecture paddle 2

$C067 Lect PADDL3 Lecture paddle 3

$C068 L/E STATEREG Registre regroupant 8 commutateurs communément utilisés sur l'Apple II.

Bit 7 : 1 si diagnostics ROM
Bit 0 : INTCXROM $C007,$C006,
Si bit à 1 la rom interne en $Cx00 est sélectionnée.
Bit 1 : ROMBANK
Ce bit doit toujours être à 0.
Ne pas modifier.
Bit 2 : LCBNK2 $C083, $C011
Si bit à 1 Bank 1 de la
carte language sélectionné. Bank 2 sinon.
Bit 3 : RDROM $C080, $C012
Si bit à 1 la ROM de la
carte language est validée sinon il s'agit de la RAM.
Bit 4 : RAMWRT $C009,$C008
Si bit à 1 la RAM uxiliaire est validée en écriture.
Bit 5 : RAMRD $C009,$C008
Si bit à 1 la RAM auxiliaire est validée en lecture.
Bit 6 : PAGE 2 $C054,$C055
Si bit à 1 la page 2 est séléctionnée.
Bit 7 : ALTZP $C009,$C016
Si ce bit est à 1 la page zéro et la pile se trouve en mémoire principale.

$C069 xxxx Réservé aux extensions futures

$C06A xxxx Réservé aux extensions futures

$C06B	xxxx	Réservé aux extensions futures
$C06C	xxxx	Réservé aux extensions futures
$C06D	L/E	TESTREG registre de mode de test
$C06E	Ecr	CLRTM dévalide le mode test.
$C06F	Ecr	ENTM valide le mode test.
$C070	Ecr	PTRIG : réinitialise la lecture des Paddles
$C071	Lect	Code de traitement des interruptions.
$C07F	Lect	RDDHIRES : le bit 7 est à 1 si le mode double haute résolution est actif. (Commutateurs SETAN3/CLRAN3)
$C080 *	Lect	Lire cette adresse pour passer en lecture RAM de la carte langage utiliser $D000 bank 2 et protéger l à RAM en écriture.
$C081 *	L/L	ROMIN : Lire deux fois cette adresse pour lire la ROM (carte langage) valider en écriture la RAM en utilisant $D000 bank 2.
$C082 *	Lect	Lire cette adresse pour valider la ROM en lecture empêcher l'écriture en RAM et utiliser $D000 bank 2
$C083 *	L/L	Lire deux fois cette adresse pour valider la RAM en lecture et en écriture avec pour $D000 bank 2
$C084 *	Lect	Idem $C080
$C085 *	L/L	Idem $C081
$C086 *	Lect	Idem $C082
$C087 *	L/L	Idem $C083
$C088 *	Lect	Lire cette adresse pour passer en lecture RAM de la carte langage utiliser $D000 bank 1 et protéger la RAM en écriture.

$C089 *	L/L	Lire deux fois cette adresse pour lire la ROM (carte language) valider en écriture la RAM en utilisant $D000 bank 1.
$C08A *	Lect	Lire cette adresse pour valider la ROM en lecture, empêcher l'écriture en RAM et utiliser $D000 bank 1
$C08B *	L/L	Lire deux fois cette adresse pour valider la RAM en lecture et en écriture avec pour $D000 bank 1
$C08C *	Lect	Idem $C088
$C08D *	L/L	Idem $C089
$C08E *	Lect	Idem $C08A
$C08F *	L/L	Idem $C08B
$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

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

Rom des cartes d'extensions, la rom en présence dépend de $C02D SLTROMSEL

$C100 $C1FF	Lect	Rom carte slot 1 ou Rom interne
$C200 $C2FF	Lect	Rom carte slot 2 ou Rom interne
$C300 $C3FF	Lect	Rom carte slot 3 ou Rom interne
$C400 $C4FF	Lect	Rom carte slot 4 ou Rom interne
$C500 $C5FF	Lect	Rom carte slot 5 ou Rom interne
$C600 $C6FF	Lect	Rom carte slot 6 ou Rom interne
$C700 $C7FF	Lect	Rom carte slot 7 ou Rom interne
$C800 $CFFF	Lect	Rom carte ou Rom interne
$CFFF	Lect	Dévalide la Rom en $C800

III

DRIVE

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 du GS.

Le GS peut supporter trois types de floppy. Deux, acceptant des disquettes 3 pouces et demi, d'une capacité de 800k, et un, des disquettes 5 pouces un quart, d'une capacité de 140k par face. Les deux versions 3 pouces 1/2 se différencient principalement par la présence ou non d'un processeur à l'intérieur, 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. Quand au lecteur 5 pouces il s'agit du duodisk ou de l'unidisk 5" 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 pouces un quart (l'utilisation des disquettes 3 pouces et demi 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 logiciel.

La structure d'une disquette 5" 1/4 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 kilos octets par piste soit 140 k par disquette. (Je rappelle qu'une face seulement de la disquette est utilisée, en retournant la disquette on gagne 140 K)

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 :

Gap 1 :

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

Secteur 0 :

.....

Secteur 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 AA AD
- 342 octets de données
- Somme de contrôle soit un octet
- Marque de donnée Epilogue soit DE AA 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 d7d6d5d4d3d2d1d1 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 6 2. Trois octets de données utiles donnant quatre octets sur disque.

Les octets de synchronisation sont des octets écrits de telle facon 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 111100. Le contrôleur ne pouvant lire que les octets commencant 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 pouces et demi et 5 pouces un quart, 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 pouces 1/4 soit 512 octets.
On a donc les capacités suivantes en blocs:

Disquette 5 pouces 1/4	:	280 blocs
Disquette 3 pouces 1/2	:	1600 blocs
Carte 1Mo ram	:	2048 blocs
Rom disk	:	jusqu'à 1024 blocs

Le format des disquettes 5 pouces 1/4 est exactement le même que sous DOS seul l'organisation des fichiers change, par contre le format des disquettes 3 pouces 1/2 diffère largement :

2 faces composées chacune de :

80 pistes composées d'un nombre variable de secteurs soit pour la

pistes 0 à 15	:	12 secteurs
pistes 16 à 31	:	11 secteurs
pistes 32 à 47	:	10 secteurs
pistes 48 à 63	:	9 secteurs
pistes 64 à 79	:	8 secteurs

Chaque secteur a la structure suivante :

Marque d'adresse : D5 AA 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 facon 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 language 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 d'unité sous la forme DSSS0000 ou 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.
$04	Nombre de paramètres différent de $03.
$27	Erreur d'E/S ou numéro de bloc invalide.
$28	Pas de périphérique correspondant à cette unité.
$2B	Le disque est protégé en écriture.
$56	Buffer invalide.

Exemple :

```
MLI             EQU     $BF00

                JSR     MLI
                DFB     $80             ;Lecture du bloc
                DW      parm_liste      ;$0000 du drive 1
                BCS     erreur          ;Slot 6

                JSR     MLI
                DFB     $81             ;Ecriture du bloc
                DW      parm_liste      ;$0000 du drive 1

                BCS     erreur          ;Slot 6

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 :

$202C DInfo : Permet d'obtenir des informations à propos d'un Device connecté au système.

$202F DRead : Permet de lire un bloc ou plus.

$202D DStatus : 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.

Paramètres de DInfo

+0 - +1	E	Nombre de paramètres (10)
+2 - +3	E	Numéro de référence du Device
+4 - +7	R	Pointeur sur le buffer devant recevoir le nom du Device.
+8 - +9	R	Caractéristique du Device. Bit 15 : 1 s'il s'agit d'un disque RAM ou ROM Bit 14 : 1 si un «device driver» a été généré. Bit 13 : réservé Bit 12 : 1 si le Device est occupé Bit 11 : réservé Bit 10 : réservé Bit 9-8 : Vitesse à laquelle peut fonctionner le Device 00 1 Mhz 01 2.6 Mhz 10 > 2.6 Mhz 11 Vitesse sans importance. Bit 7 : 1 si le Device utilise des blocs. Bit 6 : 1 l'écriture est autorisée Bit 5 : 1 la lecture est autorisée Bit 4 : réservé Bit 3 : 1 le formatage est autorisé Bit 2 : 1 le support du Device est amovible (ex une disquette pour une unité de disquette) Bit 1 : réservé Bit 0 : réservé
+10-+13	R	Capacité du Device en bloc.
+14-+15	R	Numéro du slot du Device.
+16-+17	R	Numéro d'unité du Device pour le Smartport.
+18-+19	R	Numéro de version du «Device driver»
+20-+21	R	Numéro d'identification du Device Ex : $0000 disquette 5 pouce 1/4 $0003 disquette 3 pouce 1/2
+22-+23	R	Premier numéro de Device dans la liste.
+24-+25	R	Numéro suivant de Device

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

Paramètres de DRead DWrite :

+0 - +1	E	Nombre de paramètres (6)
+2 - +3	E	Numéro de référence du Device.
+4 - +7	R	Pointeur sur le buffer
+8 -+11	E	Nombre d'octets à lire / écrire
+12-+15	E	Bloc de départ
+16-+17	E	Nombre d'octets par bloc
+18-+21	R	Nombre d'octets réellement lu/écrit

Paramètres de DStatus :

+0 - +1	E	Nombre de paramètres (5)
+2 - +3	E	Numéro de référence du Device.
+4 - +5	E	Type de requête
		$0000 status du Device $0001 paramètre de configuration $0002 wait / no wait $0003 options de formatage $0004 status de la partition
+6 -+ 9	R	Pointeur sur le résultat
+10-+13	E	Taille théorique du résultat
+14-+17	E	Taille réelle du résultat

Exemple d'utilisation de DInfo :

Affichage de la liste des Devices en ligne

```
        LDA #1
        STA NumDev
Encore
        _DInfo parm
        BCS             terminé

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

        PushPtr         NomDev+1    ; Début de la chaine

        _DrawString                 ; (fonction Quickdraw)

        JSR             CRLF        ; passe à la ligne suivante

        INC             NumDev      ; tente le numéro suivant
        BRA             Encore
```

```
terminé
        RTS

parm
        DW          10
NumDev
        DW          1
        DDW         nom
        DDW         0
        DW          0
        DW          0
        DW          0
        DW          0
        DW          0
        DW          0

nom     DW          35          ; taille du buffer nom
NomDev
        DS          33
```

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 GS (ou insérés dans un slot). Ce logiciel transforme les appels en un format intelligible par les unités intelligentes (unités pouvant interpreter des flots de commandes circulant sur le Bus du SmartPort, comme par exemple l'UniDisk 3.5). De même le SmartPort fourni une interface pour plusieurs Devices inintelligentes comme l'Apple 3.5 , les Ramcards, 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      Parms
```

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 IIe et IIc) 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 GS, $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 ...). Généralement chaque commande existe sous les deux formes.

Chaque appel au SmartPort à la forme d'un appel au MLI de Prodos, 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 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'éxécution d'une commande le bit carry est positionné et un code erreur est transmis dans l'accumulateur. Lors de l'éxécution 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.

Paramètres :

E Fourni en entrée R Reçu en sortie

E - Nombre de paramètres 1 octet : $03
E - Numéro d'Unité
(Les numéros d'unité sont alloués de façon unique à chaque Device lors de l'initialisation du SmartPort)
R - Pointeur sur un buffer devant contenir le résultat.(deux ou quatre octets suivant mode normal ou étendu)
E - type de status demandé
$00 Status d'un Device
$01 Bloc de contrôle d'un Device
$02 Etat du caractére nouvelle ligne
$03 Bloc d'information d'un Device

En retour on obtient dans le buffer les informations suivantes :

Type de status : $00 4-5 octets d'informations : 1 octet de status général.

bit 7 à 1 : Device de type bloc ,
0 : Device de type caractére
bit 6 à 1 : Ecriture autorisée
bit 5 à 1 : Lecture autorisée
bit 4 à 1 : Device en ligne ou disquette présente.
bit 3 à 1 : Formatage autorisé
bit 2 à 1 : Protection en écriture (Device de type bloc seulement)
bit 1 à 1 : Device en cours d'interruption (Apple IIc seulement)
bit 0 à 1 : Device actuellement ouvert (Device de type caractére)

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 à zéro.
Si le numéro d'unité est de zéro on obtient les informations sur le SmartPort soit 7 octets :

octet 0	Nombre de devices
octet 1	Etat d'interruption (si bit 6 à alors pas d'interruption
octets 2-3	Societé 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 : $01

Le bloc de contrôle est spécifique à chaque Device, la seule chose commune étant la présence d'un octet de longueur comme premier caractère du buffer. (Note: 0 comme valeur pour cet octet indique un buffer résultat de 256 octets et non 0. L'octet contenant la longueur du buffer résultat n'est pas compté dans celle-ci).

Type de status : $02

Etant donné qu'aucun Device de type caractère n'est encore implémenté, ce type de status est indéfini.

Type de status : $03

Cet appel retourne dans le buffer le bloc d'information d'un Device .

- 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 chaine d'identification 1 octet.
- Chaine 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 : Média amovible.
  - Bit 4 à 0 réservés.
- Mot de version : 2 octets

Exemple de couples type, sous type définis :

Type	Sous type	Device
$00	$00	Carte d'extension mémoire Apple
$00	$C0	Carte d'extension mémoire Apple GS configurée en disque RAM.
$01	$00	Unidisk 3.5
$01	$C0	Apple 3.5
$03	$E0	SCSI avec Média non amovible.

3.2.3.2 *ReadBlock : $01/$41*
WriteBlock : $02/$42

Lit/Ecrit 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.

Paramètres :

E - Nombre de paramètres 1 octet égal à $03
E - Numéro d'unité 1 octet
ER- Pointeur sur le buffer 2 à 4 octets
E - Numéro du bloc 3 à 4 octets.

Codes d' erreur en retour :

$06	BUSERR	Erreur de communication sur le SmartPort
$27	IOERROR	Erreur d'entrée sortie
$28	NODRIVE	Pas de Device connecté
$2B	NOWRITE	Disque protégé contre l'écriture
$2D	BADBLOCK	Numéro de bloc invalide.
$2F	OFFLINE	Device 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 Device pour permettre l'écriture et la lecture.
Paramètres :

E - Nombre de paramètres : 1 octet valant $01
E - Numéro d'unité 1 octet.

Codes d' erreur en retour :

$06	BUSERR	Erreur de communication sur le SmartPort
$27	IOERROR	Erreur d'entrée sortie
$28	NODRIVE	Pas de Device connectée
$2B	NOWRITE	Disque protégé contre l'écriture
$2F	OFFLINE	Device offline ou pas de disquette dans l'unité.

3.2.3.4 *Control : $04 / $44*

Envoi des informations de contrôle à un Device (demande d'éjection de disque ...) la demande peut être de type général ou spécifique à un type de Device.

Paramètres :

E - Nombre de paramètres $03
E - Numéro d'unité 1 octet

E - Pointeur sur la liste de contrôle 2 à 4 octets
E - Type de contrôle demandé.

Note : les deux premiers octets de la liste de contrôle spécifient sa taille.

Le type de contrôle dépend du Device adressé.
Les types suivants étant prédéfinis :

$00	Reset logiciel du Device.
$01	Positionnement du bloc de contrôle.
$02	Sélection du caractère nouvelle ligne.
$03	Contrôle des interruptions.

Codes d'erreur en retour :

$06	BUSERR	Erreur de comm. sur le SmartPort
$21	BADCTL	Type de contrôle invalide
$22	BADCTLPARM	Liste de paramètres invalides.
$30 > $3F	UNDEFINED	Erreurs spécifiques au Device.

3.2.3.5 *Init : $05 / $45*

Permet à l'application de réinitialiser le SmartPort.

Paramètres :

E - Nombre de paramètres $01
E - Numéro d'unité $00

Codes d'erreur en retour :

$06	BUSERR	Erreur de communication sur le SmartPort
$28	NODRIVE	Pas de Device 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 :

E - Nombre de paramétres $01
E - Numéro d'unité 1 octet

Codes d'erreur en retour :

$01	BADCMD	Commande invalide
$06	BUSERR	Erreur de communication sur le SmartPort
$28	NODRIVE	Pas de Device connecté

3.2.3.7 *Close : $07 / $47*

Cet appel indique à un Device de type caractère qu'une séquence d'Ecriture/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 :

E - Nombre de paramètres $01
E - Numéro d'unité 1 octet

Codes d'erreur en retour :

$01	BADCMD	Commande invalide
$06	BUSERR	Erreur de communication sur le SmartPort
$28	NODRIVE	Pas de Device connecté

3.2.3.8 *Read : $08 / $48*

Permet de lire un nombre d'octets donné.
L'adresse référence une localisation sur un device de type caractère. Dans le cas d'un Device 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 :

E - Nombre de paramètres $04
E - Numéro d'unité : 1 octet différent de 0
E - Pointeur du buffer 2 à 4 octets

E - Nombre d'octets à transférer 2 octets
(poids faible poids fort)
E - Adresse dans le device 3 à 4 octets

Codes d'erreur en retour :

$06	BUSERR	Erreur de communication sur le SmartPort
$27	IOERROR	Erreur d'entrée sortie
$28	NODRIVE	Pas de Device connecté
$2B	NOWRITE	Disque protégé contre l'écriture
2D	BADBLOCK	Numéro de bloc invalide.
$2F	OFFLINE	Device 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 device de type caractère. Dans le cas d'un Device 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 :

E - Nombre de paramètres $04
E - Numéro d'unité : 1 octet différent de 0
ER - Pointeur du buffer 2 à 4 octets
E - Nombre d'octets à transférer 2 octets
(poids faible poids fort)
E - Adresse dans le device 3 à 4 octets

Codes d'erreur en retour :

$06	BUSERR	Erreur de communication sur le SmartPort
$27	IOERROR	Erreur d'entrée sortie
$28	NODRIVE	Pas de Device connecté
$2B	NOWRITE	Disque protégé contre l'écriture
$2D	BADBLOCK	Numéro de bloc invalide.
$2F	OFFLINE	Device offline ou pas de disquette dans l'unité.

3.2.3.10 *Types de contrôle de l'Apple 3.5*

Il existe sept types de contrôle specifique 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.

3.2.3.10/1 *Eject : $04*

Ejecte la disquette contenue dans le drive.

E - Taille de la liste 2 octets : $0000

Exemple :

En supposant que l'unité 1 est un Apple 3.5.

```
        JSR     SmartPort
        DFB     $04             ;Control
        DW      parm
        BCS     Erreur

parm    DFB     $03
        DFB     $01
        DW      parm_ctrl
        DFB     $04             ;Code d'éjection

parm_ctl
        DW      $0000           ;taille de la liste
```

3.2.3.10/2 *SetHook : $05*

Permet de rediriger les routines internes de l'Apple 3.5 vers ses propres routines. Ceci permet notamment de lire ou d'écrire des disquettes de format non-standard.
Paramètres :

E - Taille de la liste $0004
E - Numéro du Vecteur.

$01	Lecture du champ adresse
$02	Lecture du champ donnée
$03	Ecriture du champ donnée
$04	Positionnement tête/ piste

$05 Formatage
$06 Ecriture d'un piste
$07 Vérification d'une piste

E - Nouvelle adresse du vecteur
3 octets : Poids faible
Poids fort
Numéro de banc

3.2.3.10/3 *ResetHook : $06*

Restore la valeur par défaut d'un vecteur.

Paramètres :

E - Taille de la liste $0001
E - Numéro de référence du vecteur

3.2.3.10/4 *SetMark : $07*

Change des octets dans la table des marques (marques de champ adresse, donnée ...)

Paramètres :

E - Taille de la liste 1 octet
E - Début des modification dans la table 1 octet
E - Octets modifiés

Table :

0 : Numéro de secteur $FF
1 à 4 : Marque de champ donnée $AD $AA $D5 $FF
5 à 9 : Octets de synchro $FC $F3 $CF $3F $FF
10 à 12 : Fin de donnée/adresse $FF $AA $DE
13 à 17 : gap $FF $FF $FF $FF $FF
18 à 21 : Marque de champ adresse $96 $AA $D5 $FF

Exemple : on change la marque de champ adresse en $D5 $AA $97

```
JSR     SmartPort
DFB     $04             ;Control
DW      parm
BCS     Erreur
```

```
parm      DFB     $03
          DFB     $01
          DW      parm_ctrl
          DFB     $07             ;Code d'éjection

parm_ctl
          DW      $0004           ;taille de la liste
          DFB     18
          DFB     $97,$AA,$D5
```

3.2.3.1/.5 ResetMark : $08

Permet de restaurer la valeur par défaut des paramètres de la table.

Paramètres :

E - Taille de la liste 1 octet égal au nombre d'octets à restaurer dans la table plus 1.
E - Octet de départ dans la table 1 octet

Exemple : on restore la marque de champ adresse

```
          JSR     SmartPort
          DFB     $04             ;Control
          DW      parm
          BCS     Erreur

parm      DFB     $03
          DFB     $01
          DW      parm_ctrl
          DFB     $08             ;Code d'éjection

parm_ctl
          DW      $0004           ;3 octets à restaurer
          DFB     18
```


3.2.3.10/6 *SetSides : $09*

Défini le nombre de faces de la disquette.

Paramètres :

E - taille de la liste $0001
E - Nombre de faces 1 octet
$80 2 faces
$00 1 face

3.2.3.10/7 *SetInterleave*

Defini l'entrelacement des secteurs, pour la prochaine opération de formatage.

Paramètres :

E- Taille de la liste $0001
E - Entrelacement $01 à $0C un octet

Avec un entrelacement de 1 on a la disposition suivante des secteurs sur le disque :

$0 $1 $2 $3 ..... $C

avec un entrelacement de 2 on saute un secteur sur 2

$0 $6 $1 $7 $2 $8 $3 $9 $4 $A $5 $B $6 $C

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 1/4 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.
(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 Gs/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 $BF2F. (par exemple pour appeler le device handler du slot 6 drive 1 on fait un JSR ($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 D numéro de Drive 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	Doit être $00.
$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 erreur :

$00	Pas d'erreur.
$27	Erreur d'Entrée sortie
$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	Doit être $01.
$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 erreur :

$00	Pas d'erreur.
$27	Erreur d'Entrée sortie
$28	Pas de Device 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	Doit être $02.
$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 erreur :

$00	Pas d'erreur.
$27	Erreur d'Entrée sortie
$28	Pas de Device 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	Doit être $03.
$43	Unité concernée.
$44 - $45	Inutilisé.
$46 - $47	Inutilisé.

En retour le bit carry est positioné en cas d'erreur et l'accumulateur contient le code erreur :

$00 Pas d'erreur.
$27 Erreur d'Entrée sortie
$28 Pas de Device 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            ;READ
STA     $42

LDA     #%0110000
STA     $43             ;Unité
LDA     #<buffer        ;adresse du buffer
STA     $44
LDA     #>buffer
STA     $45

LDA     #01             ;numéro du bloc
STA     $46             ;poids faible
LDA     #00


JSR     call

BCS     erreur

LDA     #$02            ;WRITE
STA     $42
LDA     #%0110000
STA     $43             ;Unité

LDA     #<buffer
STA     $44
LDA     #>buffer
STA     $45
LDA     #00
```



```
            STA     $46
            LDA     #00
            STA     $47

            JSR     call

            BCS     erreur

call        JMP     ($BF1C)         ;Adresse dans la page
                                    ;globale

buffer
            DS      512
```

3.5 L'Accès direct aux disquettes

Via les commutateurs logiciels situés en $C000/$C0FF 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 capacité 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 Drive 1	$C0EA
Sélection du Drive 2	$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 entiere, il faut accéder à deux phases. L'accès à une seule phase ne déplacant 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     Move 11

            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 Usec 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'intervales de 100 Usec utilisés
; lors du déplacement du bras.

TblTmp1  DC     H'01302824201E1D1C1C1C1C1C'
TblTmp2  DC     H'702C26221F1E1D1C1C1C1C1C'
```

3.5.1.2 *Allumage extinction du moteur*

$C0E9 Allumage du moteur
$C0E8 Extinction du moteur

Un délai suffisant doit être laissé avant toute autre opération pour permettre au moteur d'atteindre la bonne vitesse (300tpm). Pour ceci on peut employer une routine de délai fixe ou une routine surveillant le registre de donnée.

Exemple :

Extinction du moteur.

```
            LDX     #$60
            LDA     $C088,X
            LDA     $C08E,X

L25         LDY     #$08
            LDA     $C08C,X     ; à l'arrêt le contenu
                                ; du registre de donnée
                                ; est constant
L2A         CMP     $C08C,X
            BNE     L25
            DEY
            BNE     L2A
```

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 accedé 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 protg
```

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,X     ; 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 $C0ED le registre est décaler ensuite automatiquement d'un bit vers la gauche ($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 :

```
                                        NB cycle de l'instruction
                                                Calcul total
                                                cycle entre le
                                                $D5 et le $AA

        LDA     #$D5        2
        JSR     write       6
        LDA     #$AA        2       10
        JSR     write       6       12
        LDA     #$AD        2
        JSR     write       6
write
        CLC                 2       18
        PHA                 3       20
        PLA                 4       23
        STA     $C08D,X     5       27
        ORA     $C08C,X     4       32      ; Début de l'écriture
                                            ; du $D5
        RTS                 6       4
```

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 suplémentaires générant 2 bits à 0 permettant cette synchronisation.

Soit une chaine 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   $FF
                         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 ou 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 chaine de zéro avant toute information valide. La raison est toute simple, la présence de plusieurs bits à zéro consécutifs sur le disque est trés mal supporté par l'automate qui relit une valeur aléatoire.

3.5.2 Le lecteur 3 "1/2

Dans cette partie on va parler uniquement du lecteur non-intelligent l'Apple 3.5. En effet celui ci est contrôlable 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 1/4. On doit donc le sélectionner avant toute chose car le 5 1/4 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
```

• Etre 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 mo
                        ; teur
ORA     #$80
STA     $C036
```

• Ecrire un $40 en $C031 pour selectionner 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 à 1 sélectionne la tête 1
à 0 sélectionne la tête 0

Bit 6 à 1 sélectionne les drives 3 1/2
à 0 sélectionne les drives 5 1/4

Le lecteur 3 1/2 utilise aussi les mêmes commutateurs logiciels que le lecteur 5 1/4. Ces commutateurs situés de $C0E0 à $C0EF portent les mêmes noms que lors de leur emploi pour le 5 1/4 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	Opération
Off	Off	On	lecture registre de donnée
Off	On	xx	lecture registre de status
On	Off	xx	lecture registre 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 à $C0EF) écrira dans ce registre, la lecture d'une adresse paire de l'IWM ($C0E0 à $C0EE) lira ce registre.

Registre de Mode :

Attention le lecteur doit être arrêté et non seulement éteint.

Bit 7 Réservé ne pas modifier
Bit 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 1/4.
Bit 2 si à 1, le lecteur après sa désélection, restera allumé une seconde.
Bit 1 à 1 protocole de Handshake synchrone
(lecteur 5 1/4)
à 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 1/4) résultat accès registre pour le 3 1/2.
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 commmutateurs 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. A la fin des opérations de lectures de registres, repassez en mode normal en accédant à Q6 off.

Phase2	Phase1	Phase0	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
Off	On	Off 1	TK0	tête sur piste 0
Off	On	On 1	TACH	tachomètre
On	Off	Off 0	RDDATA0	flot données tête 0
On	Off	Off 1	RDDATA1	flot données tete 1
On	On	Off 0	SIDES	lecteur simple ou double face
On	On	On 1	DRVIN	lecteur installé

Ecriture dans ces registres :

Vérifiez d'abord que Phase3 est off, ensuite passez On Phase0 et Phase2, puis positionnez SEL à 0. Mettez ensuite Phase1 et Phase0 dans l'état necéssaire pour accéder aux registre, et positionnez Phase2 suivant la valeur à écrire (On pour 1, Off pour 0).
Maintenez Phase3 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 changer ni Phase1-2 ou SEL pendant que Phase3 est On, et que Phase0 et Phase1 sont On avant de modifier SEL.

NOTE : Comme dans le cas d'une lecture le lecteur doit être d'abord sélectionné.

Phase1	Phase0	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 peut prés 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'éteind. 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 Phase3 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 lecteur :

Ejection 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       ;Phase3 Off
            LDA     $C0E1       ;Phase0 On
            LDA     $C0E3       ;Phase1 On
            LDA     #$40
            STA     $C031       ;SEL à 0
;
; Paramètres pour l'éjection
;
            LDA     $C0E1       ;Phase0 On
            LDA     $C0E3       ;Phase1 On
            LDA     $C0E5       ;Phase2 On (valeur 1)
;
            LDA     $C0E7       ;Phase3 On
            JSR     WAIT        ;attente d'une demi sec
            LDA     $C0E6       ;Phase3 Off
```

Déplacement du bras

```
* SEEK
*  Allume le moteur et se déplace vers une piste
* Cyl    : Piste voulue
* CurCYl : piste actuelle
* Drive  : Lecteur concerné
```



```
SEEK    LDX     Drive
        BIT     CurCyl,X
        BPL     no

        JSR     Recall          ;Recalibre
        BCS     erreur
no      JSR     DriveOK

        SEC
        LDX     Drive           ;Calcul le nbr de
        LDA     CurCyl,X        ;pistes à déplacer
        SBC     Cyl
        BEQ     fin

        LDY     #$01            ;Piste plus petite
        BCS     inf
        LDY     #$00            ;No pst plus grand
        EOR     #$FF
        ADC     #$01

inf
        TAX
        TYA
        JSR     EcrREG          ;Sens de déplace-
                                ;ment
        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
```



```
                LDX     #$3C

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

                LDX     #$50
                JSR     seek_n

                LDX     #$50
encore
                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

; Ecriture d'un registre
EcrREG
                JSR     SetReg
Ecr2REG
                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 : Phase2  1 = ON
; Bit 1 : SEL     1 = 1
; Bit 2 : Phase0  1 = ON
; Bit 3 : Phase1  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
Ecriture d'un octet

HEF03           BIT     $C0ED
                LDA     #$FF
                STA     $C0EF

                LDY     #$07
Loop
```



```
                LDA     : DATA_S2,Y     ;Marque de champ
                                        ;donnée

    ;On regarde le registre de Handshake

    L1          BIT     $C0EC           ;On peut écrire ?
                BPL     L1

                STA     $C0ED           ;Ecrit
                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 chaine donne la séquence suivante :

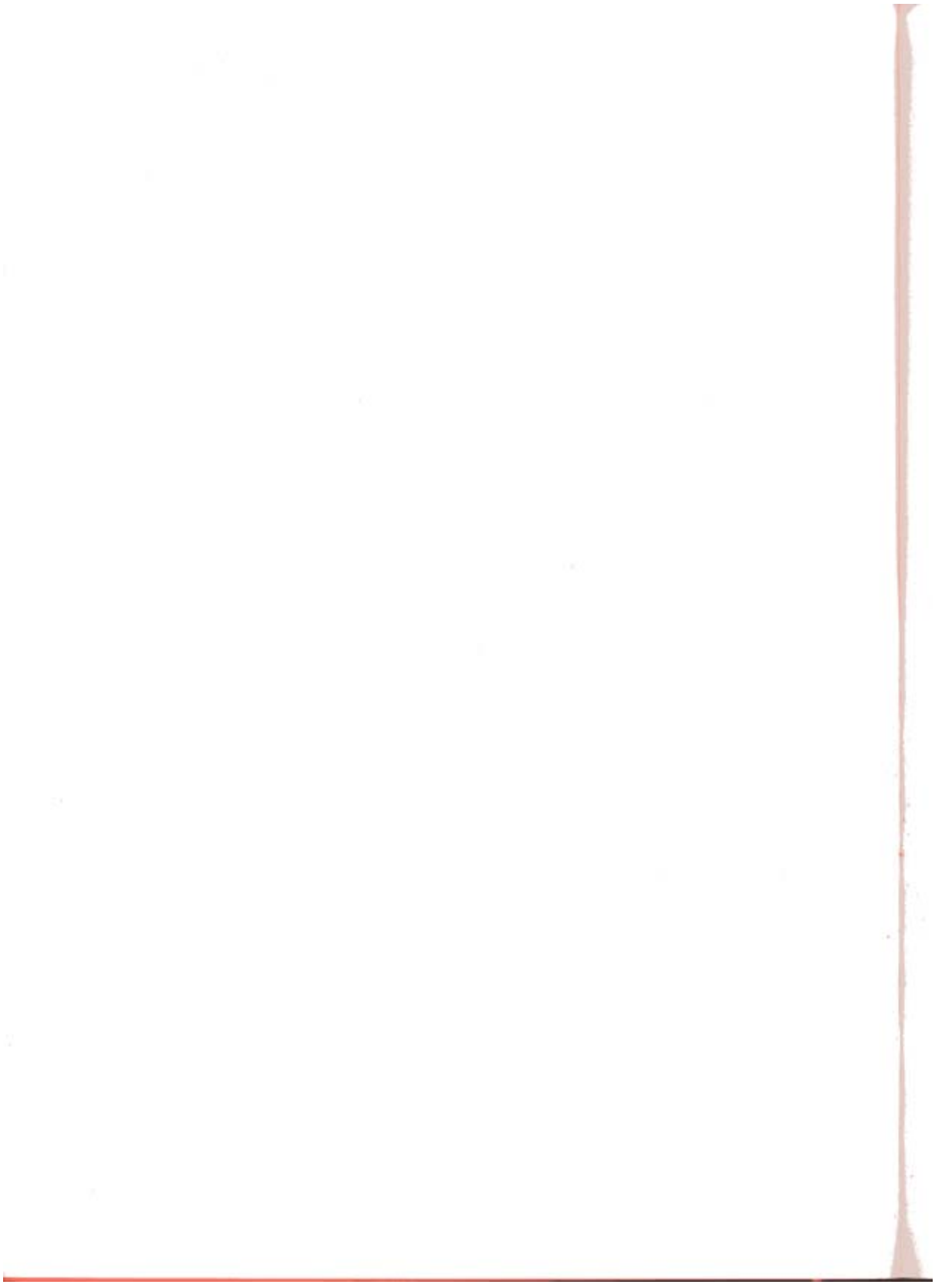
1111 1111-0011 1111-1100 1111-1111 0011-1111 1100

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 possiblité d'écriture d'un octet. Il est également possible d'écrire en se synchronisant en 16 cycles comme pour le lecteur 5 1/4.

lecture d'un octet.

```
                BIT     $C0EC   ;Mode Lecture

    Loop        LDA     $C0EE
                BPL     Loop
```



IV

GRAPHISME

Le G de GS signifie «GRAPHISME». Avec le GS, 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 boite 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 $E19FFF.

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
Bit 4 : Réservé (mis à 0)
Bit 5 : Mode fill (1=on, 0=off)
Bit 6 : Interruption (1=réclamée, 0=non réclamée)
Bit 7 : Nb de pixels horizontaux (0=320, 1=640)

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 $E19DFF 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.
Les choses se présentent donc ainsi :

Ligne	SCB	Points
00	$E19D00	$E12000
01	$E19D01	$E120A0
02	$E19D02	$E12140
03	$E19D03	$E121E0
..	...	...
198	$E19DC6	$E19BC0
199	$E19DC7	$E19C60

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 ^ 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 A, le point 01 de couleur B. 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.

Bits	Couleurs possibles
6-7	0 - 3
4-5	4 - 7
2-3	8 - 11
0-1	12 - 15

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 p.ex), 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 parmis seize couleurs en mode 640 !
La technique qu'ils utilisent 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 à $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*16*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
00	$E19E00
01	$E19E20
02	$E19E40
03	$E19E60
..	...
0E	$E19FC0
0F	$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 :

$00 00 - Noir
$F0 00 - Vert
$0F 00 - Bleu
$00 0F - Rouge
$0F 0F - Violet
$F0 0F - Jaune
$FF 00 - Bleu clair
$FF 0F - Blanc
$88 08 - Gris 8

4.1.3. Interruptions SCANLINE

Le GS, 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 (REMPLISSAGE)

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 grande surface à l'écran, il n'est accessible qu'en mode 320 points par ligne. Lorsqu'il est actif, la couleur 00 disparait, 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 sèquence suivante :

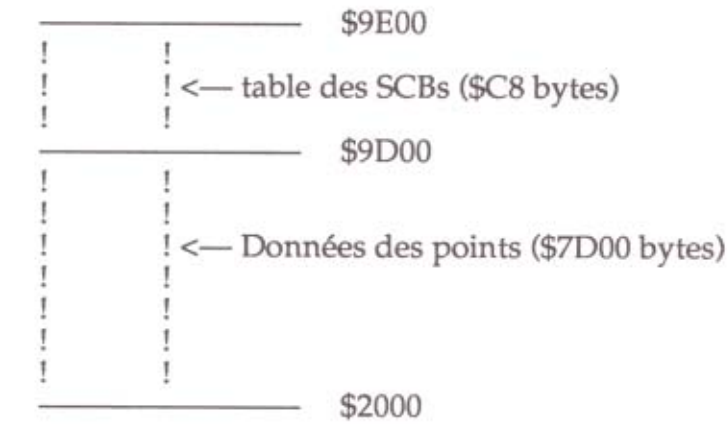
2000000 400000000 60000000000 70000

apparaitra 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 apparaitra 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 donnés graphiques en banc 01, celles-ci étant récopiées 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 copiées hardwarement 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 équivalent à :

```
STZ $C035
LDA #$FF
STAL $019E1F
```

Pour visualiser cette mémoire graphique, il faut manipuler un soft-switch, il s'agit de NEWVIDEO, à l'adresse $C029. 2 bits de ce softswitch sont importants à connaitre :

- 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, $C029 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 #$41    — Linéarise la mémoire
STA $C029      graphique

LDA #$21    — Repasse en mode
STA $C029      TEXT
```


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 TEXT, mais permet l'accès à la mémoire
0/1000<E1/2000.A000M : bouge 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

C029:41 : 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
        SEP #$30
        LDA #$41
        STA $C029          ; linéarise la mémoire pour
        REP #$30           ; pouvoir la lire
        LDX #$7FFE         ; X sert de compteur pour
:1      LDAL $E12000,X     ; copier $7FFE bytes
        STAL $001000,X
        DEX
        DEX
        BPL :1             ; tant que X<>$FFFE
        SEP #$30
        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.

```
    CLC
    XCE
    SEP #$30
    LDA #$41
    STA $C029          ; linéarise la mémoire
    REP #$30           ; pour écrire correctement
    LDX #$7FFE         ; X sert de compteur pour
:1  LDAL $001000,X     ; copier $8000 bytes
    STAL $E12000,X
    DEX
    DEX
    BPL :1             ; tant que X<>$FFFE
    SEP #$30
    RTS
```

Lorsque l'on travaille sur la page graphique, il est toujours agréable de pouvoir connaitre à tout instant l'adresse d'une ligne particulière. La formule en est simple :

Adresse = $E12000 + n ligne * $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
        REP #$30          ; Mode 16 bits
        LDX #$0000        ; X = compteur de position
        LDA #$2000        ; Adresse de départ
:1      STA TABLE,X
        INX               ; X = X + 2
        INX
        CLC
        ADC #$A0          ; Adresse = Adresse + $A0
        CMP #$9D00        ; (distance entre deux lignes)
        BNE :1            ;
        SEP #$30
        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 accèder à une ligne quelconque, il suffit de faire :

```
CLC
XCE
REP #$30
LDA Ligne_Nb      ; Numéro de la ligne
ASL               ; Multiplié par deux
TAX               ; Dans X
LDA TABLE,X       ; A contient alors
...               ; l'adresse du début de
                  ; la ligne (seize bits)
```

4.1.6 Registre monochrone / couleur

Ce registre MONOCOLOR ($C021), option DISPLAY dans le CONTROL PANEL, détermine si la sortie composite du GS doit être en couleur ou en tons de gris. Le bit 7 (le bit le plus fort) est le seul à 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 possiblité 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 parmis 16 couleurs disponibles (les couleurs sont les mêmes que celles de l'ancien mode GR, la basse résolution). On peut sélec-

tionner 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).

	Bits	Description
$C022	7-4	Couleur du texte
	3-0	Couleur du fond
$C034	7-4	Bits horloge/ ne pas modifier
	3-0	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 :

```
LDA $C034
AND #$F0
ORA #COULEUR
STA $C034
```

La table suivante donne les équivalents de chacune des couleurs en système RGB.

N° Couleur	Couleur	équivalent RGB
00	Noir	00 00
01	Rouge	03 0D
02	Bleu foncé	09 00
03	Violet	2D 0D
04	Vert foncé	72 00
05	Gris foncé	55 05
06	Bleu moyen	2F 02
07	Bleu clair	AF 06
08	Brun	50 08
09	Orange	60 0F
0A	Gris clair	AA 0A
0B	Rose	98 0F
0C	Vert clair	D0 00
0D	Jaune	F0 0F
0E	Aquamarine	F9 04
0F	Blanc	FF 0F

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 précèdement.

```
            ORG $900

COULEUR =   $00

            CLC
            XCE
            SEP #$30
            STZ COULEUR
            LDA #$41          ; Mem. graphique linéaire
            STA $C029
            REP #$30          ; Mode 16 bits
            LDA #$0000
            LDX #$7FFE
:1          STAL $E12000,X    ; Efface tout :
            DEX               ; données, table SCB
            DEX               ; palettes
            BPL :1

            SEP #$30          ; Retour en 8 bits
            LDA #$A1          ; Affiche Super Hires
            STA $C029
LOOP        LDA $C034         ; Couleur du bord
            AND #$F0          ; Epargne les bits 4 à 7
            ORA COULEUR       ; Modifié
            STA $C034         ; Restocké
            LDX COULEUR
            LDA COLTB1,X      ; Tables de valeurs RGB
            STAL $E19E00      ; On modifie la valeur de
            LDA COLTB2,X      ; la couleur 00 de la
            STAL $E19E01      ; palette 00
            BIT $C010
KBD         LDA $C000         ; attend une touche
            BPL KBD
            LDA COULEUR
            INC               ; incrémente la couleur
            AND #$0F          ; qui ne peut pas
            STA COULEUR       ; dépasser #$0F
            BRA LOOP          ; Continue


COLTB1      HEX 00,03,09,2D,72,55,2F,AF
            HEX 50,60,AA,98,D0,F0,F9,FF
```

```
COLTB2  HEX 00,0D,00,0D,00,05,02,06
        HEX 08,0F,0A,0F,00,0F,04,0F
```

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 PAL
ORA #$01      ; (met le bit 4 à 1)
STA $C02B

LDA $C02B     ; Passe en NTSC
AND #$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 50Hz (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 ou le faisceau électronique «remonte» du coin en bas à 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 à environ 2.8 MHZ, ce qui signifie qu'il exécute 2"800"000 cycles/seconde. En NTSC, l'écran est balayé 60 fois par seconde, on peut en déduire que le nombre de cycles disponibles par balayage est de : 2"800"000/60 = 46000 cycles. Si on divise ce chiffre par 262 (nb 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 mèmoire 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 GS. 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 balaient pas (lorsque le faiscaux 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,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
      BNE :1
```

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 connaitre !

(2) Le registre $C019 (RDVBLBAR) (marche sur certains 2c et sur IIgs) Ce registre est à $00 lorsque l'écran (Text, graphique..) est balayé et à $80 lorsque le processeur vidéo rafraichit les bourdures 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 qu'on est
      BMI :1        ; sur l'écran
:2    LDA $C019     ; assure qu'on passe
      BPL :2        ; sur la bordure du bas
```

Une programme de démonstration de cette technique pourrait être :

```
:1    LDA $C019     ; 65C02 synchronisé sur le
      BPL :1        ; début de la bordure du bas
      LDA #$0E      ; Bordure de couleur $0E
      STA $C034

:2    LDA $C019     ; Attend le début de l'écran
      BMI :2
      STZ $C034     ; Bordure noire
      BRA :1        ; On continue
```

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 la ligne en cours de rafraichissement. En fait, ce registre est incrémenté toutes les 2 lignes seulement, il peut prendre les valeurs suivantes :

Mode (HERZ)	Valeurs de $C02E
60	$7D à $FF
50	$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
            XCE
            SEP #$30        ; mode 8 bits

DEBUT       LDX #$00

:1          LDA $C02E       ; Position verticale
            CMP POS,X       ; attend une position
            BNE :1          ; particulière
            LDA $C034
            AND #$F0
            ORA COUL,X
            STA $C034       ; Change la couleur du
            LDA COUL,X      ; bord et celle du fond
            STA $C022       ; couleur des caractères
            INX             ; = 00
            CPX #$04        ; si X=4, on le remet à 00
            BNE :1
            BRA DEBUT

POS         HEX A0,B0,C0,D0 ; Positions des changements de
                            ; couleur
COUL        HEX 02,0F,01,00 ; Couleurs : bleu, blanc, rouge
                            ; et noir pour finir
```

Le registre $C02F (HORIZCNT) 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 souvents utilisés comme générateurs de nombres aléatoires.

Exemple : le programme suivant fournit des nombres relativement aléatoires entre $00 et $FF


```
TEMP    =  $00

           CLC
           XCE
           SEP #$30

           LDA $C02E      ; VERTCNT
           ASL            ; On extrait le nibble
           ASL            ; le plus faible
           ASL
           ASL
           EOR $C02F      ; HORIZCNT, change très
           STA TEMP       ; rapidement
           LDA $C02E
           EOR TEMP
           EOR $C02F
```

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 $E10028)
- Cette routine doit signaler qu'elle s'est chargée de l'interruption en mettant le bit 6 du registre SCANINT ($C032) à 0
- La routine d'interruption peut maintenant rendre le contrôle à 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=N° de la ligne ou
LDAL $E19D00,X     ; l'on désire
ORA  #$40          ; l'interruption
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 $E19CFF,X
     AND  #$BF
     STAL $E19CFF,X
     DEX
     BPL :1
```

3) VGCINT : $C023

Bit 1 - authorise l'interruption SCANLINE s' il est à 1
Bit 2 - authorise 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 en $E10028 (avec les autres vecteurs d'interruption IRQ, interruption une seconde en $E10054 p.ex). Si on désassemble cette zone, on trouve :

```
E10028: JMPL  $FFB5DE  (avec des ROMs 01)
```

En $FFB5DE, on trouve simplement un SEC, RTL.
Pour modifier ce vecteur, il faut changer l'adresse suivant le JMP (n de banc y compris). Par exemple, pour faire pointer le vecteur en $320, banc 00, la séquence suivante fait très bien l'affaire :

```
LDA  #$20
STAL $E10029
LDA  #$03
STAL $E1002A
LDA  #$00
STAL $E1002B
```

5) Pour signaler que l'interruption à été prise en compte, le programme de gestion doit effacer un bit du registre SCANINT.

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 à zéro (interruptions authorisée) 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, à l'interface entre l'écran graphique proprement dit et la bordure droite. Malheureusement, l'INTERRUPT MANAGER prend énormèment 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 IRQ (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 tache 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 «POINT» en interruption, puis extrait les racines carrées des nombres de 1 à 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 multitache, 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
CLILOP  LDAL $E19D00-1,X  ; d'interruption de
        AND #$BF          ; tous les SCB
        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
        STAL $E10028      ; JMPL au début du vecteur
        LDA #$00          ; Banc de la routine
        STAL $E1002B      ; d'interruption
        LDA #INTER        ; Adresse partie basse
        STAL $E10029
        LDA #>INTER       ; Adresse partie haute
        STAL $E1002A
        LDA $C023         ; Mise à un du bit 1
        ORA #$02          ; du registre VGCINT
        STA $C023         ; autorise SCANLINE
        CLI               ; assure le bit INT du registre
        SEC               ; d'état à zéro
        XCE               ; SEC XCE nécessaire avant le
        RTS               ; retour au BASIC


INTER   LDA $C032         ; Mise à zéro du bit 5
        AND #$DF          ; de SCANINT, pour que
        STA $C032         ; SCANLINE continue

        PHB               ; Sauvegarde du banc de donnée
        LDA #$E1          ; Banc mis à $E1
        PHA
        PLB

        LDX OLDPOS        ; Efface le point à
        LDA #$00          ; l'ancienne position
        STA $5DE0,X
        STA $5E80,X       ; Adresses des lignes
        STA $5E81,X       ; ou évolue le motif
        STA $5F20,X       ; du point

        LDX NEWPOS        ; Dessine le nouveau
        LDA #$0F          ; point
```



```
        STA $5DE0,X
        STA $5F20,X
        LDA #$FF        ; Byte de motif
        STA $5E80,X     ; Adresse de la ligne
        LDA #$F0
        STA $5E81,X

        PLB             ; Récupère le banc
        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 switchs impliqués :

Le vecteur se trouve en $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 zèro

Après le traitement de l'interruption, il faut remettre le switch CLRVBLINT ($C047) à zéro pour que l'interruption ait lieu à nouveau.
Le switch INTFLAG ($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
        STAL $E10033
        LDA #INTER      ; Vecteur partie basse
        STAL $E10031
        LDA #>INTER     ; Vecteur partie haute
        STAL $E10032
        LDA $C041       ; Interruption VBL
        ORA #$08        ; authorisée
        STA $C041
        CLI
        RTS             ; Revient au moniteur
                        ; ou au BASIC

INTER   STZ $C047       ; Régénère l'interruption VBL
        INC $C034       ; Incrémente la couleur de
                        ; la bordure
```



```
CLC             ; Retour à l'INTERRUPT
RTL             ; MANAGER
```

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 GS (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 principale calcule pendant plus d'un balayage.

Encore deux dernières remarques à propos des interruptions en général sur GS :

- 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 GS. 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 GS, 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 mis en ligne d'attente. Ceci peut s'avérer très génant 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 switchs. 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 récriture 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 mouvement 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 obligè de recourir à des boucles en assembleur. Avec l'arrivèe 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 recopie 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	nb 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 - Dépile 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 ! . Elle 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)

```
CLC
XCE
REP #$30
LDA $C068
ORA #$30
STA $C068
LDA #$2000
TCD
...
```

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
                       ; ]POS est mise à zèro

        LUP $7F        ; Boucle répétée $7F fois

        LDA ]POS+2     ; Assemble l'instruction LDA
                       ; p. zéro avec l'adresse POS
        STA ]POS       ; idem pour STA avec POS+2

]POS    = ]POS+2       ; POS et 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
        LDY #$1800     ; en banc 01: adresse $011800
        LDA #AUXEND-AUX
```



```
        MVN $000000,$010000

        PHK             ; Restore B, perturbé
        PLB             ; 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 avec
        DEX             ; le shadowing on
        BPL COPY


        SEP #$30
SYNC    LDA $C02E
        CMP #$D8        ; Attend une position du
        BNE SYNC        ; balayage

        REP #$30        ; Mode 16 bits

        JSL $011800     ; 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 assem
                        ; blée 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 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
LOOP        TCD             ; D commence à $8900
                            ; (un multiple de $100)
]POS        = $00

            LUP $7F

            LDA ]POS+2
            STA ]POS

      ]POS = ]POS+2

            —^

            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         LDA #$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 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 auxilliaire
            LDY #$1800
```

```
        LDA #AUXEND-AUX
        MVN $000000,$010000

        PHK
        PLB

        LDX #$7FFE          ; Recopie l'image du
COPY    LDAL $E12000,X      ; banc E1 en banc 01
        STAL $012000,X
        DEX
        DEX
        BPL COPY

CONT    SEP #$30
SYNC    LDA $C02E           ; Attends une valeur du
        CMP #$92            ; balayage
        BNE SYNC

        REP #$30            ; Sauve la ligne la plus
        LDX #$0062          ; haute de la fenêtre
:1      LDAL $01341E,X      ; dans une zône
        STA $9000,X         ; tampon : $9000
        DEX
        DEX
        BPL :1

        JSL $011800         ; Bouge toutes les lignes
                            ; d'un cran vers le haut
        LDA #$0000          ; Remet la page zéro au bon
        TCD                 ; endroit

        SEP #$30

        LDA $C000           ; Si une touche est pressée
        BMI END             ; on arrête tout

        REP #$30

        LDX #$0062          ; Recopie la ligne sauvée
:2      LDA $9000,X         ; tout en bas de la
        STAL $015C1E,X      ; fenêtre
        DEX
        DEX
        BPL :2
        BRA CONT

END     LDA #$21        ; Fin, on repasse en mode
```



```
            STA $C029       ; text
            RTS

                            ; Routine SCROLL, en RAM auxilliare
AUX         TSX             ; Sauve le pointeur de pile

            LDA $C068       ; P. 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

            LUP $32

            PEI ]POS        ; Crée les instructions PEI

]POS        = ]POS-2        ; PEI $62
            —^              ; PEI $60
                            ; PEI $5E ....
            TSC             ; Fin d'une ligne
            CLC
            ADC #$104       ; Ligne destination suivante
            TCS
            TDC
            ADC #$A0        ; ligne source suivante
            CMP #$5D00
            BCC LOOP        ; Terminé ?

            LDA $C068       ; Remet la page zéro et la
            AND #$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 Adress, 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 correspondants à des points de couleur de l'objet. Lorsque l'on effectue le ET logique (AND) entre le décor et le masque, une silhouette noir 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 claviers 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
OLDPOS = $02

            CLC
            XCE
            SEP #$30
            LDA #$A1        ; Super hires affichée
            STA $C029
            STZ $C035       ; Shadowing actif
            LDA #$01        ; Le banc «données» est le
            PHA             ; banc 01, STA $1000 signifie
            PLB             ; maintenant STAL $011000

            REP #$30
            STZ NEWPOS  ; Position du sprite de 00 à $7A7C
            LDX #$7FFE
COPY        LDAL $E12000,X      ; Copie la mémoire
            STAL $012000,X      ; graphique du banc
            DEX                 ; $E1 en banc $01
            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        ; la redessine à sa nouvelle
            SEP #$30        ; position
KBD         LDA $C000       ; Lit le clavier
            BPL KBD
            BIT $C010
            CMP #$95        ; Flèche à droite
            BEQ RIGHT
            CMP #$88        ; Flèche à gauche
            BEQ LEFT
            CMP #$8B        ; Flèche en haut
            BEQ UP
            CMP #$8A        ; Flèche en bas
            BNE KBD
```



```
DOWN    REP #$30
        LDA NEWPOS      ; Change la position
        CLC
        ADC #$A0        ; de la croix d'après
        BRA CONT
UP      REP #$30        ; la touche pressée
        LDA NEWPOS
        SEC
        SBC #$A0
        BRA CONT
RIGHT   REP #$30
        LDA NEWPOS
        INC
        BRA CONT
LEFT    REP #$30
        LDA NEWPOS
        DEC
        BRA CONT

CONT    BPL OK1         ; Vérifie le non-dépassement
        LDA #$0000      ; des limites de l'écran
OK1     CMP #$7A7C
        BCC OK2
        LDA #$7A7C
OK2     STA NEWPOS
        BRA LOOP        ; Continue

        MX 11           ; Nécessaire, car la suite est en 8 bits

WAIT    LDA $C019       ; Attend le bas de
        BMI WAIT        ; l'écran graphique
W2      LDA $C019
        BPL W2
        RTS             ; Et revient

        MX 00           ; 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
LDA $A004          ; sauvegarde
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 quatres 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 $A0 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

bizares 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 apparaitre des menus déroulants presque instantanément, ces derniers étant tout d'abord dessinés en banc 01 avec le shadowing inactif.

- La deuxième cas où le shadowing peut nous être utile est celui ou 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 générique, des textes apparaissent sur une image, puis ils sont effacés et d'autres les remplacent, l'image restant intact.

Voyons maintenant les techniques de recopie rapide 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 :

PORGRAMME «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=0000
                        ; pour ne pas perturber les
]POS    = $00           ; bits recopiés
        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
        BEQ END         ; à copier
        BRL LOOP
END     LDA #$30        ; Page zéro à nouveau en
        TRB $C068       ; banc 00
        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 GS.

Les deux programmes qui suivent sont des programmes prototypes, qui surpassent les possibilités «normales» du GS.

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 évènements 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, additioné 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 GS des images provenant d'autres machines avec plus de 16 couleurs (images AMIGA 32 couleurs, VGA 256 couleurs etc..). Cependant, le GS est limité à 16 couleurs par ligne, maximum qui ne pourra vraisemblablement jamais être dépassé, car le processeur vidéo du GS 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 boite 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 pourcents 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 apparaitront donc forcément avec l'évolution du GS, 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 déccélère si il en trouve une (voir le chapitre «TRANSWARP»).

«MULTICOULEUR»

```
            ORG $900

            CLC
            XCE
            SEP #$30
            LDA #$A1
            STA $C029

            REP #$30

            LDAL $BCFF00        ; Cherche une carte
            CMP #'TW'           ; accélératrice
            BNE NOTR            ; TRANSWARP (tm)
            LDAL $BCFF02        ; et la met hors
            CMP #'GS'           ; d'usage le cas
            BNE NOTR            ; échéant

            LDA #$28
            JSL $BCFF24

NOTR        LDA #$0000          ; Efface l'écran
            LDX #$7FFE
CLR         STAL $E12000,X
            DEX
            DEX
            BPL CLR

            SEP #$30
            LDX #$C8
CLILOP      LDAL $E19D00-1,X    ; Met le bit
            AND #$BF            ; d'interruption de
            STAL $E19D00-1,X    ; tous les SCB à 00
            DEX                 ; pour n'avoir qu'une
            BNE CLILOP          ; seule interruption
            SEI
            LDAL $E19D30        ; Interruption à la
            ORA #$40            ; ligne $30
            STAL $E19D30
            LDA #$5C            ; Code pour 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             ; Elle peut avoir lieu.
        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 00
```

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 Prgramme «ENTRELACAGE»

Le nombre de couleurs différentes que peut générer le processeur vidéo du GS 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'intensitè 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 authorisant : 31^3 = 29791 couleurs !
Ce qui multiplie par plus de 7 le nombre de couleurs de base. Cette technique pourrait s'appeler «ENTRELACAGE COULEUR» par analogie avec l'entrelaçage tout court. Un mode graphique entrelacé posséde 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 apparait, 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, légérement 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 ...
..... ..... ..... ..... ...
FFFFFFFFFFFFFFFFFFFF...
FFFFFFFFFFFFFFFFFFFF...
```

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 LACE        ; normal ou entrelacé

CONT    LDX #$21        ; Remet la palette à 00
        LDA #$00        ; sur toutes les lignes
PAL0    STAL $E19D53,X      ; du dessin
        DEX
        BPL PAL0

        LDA LACE        ; Si lace <> 00 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 sur
            BRA W1       ; 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'interrom-

pre 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 N 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 & décompacter une image :

```
10 PRINT CHR$(4);»BLOAD IMAGE,A$2100,T$C0"
20 PRINT CHR$(4);»BRUN DECOMPACTEUR»
```

DECOMPACTEUR

```
          ORG $2000

          LDA #$A1
          STA $C029
          STZ $EB
          STZ $ED
          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 #$00C7
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 (comprenant 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 à 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
        CMP "TW"            ; TRANSWARP : la séquence
        BNE :1              ; "TWG" = $54574753
        LDAL $BCFF02        ; à l'adresse $BCFF00
        CMP "GS"
        BNE :1              ; Si la carte est présente,
        JSL $BCFF20         ; On regarde la vitesse
        STA SAVSPEED        ; et on la sauve. LDA #$28
                            ; On force la vitesse à 2.8
                            ; MHZ, vitesse du processeur
        JSL $BCFF24         ; d'un GS en mode FAST

:1  .
    .   PROGRAMME PRINCIPAL


        REP #$30            ; Fin, passage en 16 bits
        LDA SAVSPEED        ; Si SAVSPEED=00, pas de
        BEQ :2              ; TRANSWARP, on sort
        JSL $BCFF24         ; Sinon, on remet la vitesse
:2      RTS                 ; avant de sortir
```


$BCFF20 - GETSPEED, On récupère la vitesse dans A 16 bits
$BCFF24 - STORESPEED, On force la vitesse contenue dans A

Il existe encore d'autres appels possibles, entre autres pour connaitre les vitesses disponibles... Mais ces 2 suffisent amplement. 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 du GS, 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 de comparer le GS à ses concurrents, d'un point de vue graphique.

Machine	Résolution	Nb couleurs	RGB
GS	320 * 200	16 *1	12 bits
GS	640 * 200	04	
ST	320 * 200	16	9 bits
ST	640 * 200	04 *2	
ST	640 * 400	02	
AMIGA	320 * 200 *3	32	12 bits
AMIGA	640 * 400	16	
AMIGA	320 * 400	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 bit = 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 GS semblent si dérisoires par rapport à des animations AMIGA ou même 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. 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 quadruplée, passant de 8 à 32k ! Tout le monde semble d'accord pour dire que la vitesse du GS 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 tous ces limitations comme un encouragement à aller le plus loin possible avec ce qui nous est donné, quelles que soient les techniques employées.

