# DE RE ATARI®

ANNO DOMINI MCMLXXXI

## UN GUIDE POUR UNE PROGRAMMATION PERFORMANTE DES ORDINATEURS ATARI

ATARI®

ATARINSIDE

# DE RE ATARI

ANNO DOMINI MCMLXXXI

## UN GUIDE POUR UNE PROGRAMMATION PERFORMANTE DES ORDINATEURS ATARI

© 1981 ATARI, INC.

APX 90009

ATARINSIDE

# AVERTISSEMENT

Ce livre a été conçu par le Département Logiciel d'ATARI, à Sunnyvale, en Californie. Lors de sa traduction en français, nous avons délibérément choisi de ne pas traduire certains mots ou certains termes. En effet, de nombreux mnémoniques utilisés dans les ouvrages ATARI, dans la presse spécialisée ou dans les livres relatifs aux ordinateurs ATARI utilisent ces termes anglais condensés. Les traduire aurait entraîné un très grand désarroi du lecteur francophone. Par exemple, les lettres PM (Player Missile) devraient être traduites par MM (Mobile Missile) ou JM (Joueur Missile). Or, tous les livres de programmation (de jeux, entre autres) parlent de PM et bien sûr jamais de MM ni de JM.

D'autre part, la traduction en français aurait donnée lieu, dans certains cas, à des périphrases trop longues et à des risques de confusion dans l'esprit du lecteur.

Par contre, afin de rester clair et précis, nous avons modifié le glossaire à la fin de ce manuel afin de le compléter et d'expliciter ces termes difficilement traduisibles. Nous espérons ainsi donner à tout lecteur francophone une très bonne compréhension des ordinateurs ATARI tout en le laissant proche des termes utilisés par le monde.

Si vous possédez la cartouche Editeur/Assembleur ATARI ou le programme Macro Assembleur, vous retrouverez les mêmes mnémoniques et expressions dans les modes d'emploi de ces programmes.

ATARINSIDE

# SOMMAIRE



ATARINSIDE

ATARINSIDE

# PREFACE

Ce manuel s'applique essentiellement aux ordinateurs ATARI 400 et ATARI 800. En fait, il s'applique également aux ordinateurs ATARI 600XL et ATARI 800XL. Mais certaines adresses mémoire peuvent être différentes.

L'objet de ce manuel consiste à décrire en détail le fonctionnement et la manière d'utiliser les possibilités des machines ATARI. En raison de la puissance des ordinateurs ATARI, les explications sont parfois longues. De plus, elles demandent au lecteur un ensemble de connaissances, comme par exemple une excellente pratique du BASIC ATARI, et de bonnes connaissances en Assembleur 6502. Ce livre ne s'applique donc pas au programmeur débutant. Un glossaire placé en fin de ce livre défini certains termes, mais les plus usités ne sont pas explicités car ils font parti du vocabulaire que tout programmeur sérieux doit connaître.

Ce livre est conçu pour expliquer les idées de base et les possibilités plutôt que de définir des registres, des codes de contrôle ou de donner des programmes tout faits.

Le titre «DE RE ATARI» (prononcez dé ré) se réfère tout simplement à certains manuscrits écrits en latin au moyen âge et dont le titre était «DE RE ceci» ou «DE RE cela». Ainsi, «DE RE METALLICA» décrivait la métallurgie de l'époque. «DE RE» sous-entend donc «tout sur ...».

Ce livre a été écrit par le groupe Support et Développement Logiciel. Chris Crawford a rédigé les chapitres 1 à 6, ainsi que les additifs A et B. Lane Winner a conçu les chapitre 10 et additif D avec l'assistance de Jim Cox. Amy Chen a écrit l'additif C ; Jim Dunion, les chapitres 8 et 9 ; Kathleen Pitta, l'additif E. Bob Fraser, le chapitre 7. Enfin, Gus Makreas a préparé le glossaire. Le résultat final présente quelques petites imperfections mais nous sommes tout de même fiers de l'ensemble.

ATARINSIDE

# 1 - VUE GENERALE DU SYSTEME

Les Ordinateurs-Maison ATARI constituent la deuxième génération des ordinateurs personnels. En premier lieu, ce sont des ordinateurs destinés à être utilisés directement par le grand public. Le souci constant lors de leur étude a été de les rendre simple d'emploi pour l'utilisateur, et cela se retrouve sur la machine elle-même : l'ordinateur est protégé contre des erreurs de branchement (connecteurs avec détrompeurs) ; la touche RESET est suffisamment difficile d'accès pour éviter toute manipulation intempestive ; toute l'électronique est soigneusement blindée pour éviter toute interférence avec l'ordinateur de votre voisin.

L'ordinateur ATARI possède de grandes possibilités graphiques ; l'être humain réagit mieux à des images qu'à du texte. La synthèse sonore a fait également l'objet de soins attentifs de la part des concepteurs. Là aussi, l'être humain peut réagir plus rapidement à un événement sonore qu'à un message textuel. Enfin, l'ordinateur ATARI possède des entrées pour manette à levier ou à volant afin de permettre des manipulations autre que par le clavier. Il est nécessaire de bien comprendre que l'ordinateur ATARI n'a pas reçu ces possibilités comme autant de greffes, mais que ces éléments constituent la base même de cet ordinateur et représentent toute une philosophie orientée vers le confort de l'utilisateur. Le programmeur qui ne comprend pas cela ne saura jamais exploiter les immenses possibilités de sa machine.

Le schéma synoptique de l'ordinateur ATARI 400/600/800 est très différent de celui des autres systèmes existants. Bien sûr, il comprend un microprocesseur (dérivé du 6502), un ensemble de mémoires mortes et vives, et un circuit d'entrées-sorties (PIA). Mais il utilise en plus trois circuits particuliers à grande densité d'intégration, désignés par leur nom : ANTIC, CTIA, et POKEY. Ces circuits ont été conçus par les laboratoires d'ATARI pour soulager le microprocesseur de tâches fastidieuses, d'où une amélioration de la vitesse de calcul. Chacun de ces circuits possède 40 broches et utilise une surface de silicium pratiquement aussi grande que le microprocesseur lui-même. C'est dire qu'ensemble, ils concentrent une puissance exceptionnelle. Etre capable de parfaitement programmer les ordinateurs ATARI revient principalement à utiliser correctement ces boîtiers.

L'ANTIC est un microprocesseur spécialisé dans l'affichage TV. C'est un vrai microprocesseur ; il possède son jeu d'instructions, un programme ou liste d'instructions (appelée Display List), et des données. La Display List et les données sont écrites en mémoire vive par le 6502. L'ANTIC récupère ces informations en DMA puis génère une suite de commandes destinées au CTIA.

Le CTIA génère l'image vidéo en PAL ou en NTSC selon les pays. L'ANTIC contrôle directement la plupart des actions du CTIA mais le 6502 peut être programmé pour agir directement sur le CTIA, partiellement ou totalement. Le CTIA réalise de lui-même un certain nombre de fonctions, comme la gestion des registres couleurs, les graphiques utilisant des Players-Missiles, et la détection de collisions.

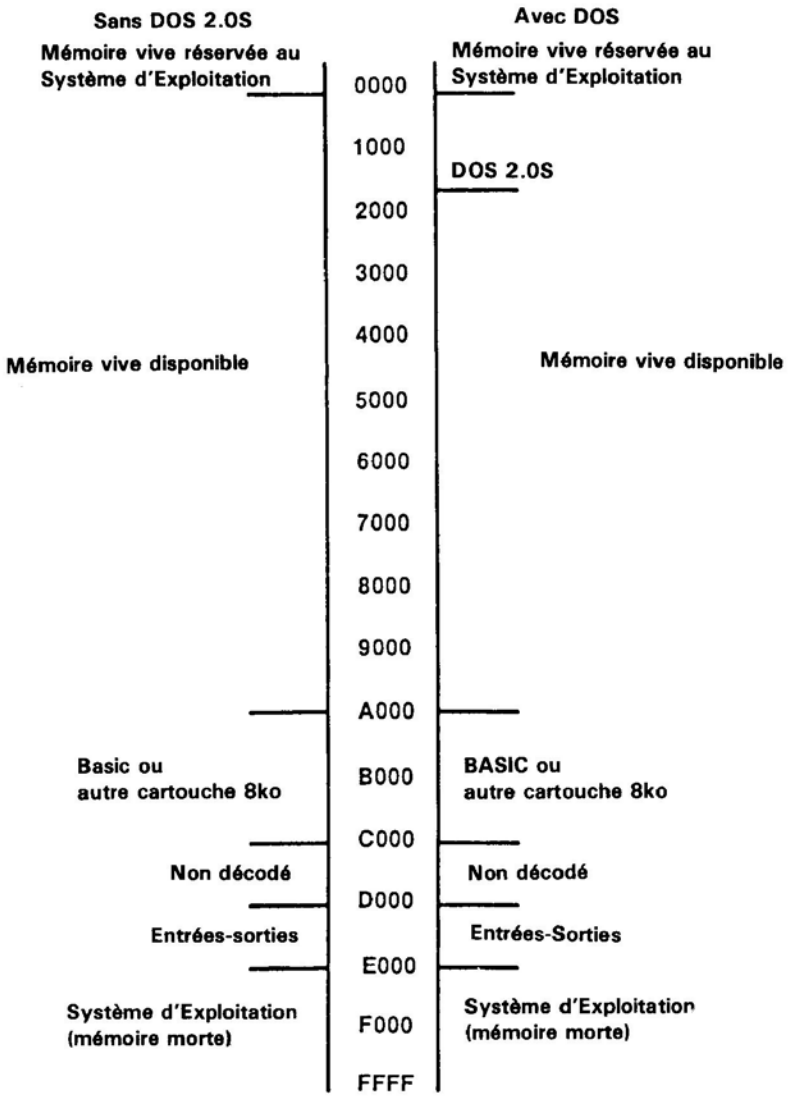
Le POKEY est un circuit d'entrées-sorties. Il gère le bus série, génère tous les sons, scrute le clavier et produit des nombres aléatoires. Il convertit également en numérique les entrées des commandes à volant et contrôle les interruptions.

Ces quatre circuits LSI fonctionnent simultanément. Une étude précise lors de la répartition de leur tâche entre eux a permis d'éviter des conflits sur les bus. Le seul conflit qui subsiste apparaît lorsque l'ANTIC accède à la mémoire pour y chercher ses informations. Pour effectuer cette opération, il doit arrêter le 6502 et prendre le contrôle des bus.

Comme dans tout système à base de 6502, les entrées-sorties sont implantées dans l'espace mémoire. Le dessin ci-dessous représente une cartographie simplifiée de la mémoire ; la figure suivante donne le schéma synoptique des ordinateurs ATARI.

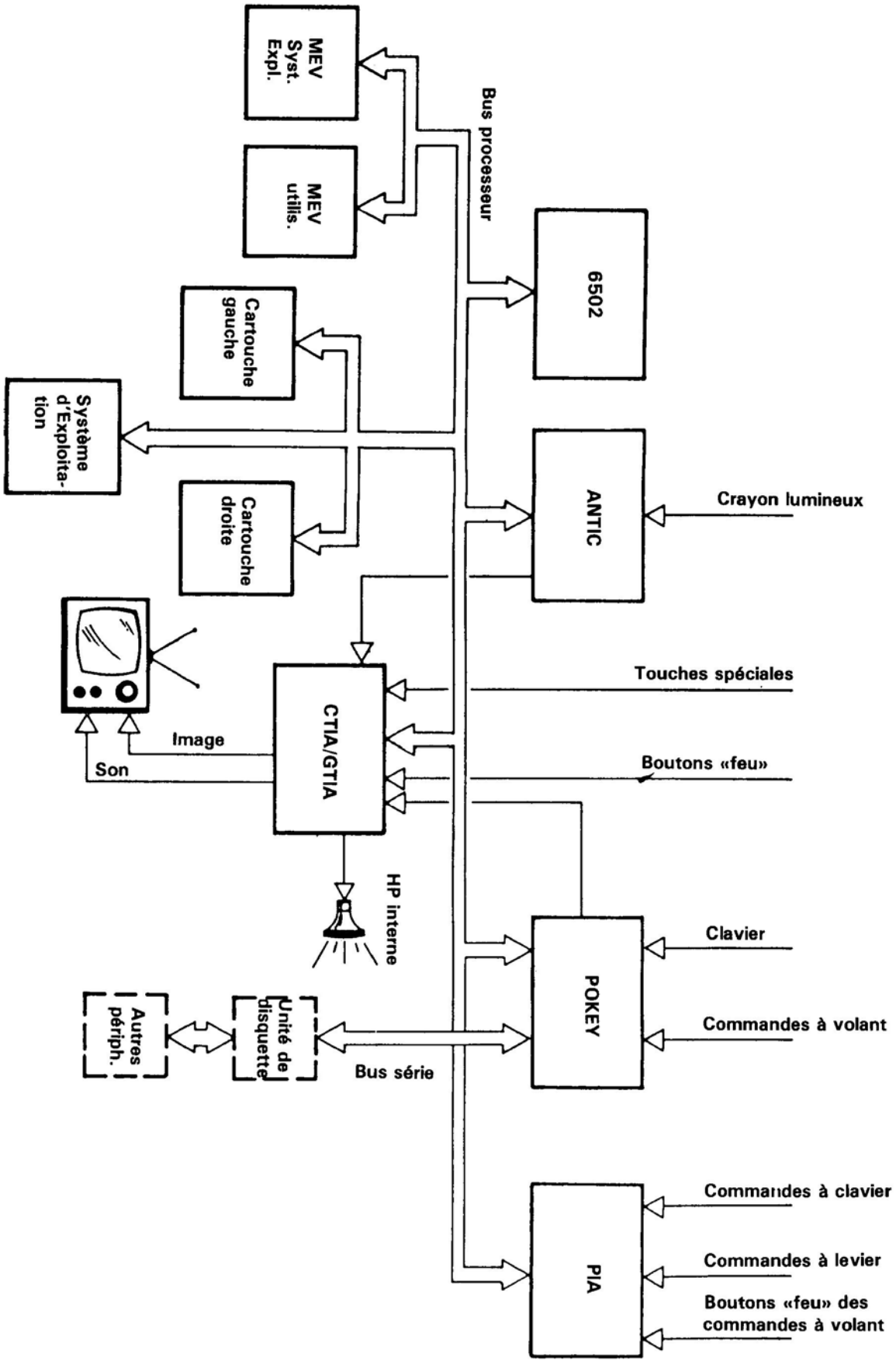


ATARINSIDE


Sans DOS 2.0S
Avec DOS
Mémoire vive réservée au Système d'Exploitation
Mémoire vive réservée au Système d'Exploitation
0000
1000
DOS 2.0S
2000
3000
4000
Mémoire vive disponible
Mémoire vive disponible
5000
6000
7000
8000
9000
A000
Basic ou autre cartouche 8ko
B000
BASIC ou autre cartouche 8ko
C000
Non décodé
Non décodé
D000
Entrées-sorties
Entrées-Sorties
E000
Système d'Exploitation (mémoire morte)
F000
Système d'Exploitation (mémoire morte)
FFFF



ATARINSIDE






ATARI INSIDE

# 2 - ANTIC ET LA DISPLAY-LIST

## L'AFFICHAGE TV

Pour comprendre les possibilités graphiques des ordinateurs ATARI, il est d'abord nécessaire de connaître les principes de fonctionnement d'un téléviseur. Un téléviseur utilise un système d'affichage à balayage. Un faisceau d'électrons est généré à l'arrière du tube cathodique pour être projeté sur le devant du tube, appelé écran, et recouvert de luminophores. A l'endroit de l'impact, le luminophore placé sur l'écran est excité par le flux d'électrons et en conséquence, émet une lumière, proportionnelle au nombre d'électrons qui le percutent. Sur son trajet, le flux d'électrons passe dans deux champs magnétiques perpendiculaires et formés par des bobines placées convenablement sur le col du tube cathodique. Un jeu de bobines dévie le faisceau verticalement, un autre jeu le dévie horizontalement. En combinant les deux effets, on peu donc «viser» n'importe quel point de l'écran. Le principe du balayage consiste à décrire régulièrement et systématiquement tout l'écran.

Le faisceau démarre au coin supérieur gauche de l'écran et trace une ligne horizontale sur l'écran. Pendant ce mouvement, l'intensité du faisceau est modulée de manière à exciter ou non les luminophores sur lesquels passe ce faisceau. Puis le faisceau est «éteint» pendant que les bobines de déflection le ramènent très rapidement à gauche de l'écran, légèrement plus bas que précédemment. Et le cycle se poursuit jusqu'à dessiner une image complète sur l'écran. Puis le faisceau revient à son point de départ et trace une nouvelle image. En répétant l'opération 50 fois par seconde, la vitesse est suffisante pour donner l'impression d'une image stable, non hachée. Pour diminuer encore l'effet de scintillement, une image TV (soit 625 lignes de balayage) est divisée en deux images : l'une, constituée de toutes les lignes impaires, est émise en premier ; puis vient la même image de départ dans laquelle on ne garde cette fois que les lignes paires. En alternant lignes paires et lignes impaires, l'émetteur TV émet donc 50 demi-images par seconde, une demi-image comportant 312 lignes environ. On dit que le balayage est entrelacé. En fait, dans le cas des ordinateurs ATARI, le CTIA génére 50 fois par seconde une demi-image de 312 lignes (toujours les mêmes). La définition est ainsi suffisante et l'électronique s'en trouve nettement simplifiée.

Et maintenant, voici quelques définitions : un seul passage du faisceau sur l'écran de la gauche vers la droite est appelé ligne de balayage horizontal. Une ligne constitue l'unité de mesure de la verticale de l'image. Ainsi, on parle d'une image de 312 lignes. Le temps pendant lequel le faisceau revient de la droite vers la gauche pour passer d'une ligne à la suivante, est appelée «blanking horizontal». Le temps pendant lequel le faisceau revient du coin inférieur droit au coin supérieur gauche s'appelle «blanking vertical». Le traçage complet d'une image dure 16684 microsecondes (en NTSC 60 Hz). Le blanking vertical dure environ 1400 microsecondes. Le blanking horizontal prend 14 microsecondes ; et la période d'une ligne est 64 microsecondes.

La grande majorité des téléviseurs dilate l'image de façon à ce que les bords de l'image n'apparaissent pas sur l'écran. Cela devient gênant pour les ordinateurs car vous risquez de ne pas pouvoir lire les premiers caractères de chaque ligne, ceux ci étant tracés hors écran. Pour cette raison, l'image générée par l'ordinateur doit être plus petite que celle théoriquement utilisable par le téléviseur. Ainsi, 192 lignes de balayage seulement peuvent être exploitées pour l'image de l'ordinateur. Une image ne pourra dont exéder 192 points en définition verticale. Aller au-delà ne servirait à rien car le téléviseur serait incapable de restituer l'intégralité de l'image. A l'intérieur d'une ligne de balayage, l'unité est l'impulsion d'horloge couleur. Vous spécifiez la largeur de l'image en indiquant le nombre d'impulsions par ligne. Il y en a 228 pour une ligne, dont 176 environ réellement visibles. Ainsi, la résolution maximale d'un téléviseur couleur standard est 176 cellules de couleur fondamentale, ou pixels. Avec un ordinateur ATARI, il est possible d'aller au-delà et de contrôler la résolution horizontale par demi-période d'horloge ; cela donne 352 pixels en résolution horizontale. Etant située au delà des possibilités du TV, cette résolution produit sur l'écran des aberrations chromatiques, normalement gênantes, mais qu'un programmeur avisé peut au contraire rechercher pour produire de nouvelles teintes.

## ORDINATEURS ET TELEVISEURS

Le problème fondamental de tout micro-ordinateur relié à un téléviseur réside dans le fait que le téléviseur ne se souvient pas de l'image qu'il dessine. En conséquence, l'ordinateur doit réaliser cette opération et envoyer en permanence vers le TV un signal vidéo représentant 50 images par seconde. Cette opération doit s'effectuer sans faillir et sans aucun retard, sinon l'image ne serait pas stable. En conséquence, la plupart des micro-ordinateurs possèdent une architecture particulière calquée sur le schéma suivant :

microprocesseur → mémoire écran → circuits vidéo → téléviseur

Le microprocesseur compose l'image sous forme binaire dans la mémoire écran. Les circuits vidéo lisent en permanence cette mémoire et convertissent l'information binaire en signal approprié pour le téléviseur. La mémoire écran représente dans l'ordre, les caractères destinés à l'affichage depuis le coin supérieur gauche de l'écran (premier octet), jusqu'au coin inférieur droit de l'écran (dernier octet) en suivant les lignes de balayage.



ATARINSIDE

La qualité de l'image obtenue dépend de deux facteurs : la qualité des circuits vidéo et la taille de la mémoire écran. L'arrangement le plus simple est celui utilisé sur les machines TRS80-1 et PET (marques déposées respectivement de Radio Shack Co et de Commodore Business Machines). Chacune de ces machines allouent 1 ko de mémoire vive à la mémoire écran. Les circuits vidéo lisent régulièrement cette zone et convertissent, grâce au générateur de caractères placé en mémoire morte, ces informations en points lumineux reconstituant des caractères sur l'écran TV. Chaque octet représente un caractère ; il peut donc exister 256 caractères différents. Avec une mémoire écran de 1 ko, un millier de caractères peuvent être affichés à l'écran. APPLE utilise des circuits vidéo plus sophistiqués. Trois modes d'affichage sont possibles : texte, graphique basse résolution, graphique haute résolution. En mode texte, le système fonctionne comme le PET ou le TRS80. En mode graphique basse résolution, l'interprêtation de chaque octet varie selon sa position dans la mémoire. Pour les 20 premières lignes de caractères (160 lignes de balayage), chaque octet est interprêté comme une perte de couleur, une couleur par pixel. Dans le mode haute résolution, un bit correspond à un pixel. Si le bit vaut 0, le point correspondant est allumé. Si le bit vaut 1, le pixel prend une certaine couleur. D'autres sophistications viennent en réalité compliquer cette description mais ce qu'il faut retenir est l'astuce d'interprêter la même mémoire écran de plusieures manières différentes. Un octet peut représenter un caractère, deux carrés de couleur ou sept points élémentaires de l'écran.

## ANTIC, UN MICROPROCESSEUR VIDEO

Le système de la Display List utilisé sur les ordinateurs ATARI représente une généralisation de ces systèmes. Ainsi, 14 modes d'affichage différents sont prévus. De plus, l'utilisateur n'est plus contraint de choisir entre un écran graphique et un écran texte : tous les modes graphiques peuvent être mélangés sur l'écran. Une troisième différence importante consiste à pouvoir déplacer la mémoire écran dans la mémoire de l'ordinateur, même pendant l'exécution du programme. Les autres machines utilisent des zones mémoires fixes.

Tout cela est rendu possible grâce au microprocesseur spécialisé ANTIC. ATARI a conçu ce circuit particulier très élaboré uniquement pour préparer l'image vidéo. Comme tout microprocesseur, ANTIC a son propre jeu d'instructions, son programme appelé Display List et ses données. La Display List spécifie 3 choses : où se situe la mémoire écran, de quelle manière la mémoire écran doit-elle être interprêtée, et quelles options d'affichage doivent être implémentées (si nécessaire). Pour comprendre le fonctionnement de la Display List, il est nécessaire de considérer l'écran texte ou graphique décrit précédemment comme une image homogène dans un mode déterminé et de considérer la Display List comme une pile de modes graphiques, un mode regroupant plusieures lignes de balayage horizontal. Une ligne de caractères en mode graphique 2 utilise par exemple 16 lignes de balayage, tandis qu'une ligne en graphique 7 utilise seulement 2 lignes de balayage. Les modes graphiques obtenus par le Basic sont homogènes : l'écran tout entier passe dans le mode sélectionné. Mais ne vous limitez pas à ces apparences : vous pouvez en fait mélanger n'importe quelle suite de modes sur l'écran : titre en mode 2 en haut, dessin haute résolution au milieu, légendes en mode 0, et bas de page en mode 1... La Display List vous permet de créer la séquence que vous désirez.

Le jeu d'instructions de l'ANTIC est très simple. Il existe quatre types d'instructions : mode graphique, mode caractère, ligne vide et saut. Les instructions en mode graphique demandent à l'ANTIC d'afficher une ligne composée de pixels colorés (pas de caractère). Les instructions en mode caractère demandent à l'ANTIC d'afficher des caractères. Les instructions de ligne vide provoquent l'affichage d'un certain nombre de lignes de balayage avec comme couleur celle du fond de l'image. Les instructions de saut sont analogues à l'instruction JMP du 6502 ; elles repositionnent le compteur ordinal de l'ANTIC.

Quatre options peuvent également être spécifiées : interruption de Display List (DLI), chargement du compteur mémoire (LMS), défilement vertical et défilement horizontal.

Les instructions en mode graphique provoquent l'affichage de lignes graphiques composées de pixels allumés ou éteints. La couleur affichée provient d'un registre couleur dans lequel on a placé auparavant un certain code (la couleur). Le choix du registre couleur est spécifié par la valeur lue dans la mémoire écran. Dans les modes graphiques à 4 couleurs (modes 3, 5 et 7 du Basic ; modes 8, A, D et E de l'ANTIC), une paire de bits est nécessaire pour coder une couleur.

| Valeur d'une paire de bits | Registre couleur utilisé |
|---|---|
| 00 (0) | COLBAK |
| 01 (1) | COLPF0 |
| 10 (2) | COLPF1 |
| 11 (3) | COLPF2 |

Comme deux bits suffisent pour spécifier la couleur d'un pixel, quatre pixels sont encodés sur un octet. Prenons un exemple. Supposons qu'un octet de la mémoire écran vaille $1B :

$$\$1B = 00011011 = 00\ 01\ 10\ 11$$

Le premier pixel correspondant à cet octet recevra la couleur de fond (COLBAK) ; le second, la couleur COLPF0 ; le troisième, la couleur COLPF1 ; et le quatrième, la couleur COLPF2.

Dans les modes à deux couleurs (modes 4, 6 et 8 du Basic ; modes 9, B, C et F de l'ANTIC), chaque bit détermine un des deux registres couleurs possible : 0 détermine l'usage de la couleur de fond (COLBAK), 1 l'usage de COLPF0. Huit pixels sont décrits par un octet.

Il existe huit modes graphiques différents. Ils diffèrent dans le nombre de couleurs utilisées (2 ou 4), la hauteur d'une


ATARINSIDE

ligne graphique (1, 2, 4 ou 8 lignes horizontales), et le nombre de colonnes adressables dans une ligne graphique (40, 80, 160 ou 320). Certains modes graphiques donnent une haute résolution, d'autres occupent peu de place en mémoire. La figure 2-1 résume les diverses possibilités offertes par l'ANTIC.

Les instructions en mode caractères demandent à l'ANTIC d'interpréter la valeur placée dans un octet comme un caractère. Il existe six modes caractères ; ils sont décrits en détail dans le chapitre 3.

Les instructions de ligne vide produisent des lignes de balayage dont la couleur est celle du fond de l'image. Il existe huit instructions de ce type, provoquant l'affichage de 1 à 8 lignes vides (c'est-à-dire de 1 à 8 lignes de balayage de couleur uniforme).

Figure 2-1 Les différents modes de l'ANTIC

| Mode ANTIC | Mode BASIC | Nombre de couleurs | Lign. de balay./ Lign. d'affichage | Nombre de col./ Lign. d'affichage | Octets/ Ligne | Octets/ Ecran |
|---|---|---|---|---|---|---|
| 2 | 0 | 2 | 8 | 40 | 40 | 960 |
| 3 | — | 2 | 10 | 40 | 40 | 760 |
| 4 | — | 4 | 8 | 40 | 40 | 960 |
| 5 | — | 4 | 16 | 40 | 40 | 480 |
| 6 | 1 | 5 | 8 | 20 | 20 | 480 |
| 7 | 2 | 5 | 16 | 20 | 20 | 240 |
| 8 | 3 | 4 | 8 | 40 | 10 | 240 |
| 9 | 4 | 2 | 4 | 80 | 10 | 480 |
| A | 5 | 4 | 4 | 80 | 20 | 960 |
| B | 6 | 2 | 2 | 160 | 20 | 1920 |
| C | — | 2 | 1 | 160 | 20 | 3840 |
| D | 7 | 4 | 2 | 160 | 40 | 3840 |
| E | — | 4 | 1 | 160 | 40 | 7680 |
| F | 8 | 2 | 1 | 320 | 40 | 7680 |

Il existe deux instructions de saut. La première (JMP) est un saut direct ; elle recharge le compteur ordinal avec une nouvelle valeur (adresse) constituant l'opérande de l'instruction JMP. Sa seule fonction est de résoudre un petit problème hardware : le compteur ordinal de l'ANTIC possède seulement 10 bits pour le comptage proprement dit et 6 bits pour le segment mémoire considéré. Aussi, la Display List ne peut se situer à cheval sur une frontière multiple de 1 ko sans utiliser l'instruction JMP pour passer cette frontière (changement de segment). Notez que cela sous-entend que la Display List n'est pas totalement relogeable.

La seconde instruction de saut (JVB) est plus couramment utilisée. Elle recharge le compteur ordinal avec la valeur donnée par l'opérande et attend l'impulsion de Blanking vertical. Cette instruction est normalement utilisée pour terminer la Display List et renvoyer ce programme à son début. Cela provoque une boucle infinie, un passage dans la boucle correspondant à une image complète. Cette instruction permet une parfaite synchronisation entre le déroulement du programme d'affichage et le balayage sur l'écran télé. Les instructions JMP et JVB sont toutes sur 3 octets ; le premier constitue l'instruction, les deux suivants, l'adresse (poids faibles/poids forts).

Les quatre options spéciales dont nous avons parlé précédemment seront détaillées dans les chapitres 5 et 6. Toutefois, parlons déjà un peu de l'instruction de chargement du compteur (LMS). Cette option est validée en positionnant à 1 le bit 6 de l'octet instruction de la Display List d'un mode texte ou graphique. Quand l'ANTIC décode une telle instruction, il charge son compteur interne avec les deux octets suivants qui lui donnent l'adresse de début de la mémoire écran correspondante. L'instruction LMS est codée sur 3 octets : l'instruction elle-même sur un octet, et l'adresse sur deux octets.

Dans une Display List simple, l'instruction LMS est utilisée une fois seulement, au début de la Display List, afin d'indiquer à l'ANTIC où commence la mémoire écran. Il peut être nécessaire d'utiliser d'autres instructions LMS, notamment lorsque la mémoire écran dépasse 4 ko. En effet, le compteur d'adresse interne à l'ANTIC fonctionne sur 12 bits, soit sur une page de 4 ko. Un segment complémentaire de quatre bits existe, mais sa valeur n'est ni incrémentée ni décrémentée par l'ANTIC. L'instruction LMS permet de changer ce segment en même temps que repositionner le compteur lui-même. les instructions LMS permettent bien d'autres choses que nous verrons plus loin.

## CONSTRUCTION D'UNE DISPLAY LIST

Chaque Display List doit commencer avec trois instructions de ligne vide, chacune générant huit lignes horizontales. Cela permet de centrer l'image verticalement sur le téléviseur. Puis la première ligne d'affichage doit être exécutée. Simultanément, l'instruction LMS est utilisée pour déterminer l'adresse de départ de la mémoire écran. puis la Display List se continue, en déterminant le mode pour chaque ligne d'affichage de l'écran (une ligne d'affichage contient de une à seize lignes de balayage selon le mode choisi). Le nombre maximum de lignes de balayage ne peut dépasser 192. Sinon, l'image ne sera pas stable. Afficher moins de 192 lignes de balayage ne provoque aucun problème, si ce n'est une réduction du temps d'éxécution du 6502. Le programmeur doit donc vérifier le nombre de lignes de balayage. Enfin, la Display List est terminée par une instruction JVB qui lui permet de boucler. Voici un exemple typique de Display List pour un écran standard homogène en mode 0 (numérotation Basic) :


ATARINSIDE

```
70      Huit lignes de balayage vide
70      Huit lignes de balayage vide
70      Huit lignes de balayage vide
42      Mode ANTIC 2 (mode Basic 0) + LMS
20      La mémoire écran commence en 7C20
7C
02      Deuxième ligne d'affichage en mode ANTIC 2
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
02
41      Saut au début de la Display List
E0      en FBE0 avec attente
7B      de synchronisation
```

Comme vous pouvez le constater, cette Display List est courte : seulement 32 octets. La plupart des Displays Lists sont inférieures à 100 octets et leur composition reste simple.

Pour concevoir votre propre Display List, vous devez d'abord étudier le format de l'affichage. Utilisez pour cela un papier et un crayon. Dessinez l'écran et sa partition entre les différents modes puis traduisez cela en lignes d'affichage puis en lignes de balayage. Ecrivez ensuite la séquence d'octets correspondants pour l'ANTIC. Ajoutez trois lignes d'affichage vide (24 lignes de balayage) (valeur $70) au début de la liste. Positionnez le bit 6 du code de la première ligne (c'est-à-dire ajoutez-lui $40). Insérez à la suite de cet octet ainsi modifié l'adresse (poids faible, poids fort) de début de la mémoire écran. A la fin de la Display List, ajoutez l'instruction JVB ($41) suivi de l'adresse du premier octet de la Display List. Rappelez-vous que vous ne devez pas franchir la limite d'un bloc de 1 ko de mémoire, sinon vous devez utiliser l'instruction JMP avant cette limite. Ensuite, vous placez cette Display List en mémoire, à l'emplacement que vous avez déterminé dans l'instruction JVB. Puis arrêtez le fonctionnement de l'ANTIC pendant une fraction de seconde, le temps de lui changer son pointeur de Display List. Ecrivez pour cela 0 à l'adresse SDMCTL ($22F). Placez l'adresse de départ de la nouvelle Display List aux adresses $230 et $231 (respectivement poids faible et poids fort). Enfin, redémarrez l'ANTIC en écrivant $22 dans SDMCTL. Lors du Blanking vertical, le système d'exploitation rechargera le compteur de l'ANTIC avec les valeurs placées aux adresses $230 et $231.

## ECRITURE SUR UN ECRAN DONT VOUS AVEZ CONÇU LA DISPLAY LIST

La mémoire écran peut être placée n'importe où dans le champ adressable de l'ordinateur. Normallement, la première instruction d'affichage de la Display List indique l'emplacement de la mémoire écran (c'est le rôle de l'instruction LMS). Toutefois, l'ANTIC peut effectuer une nouvelle instruction LMS à chaque ligne d'affichage si celà est nécessaire. De cette manière, toute la mémoire de l'ordinateur peut devenir mémoire écran. Cela peut permettre également la mise en place de fenêtres texte indépendantes.

Cependant, il existe plusieurs restrictions concernant la mise en place de votre mémoire écran. Premièrement, elle ne peut pas empiéter d'un bloc de 4 ko sur un autre. Ses adresses ne peuvent donc pas être à cheval sur des valeurs $1000, $2000, etc. Si vous ne pouvez éviter cela (c'est notamment le cas dans le mode Basic 8 qui utilise 8 ko de mémoire vive), vous devez utiliser avant cette frontière une instruction LMS pour repositionner le compteur interne de l'ANTIC. Deuxièmement, si vous désirez utiliser les sous-programmes du système d'exploitation relatifs à l'écran, vous devez respecter leurs règles d'utilisations. C'est particulièrement délicat lorsque dans un programme Basic, vous exploitez une Display List modifiée. Si vous modifiez une Display List standard et si vous tentez d'utiliser les instructions basic PRINT ou PLOT, le système d'exploitation les exécutera en supposant que la Display List est toujours celle d'origine. Cela provoquera des troubles à l'écran.

L'affichage ne sera pas correct dans trois cas bien précis : premièrement, le Basic peut refuser la tentative d'écriture sur l'écran car celà lui est impossible compte tenu du mode graphique qu'il croit utiliser. Le système d'exploitation mémorise le code du mode graphique à l'adresse $57. Mais si vous avez modifié la Display List, il n'y a plus correspondance entre le mode graphique exécuté par l'ANTIC et la valeur stockée en $57. Vous devez alors tromper le système d'exploitation en plaçant en $57 le mode graphique Basic adéquat (et non pas le mode graphique ANTIC).


ATARINSIDE

Le second problème survient lorsque vous mélangez des lignes d'affichage nécessitant des tailles mémoire différentes. Certaines lignes d'affichage utilisent 40 octets par ligne, d'autres 20, d'autres 10. Voyons ce qui se passe lors d'un mélange. Supposons que vous utilisez l'instruction PRINT. Au début, tout va bien tant qu'il y a correspondance entre le nombre d'octets utilisés par l'ANTIC et le nombre d'octets placés en mémoire vive (40) pour une ligne d'affichage. Mais supposons que la Display List signale une ligne d'affichage en mode Basic 1 (20 caractères, donc 20 octets par ligne). L'ANTIC décale les caractères suivants de 20 positions vers la droite. En effet, le système d'exploitation croyant que toutes les lignes utilisent 40 octets, insère 20 blancs pour compléter les 20 octets précédents dans la mémoire écran. Mais l'ANTIC ne lira, lui, que 20 octets pour constituer la ligne en mode 1. Lorsqu'il lira les 20 blancs excédentaires, il exécutera donc un décalage de 20 espaces de toute la suite de l'affichage.

La seule manière efficace de résoudre ce problème est d'éviter de mélanger des Displays Lists modifiées avec les instructions PRINT et PLOT du Basic. Vous pouvez aussi organiser de vous-même l'écran en groupe de lignes de même longueur en mémoire (par exemple, insérez 2 lignes de 20 octets par ligne dans un groupe de lignes de 40 octets par ligne).

Cette solution présente l'inconvénient de favoriser le troisième problème : les décalages verticaux. Le système d'exploitation positionne les caractères verticalement sur l'écran, en calculant le nombre d'octets qu'il faut sauter pour passer d'une ligne à l'autre, et cela en commençant au début de la mémoire écran. Dans un affichage standard de 40 octets par ligne, le basic positionnera les caractères sur la dixième ligne en sautant 360 octets depuis le début de la mémoire écran. Si vous avez inséré 4 lignes de 10 octets (soit pour le système d'exploitation, une seule ligne de 40 octets), l'affichage sera en fait décalé de 3 lignes vers le bas et vous écrirez sur la treizième ligne. Enfin, les différents modes provoquent l'utilisation d'un nombre différent de lignes de balayage pour un point élémentaire (caractères graphiques) sur l'écran.

Comme vous pouvez le constater, le mélange de différents modes d'affichage peut être difficile à utiliser en Basic. Souvent, vous devrez modifier la valeur placée à l'adresse $57 (mode Basic courant pour le système d'exploitation). Afin d'utiliser les instructions PRINT ou PLOT dans une fenêtre déterminée et homogène, placez le numéro du mode Basic correspondant à cette fenêtre à l'adresse $57 puis l'adresse du premier octet de cette fenêtre aux adresses $58 et $59 (poids faible et poids fort). En mode texte, utilisez l'instruction POSITION 0,0 pour amener le curseur à cette position dans la fenêtre et utilisez ce nouveau point comme départ des nouvelles coordonnées d'affichage. Dans les modes graphiques, il n'est pas nécessaire d'utiliser POSITION 0,0 puisque toutes les instructions PLOT et DRAWTO sont déjà calculées en partant du coin supérieur gauche de la fenêtre.

Les modifications de Display List sont très utiles pour réaliser des affichages plus attractifs. Par exemple, vous pouvez concevoir un écran avec un titre en mode Basic 2, un sous-titre en mode 1, du texte en mode 0, un dessin en mode 8 et à nouveau du texte en mode 0. Un bon exemple de cette technique est fourni par l'écran utilisé dans le programme ATARI ETATS et CAPITALES des ETATS-UNIS.

Les problèmes mentionnés ci-dessus demandent donc beaucoup d'attention au programmeur lorsqu'il travaille en Basic. En Assembleur, il suffit d'organiser l'écran en une série de fenêtres, chacune ayant sa propre mémoire écran indiquée à l'ANTIC par une instruction LMS.

## APPLICATIONS DES DISPLAYS LISTS

Une application simple de Display List modifiée consiste à insérer des lignes (vides) de balayage sur l'écran. Cela améliore la visibilité en augmentant l'écrat entre les lignes de texte.

Vous pouvez également obtenir des résultats inaccessibles depuis le Basic. Par exemple, il existe 3 modes texte supportés par l'ANTIC mais que le Basic n'autorise pas. Seule l'utilisation d'une Display List modifiée permet l'accès à ces modes particuliers.

Les interruptions de Display List ainsi que le défilement de l'image verticalement et horizontalement ne sont possibles qu'après avoir modifié la Display List. Nous verons cela en détail dans les chapitres 5 et 6.

L'utilisation astucieuse de l'instruction LMS offre de nouvelles perspectives au programmeur créatif. Par exemple, en modifiant l'argument d'une instruction LMS pendant la période de Blanking vertical, le programmeur peut alterner diverses images. Cela peut être réalisé à basse vitesse (on passe ainsi d'un écran à un autre sans la perte de temps dû au tracé du second écran, mais cela sous-entend que toutes les images nécessaires existent simultanément en mémoire). Mais une cadence plus rapide peut être choisie afin d'obtenir des effets d'animation : il suffit de changer deux octets au début de la Display List (l'argument de la première instruction LMS) pour passer instantanément d'une image composée de centaines, voire de milliers d'octets à une autre.

Il est également possible de superposer des images. En tenant compte de la rémanence du tube cathodique et de celle de l'oeil humain, vous pouvez commuter suffisamment vite entre deux ou trois images pour donner l'impression d'une seule image composée et stable à l'écran. Vous augmentez ainsi la résolution, le nombre de couleurs et d'objets, etc. Toutefois, chaque image étant affichée peu de temps à l'écran, il est nécessaire que chacune soit bien contrastée afin d'être encore visible dans le résultat final. D'où un choix nécessairement astucieux des couleurs et des luminances. Ce procédé permet également la création de nouvelles nuances colorées. Vous obtenez en effet une palette plus large de luminances.

Une dernière suggestion concerne une idée intéressante mais trop rarement utilisée : la Display List dynamique. Il s'agit d'une Display List modifiée par le 6502 durant les périodes de Blanking vertical. Cela peut permettre la modification de la Display List au fur et à mesure que le curseur est déplacé verticalement (c'est le cas du programme ATARI EDITEUR DE PROGRAMMES ET DE TEXTES). Cette technique reste délicate à utiliser, mais ses attraits sont nombreux.



ATARINSIDE

# 3 - GRAPHIQUES, REGISTRES COULEURS ET JEUX DE CARACTERES

NOTE : Dans ce manuel, nous utilisons certains termes bien précis. Ainsi, le registre couleur de fond contient le code de la couleur du fond de l'écran (le bleu en mode Basic 0). Nous désignons par registre couleur image les quatre autres registres couleur utilisés totalement ou en partie dans tous les modes graphiques. Nous parlerons de registres couleur Players pour les registres réservés aux Players Missiles.

La notion d'indirection est particulièrement intéressante en programmation. Dans le langage Assembleur du 6502, plusieurs niveaux d'indirection sont autorisés : adressage page zéro, adressage indirect, adressage indirect indexé. Cela autorise la conception de programmes plus concis, plus rapides (en temps d'éxécution) et plus efficaces. Il est plus facile de pointer vers une table de valeurs plutôt que de répéter autant de fois que nécessaire la valeur adéquate. En changeant le pointeur, il devient ainsi très facile de modifier le contenu même du programme.

## REGISTRES COULEURS

Le système d'exploitation des ordinateurs ATARI utilise ce processus très souvent. Ainsi, le début de la mémoire vive n'est constitué pratiquement que de pointeurs. Un pointeur susceptible d'être modifié et utilisé lors d'un adressage indirect, s'appelle un vecteur. Enfin, un grand nombre de registres hardware (registres intégrés dans l'ANTIC ou dans le CTIA) possède un double en mémoire vive. Ces doubles s'appellent registres cache.

Un registre couleur ne désigne pas directement une couleur. C'est plutôt le contenant, le récipient dans lequel on verse la peinture de la nuance voulue. Un registre couleur peut donc recevoir n'importe quelle couleur. Par contre la couleur du fond de l'écran (par exemple) sera toujours prise dans le même registre couleur.

L'ordinateur ATARI possède 9 registres couleurs : 4 sont réservés aux Players Missiles, et nous les étudierons au chapitre 4. Les 5 autres sont utilisés ou non en fonction du mode graphique utilisé. Par exemple, dans le mode graphique 0, seulement 2 1/2 registres sont utilisés : celui donnant la teinte du fond, celui donnant la teinte de la bordure et celui (utilisé à moitié) donnant la luminance des caractères (la couleur étant la même que celle du fond). La couleur des caractères vient ainsi du registre 2, leur luminance du registre 1. Les registres couleurs hardware sont placés dans le CTIA aux adresses $D016 à $D01A. Ils sont doublés par des registres caches (aux adresses 708 à 712 en décimal) et durant chaque période de Blanking vertical, les valeurs de ces registres caches sont recopiées dans le CTIA. Le tableau ci-dessous donne les adresses et les noms mnémoniques de ces registres.

| Image | Registre CTIA Etiquette | Adresse | Registres cache Etiquette | Adresse |
|---|---|---|---|---|
| PLAYER 0 | COLPM0 | D012 | PCOLR0 | 2C0 |
| PLAYER 1 | COLPM1 | D013 | PCOLR1 | 2C1 |
| PLAYER 2 | COLPM2 | D014 | PCOLR2 | 2C2 |
| PLAYER 3 | COLPM3 | D015 | PCOLR3 | 2C3 |
| OBJET 0 | COLPF0 | D016 | COLOR0 | 2C4 |
| OBJET 1 | COLPF1 | D017 | COLOR1 | 2C5 |
| OBJET 2 | COLPF2 | D018 | COLOR2 | 2C6 |
| OBJET 3 | COLPF3 | D019 | COLOR3 | 2C7 |
| FOND | COLBK | D01A | COLOR4 | 2C8 |

Dans deux cas seulement, le programmeur écrira directement dans le CTIA. Le premier et le plus courant correspond à une interruption de Display List telle que nous la verrons au chapitre 5 : cela autorise un changement de valeur dans le registre couleur en cours de tracé d'image. Le second cas apparaît lorsque le programmeur inhibe la partie du système d'exploitation transférant à chaque Blanking vertical les valeurs des registres caches vers le CTIA. Nous verrons cela au chapitre 8.

Les couleurs sont codées dans les registres couleurs d'une manière simple : les quatre bits de poids forts donnent la teinte de base ; cette valeur est identique au second paramètre de la commande SETCOLOR du Basic. Les quatre bits de poids faible donnent la luminance ; cette valeur correspond au troisième paramètre de SETCOLOR. Le bit de poids le plus faible (D0) n'est pas significatif dans le cas du CTIA. 128 nuances sont donc possibles au total (256 dans le cas du GTIA). Dans ce livre, nous désignons par «nuance» ou «teinte» un couple couleur + luminance.

Lorsqu'une nuance est placée dans le registre couleur, elle apparaît sur l'écran chaque fois que le registre la contenant est désigné. Dans les modes graphiques qui utilisent quatre registres couleurs, les données placées dans la mémoire écran indiquent quel registre utiliser en codant le numéro du registre sur deux bits. Un octet peut donc définir la couleur de quatre points graphiques adjacents sur une même ligne de balayage (on appelle «point graphique» l'unité élémentaire



ATARI INSIDE

de dessin utilisée dans le mode considéré ; par exemple en mode Basic 0, un point graphique vaut un caractère et se compose de 8 × 5 points à l'écran ; en mode Basic 7, un point graphique se compose de 2 × 2 points à l'écran ; le point à l'écran représentant la plus petite quantité vidéo définie par l'utilisateur).

Dans les modes Basic 1 et 2, la sélection du registre couleur est réalisée par les deux bits de poids fort de chaque octet, les six bits restants définissant le caractère. Voilà pourquoi dans ces modes, 64 caractères seulement sont utilisables.

Vous pouvez facilement manipuler ces registres en temps réel de manière à produire des effets attrayants. En voici l'exemple le plus simple :

```
FOR I = 0 TO 254 : POKE 712,I : NEXT I
```

La couleur de la bordure de l'écran change rapidement. Cela est plaisant et certainement attrayant. Cette technique de base peut maintenant être étendue de moult manières. On peut par exemple dessiner en quatre couleurs puis faire tourner les couleurs dans les registres. Cela crée une certaine animation sans avoir besoin de redessiner l'écran. On comprend sur cet exemple l'intérêt de posséder des registres couleur plutôt que de coder directement dans la mémoire écran la couleur voulue : dans ce dernier cas, il aurait fallu transformer des centaines, voire des miliers d'octets. Le programme suivant illustre cette idée.

```
10 GRAPHICS 23
20 FOR X=0 TO 39
30 FOR I=0 TO 3
40 COLOR I
50 PLOT 4*X+I,0
60 DRAWTO 4*X+I,95
70 NEXT I
80 NEXT X
90 A=PEEK(712)
100 POKE 712,PEEK(710)
110 POKE 710,PEEK(709)
120 POKE 709,PEEK(708)
130 POKE 708,A
140 GOTO 90
```

Une application intéressante des registres couleurs permet de colorer astucieusement les différents écrans d'un programme. Par exemple, dans un programme de gestion du budget, les écrans correspondants aux recettes reçoivent une bordure verte ; ceux des dépenses ont une bordure rouge ; ceux des statistiques et des résultats, une bordure jaune. L'écran peut clignoter en rouge lorsque l'on appuie sur une mauvaise touche. Les caractères colorés des modes Basic 1 et 2 permettent toute une animation ; les mots importants peuvent ainsi être facilement mis en évidence.

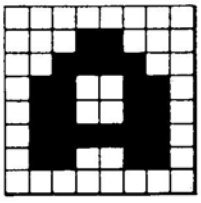
Enfin, il est possible sans grande difficulté d'augmenter le nombre de couleurs apparaissant à l'écran au-delà de la limite des cinq registres existants. Celà, grâce aux interruptions de Display List que nous verrons au chapitre 5. Un simple registre permet alors l'affichage de 128 nuances sur un même écran.

## JEUX DE CARACTERES

Dans un ordinateur ATARI, le jeu de caractères peut être totalement redéfini. De plus, on peut même définir simultanément en mémoire plusieurs jeux de caractères, et en changeant un seul vecteur, passer instantanément d'un jeu à l'autre même en cours d'image.

A la mise sous tension, le système d'exploitation figé en mémoire morte fourni un jeu standard de 256 caractères. Mais l'utilisateur peut le modifier en trois étapes.

Tout d'abord, le programmeur redéfinit la forme de chaque caractère. C'est l'opération la plus longue à réaliser. Chaque caractère doit tenir dans une grille de 8 × 8 points élémentaires, ce qui correspond à 8 octets en mémoire, un octet définissant 8 points horizontalement. En voici un exemple :

| Image | Représentation binaire | Représentation hexa |
|---|---|---|
|  | 00000000 | 00 |
| | 00011000 | 18 |
| | 00111100 | 3C |
| | 01100110 | 66 |
| | 01100110 | 66 |
| | 01111110 | 7E |
| | 01100110 | 66 |
| | 00000000 | 00 |



ATARINSIDE

Un jeu complet se compose de 128 caractères différents (les 128 autres étant fournis par l'inversion vidéo des précédents). Cet ensemble occupe donc 1024 octets en mémoire. Cette zone ne doit pas être à cheval sur deux blocs de 1 k. Pour les modes Basic 1 et 2, le jeu de caractères se réduit à 64, soit 512 octets, cette zone ne devant pas être à cheval sur deux blocs de 512 octets. La redéfinition de caractères est une tâche assez importante, et bien sûr de durée proportionnelle au nombre de caractères redéfinis. Heureusement, dans son catalogue de logiciels, ATARI propose un éditeur de caractères performant (dans le catalogue APX, «Générateur de caractères») simplifiant votre travail.

Lorsque le nouveau jeu de caractères est placé dans la mémoire, vous devez signaler à l'ANTIC la nouvelle adresse de début de ce jeu. Pour cela, inscrivez le numéro de la page mémoire (octet de poids fort de l'adresse) où commence le nouveau jeu de caractères à l'adresse $D409 (54281 en décimal) ou dans la mémoire cache correspondante (CHBAS) à l'adresse $2F4 (756 en décimal).

Enfin, la dernière étape consiste bien évidemment à mettre en pratique ce jeu de caractères, soit par l'instruction PRINT, soit en plaçant les bons codes directement dans la mémoire écran.

Il existe une possibilité intéressante de l'ANTIC non accessible depuis le Basic : les caractères multicolores. Les modes Basic 1 ou 2 autorisent 5 couleurs sur l'écran, mais un caractère utilise une couleur et une seule. Par contre, les modes ANTIC 4 et 5 permettent d'obtenir des caractères utilisant chacun plusieurs couleurs. La matrice d'un tel caractère se compose de 4 points graphiques de large, un point graphique vallant 2 points élémentaires. Chaque point graphique reçoit sa propre couleur, à choisir parmi 4 registres dont celui de la couleur de fond. Chaque octet du jeu de caractères est éclaté en 4 couples de bits ; chaque couple sélectionne un registre couleur pour le point correspondant. Le bit de poids le plus fort (D7) permet de choisir comme quatrième registre, le quatrième ou le cinquième registre couleur objet. Cette sélection est valide pour les quatre points graphiques de l'octet. En combinant ces possibilités, on peut utiliser 5 couleurs pour un caractère. La sélection des registres couleur s'effectue selon le tableau ci-dessous :

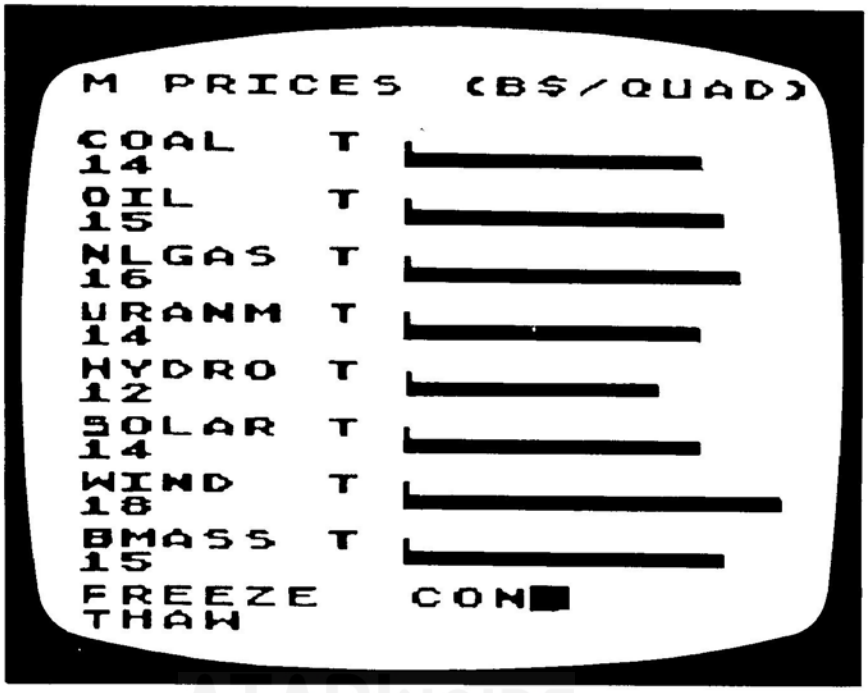
| Couple de bit | D7 = 0 | D7 = 1 |
|---|---|---|
| 00 | COLBAK | COLBAK |
| 01 | PF0 | PF0 |
| 10 | PF1 | PF1 |
| 11 | PF2 | PF3 |

Notez qu'afficher des caractères ordinaires en 5 couleurs peut sembler tordu... Mais pensez à la redéfinition de caractères et vous pourrez concevoir des objets très finement colorés en utilisant peut de place en mémoire.

Un autre mode intéressant est le mode ANTIC 3. Il autorise en effet 10 lignes de balayage par ligne d'affichage mais comme les caractères sont définis sur seulement 8 lignes de balayage, les 2 lignes inférieures sont normalement vides. Toutefois, si un caractère du dernier quart du jeu de caractères est affiché, les 2 lignes supérieures de ce caractère seront vidées et transférées sur les deux lignes inférieures. Cela permet la création de lettres minuscules avec jambages.

## APPLICATIONS DES JEUX DE CARACTERES

Le fait de pouvoir modifier des caractères constitue une des possibilités très intéressante des ordinateurs ATARI. La première application consiste à créer des caractères plus élégants (écriture anglaise) ou plus techniques (alphabet grec). On peut ensuite concevoir des caractères spéciaux pour augmenter la résolution graphique tout en restant dans un mode texte, plus économe en mémoire. Ainsi, le jeu «Ministre de l'Energie» redéfinit des caractères pour pouvoir tracer des lignes horizontales avec une résolution de l'ordre du point élémentaire, alors que le texte est en mode Basic 1. Les caractères redéfinis sont les moins usités (&, £, etc). Un caractère représente un trait vertical d'un point élémentaire de large ; un second représente un trait de deux points de large ; et ainsi de suite jusqu'à huit points de large. Il suffit ensuite de les combiner simplement pour obtenir la longueur adéquate. La figure ci-dessous montre le résultat obtenu. Le mélange caractères/graphique n'est qu'apparent, l'écran tout entier utilise des caractères.






ATARINSIDE

On peut aller encore plus loin en créant des images spéciales. Par exemple, avec un jeu de caractères bien dessinés, il est possible de reproduire des rivières, un rivage, des forêts, etc et de construire ainsi une carte géographique. Il faut penser dans ce cas à définir 5 à 8 aspects différents du même dessin de base de manière à éviter toute monotonie dans l'affichage. La carte du jeu «Front de l'Est» utilise cette technique. On obtient par la même occasion et avec une Display List adéquate, jusqu'à 18 couleurs sur l'écran.


Vous pourriez tout aussi bien créer des caractères électroniques et concevoir ainsi des schémas à l'écran. Les plus gros composants seront formés de plusieurs caractères. Vous pouvez envisager un jeu de caractères architecture possédant les caractères «porte», «mur», «coin», etc. Les possibilités graphiques permises par la redéfinition des caractères n'ont pas encore été pleinement explorées.

Les caractères peuvent être retournés verticalement en plaçant la valeur 4 à l'adresse 755 (en décimal). Cette particularité trouve une application dans la représentation des cartes à jouer. La moitié supérieure de la carte est affichée normalement, puis une interruption de Display List située au milieu de l'affichage de la carte, inverse les caractères pour la moitié inférieure du dessin. Vous créez ainsi des images avec réflexion, comme dans un miroir ou sur un plan d'eau.

Allons encore plus loin en modifiant le jeu de caractères pendant le déroulement du programme. Un jeu de caractères utilisant 512 ou 1024 octets, vous pouvez en placer plusieurs en mémoire et passer de l'un à l'autre à des cadences différentes selon vos besoins : supérieure à 1 seconde, 1/50e de seconde à 1 seconde, inférieure à 1/50e de seconde.

Les commutations à faible vitesse permettent un changement de scène ; par exemple, un programme de voyage dans l'espace peut utiliser un type de caractères pour une planète, un autre type pour une autre planète, etc. Un jeu d'aventure modifie le set de caractères de salle en salle.

Une commutation à vitesse plus rapide donne une animation à l'écran ; ainsi, dans Space Invaders, les envahisseurs de l'espace sont en réalité composés de caractères. En les changeant à chaque seconde, le programmeur les anime. Il n'existe que 6 dessins de monstres, chacun ayant 4 incarnations différentes.

Une animation encore plus rapide sur un écran tout entier s'obtient en dessinant dans un premier temps l'écran, puis en effectuant des rotations rapides sur les jeux de caractères. Si chaque caractère est dessiné avec de légères différences par rapport à ses homologues dans les autres sets, cela donnera l'impression de voir tout l'écran s'animer. Lorsque tous les caractères voulus sont dans la mémoire, l'animation s'effectue en modifiant un seul octet à l'adresse 756. Quoi de plus simple! En supposant que CHARBAS(I) représente un tableau de 10 pointeurs vers 10 jeux de caractères, une animation en Basic s'écrirait :

```
1000  FOR I=1 TO 10
1010  POKE 756,CHARBAS(I)
1020  NEXT I
1030  GOTO 1000
```

Une animation encore plus rapide donne la possibilité d'afficher sur le même écran plusieurs jeux de caractères différents. Cela, en utilisant les interruptions de Display List. Nous verrons cela en détail dans le chapitre 5.



ATARINSIDE

L'utilisation de plusieurs jeux de caractères pour créer des graphiques sophistiqués et des animations présente de nombreux avantages et quelques limitations. Le plus gros avantage : très peu de mémoire vive suffit pour obtenir un graphisme détaillé. Un jeu de caractères redéfinis, en mode Basic 2, peut donner autant de détails avec une couleur de plus qu'en mode Basic 7. D'autre part, 200 octets suffiront pour définir cette image contre 4000 en mode 7. Un jeu de caractères n'utilisant en ce cas que 512 octets, il n'est pas déraisonnable d'en installer plusieurs simultanément en mémoire.

Les manipulations de caractères s'exécutent rapidement car peu de données sont à modifier à chaque fois. Toutefois, les graphiques réalisés par des caractères redéfinis sont moins souples à utiliser et à mettre au point : une modification d'apparence mineure peut nécessiter la retouche de plusieurs caractères. Dans certains cas, comme par exemple la représentation d'une fonction quelconque, il est impossible d'utiliser des caractères graphiques.


ATARINSIDE

# 4 - ANIMATION PAR PLAYER-MISSILE

## DIFFICULTES D'UNE ANIMATION RAPIDE

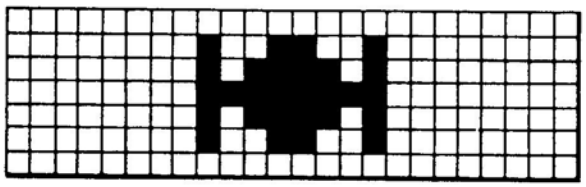
L'animation est l'une des possibilités les plus importantes de l'ordinateur-maison. Une activité sur l'écran améliore nettement l'attrait du programme. L'animation est l'un des points les plus importants de tout jeu vidéo et d'ailleurs une image animée donnera davantage d'informations qu'une image statique (voyez par exemple le jeu Centrale Nucléaire). En dehors du domaine du jeu, une animation peut aussi attirer l'attention sur un point dangereux ou délicat. Un processus peut être montré en fonctionnement dynamique : un dessin animé vaut mieux qu'un long discours. L'animation doit donc être considérée comme un élément important dans les possibilités graphiques de n'importe quel ordinateur.

La façon traditionnelle de réaliser une animation avec un ordinateur personnel consiste à déplacer l'image dans la mémoire écran. Cela nécessite deux étapes. Tout d'abord, le programme doit effacer l'ancienne image en écrivant à sa place les valeurs du fond de l'image. Puis il doit écrire l'image à son nouvel emplacement dans la mémoire écran. En répétant ce processus rapidement maintes et maintes fois, l'image donne l'impression de se déplacer sur l'écran.

Il existe toutefois deux problèmes importants avec cette technique : en premier lieu, si l'objet est créé dans un mode graphique utilisant des points élémentaires de grande surface, le mouvement sera saccadé. L'image saute d'une position à une autre, trop éloignée. Avec certains ordinateurs, la solution consiste à utiliser un mode graphique donnant une meilleure résolution et un point élémentaire plus petit.

Le second problème est plus important. L'image est une représentation en deux dimensions mais la mémoire de l'ordinateur est organisée en une suite de cases mémoire consécutives et décrites séquentiellement d'une seule manière. On peut considérer que la mémoire donne une représentation unidimensionnelle de l'image. Cela veut dire que l'image d'un objet sur l'écran ne sera pas représentée par une suite continue d'octets en mémoire. En voici un exemple :

IMAGE



Octets correspondants en mémoire vive

00 00 00
00 99 00
00 BD 00
00 FF 00
00 BD 00
00 99 00
00 00 00

Représentation dans la mémoire vive

00 00 00 00 99 00 00 BD 00 00 FF 00 00 BD 00 00 99 00 00 00 00

Les octets représentant l'objet se retrouvent isolés les uns des autres.

Cela ne semble pas très gênant tant que vous ne cherchez pas à déplacer l'objet. Par contre, pour une animation, nous découvrons de nouvelles complications : votre programme doit recalculer les adresses de tous les points de l'objet, et cela n'est pas nécessairement évident. Voici par exemple un sous-programme en Assembleur permettent de retrouver l'adresse d'un octet en partant du début de la mémoire écran, en connaissant les coordonnées XPOS et YPOS de l'objet correspondant sur l'écran (on suppose qu'une ligne est composée de 40 octets).



ATARI INSIDE

```
LDA SCRNRM      Adresse du début de la mémoire écran
STA POINTR      Pointeur en page zéro
LDA SCRNRM+1    Poids fort de l'adresse
STA POINTR+1    Poids fort du pointeur
LDA #$00
STA TEMPA+1     Registre temporaire
LDA YPOS        Position verticale
ASL A           × 2
ROL TEMPA+1     Décalage avec retenue dans TEMPA + 1
ASL A           × 4
ROL TEMPA+1     Décalage
ASL A           × 8
ROL TEMPA+1     Décalage
LDX TEMPA+1     Sauvegarde de YPOS*8
STX TEMPB+1     dans TEMPB
STA TEMB        Poids faible
ASL A           × 16
ROL TEMPA+1
ASL A           × 32
ROL TEMPA+1
CLC
ADC TEMPB       Addition de YPOS*8 avec YPOS*40
STA TEMPB
LDA TEMPA+1     Calcul sur poids fort
ADC TEMPB+1
STA TEMPB+1
LDA TEMPB       TEMPB contient l'offset depuis le début de la mémoire écran jusqu'à
CLC             l'octet considéré
ADC POINTR
STA POINTR
LDA TEMPB+1
ADC POINTR+1    Addition au pointeur
STA POINTR+1
LDY XPOS
LDA (POINTR),Y  On y accède!
```

Ce programme n'est sûrement pas la manière la plus élégante et la plus rapide pour résoudre le problème. Mais il montre clairement la complexité de la tâche à exécuter. La gestion de l'écran occupe en conséquence le microprocesseur pendant une grande partie de son temps. Le programme ci-dessus utilise plus de 100 cycles machines pour accéder à un seul octet sur l'écran. Pour déplacer effectivement une image qui occuperait 50 octets, cela prendrait plus de 10 millisecondes. Si vous désirez au même moment un déplacement doux de plusieurs objets, le microprocesseur n'aura plus le temps de calculer, de produire des effets sonore, ou de scruter le clavier. Bien que des animations soient possibles de cette manière, vous êtes en fait rapidement limité à un ou deux objets, à des déplacements lents ou à de petits objets, et à peu de calcul.

## LES PRINCIPES DE BASE DES PLAYERS-MISSILES

La solution utilisée par les ordinateurs ATARI réside dans l'utilisation des Players-Missiles. Afin de bien comprendre cette technique, il est nécessaire de comprendre l'essence même du problème de l'animation : l'image écran est en deux dimensions alors que sa représentation en mémoire est unidimensionnelle. La solution consiste à créer un objet dont la représentation soit unidimensionnelle aussi bien à lécran que dans la mémoire vive de l'ordinateur. Cet objet que nous appelons un Player ou un Missile apparaît en mémoire comme une table, située en dehors de la mémoire écran, et possédant 128 ou 256 octets de long. Cette table est ensuite recopiée directement sur l'écran. Elle apparaît comme une bande verticale descendant sur l'écran du haut vers le bas. Chaque octet dans la table représente une ou deux lignes de balayage, le choix restant au programmeur. Si un bit de l'octet est à 1, le point correspondant sur l'écran prend la couleur du registre couleur associé au Player (et indépendant des registres couleur de l'image elle-même). Si le bit est à 0, l'image écran n'est pas modifiée. En réalité, on comprend qu'un player est unidimensionnel au niveau de l'octet : il possède quand même une certaine largeur (8 bits).

Pour dessiner l'image d'un Player sur l'écran, vous devez d'abord dessiner l'objet sur une feuille de papier quadrillé, avec 8 cases en largeur. Donnez la valeur 1 à tous les carrés noircis, 0 aux autres. Vous obtenez ainsi 8 valeurs binaires


ATARINSIDE

(1 octet) par ligne de cases du Player. Lorsque vous avez terminé, vous possédez la table représentant le Player (selon la résolution du Player, cette table pourra contenir 128 ou 256 octets au maximum). Puis vous devez choisir une portion de la mémoire vive de l'ordinateur que vous réserverez au Player. Commencez par écrire une suite de zéros afin d'effacer tout dessin aléatoire. Recopiez ensuite à cet endroit la table du Player. Un Player utilise toujours 128 ou 256 octets. Si votre table est plus petite, les octets restants seront donc à 0. La position relative de votre table dans cet espace de 128 ou de 256 octets déterminera la position verticale du Player sur l'écran. En pratique, le premier octet de la table sera situé quelques octets ou dizaines d'octets après le début de la zone mémoire réservée au Player.

## DEPLACEMENT VERTICAL

Un déplacement vertical est obtenu en décalant la table dans la zone mémoire du Player. Le principe à utiliser ressemble à celui vu au début de ce chapitre. Alors qu'avons-nous gagné? En fait, si nous y regardons de plus près, il existe une grosse différence : comme tous les octets de la table sont cette fois **consécutifs**, le déplacement ne nécessite plus aucun calcul. Le programme Assembleur consiste à recopier un bloc mémoire à une autre adresse, ce qui est particulièrement simple et rapide. Le long programme vu précédemment devient maintenant :

```
      LDX #$01
LOOP  LDA PLAYER,X
      STA PLAYER-1,X
      INX
      BNE LOOP
```

Ce programme nécessite seulement 4 millisecondes pour déplacer le Player en entier, soit 256 octets, alors qu'auparavant, il fallait une période valant plus du double pour 50 octets seulement. Si l'objet est de petite dimension, on peut réduire le nombre de boucles dans ce programme à ce qui est juste nécessaire. Le Player est alors déplacé facilement en moins de 200 microsecondes. On constate ainsi que déplacer un Player verticalement est une opération particulièrement simple et rapide.

## DEPLACEMENT HORIZONTAL

Le déplacement horizontal est encore plus simple à réaliser qu'un déplacement vertical. Il existe en effet un registre dans le CTIA dans lequel il suffit de placer la position horizontale du Player. Changez la valeur dans ce registre et le Player prend immédiatement sa nouvelle position. Les déplacements horizontaux et verticaux sont indépendants. Vous pouvez les combiner de la manière que vous désirez.

L'unité de déplacement dans un sens horizontal correspond à une impulsion d'horloge. Ainsi, le fait d'ajouter 1 à la valeur contenue dans le registre de position déplacera le Player d'un point élémentaire vers la droite. Rappelons qu'une ligne de balayage se compose de 228 points élémentaires. Toutefois, certains de ces points ne correspondent pas à un affichage sur l'écran, car il faut tenir compte de l'expansion produite volontairement par le téléviseur (afin de faire disparaître les bords disgracieux des images). En conséquence, les points 0 à 47 disparaissent à gauche de l'écran, tandis que les positions situées au-delà de 208 disparaissent à droite. Pour être sûr de faire apparaître un Player sur l'écran, donnez-lui une position horizontale comprise entre 60 et 200. Enfin, n'oubliez pas que la solution la plus simple pour faire disparaître un Player consiste à ramener sa position horizontale à 0, ce qui utilise deux instructions en Assembleur.

## AUTRES POSSIBILITES DES PLAYERS-MISSILES

Bien évidemment, un Player-Missile permet une animation à grande vitesse. Mais les concepteurs des ordinateurs ATARI ne se sont pas arrêtés en si bon chemin et voyons maintenant les autres possibilités de ce système Players-Missiles.

Tout d'abord, il existe 4 Players-Missiles indépendants les uns des autres, chacun possédant sa propre mémoire écran. Ils sont désignés par les mnémoniques P0 à P3. Nous pouvons les positionner les uns à côté des autres pour former un Player de 32 bits de large, chaque constituant pouvant encore se déplacer indépendamment.

Chaque Player possède son propre registre couleur référencé par son nom mnémonique : COLP0 à COLP3. Ainsi, le nombre de couleurs utilisables sans ruse particulière augmente facilement. Chaque registre possède son cache en mémoire vive : PCOLR0 à PCOLR3.

Toutefois, il n'est pas possible de réaliser des Players multicolores sans utiliser les interruptions de Display List.


ATARI INSIDE

Après avoir défini le Player, vous pouvez modifier sa largeur : normal, double ou quadruple, grâce aux registres SIZEP0 à SIZEP3. La définition reste sur 8 bits mais l'affichage s'effectue sur 8, 16 ou 32 points élémentaires. Vous pouvez également choisir entre deux résolutions verticales : un octet génère une ou deux lignes de balayage. Pour une ligne de balayage par octet, la résolution est maximale et la table du Player utilise 256 octets. Pour deux lignes par octet, la table n'occupe plus que 128 octets. Par contre, il faut noter que la résolution verticale est la même pour tous les Players. Ce paramètre est contrôlé par le bit D4 du registre DMACTL.

Pour une résolution d'une ligne de balayage par octet, les 32 premiers octets de la table correspondent à la zone située au-dessus de la bordure standard et les 32 derniers, à la partie inférieure de l'écran. Lorsque la résolution passe par deux lignes par octet, ces valeurs sont à diviser par 2.

## MISSILES

Un autre perfectionnement : les Missiles. A chaque Player est associé un objet de deux bits de large, qu'on appelle Missile. Il utilise la même couleur que le Player auquel il est lié. Les dessins des missiles sont enregistrés en mémoire vive, juste avant la zone attribuée au Player. Les quatre Missiles sont définis dans la même table : quatre missiles à deux bits chacun = un octet de large. Les missiles peuvent être déplacés indépendemment des Players. Ils ont leur propre registre de position horizontale, et leur registre de taille (normale, double ou quadruple largeur) SIZEM. Toutefois, à l'inverse des Players, la taille est la même pour tous les missiles.

Ces objets servent à représenter des balles, des boulets, et... pourquoi pas des missiles. Ils peuvent être regroupés pour former un cinquième Player ; sa couleur est alors déterminée par le registre couleur image n°3. On effectue cette opération en positionnant à 1 le bit D4 du registre de contrôle priorité PRIOR. Notez que les missiles se déplacent encore indépendemment les uns des autres à l'horizontal lorsque cette option est en action. Le couplage ne détermine que leur registre couleur commun.

Vous déplacez verticalement un missile de la même manière qu'un Player : en déplaçant les octets composant le missile à l'intérieur de la mémoire vive correspondante. Cela peut être difficile à réaliser car les missiles sont tous dans la même zone mémoire. Pour accéder à un seul missile sur les quatre, vous devez accéder au bit correspondant en masquant les autres missiles.

## PRIORITES ENTRE LES PLAYERS ET LES AUTRES OBJETS CONSTITUANT L'IMAGE

Une possibilité importante des Players-Missiles réside dans le fait qu'ils sont totalement indépendants du reste de l'écran. En conséquence, vous pouvez les faire apparaître et les utiliser quel que soit le mode graphique. Mais cela pose un problème. Que se passe-t-il si le Player apparaît sur une partie de l'écran qui n'est pas vide (c'est-à-dire qui n'utilise pas seulement la couleur de fond)? Quelle image a la priorité : l'objet fixe ou le Player?

Vous pouvez vous-même déterminer cette priorité. Ainsi, tous les Players peuvent être prioritaires sur les autres objets de l'écran (un objet est le terme imagé que nous utilisons ici pour représenter une partie non vide de l'écran et qui n'est ni un Player, ni un Missile ; un objet utilise donc un ou plusieurs registres couleur standard ; c'est grâce à cela que le CTIA fait la différence entre un objet et le fond de l'écran). Il est ainsi possible de voir passer un Player sur des caractères. Vous pouvez au contraire donner priorité aux registres couleurs standards (sauf celui de la couleur de fond) : COLPF0 à COLPF3 (ou leurs mémoires cache : COLOR0 à COLOR3). Enfin, la priorité peut être différente pour chaque Player : le Player 0 peut ainsi donner l'impression de passer devant (ou sur) les caractères tandis que le Player 2 passera derrière!

Les priorités sont déterminées par le contenu du registre de contrôle PRIOR (adresse 53275) ou de sa mémoire cache GPRIOR (adresse 623).

| ORDRES DE PRIORITE (DU PLUS PRIORITAIRE AU MOINS PRIORITAIRE) | VALEUR DECIMALE |
|---|---|
| Players 0 à 3 — Registres couleurs image 0 à 3 — Fond | 1 |
| Players 0 et 1 — Registres couleurs image 0 à 3 — Players 2 et 3 — Fond | 2 |
| Registres couleurs image 0 à 3 — Players 0 à 3 — Fond | 4 |
| Registres couleurs image 0 et 1 — Players 0 à 3 — Registres couleurs image 2 et 3 - Fond | 8 |

Enfin, il est possible de déterminer une collision entre deux objets. Ceci est fondamental dans les jeux vidéo. Vous pouvez ainsi vérifier si un mobile (Player ou Missile) a rencontré quelque chose d'autre. On peut détecter une collision Missile — Player, Missile — Objet, Player — Player, et Player — Objet. On ne peut pas détecter une collision Missile — Missile. Au total, 54 possibilités de collisions sont détectables et à chacune correspond un bit qui se positionne à 1 si la collision a eu lieu. Ces 54 bits sont répartis en 16 registres dans le CTIA. Seuls les 4 bits de poids faibles dans chaque registre sont utilisés et certains sont sans signification. Ces registres sont accessibles en lecture seulement. On ne peut donc remettre à zéro la détection de collision en écrivant 0 dans ces registres. Il faut écrire pour cela n'importe quelle valeur dans le registre HITCLR (adresse 53278) et tous les registres collision sont alors remis à zéro simultanément. Notez que les registres collision, utilisés en écriture, correspondent à d'autres fonctions du CTIA qui n'ont rien à voir avec les collisions. Deux noms mnémoniques peuvent ainsi être différents pour désigner une même adresse mémoire :

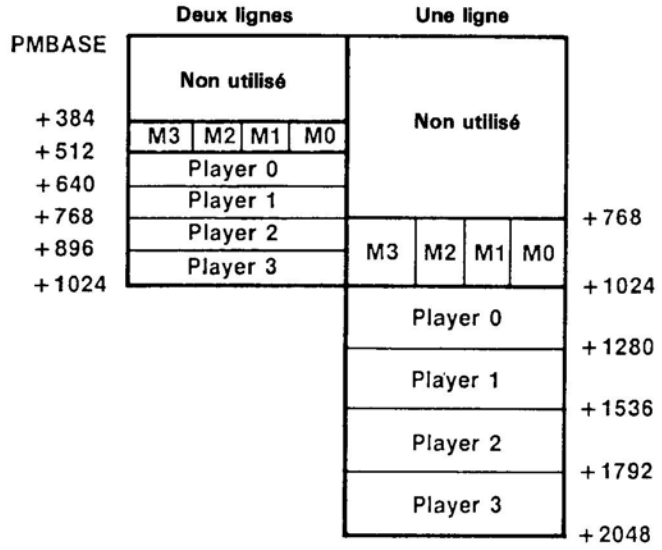

ATARINSIDE

l'un correspond à un cycle d'écriture, l'autre, à un cycle de lecture. Les registres collisions vont de l'adresse 53248 ($D000) à 53263 ($D00F).

En termes «hardware», une collision est détectée lorsque deux registres couleur sont utilisés pour le même point (on ne tient pas compte du registre couleur de fond). Toutefois, le bit de collision ne passe pas à 1 tant que la partie de l'écran montrant la collision n'a pas été dessinée. Cela veut dire qu'une détection de collision n'est significative que lorsqu'il s'est écoulé 16 ms au minimum (1 image vidéo complète) depuis le dernier mouvement du Player. Une astuce consiste à déplacer le Player et à tester les collisions durant une période de Blanking vertical (reportez-vous au chapitre 8). Dans ce cas, on commencera par vérifier la détection de collision, puis on remet à zéro ces registres en écrivant n'importe quoi dans HITCLR. Enfin, on déplacera les Players. Ce sous-programme s'exécutera toujours avec une période d'une image complète puisqu'il est exécuté entre deux images ; la détection de collision sera donc toujours significative.

Un certain nombre d'étapes sont nécessaires pour utiliser les Players-Missiles. Premièrement, vous devez déterminer une zone en mémoire vive que vous réserverez au Player-Missile, et vous l'indiquerez à l'ordinateur en remplissant convenablement (poids fort de l'adresse) le registre PMBASE de l'ANTIC (adresse 54279). Si vous utilisez une résolution d'une ligne (POKE 559,62), cette zone mémoire utilisera 2048 octets. Avec une résolution de 2 lignes (POKE 559,46), elle ne sera que de 1024 octets. Une bonne habitude consiste à placer cette zone mémoire juste avant la mémoire écran.

Voici la cartographie de cette zone :




Le pointeur du début de cette zone est placé comme nous l'avons vu, dans PMBASE. En raisons de certaines limitations internes de l'ANTIC, PMBASE doit recevoir une adresse multiple de 2 ko pour une résolution d'une ligne, ou multiple de 1 ko pour une résolution sur deux lignes. Si vous n'utilisez pas tous les Players et/ou aucun des Missiles, les zones de mémoire correspondantes redeviennent libres pour une autre utilisation (un programme Assembleur, par exemple). Notez qu'il n'est pas possible de déplacer verticalement les Players en modifiant PMBASE.

L'étape suivante consiste à effacer (en la remplissant de 0) la mémoire vive réservée aux Players et aux Missiles que vous allez utiliser. Puis il faut dessiner les Players-Missiles en mémoire.

Enfin, il faut déterminer les paramètres des Players en indiquant la couleur du Player, sa position horizontale et sa largeur. Si cela est nécessaire, déterminez les priorités. Informez l'ANTIC de la résolution verticale voulue en positionnant le bit D4 du registre DMACTL (adresse 54272) ou sa mémoire cache SDMCTL (adresse 559). Pour une résolution sur une ligne, D4 vaut 1. Finalement, donnez l'autorisation à l'ANTIC d'afficher ces Players en positionnant à 1 le bit D2 de DMACTL (pour les Missiles) et D3 (pour les Players). Veillez à ne pas modifier lors de cette opération les autres bits de ce registre. Voici un exemple de programme en Basic permettant la mise en place d'un Player et son déplacement avec un joystick.


ATARINSIDE

```
1   PMBASE=54279:REM                          Pointeur début mémoire des PM
2   RAMTOP=106:REM                            Pointeur fin de la mémoire vive pour le système d'exploitatic
3   SDMCTL=559:REM                            Mémoire cache du registre DMACTL
4   GRACTL=53277:REM                          Registre de commande des graphiques du CTIA
5   HPOSP0=53248:REM                          Position horizontale de P0
6   PCOLR0=704:REM                            Mémoire cache du registre couleur de P0
10  GRAPHICS 0:SETCOLOR 2,0,0:REM             Couleur de fond noire
20  X=100:REM                                 Position horizontale en Basic du Player
30  Y=48:REM                                  Position verticale en Basic du Player
40  A=PEEK(RAMTOP)-8:REM                      Détermine zone mémoire des Players 2 ko en-dessous de fin
50  POKE PMBASE,A:REM                         Indique à l'ANTIC l'emplacement de la mémoire PM  mémoi
60  MYPMBASE=256*A:REM                        Mémorisation de cette adresse
70  POKE SDMCTL,46:REM                        Autorise le DMA sur PM avec une résolution sur 2 lignes
80  POKE GRACTL,3:REM                         Autorise l'affichage des PM
90  POKE HPOSP0,100:REM                       Déclare la position horizontale
100 FOR I=MYPMBASE+512 TO MYPMBASE+640:REM    Efface le Player de la mémoire
110 POKE I,0
120 NEXT I
130 FOR I=MYPMBASE+512+Y TO MYPMBASE+518+Y
140 READ A:REM                                Dessin du Player
150 POKE I,A
160 NEXT I
170 DATA 8,17,35,255,32,16,8
180 POKE PCOLR0,88:REM                        Player en rose
190 A=STICK(0):REM                            Lire Joystick
200 IF A=15 THEN GOTO 190:REM                 Si inactif, recommence
210 IF A=11 THEN X=X-1:POKE HPOSP0,X
220 IF A=7 THEN X=X+1:POKE HPOSP0,X
230 IF A<>13 THEN GOTO 280
240 FOR I=8 TO 0 STEP -1
250 POKE MYPMBASE+512+Y+I,PEEK(MYPMBASE+511+Y+I)
260 NEXT I
270 Y=Y+1
280 IF A<>14 THEN GOTO 190
290 FOR I=0 TO 8
300 POKE MYPMBASE+511+Y+I,PEEK(MYPMBASE+512+Y+I))
310 NEXT I
320 Y=Y-1
330 GOTO 190
```

Une fois que les Players sont affichés à l'écran, ils sont difficiles à effacer. Cela est dû à la procédure suivie pour leur affichage. Tout d'abord, l'ANTIC lit les données définissant les Players-Missiles dans les mémoires vives (si cela lui est autorisé par DMACTL) puis il envoie ces données vers le CTIA en synchronisme avec l'horloge et l'image télé (si celà est autorisé par GRACTL). Le CTIA affiche ensuite le contenu de ses registres Players et Missiles (GRAPF0 à GRAPF3 et GRAFM) chargés par l'ANTIC. De nombreux programmeurs tentent de supprimer les Players-Missiles en mettant à 0 les bits de contrôle des registres DMACTL et GRACTL. Cela n'est pas suffisant, car en opérant ainsi, le programmeur interdit simplement à l'ANTIC de relire le Player-Missile. Mais l'ancienne image subsiste toujours ; au mieux, elle ne se déplace plus verticalement. En conséquence, pour obtenir l'effacement complet des Players-Missiles, il faut effacer les contenus des registres du CTIA. Il est en fait plus simple de laisser le Player actif et de le positionner en dehors de l'écran (position horizontale = 0). Dans cette solution, l'ANTIC continue de relire le dessin du Player en mémoire, ce qui gaspille quand même 70 000 cycles machine par seconde.

## APPLICATIONS DES PLAYERS-MISSILES

Les Players-Missiles autorisent un très grand nombre de possibilités. Bien sûr, ils sont à la base de toute animation, mais ils présentent quelques limitations : il n'existe que quatre Players et chacun n'est défini que par 8 bits de large. Si vous désirez une plus grande largeur, vous pouvez, soit placer plusieurs Players côte à côte, soit revenir à une animation traditionnelle par déplacements en mémoire gérés par le microprocesseur.


ATARINSIDE

Il est possible de shunter l'ANTIC dans son fonctionnement et d'écrire directement les données d'un Player-Missile dans les registres GRAPFn du CTIA. Le programmeur a un contrôle total des Players-Missiles mais au prix d'efforts soutenus. En effet, les registres du CTIA ne sont pas chargés n'importe quand par l'ANTIC : l'écriture dans ces registres doit être rigoureusement synchrone avec les lignes de balayage. Le programmeur doit donc maintenir en mémoire les dessins de ses Players-Missiles et les transférer dans les regsitres de CTIA au moment exact. Le 6502 peut, heureusement, être synchronisé sur l'image TV (voir chapitre 5). Mais quel travail! Pour finalement des améliorations de performance peu sensibles. Le programmeur qui dédaigne la puissance de l'ANTIC doit dépenser beaucoup d'énergie.

Les Players sont également utilisés pour produite des déplacements en pseudo 3D. Il suffit de jouer sur la largeur des Players. On peut ainsi établir une table représentant un Player sur 6 bits de large, et une autre table représentant ce Player sur 8 bits. On affiche ensuite successivement sur l'écran le Player en 6 bits de large, puis on double sa largeur, se qui donne 12 points d'affichage. On passe au Player sur 8 bits en largeur double, soit 16 points d'affichage, et en quadruplant les largeurs, on obtient 24 et 32 points. Le joueur reçoit l'impression que l'objet s'approche de lui. Cette technique a été particulièrement développée dans STAR RAIDERS. Les Zylons sont constitués de deux Players sur 16 bits, les transistions d'une taille à une autre étant ainsi encore plus douces.

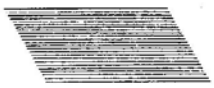
Les Players-Missiles offrent de nombreuses possibilités en dehors de l'animation. Vous pouvez les mettre en œuvre pour augmenter le nombre de couleurs à l'écran : en autorisant quatre couleurs de plus par ligne d'affichage. Bien sûr, la résolution sur 8 bits limite quelque peu leur champ d'application. Mais voici une utilisation possible. Définissez un Player en quadruple largeur et positionnez-le sur l'écran. Puis changez les priorités de manière à ce que le Player ait une priorité inférieure à celle des registres couleurs image. Intervertissez maintenant la couleur dans ces registres avec celle du fond, si bien que la couleur apparente du fond est en fait donnée par un registre couleur image. Le Player disparaît donc derrière cette nouvelle fausse couleur de fond. Maintenant, découpez une fenêtre dans ce faux fond en redessinant avec la couleur du véritable registre fond. Le Player réapparaît dans cette fenêtre et dans cette fenêtre seulement. De cette manière, la résolution horizontale apparente du Player est supérieure à 8 bits. En voici un exemple simple :

```
1   RAMTOP=106:REM          Pointeur du haut de la mémoire vive
2   PMBASE=54279:REM        Registre pointeur ANTIC de la zone mémoire des Players-Missiles
3   SDMCTL=559:REM          Registre cache de DMACTL
4   GRACTL=53277:REM        Registre de contrôle du CTIA
5   HPOSP0=53248:REM        Registre de la position horizontale de P0
6   PCOLR0=704:REM          Registre cache du registre couleur P0
7   SIZEP0=53256:REM        Registre contrôle de la largeur des Players
8   GPRIOR=623:REM          Registre contrôle des priorités
10  GRAPHICS 7
20  SETCOLOR 4,8,4
30  SETCOLOR 2,0,0
40  COLOR 3
50  FOR Y=0 TO 79:REM       Cette boucle remplit l'écran
60  PLOT 0,Y
70  DRAWTO 159,Y
80  NEXT Y
90  A=PEEK(RAMTOP)-20:REM   Réserve pour GR.7
100 POKE PMBASE,A
110 MYPMBASE=256*A
120 POKE SDMCTL,46
130 POKE GRACTL,3
140 POKE HPOSP0,100
150 FOR I=MYPMBASE+512 TO MYPMBASE+640
160 POKE I,255:REM          Les 8 bits de large du Player sont utilisés
170 NEXT I
180 POKE PCOLR0,88
190 POKE SIZEP0,3:REM       Quadruple largeur
200 POKE GPRIOR,4:REM       Mise en place de la priorité
210 COLOR 4
220 FOR Y=30 TO 40
230 PLOT Y+22,Y
240 DRAWTO Y+43,Y
250 NEXT Y
```



ATARINSIDE

Ce programme produit le résultat suivant :

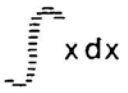


## CARACTERES SPECIAUX

On utilise les Players-Missiles dans la définition de caractères particuliers. Vous pourriez redéfinir le jeu de caractères mais il est parfois plus simple ou plus performant d'utiliser un Player. Ainsi, le symbole d'une intégrale, un indice, ou des curseurs particuliers sont conçus de cette manière. En voici un exemple :

```
1   RAMTOP=106:REM          Pointeur du haut de la mémoire vive
2   PMBASE=54279:REM        Registre pointeur ANTIC de la zone mémoire des Players-Missiles
3   SDMCTL=559:REM          Registre cache de DMACTL
4   GRACTL=53277:REM        Registre de contrôle du CTIA
5   HPOSP0=53248:REM        Registre de la position horizontale de P0
6   PCOLR0=704:REM          Registre cache du registre couleur P0
10  GRAPHICS 0:A=PEEK(RAMTOP)-16:REM   Réserve de place pour résolution sur une ligne
20  POKE PMBASE,A
30  MYPMBASE=256*A
40  POKE SDMCTL,62
50  POKE GRACTL,3
60  POKE HPOSP0,102
70  FOR I=MYPMBASE+1024 TO MYPMBASE+1280
80  POKE I,0
90  NEXT I
100 POKE PCOLR0,140
110 FOR I=0 TO 15
120 READ X
130 POKE MYPMBASE+1100+I,X
140 NEXT I
150 DATA 14,29,24,24,24,24,24,24
160 DATA 24,24,24,24,24,24,184,112
170 ?" ":REM                Efface l'écran
180 POSITION 15,6
190 ?"xdx"
```

Ce programme produit le dessin de l'intégrale :




N'hésitez pas à utiliser les Players-Missiles comme curseur. Grâce à leur déplacement possible point par point ne perturbant pas le contenu de l'écran, ils sont idéaux pour ce genre d'application. Vous pouvez modifier sa couleur en cours de déplacement, et par le jeu des priorités, vous faites apparaître ou non ce qui est sous lui (par exemple un caractère).

En résumé, les Players-Missiles offrent de grandes possibilités au programmeur. Leur première utilité réside dans les jeux d'actions, mais ils peuvent servir dans des cas plus sérieux : amélioration de la résolution d'un dessin, augmentation du nombre de couleur, dessin de caractères spéciaux ou de curseurs (le programme APX «EDITEUR DE PROGRAMMES ET DE TEXTES» livré avec le Macro-Assembleur ATARI en est un bon exemple). Alors, n'hésitez pas à les utiliser.



ATARINSIDE

# 5 - INTERRUPTIONS DE DISPLAY LIST

L'interruption de Display List (DLI) représente un atout considérable pour le programmeur d'un ordinateur ATARI. Bien que d'une mise en œuvre délicate demendant une bonne maîtrise de l'Assembleur et des autres caractéristiques de la machine, les Interruptions de Display List, si elles n'apportent pas en elles-mêmes de nouvelles possibilités, permettent à celles que nous avons vu précédemment de développer toutes leurs puissances.

## PRINCIPE DE FONCTIONNEMENT

Les interruptions de Display List utilisent astucieusement le fait que le balayage TV est de nature séquentielle. Une image TV se construit sur l'écran de haut en bas et un cycle complet dure environ 17000 microsecondes, temps court à l'échelle humaine, mais très long au niveau du microprocesseur. L'ordinateur peut donc facilement modifier les paramètres de l'affichage ou le contenu de la mémoire écran pendant le dessin même de l'image. On atteint ainsi une fréquence des modifications de l'ordre de 50 par secondes. Mais, et c'est là que tout se complique, les changements doivent intervenir au bon moment lorsque l'image se dessine. On pressent donc l'importance de la synchronisation entre le fonctionnement du microprocesseur responsable de ces modifications et le fonctionnement du couple ANTIC/CTIA responsable du tracé de l'image TV. Une solution consisterait à concevoir un programme pour le 6502 s'exécutant rigoureusement 50 fois par secondes (60 fois dans les régions où le secteur est à 60 Hz). Outre le fait que ce n'est pas simple, le microprocesseur ne réaliserait plus que cette tâche! Une meilleur solution consiste à interrompre le 6502 dans son travail ordinaire chaque fois qu'il est temps de modifier quelque chose à l'écran. Le 6502 délaisse son premier travail pour quelques microsecondes, exécute les modifications prévues et reprend sa tâche là où il l'avait laissée. L'interruption qu'il reçoit doit être produite au bon moment, en synchronisme parfait avec le tracé de l'image. Cette interruption particulière est générée par l'ANTIC et le programmeur la provoque en plaçant dans la Display List une instruction spéciale appelée interruption de Display List (DLI).

Le timing et l'exécution de n'importe quel processus d'interruption sont complexes. Cependant, nous allons décrire dans un premier temps la suite des événements produits par une DLI fonctionnant correctement. Le processus commence lorsque l'ANTIC reconnaît l'interruption de Display List, c'est-à-dire lorsque le bit de poids le plus fort (D7) est à 1 dans un octet instruction de la Display List. l'ANTIC attend alors que la dernière ligne de balayage, constituant la ligne de caractères ou de points graphiques en cours d'affichage, soit terminée. Puis l'ANTIC examine le registre NMIEN (adresse 54286) pour voir si les interruptions de Display List sont autorisées. Si le bit D7 est à 0, l'ANTIC ignore l'interruption et continue son travail habituel. Si ce bit est à 1, l'ANTIC valide la ligne NMI du 6502. Puis il poursuit son affichage. Le 6502 suspend la tâche en cours, et passe, via le vecteur NMI, au sous-programme de traitement des interruptions du système d'exploitation. Ce programme établit l'origine de l'interruption NMI (touche RESET, DLI ou Blanking vertical). Dans le cas d'une DLI, le programme passe, via le vecteur situé en mémoire vive aux adresses $200 et $201 (poids faible, poids fort), au programme de gestion de la DLI écrit par le programmeur. Le microprocesseur transforme alors les paramètres voulus (registres couleurs, positions de Players-Missiles, jeux de caractères, etc) puis l'instruction RTI le ramène à son travail de départ.

La mise en place d'une DLI se décompose en plusieurs étapes. Tout d'abord, vous devez écrire le programme en Assembleur qui s'exécutera lors de la DLI. Ce programme doit commencer par placer dans la pile tous les registres du 6502 qui seront utilisés par le programme car la routine d'interruption du système d'exploitation n'effectue pas cette opération (le 6502 empile automatiquement le registre d'état lors d'une interruption). Le programme doit être court et rapide. Il devrait en principe modifier seulement quelques registres. Il doit se terminer en dépilant dans le bon ordre tous les registres du 6502.

Dans un deuxième temps, vous placez ce programme en mémoire. La page 6, toujours disponible pour l'utilisateur, est idéale pour cela. Placez ensuite l'adresse de début de ce programme aux adresses $200 et $201. Déterminez verticalement le point sur l'écran où doit s'exécuter la DLI, trouvez l'octet correspondant dans la Display List et positionnez le bit D7 à 1. Enfin, autorisez la reconnaissance de la DLI par l'ANTIC en mettant à 1 le bit 7 du registre NMIEN ($D40E). L'interruption sera validée dès cet instant.

## TEMPS D'EXECUTION D'UNE DLI

Comme dans tout programme de gestion d'interruption, des considérations de temps doivent être prises en compte car certaines phases peuvent devenir critiques. L'ANTIC n'envoie pas l'interruption au 6502 immédiatement après avoir lu l'instruction d'interruption. Un premier retard est dû à l'attente, par l'ANTIC, de la fin de la dernière ligne de balayage appartenant à la ligne de points graphiques en cours d'affichage. Cette ligne graphique (ligne de caractères ou ligne de points graphiques) correspond à l'octet de la Display List contenant la DLI. Il y a donc décalage (et retard) entre la DLI et son effet à l'écran. Il faut également tenir compte du sous-programme inclu dans le système d'exploitation et gérant les interruptions. Ainsi, lorsque votre programme de DLI commencera à s'exécuter, le faisceau d'électrons aura



ATARI INSIDE

commencé le tracé de la ligne de balayage suivante. Si votre programme modifie un registre couleur, vous pourriez voir l'ancienne couleur à gauche sur la ligne, la nouvelle à droite. Toutefois, en raison du temps de réponse variable du 6502 à une interruption, la frontière entre ces deux couleurs ne sera pas précise, ce qui ne manquera pas de produire un certain flou à cet endroit.

Il existe une solution à ce problème. Elle est fourni par l'utilisation du registre WSYNC (attente pour synchronisation horizontale). Chaque fois que ce registre est adressé d'une manière quelconque (lecture ou écriture), l'ANTIC passe à 0 la ligne READY du 6502. Celui-ci suspend alors immédiatement toute activité jusqu'à ce que l'ANTIC libère la ligne READY, ce qui se produit à la fin de chaque ligne de balayage horizontal. Cela revient à bloquer le 6502 jusqu'à ce que le faisceau d'électrons atteigne le côté droit de l'écran. Si vous insérez une instruction STA WSYNC juste avant l'instruction qui modifiera la valeur placée dans le registre couleur, la couleur changera pendant que le faisceau, éteint, reviendra sur la gauche de l'écran. Notez que le changement de couleur aura lieu une ligne de balayage plus bas sur l'écran, mais il sera net et propre. Compte tenu du fait que la DLI s'exécute après que la ligne d'affichage correspondante soit terminée, vous ne pourrez pas modifier la couleur dans un registre avant la fin de la première ligne d'affichage (car le système d'exploitation commence toujours une ligne en plaçant dans les registres couleurs le contenu des mémoires cache).

En résumé, un programme de DLI doit d'abord sauvegarder les registres du 6502 dans la pile, puis charger ces registres avec les valeurs qu'il faudra placer dans les registres couleur. Le 6502 exécutera une instruction STA WSYNC puis place les nouvelles valeurs dans les registres appropriés de l'ANTIC ou du CTIA. Finalement, il faut dépiler et exécuter une instruction RTI. Cette procédure garantira une modification des registres couleurs au début de la ligne voulue, alors que le faisceau d'électrons est encore invisible sur l'écran.

## EXEMPLE DE DLI

Voici un exemple simple d'utilisation de DLI :

```
10 DLIST=PEEK(560)+256*PEEK(561):REM   Trouve où commence la Display List
20 POKE DLIST+15,130:REM               Insère une DLI
30 FOR I=0 TO 19:REM                   Mise en place du programme de DLI
40 READ A:POKE 1536+I,A:NEXT I
50 DATA 72,138,72,169,80,162,88
60 DATA 141,10,212,141,23,208
70 DATA 141,24,208,104,170,104,64
80 POKE 512,0:POKE 513,6:REM           Chargement du vecteur d'interruption avec l'adresse de début
                                       du programme
90 POKE 54286,192:REM                  Autorise la DLI
```

Les lignes 50, 60 et 70 représentent le programme Assembleur suivant :

```
PHA          Sauvegarde de l'accumulateur
TXA
PHA          Sauvegarde du registre X
LDA #$50     Luminance foncée pour les caractères
LDX #$58     Rose
STA WSYNC    Attente
STA COLPF1   Mise en place
STX COLPF2   des couleurs
PLA
TAX
PLA          Dépile
RTI          Et fin
```

C'est un programme très simple modifiant la couleur du fond de l'écran de bleu à rose. Pour garder une bonne lisibilité, la luminance des caractères est modifiée également. Mais vous pouvez vous demander pourquoi la couleur de l'écran reste bleue dans le haut de l'image. Nous avons vu précédemment qu'à la fin de chaque image, le système d'exploitation replace les valeurs des mémoires cache dans les registres couleur. Le bleu réapparaît donc à chaque image dans le haut de l'écran. Si vous désirez que le haut de l'écran soit vert, changez simplement le coutenu du registre couleur. Notez que grâce aux DLI, vous pouvez modifier plusieurs fois les couleurs par image en modifiant directement les registres hardware. Par contre, vous ne pourrez jamais modifier la première ligne d'affichage dans le haut de l'écran par une DLI.


ATARINSIDE

# MODE ATTENTE

En modifiant le contenu des registres harware, vous supprimez le mode attente. Rappelons que ce mode est l'une des possibilités fournie par le système d'exploitation. Après environ une dizaine de minutes sans manipulation du clavier, les teintes sur l'écran changent de couleur toutes les 10 secondes environ avec une luminosité réduite. Cette astuce évite de marquer toujours les mêmes luminophores du tube cathodique par une image fixe. Vous ne risquez donc pas d'abîmer l'écran de votre téléviseur. Il est facile d'intégrer ce mode attente dans les DLI. Il suffit d'ajouter simplement 2 lignes en Assembleur :

| Ancienne version | Nouvelle version |
|---|---|
| LDA NEWCOL | LDA NEWCOL |
| STA WSYNC | EOR COLRSH |
| STA COLPF2 | AND DRKMSK |
| | STA WSYNC |
| | STA COLPF2 |

DRKMSK et COLRSH sont des adresses en page zéro ($4E et $4F) gérées par le système d'exploitation durant chaque interruption de Blanking vertical. Tant que le mode attente n'est pas actif, COLRSH vaut 0 et DRKMSK vaut FF. Lorsque le mode attente devient actif, COLRSH reçoit une valeur aléatoire toutes les 4 secondes et DRKMSK reçoit $F6. COLRSH modifie la couleur et DRKMSK masque le poids fort de la luminance.

# RETOUR SUR LE TEMPS D'EXECUTION

La mise en place des deux nouvelles lignes vues au paragraphe précédent complique encore un peu plus un problème délicat à résoudre : la rapidité d'exécution d'une DLI. Le traitement se divise en 3 phases : la première couvre la période allant du début du programme de DLI jusqu'à l'instruction STA WSYNC. Pendant ce temps, le faisceau d'électrons trace la dernière ligne de balayage de la ligne d'affichage en cours. La phase 2 commence à STA WSYNC et se termine lorsque le faisceau commence le tracé de la ligne suivante. C'est la période de Blanking horizontal pendant laquelle les modifications sur les registres doivent avoir lieu. La phase 3 commence avec le balayage sur l'écran et se termine lors de l'exécution de l'instruction RTI. Son exécution n'est pas critique.

Une ligne de balayage horizontal dure pendant 114 cycles horloge du microprocesseur. Une interruption de DLI atteint le 6502 durant le 8e cycle. Le 6502 a besoin de huit à 14 cycles pour répondre à l'interruption. Le sous-programme du système d'exploitation dure sur 11 cycles machine. Pendant ce temps, la mémoire vive (dynamique) doit être rafraîchie, ce qui prendra 1 à 3 cycles supplémentaires. Le sous-programme de DLI commencera donc 28 à 36 cycles après le début de la ligne de balayage. Conservons le cas le plus extrême : 36 cycles. D'autre part, pour obtenir une synchronisation correcte, l'instruction STA WSYNC doit être atteinte au plus tard au 100e cycle : cela réduit notre temps pour la phase 1 de 14 cycles. Finalement, l'ANTIC utilisera quelques cycles pour rafraîchir la mémoire : 9 environ. Il reste donc au maximum 55 cycles disponibles en phase 1. Mais cela n'est vrai que pour des lignes de balayage vides. Sinon, il faut compter 1 cycle utilisé par l'ANTIC pour chaque octet lu dans la mémoire écran. Dans les modes 0, 7 et 8, il faut 40 octets par ligne. Il ne reste alors dans ces modes graphiques plus que 15 cycles libres pour la phase 1.

La phase 2, critique, dure 27 cycles. Comme toujours, quelques cycles sont perdus pour le rafraîchissement et l'accès en DMA. Les Players-Missiles, s'ils sont utilisés, nécessiteront 5 cycles supplémentaires. Une instruction de Display List utilise 1 cycle. Si l'option LMS est utilisée, 2 cycles supplémentaires seront nécessaires pour lire l'opérande. Finalement, 17 à 26 cycles machine sont disponibles pour la phase 2.

Le problème de temps devient évident. Pour charger une couleur dans un registre en tenant compte du mode attente, 14 cycles sont nécessaires. Sauvegarder les registres A, X et Y et manipuler 3 registres couleur prendra 47 cycles. En conséquence, le programmeur désirant effectuer de nombreuses opérations durant une DLI rencontrera de nombreuses difficultés. S'il s'agit par contre de modifier un seul registre, tout ira bien.

Pour le programmeur exigeant en temps, il n'existe pas de recette miracle. Toutefois, on peut légèrement empiéter sur le début de la phase 3, car le faisceau d'électrons n'est pas encore arrivé sur la partie visible de l'écran. De même, une couleur qui n'est utilisée que sur la fin de la ligne de balayage pourra être changée alors que le spot est déjà sur l'écran en début de ligne. Une autre solution consiste à couper une longue DLI en deux parties plus petites. On peut également insérer une ligne vide à l'écran (cette ligne aura la couleur de fond) avec le bit de DLI positionné à 1 pour allonger au maximum la phase 1 (on gagne jusqu'à 40 cycles!).

Une autre solution consiste à reporter le traitement du mode attente dans les périodes de Blanking vertical. Pour cela, il faut placer 2 tables de couleur en mémoire vive : la première contient les valeurs utilisées par les DLI, la seconde mémorise les valeurs de la première, modifiées éventuellement par le mode attente. Durant la période de Blanking vertical, le programmeur transfère les couleurs de la deuxième table vers la première. Les programmes de gestion de DLI sont ainsi allégés.



ATARINSIDE

# DLI MULTIPLES

Il est souvent désirable de mettre en place plusieurs DLI sur un même écran. Malheureusement, il existe un seul vecteur d'indirection vers la routine de traitement de l'interruption. Le programme de DLI doit donc savoir à quelle instruction il est en train de répondre : est-ce la première, la deuxième, la nième de l'image?

Si toutes les DLI effectuent les mêmes opérations avec des valeurs différentes, il suffit d'organiser les données en tables et de gérer un compteur/pointeur qui décrira cette table. La dernière valeur du pointeur correspondra à l'exécution de la dernière DLI, et un test sur cette valeur permettra la remise à zéro du pointeur. En voici un exemple :

```
        PHA
        TXA
        PHA
        INC COUNTR
        LDX COUNTR
        LDA COLTAB,X      Utilise la page 2 en table de couleur
        STA WSYNC         Attente pour synchronisation
        STA COLBAK
        CPX #$4F          Dernière ligne?
        BNE ENDDLI        Non, fin
        LDA #$00          Oui, RAZ compteur
        STA COUNTR
ENDDLI  PLA
        TAX               Dépile X
        PLA               Dépile l'accumulateur
        RTI               Fin d'interruption
```

Le programme Basic appelant ce programme Assembleur peut s'écrire :

```
10 GRAPHICS 7
20 DLIST=PEEK(560)+256*PEEK(561):REM    Trouve l'adresse de la Display List
30 FOR J=6 TO 84:REM                    Mise en place d'une DLI à chaque ligne d'affichage
40 POKE DLIST+J,141:REM                 En Basic mode 7
50 NEXT J
60 FOR J=0 TO 30
70 READ A:POKE 1536+J,A:NEXTJ:REM       Mise en place du programme de DLI
80 DATA 72,138,72,238,32,6,175,32,6
90 DATA 189,0,240,141,10,212,141,26,208
100 DATA 224,79,208,5,169,0
110 DATA 141,32,6,104,170,104,64
120 POKE 512,0:POKE 513,6:REM           Initialisation du vecteur
130 POKE 54286,192:REM                  Autorisation des DLI
```

Ce programme affichera 80 couleurs différentes sur l'écran.

Mais d'autres solutions sont également possibles. L'une d'entre-elles consiste à utiliser un compteur d'interruption de Display List pour effectuer un branchement au sous-programme adéquat (ceci est équivalent à l'instruction Basic ON X GOTO ...). Par contre, cette manière d'opérer ralentit toutes les DLI. Une autre solution consiste à écrire à la fin de chaque DLI, l'adresse de la suivante en $200 et $201. Vous pouvez effectuer cette opération durant la phase 3 sans aucun problème, ce qui présente l'avantage considérable de ne pas allourdir ou rallonger les tâches des phases 1 et 2. Il faut simplement veiller à une chose : lorsque vous autorisez à l'ANTIC la reconnaissance des DLI, le vecteur en $200, $201 doit pointer vers le premier programme de DLI, et l'autorisation doit être donnée durant une période de Blanking vertical ; sinon, il n'y aura pas correspondance entre le numéro d'ordre du programme de la DLI et le numéro d'ordre de la DLI elle-même. Si la première DLI n'est pas critique en temps, vous pouvez placer dans son programme une boucle venant lire l'état du registre VCOUNT (adresse 54283) qui indique le numéro de la ligne de balayage en cours. La boucle se termine lorsque VCOUNT atteint la valeur correspondant à la dernière ligne de balayage de l'interruption de Display List. Dès ce moment, il y a correspondance définitive entre les interruptions et leur programme de gestion. Il n'est plus nécessaire d'utiliser VCOUNT pour les images suivantes, la boucle correspondante peut donc être sautée.



ATARINSIDE

Le «bip» généré par le système d'exploitation chaque fois que l'on enfonce une touche au clavier interfère avec le déroulement des DLI. En effet, la durée de ce bip est produite par plusieurs instructions STA WSYNC. Cela peut perturber le déroulement des programmes de DLI, provoquant une modification des couleurs une ou deux lignes de balayage plus bas que prévu, et ce, pendant une fraction de seconde. Il n'existe pas de solution simple à ce problème, si ce n'est qu'on le méprise avec dédain. Sinon, on peut à nouveau lire VCOUNT et n'autoriser au programme de s'exécuter que si VCOUNT présente la bonne valeur. On peut aussi déconnecter le programme de gestion du clavier et réaliser son propre programme à la place. Ce peut être un travail fort long. La dernière solution consiste à ne pas utiliser le clavier en masquant convenablement les interruptions (bit 6 à 0 aux adresses 16 et 53774).

# PROGRAMME D'AFFICHAGE

Les DLI ont été conçus pour remplacer une technique logicielle et matérielle plus primitive : le programme d'affichage, c'est-à-dire une boucle exécutée par le 6502, synchronisée précisément sur le cycle d'affichage d'un téléviseur. En utilisant le registre VCOUNT et en consultant une table contenant l'ensemble des modifications à apporter sur l'écran en fonction des valeurs de VCOUNT, le 6502 peut facilement contrôler toutes les valeurs graphiques d'un écran complet. Par contre, il faut payer très cher cette puissance : le 6502 n'est plus disponible pour effectuer des calculs pendant le temps d'affichage, soit pendant 75% du temps. En outre, aucun calcul ne doit durer plus de 4000 cycles machines, temps disponible dans une période de Blanking vertical. Cette restriction sous-entend qu'un programme d'affichage ne peut être exploité qu'avec des programmes nécessitant peu ou pas de calcul. Par exemple, le programme de jeu ATARI Basket Ball utilise un programme d'affichage : il nécessite peu de calculs mais de nombreuses couleurs. Les Players multicolores de ce jeu ne peuvent être réalisés en utilisant les DLI car celles-ci sont synchronisées sur les lignes de balayage composant l'image de fond, et non sur les positions des Players.

Il est possible d'utiliser un programme d'affichage pour une seule ligne de balayage et de changer ainsi les registres couleur durant le tracé de la ligne. De cette manière, on peut afficher plusieurs couleurs sur l'écran en modifiant un seul registre. La position horizontale du changement de couleur est déterminée par le temps écoulé entre le début de la ligne (et donc le début du programme) et l'instruction réalisant le changement de couleur). Ainsi, en comptant soigneusement le nombre de cycles machines, le programmeur positionne de nouveaux graphiques sur l'écran. Malheureusement, ce procédé est délicat à mettre en œuvre en pratique. En raison de l'accès mémoire en DMA par l'ANTIC, il est pratiquement impossible de savoir combien de cycles se sont écoulés ; un simple comptage ne suffit pas. Si le DMA de l'ANTIC est hors service, le 6502 peut assurer un contrôle total de l'affichage mais il doit alors réaliser toutes les fonctions habituelles de l'ANTIC. Pour cette raison, un programme d'affichage reste très difficile à mettre en œuvre. Toutefois, si les deux objets de couleurs différentes sont suffisamment séparés, disons par plus de 20 cycles couleur, cette technique peut devenir exploitable.

# APPLICATIONS DES INTERRUPTIONS DE DISPLAY LIST

Vous avez pu vous rendre compte à la lecture des paragraphes précédents, de toute la puissance que l'on peut obtenir des interrutions de Display List pour modifier le contenu des registres. Vous mettez en place de nombreuses couleurs, des graphiques et des effets spéciaux. Chaque registre peut être modifié à chaque interruption. Cela s'applique aussi bien aux registres couleurs images qu'aux registres des Players-Missiles. Vous bénéficiez de 9 registres couleurs, chacun affichant jusqu'à 128 couleurs. Est-ce assez? Bien sûr, un programme classique n'exploitera pas toutes ces couleurs. Trop de DLI finissent par ralentir le programme tout entier. Quelquefois, le dessin même de l'image s'accomode mal de DLI. En pratique, il est simple de faire apparaître 12 couleurs ; il faut bien étudier le programme pour 24 couleurs ; aller au-delà oblige le programmeur à certaines concessions.

Les interruptions de Display List servent dans bien d'autres domaines, comme par exemple pour augmenter la puissance des Players-Missiles. La position horizontale d'un Player peut être modifiée par une DLI. De cette manière, un Player peut être repositionné sur l'écran, ce qui lui donne plusieurs incarnations. Si vous interprêtez un Player comme une colonne verticale composée d'images, une DLI devient une paire de ciseaux coupant la colonne en plusieurs éléments et les répartissant sur l'écran. Bien sûr, il n'est pas possible de placer deux parties du Player sur la même ligne horizontale. Si votre affichage exploite des objets graphiques n'apparaissant jamais sur la même ligne horizontale, un seul Player peut suffire.

Une autre utilisation des DLI consiste à modifier la largeur des Players ou leur priorité. Utilisez pour cela un masque sur le registre de priorité afin de changer seulement les bits nécessaires (voir chapitre 4).

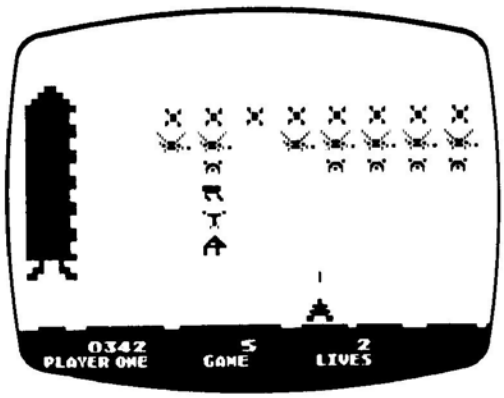
La dernière application des DLI est de modifier le jeu de caractères durant l'affichage de l'écran. Vous utilisez par exemple l'anglaise pour un titre dans le haut de l'écran, et des caractères bâtons dans la partie inférieure. Manipulez aussi le bit d'effet miroir pour inverser (tête en bas) des lignes de caractères.

Une utilisation correcte des DLI repose sur une étude soignée et détaillée de l'affichage écran. Le concepteur doit examiner l'architecture verticale de l'image. Le balayage TV n'est pas symétrique dans les deux dimensions : le tracé horizontal est nettement plus rapide (262 fois) que le tracé vertical. Les ordinateurs ATARI ont été conçus en tenant compte de ces dissymétries. N'oubliez pas qu'une image TV n'est pas une feuille de papier, plane et blanche, sur laquelle vous dessinez. Elle est constituée en fait de fines lignes horizontales, chacune composée de plusieurs paramètres. Le programmeur recherchant à tout prix une image isotropique perd de nombreuses possibilités. Vous obtiendrez des résultats optimum en organisant vos informations verticalement sur l'écran. Vous pourrez alors pleinement exploiter les DLI.

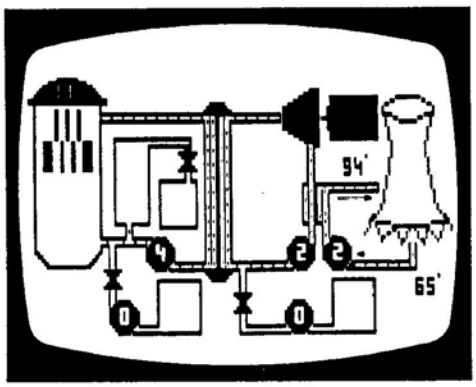
- La page suivante vous montre quelques écrans de divers programmes et vous donne des estimations du degré d'architecture verticale utilisée.



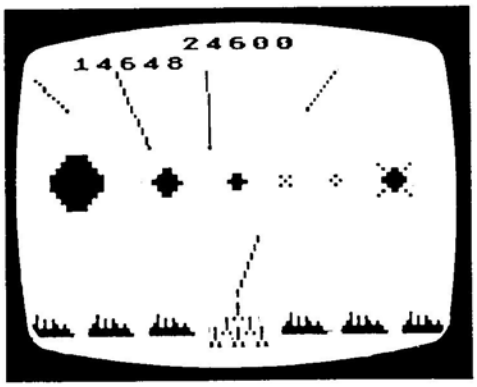
ATARI INSIDE



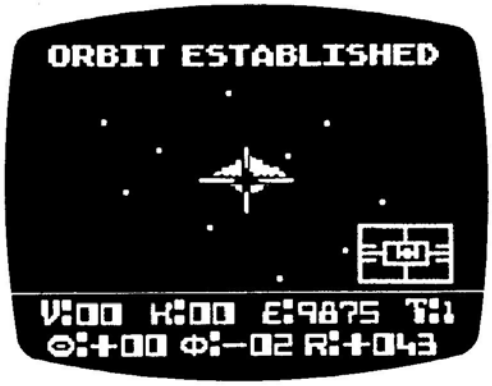

SPACE INVADERS
(Marque déposée de Taito America Corporation)
***BEAUCOUP***



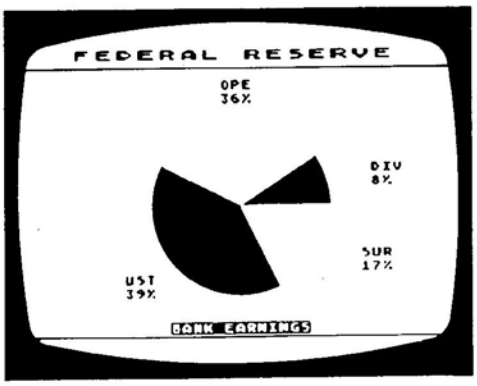

CENTRALE NUCLEAIRE
(Une simulation d'un réacteur nucléaire)
***PEU***



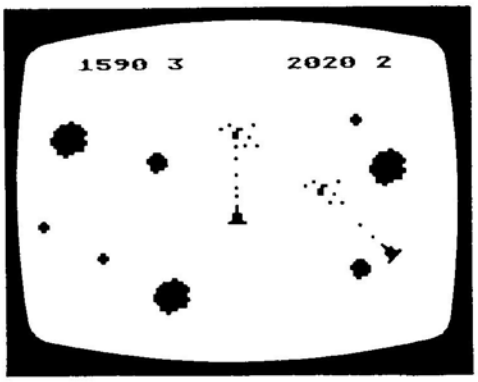

MISSILE COMMAND
***QUELQUES UNS***




STAR RAIDERS
***PEU***




GRAPHES
***AUCUN***




ASTEROIDS
***AUCUN***



ATARINSIDE

# 6 - DEFILEMENTS ET DEPLACEMENTS

Bien souvent, l'ensemble des informations qu'un programmeur veut afficher dépasse les possibilités d'un écran. Une solution consiste à faire défiler les informations à l'écran. Par exemple, le listing d'un programme Basic se déroule du bas vers le haut de l'écran. Tous les ordinateurs personnels utilisent ce genre de défilement. Mais les ordinateurs ATARI possèdent deux possibilités complémentaires de défilement très intéressantes. Le premier est le défilement «grossier» obtenu par l'instruction LMS ; le second est le défilement «fin».

Les ordinateurs standards utilisent un défilement grossier, c'est-à-dire un déplacement du texte caractère par caractère, en déplaçant les octets dans la mémoire écran. Le résultat donne une image sautillante qui peut devenir fatigante. De plus, il est obtenu en jouant sur des centaines, voire des milliers d'octets, ce qui constitue une tâche lente et fastidieuse.

Quelques ordinateurs personnels donnent un déplacement plus fin en dessinant une image en haute résolution puis en déplaçant cette image dans la mémoire écran. Bien qu'une résolution supérieure dans le défilement soit atteinte, l'image utilise un nombre supérieur d'octets ; le transfert de cette zone mémoire est plus long et le programme principal est ralenti. Le problème fondamental réside dans le déplacement des données de la mémoire écran : quel travail!

Les ordinateurs ATARI 400, 600 et 800 fonctionnent selon un meilleur principe : déplacer la mémoire écran autour des données. Les instructions LMS de la Display List permettent en effet la modification du pointeur d'adresse de l'ANTIC vers la mémoire écran. L'instruction LMS indique à l'ANTIC où se situe la mémoire écran (voir chapitre 2). Une Display List standard possède une seule instruction LMS placée à son début. La zone mémoire ainsi désignée fournie une suite d'octets dessinant l'écran ligne par ligne. En manipulant les deux octets constituant l'opérande de l'instruction LMS, un défilement primitif est obtenu. En effet, cela déplace la mémoire affichée à l'écran par dessus les données. Ainsi, en manipulant seulement 2 octets, vous produisez un effet identique à un déplacement de toute la mémoire vive. En voici un exemple :

```
10 DLIST=PEEK(560)+256*PEEK(561):REM   Trouve la Display List
20 LMSLOW=DLIST+4:REM                 Trouve l'adresse de poids faible de l'opérande de LMS
30 LMSHIGH=DLIST+5:REM                Trouve l'adresse de poids fort de l'opérande de LMS
40 FOR I=0 TO 255:REM                 Boucle extérieure
50 POKE LMSHIGH,I
60 FOR J=0 TO 255:REM                 Boucle intérieure
70 POKE LMSLOW,J
80 FOR Y=1 TO 50:NEXT Y:REM           Boucle de temporisation
90 NEXT J
100 NEXT I
```

Ce programme déplace la mémoire écran sur tout l'espace adressable de l'ordinateur. L'écran est donc successivement composé de toutes les données placées dans l'ordinateur. Vous remarquerez que le défilement obtenu est à la fois horizontal et vertical. Un défilement vertical pur est obtenu en ajoutant ou en soustrayant un nombre fixe (la longueur d'une ligne en octet) à l'opérande de LMS. Voici ce que cela donne :

```
10 GRAPHICS 0
20 DLIST=PEEK(560)+256*PEEK(561)
30 LMSLOW=DLIST+4
40 LMSHIGH=DLIST+5
50 SCREENLOW=0
60 SCREENHIGH=0
70 SCREENLOW=SCREENLOW+40:REM          Ligne suivante
80 IF SCREENLOW<256 THEN GOTO 120:REM  Dépassement?
90 SCREENLOW=SCREENLOW-256:REM         Oui ; ajuste le pointeur
100 SCREENHIGH=SCREENHIGH+1
110 IF SCREENHIGH=256 THEN END
120 POKE LMSLOW,SCREENLOW
130 POKE LMSHIGH,SCREENHIGH
140 GOTO 70
```

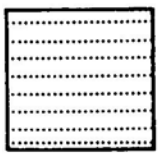


ATARINSIDE

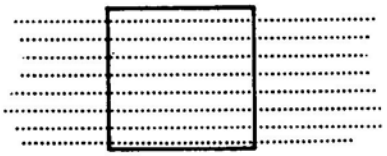
# DEFILEMENT HORIZONTAL

Un défilement purement horizontal n'est pas aussi simple à réaliser qu'un défilement purement vertical. Le problème est que la mémoire écran est organisée d'une manière sérielle. Les octets de la mémoire écran sont chaînés, ceux d'une ligne suivant immédiatement ceux de la ligne précédente. Vous pouvez réaliser un défilement horizontal en décalant tous les octets vers la gauche. Cela s'obtient en décrémentant l'opérande de l'instruction LMS. Toutefois, l'octet disparaissant à gauche d'une ligne se retrouvera à l'extrême droite de la ligne précédente.

La solution consiste à étendre la mémoire écran et à la diviser en une suite de zones indépendantes correspondant aux lignes d'affichage. Le dessin suivant illustre cette idée.



Arrangement standard



Arrangement pour un défilement horizontal

A gauche, vous voyez l'arrangement normal de la mémoire écran. La mémoire, adressée en série, donne une représentation unidimensionnelle de l'écran. A droite, voilà l'arrangement dont nous avons besoin pour un défilement horizontal. La mémoire est toujours unidimensionnelle, mais elle est utilisée différemment : pour chaque ligne, la mémoire utilise davantage d'octets que ce qui est réellement utilisé pour l'écran. Ce n'est pas un accident ; c'est normal puisque le rôle du défilement est d'afficher plus d'informations qu'il n'est possible ; il faut donc bien augmenter la taille de la mémoire écran. Avec cet arrangement, vous pouvez mettre en place un véritable défilement horizontal, sans subir le défilement vertical de l'approche précédente.

Pour implémenter un défilement horizontal pur, la première étape consiste à déterminer la longueur maximale d'une ligne et d'allouer la mémoire en conséquence. Puis vous devez écrire une Display List nouvelle, avec une instruction LMS sur chaque ligne d'affichage. Cette Display List sera plus longue que d'habitude, mais elle ne posera pas de problème pour son écriture. Quelles valeurs devez-vous donner aux opérandes des instructions LMS? Tout simplement l'adresse du premier octet de chaque ligne horizontale d'écran. Une fois que la Display List est en place, il faut donner à l'ANTIC son adresse de départ, puis mettre en place les données devant être affichées. Pour réaliser un défilement, chaque opérande de chaque instruction LMS dans la Display List doit être incrémenté pour un défilement vers la droite, ou décrémenté pour un défilement vers la gauche. La logique du programme doit vérifier que l'image ne sort pas des limites allouées. Sinon, vous verrez apparaître n'importe quoi. N'oubliez pas qu'un opérande LMS pointe le premier octet correspondant au premier caractère affiché à gauche sur l'écran, dans la ligne considérée. La valeur maximale de l'opérande pour une ligne donnée est déterminée par l'adresse du dernier octet de cette ligne moins le nombre d'octets composant une ligne à l'écran. N'oubliez pas non plus que l'opérande ne peut pas passer par une valeur multiple de 4 ko, sinon l'affichage donnera n'importe quoi.

Comme ce processus est un peu délicat, prenons un exemple. Tout d'abord, nous devons choisir une longueur maximale pour une ligne. Prenons 256 octets, ce qui simplifiera le calcul des adresses. Chaque ligne horizontale utilisera donc une page en mémoire vive. Nous emploierons le mode Basic 2, ce qui donne 12 lignes d'affichage. La taille mémoire utilisée atteint 3 ko. Dans un but de simplicité (et pour être sûr de ne pas avoir un affichage vide dans cet exemple) nous utiliserons les 3 premiers ko de l'espace adressable du 6502 (ce qu'il ne faut pas faire normalement car cette zone est utilisée en partie par le système d'exploitation, le DOS, etc). Nous positionnerons la Display List en page 6 ; la Display List elle-même apparaîtra donc à l'écran. Les valeurs initiales pour les opérandes des instructions LMS seront faciles à calculer : les poids faibles seront toujours nuls, les poids forts seront (dans cet ordre) 0, 1, 2, etc. Le programme suivant effectue toutes ces opérations et réalise un défilement horizontal :

```
10 REM first set up the display list
20 POKE 1536,112:REM
30 POKE 1537,112:REM
40 POKE 1538,112:REM
50 FOR I=1 TO 12:REM
60 POKE 1536+3*I,71:REM
70 POKE 1536+3*I+1,0:REM
80 POKE 1536+3*I+2,I:REM
90 NEXT I
```

8 lignes vides
8 lignes vides
8 lignes vides
Boucle pour créer la Display List
Avec 12 lignes en mode Basic 2 Avec l'instruction LMS à chaqu
Poids faible de l'opérande ligne
Poids fort de l'opérande



ATARINSIDE

```
100 POKE 1575,65:REM                    Instruction JVB de l'ANTIC
110 POKE 1576,0:REM                     La Display List commence en $0600
120 POKE 1577,6
130 REM tell ANTIC where display list is
140 POKE 560,0
150 POKE 561,6
160 REM now scroll horizontally
170 FOR I=0 TO 235:REM                  Boucle sur les poids faibles
175 REM Nous utilisons 235 - et pas 255 - car la largeur de l'écran est de 20 caractères
180 FOR J=1 TO 12:REM                   pour chaque ligne d'affichage
190 POKE 1536+3*J+1,I:REM               Mise en place des nouveaux poids faibles
200 NEXT J
210 NEXT I
220 GOTO 170:REM                        Boucle sans fin
```

Les données défilent de la droite vers la gauche. Quand la fin de la mémoire est atteinte, le processus recommence tout simplement. La Display List est affichée sur la sixième ligne en partant du haut de l'écran. Elle apparaît comme une suite de guillemets.

L'étape suivante consiste à combiner les deux sens de défilement pour obtenir un déplacement en diagonale. Le défilement horizontal est obtenu en ajoutant ±1 aux opérandes des instructions LMS. Le défilement vertical est obtenu en ajoutant ±L à ces mêmes opérandes (L étant la longueur en octets d'une ligne horizontale complète, soit 256 dans notre exemple précédent). Par exemple, si la longueur est 256 et que nous désirions un défilement vers le bas et vers la droite, nous devons ajouter 256 + (−1) = 255 à chaque opérande de chaque instruction LMS. C'est une addition sur 2 octets avec retenue ; le programme Basic précédent évite cette difficulté qui n'en est plus une lorsqu'on travaille en Assembleur.

Toutes sortes de combinaisons sont possibles en manipulant différemment les octets des opérandes. Les lignes peuvent défiler à des vitesses différentes les unes par rapport aux autres et cela en jouant seulement sur quelques octets. Bien sûr, cela peut être atteint avec un affichage conventionnel mais le gros avantage des instructions LMS réside dans la vitesse d'exécution : plutôt que de manipuler des milliers d'octets, changez seulement quelques pointeurs.

## DEFILEMENT FIN

La seconde possibilité intéressante de défilement pour les ordinateurs ATARI réside dans le défilement fin. Nous appelons défilement fin un déplacement de l'écran s'effectuant point élémentaire par point élémentaire. Un défilement grossier procède par bonds de la taille d'un caractère tandis qu'un défilement fin procède par bonds d'une ligne de balayage verticalement, d'une période d'horloge couleur horizontalement. Pour effectuer un défilement fin sur une grande distance, vous devez combiner les 2 types.

Deux étapes sont nécessaires pour mettre en œuvre un défilement fin. Tout d'abord, vous devez l'autoriser en positionnant convenablement les bits des instructions placées dans la Display List, et correspondant aux lignes que vous voulez faire défiler. S'il s'agit de tout l'écran, vous devez donc modifier toutes les instructions de la Display List. Le bit D5 autorise, s'il est à 1, un défilement fin vertical ; le bit D4 autorise un défilement fin horizontal. Puis vous devez stocker dans les registres de défilement, les valeurs déterminant les pas de défilement. Il existe deux registres : le registre pour défilement horizontal HSCROL situé à l'adresse $D404 ; le registre pour défilement vertical VSCROL situé à l'addresse $D405. Pour un déplacement horizontal, placez dans HSCROL le nombre de période d'horloge correspondant à l'amplitude du défilement souhaité. Pour un déplacement vertical, placez dans VSCROL le nombre de lignes de balayage sautées à chaque fois. Ces valeurs seront alors appliquées à toutes les lignes d'affichage dont les instructions dans la Display List valideront les déplacements.

Deux éléments viennent compliquer ce fonctionnement lors d'un défilement fin. Les deux sont dus au fait qu'un affichage partiellement décalé montre davantage d'informations que ne le permet l'affichage normal. Considérez par exemple une ligne composée de 40 caractères ; déplaçons cette ligne d'un demi-caractère sur la gauche. La moitié du premier caractère disparaît à gauche de l'écran. Le quarantième caractère se déplace vers la gauche. Mais que mettre à sa place? La moitié d'un nouveau caractère devrait apparaître à droite. Ce qui donne 41 caractères par ligne. Que faire?

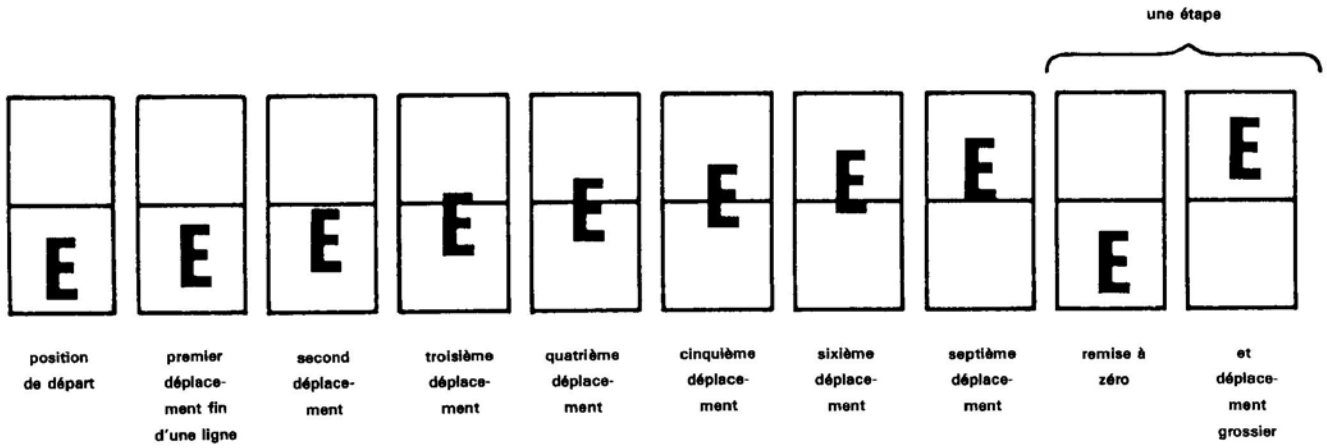
Si vous réalisez un déplacement grossier, le quarante et unième caractère apparaîtra soudainement à l'écran lorsque le premier aura disparu. La ligne compte toujours 40 caractères mais cette apparence soudaine provoque un saut de la ligne. La solution a déjà été prévue dans les circuits de l'ordinateur. Vous avez le choix entre trois options pour la largeur des lignes : l'image étroite (128 cycles d'horloge couleur de large), l'image normale (160 cycles d'horloge couleur de large) et l'image large (192 cycles). Vous choisissez l'option que vous désirez en positionnant à 1 le bit D1, D2 ou D3 du registre DMACTL. Lors d'un défilement fin horizontal, l'ANTIC utilise automatiquement plus d'informations qu'il ne peut afficher. Par exemple, si DMACTL sélectionne une largeur normale d'écran, ce qui correspond à 40 octets par ligne dans le mode Basic 0, l'ANTIC élargira l'écran, jusqu'à 48 octets par ligne. N'oubliez pas dans ce cas d'organiser la mémoire écran en lignes de 48 octets afin de ne pas avoir de «trous» à l'affichage.



ATARINSIDE

Le problème équivalent pour un défilement vertical peut être résolu de deux manières. La manière la plus simple consiste à l'ignorer. Vous n'aurez pas de demi-hauteur de ligne aux deux extrémités de l'affichage. Par contre, les lignes apparaîtront soudainement dans le bas de l'écran. La manière la plus élégante consiste à réserver une ligne d'affichage pour former un buffer. Vous effectuez cela en ne positionnant pas le bit de défilement vertical dans la dernière instruction de Display List de la zone défilant verticalement. Il n'y a plus de saut à l'écran mais l'image est raccourcie d'une ligne d'affichage. Un avantage de l'utilisation des déplacements d'écran devient maintenant évident. Il est possible de créer des images possédant plus de 192 lignes de balayage. Ceci pourrait être désastreux avec un affichage statique, mais avec un déplacement de l'image, les zones situées au-dessus ou en-dessous de la région affichée peuvent toujours être déplacées sur l'écran.

On ne peut pas réaliser de déplacement fin sur une grande amplitude. La limite verticale est de 16 lignes de balayage ; la limite horizontale de 16 périodes d'horloge couleur. Si vous essayez d'aller au-dela, l'ANTIC ignore simplement les bits de poids fort des registres de défilement. Pour obtenir un défilement fin sur une grande amplitude, vous devez combiner un déplacement fin avec un déplacement grossier. Il faut que votre programme mémorise la valeur de déplacement fin et lorsque l'amplitude du déplacement atteint la taille d'un caractère ou d'un point graphique, vous remettez à zéro le registre de déplacement et vous effectuez un déplacement grossier d'une case.




Le programme ci-dessous illustre l'utilisation d'un défilement fin :

```
1 HSCROL=54276
2 VSCROL=54277
10 GRAPHICS 0:LIST
20 DLIST=PEEK(560)+256*PEEK(561)
30 POKE DLIST+10,50:REM          Autorise les 2 défilements
40 POKE DLIST+11,50:REM          Pour 2 lignes d'affichage
50 FOR Y=0 TO 7
60 POKE VSCROL,Y:REM             Déplacement vertical
70 GOSUB 200:REM                 Retard
80 NEXT Y
90 FOR X=0 TO 3
100 POKE HSCROL,X:REM            Déplacement horizontal
110 GOSUB 200:REM                Retard
120 NEXT X
130 GOTO 40
200 FOR J=1 TO 200
210 NEXT J:RETURN
```

Ce programme exécute un déplacement fin à basse vitesse. Il met en évidence plusieurs problèmes liés à l'utilisation des défilements. Tout d'abord, les lignes affichées en-dessous de la fenêtre déplacée sont décalées vers la droite. Cela est dû au fait que l'ANTIC utilise automatiquement 48 octets par ligne au lieu de 40. Ce problème est simple à résoudre en pratique. Il suffit d'organiser la mémoire écran en lignes de 48 octets. Le second problème apparaît lorsque les registres de défilement sont modifiés alors que l'ANTIC est au milieu de son processus d'affichage. Cela perturbe l'ANTIC et provoque un saut de l'image. La solution consiste à modifier les regsitres de défilement seulement lors des périodes de Blanking vertical. Cela ne peut être obtenu qu'avec des sous-programmes écrits en Assembleur. En fait, un défilement fin de l'image se réalise pratiquement en Assembleur.



ATARINSIDE

# APPLICATIONS DES DEPLACEMENTS D'IMAGE

Les applications de cette technique sont nombreuses. Une application évidente réside dans l'affichage de grandes cartes composées de caractères graphiques. Ainsi, en utilisant le mode graphique Basic 2, j'ai créé une très grande carte de la Russie pour le programme «Front de l'Est» ; elle contient l'équivalent de 10 images plein écran. L'écran devient une fenêtre que l'on déplace sur la carte en manipulant une commande à levier. Ce procédé est très économe en mémoir ; la carte toute entière avec ses données, la Display List, et les différents jeux de caractères n'utilisent au total que 4 ko de mémoire vive.

Il y a bien d'autres applications de cette technique : n'importe quelle grande image peut être observée de cette manière. Nous vous conseillons d'utiliser des modes texte associés à des jeux de caractères redéfinis afin d'utiliser le moins de mémoire possible pour pouvoir gérer la plus grande carte réalisable. De grands schémas électroniques peuvent être présentés ainsi. La commande à levier pourrait servir afin de déplacer la fenêtre sur tout le schéma et pour pointer les composants particuliers que l'utilisateur veut adresser.

De grandes zones de texte sont également affichables de cette manière bien qu'il ne soit pas pratique de lire des zones continues de texte en déplaçant l'image en permanence. Par contre, une idée particulièrement intéressante consisterait à utiliser les déplacements d'image pour des menus. Le programme commencerait par une phrase de bienvenue avec des signes indiquant des sous-menus et pointant vers d'autres régions de l'image. «Déplacez-vous dans cette direction pour additionner», «déplacez-vous dans cette direction pour soustraire». L'utilisateur se déplacerait ainsi progressivement par un déplacement fin de menus en menus. Lorsqu'il désire faire un choix, il place un curseur sur l'option désirée et appuie sur le bouton rouge. Bien que ce système ne puisse s'appliquer à tous les programmes, il pourrait être intéressant de la développer chaque fois que cela est possible.

Deux autres applications n'ont pas encore été complètement explorées. La première concerne un défilement fin sélectif dans lequel différentes lignes d'affichage n'ont pas toutes les mêmes défilements autorisés. Normalement, vous déplacez tout l'écran ; mais il n'est pas indispensable de respecter cette tradition. Vous pouvez déplacer une ligne horizontalement, et verticalement un autre groupe de lignes. La seconde application consisterait à utiliser les interruptions de Display List pour modifier les registres HSCROL et VSCROL. Toutefois, modifier VSCROL durant un affichage est une opération délicate. L'ANTIC n'appréciera certainement pas et vous obtiendrez des résultats indésirables. Changer doit être plus simple.



ATARI INSIDE

# 7 - SONS

Les ordinateurs ATARI 400, 600, 800 sont pourvus de circuits générateurs d'effets sonores performants. Il existe à l'intérieur de chaque ordinateur quatre voies sonores réglables indépendemment les unes des autres et pouvant jouer simultanément. Chaque canal est équipé d'un registre fréquence déterminant la note, et d'un registre commande déterminant le niveau sonore et le spectre de bruit. Plusieurs options vous permettent d'insérer des filtres passe-haut, de modifier les fréquences horloge, de choisir d'autres modes de fonctionnement, et de modifier les compteurs polynomiaux.

## DEFINITION DES TERMES ET CONVENTIONS

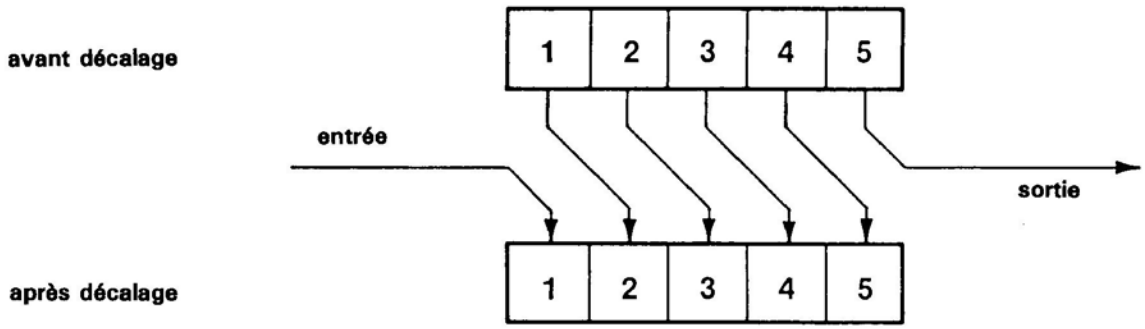
Afin de bien nous faire comprendre, rappelons quelques conventions :

| | |
|---|---|
| 1 Hz (Hertz) | correspond à 1 cycle par seconde |
| 1 kHz (kilo-Hertz) | correspond à 1000 cycles par seconde |
| 1 MHz (méga-Hertz) | correspond à 1 000 000 de cycles par seconde |

Comme nous travaillons sur des signaux carrés, un cycle correspond à un signal dont l'amplitude monte très rapidement jusqu'à un certain niveau, se maintient à ce niveau pendant un certain temps, puis redescend rapidement à zéro pour rester à ce niveau pendant également un certain temps. En envoyant un cycle de ce genre au haut-parleur du téléviseur, on entendra un petit «clic».

Un signal est composé d'une suite continue de cycles (dû moins dans notre exposé). Il existe différents types de signaux réguliers qui se distinguent les uns des autres par la forme du cycle de base. Les ordinateurs ATARI produisent des signaux carrés. Un chanteur produit des ondes sinusoïdales et certains instruments de musique des signaux triangulaires.

Un registre à décalage ressemble à une case mémoire (il mémorise une information binaire) présentant en plus la particularité de pouvoir décaler tous ses bits vers la droite d'une position lorsqu'il en reçoit l'ordre ; ainsi, le bit 5 prendra la place du bit 4, le bit 4 celle du bit 3, etc. Le bit le plus à droite est récupéré en sortie du registre à décalage et le bit le plus à gauche reçoit l'information binaire placée à l'entrée du registre.




Le terme AUDF1-4 doit être compris comme «n'importe lequel des registres fréquence 1 à 4». Leurs adresses sont $D200, $D202, $D204, $D206 (soit, en décimal : 53760, 53762, 53764, 53766).

Le terme AUDC1-4 doit être compris comme «n'importe lequel des registres commande 1 à 4». Leurs adresses sont : $D201, $D203, $SD205, $D207 (soit en décimal : 53761, 53763, 53765, 53767).

La fréquence est le nombre de cycles par seconde. Ainsi, une fréquence de 100 Hz indique qu'en 1 seconde, 100 impulsions (dans le cas de notre signal carré) sont envoyées au haut-parleur. Plus le nombre est élevé, plus la note correspondante est élevée. Par exemple, une chanteuse peut atteindre 5 kHz tandis qu'un meuglement se situe aux alentours de 200 Hz. Pour plus de simplicité, nous confondrons les termes «fréquence», «note», «tonalité» et «hauteur».

«Bruit» et «distorsion» sont utilisés indifféremment l'un pour l'autre bien que leur sens ne soit pas le même. le terme de «bruit» correspond d'ailleurs mieux à la fonction réalisée par les ordinateurs ATARI.

«L'interruption 60 Hz» (ou 50 Hz pour les pays européens) désigne l'interruption exécutée pendant une période de Blanking vertical (c'est-à-dire entre 2 images). Nous reverrons en détail cela ultérieurement.

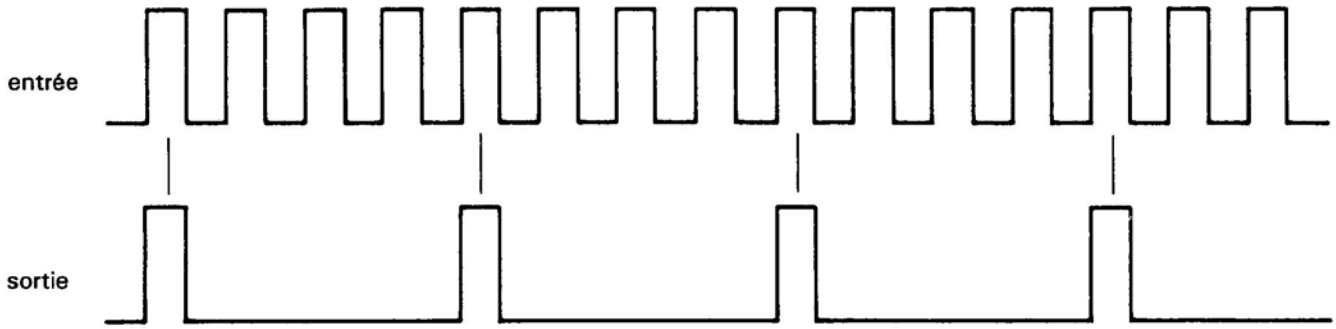
La majorité des exemples sont en Basic. Tapez-les exactement comme ils sont écrits. S'il n'y a pas de numéro de ligne, n'en ajoutez pas ; et s'il y en a plusieurs sur la même ligne, faites de même.

ATARINSIDE

Le son est généré dans l'ordinateur ATARI par un circuit particulier appelé POKEY qui contrôle également le bus d'entrées/sorties série et le clavier. Le POKEY doit être initialisé avant de pouvoir l'utiliser. Une initialisation est nécessaire après toute opération sur le bus série (cassette, unité de disquette, imprimante ou opération RS 232). Pour initialiser le POKEY en Basic, exécutez simplement une instruction de remise à zéro d'un canal, par exemple SOUND,0,0,0,0. En Assembleur, placez la valeur 0 dans le registre AUDCTL ($D208 = 53768), et un 3 dans le registre SKCTL ($D20F = 53775, registre cache à l'adresse $232 = 562).

## AUDF1-4

Chaque canal audio possède un registre fréquence qui contrôle la hauteur de la note jouée par l'ordinateur. Ce registre contient le nombre N utilisé par un circuit diviseur par N. Cette division n'est pas à prendre dans le sens mathématique ; elle correspond à quelque chose de plus simple : chaque fois que le circuit a reçu N impulsions, une impulsion en sort. La figure ci-dessous donne un exemple de division par 4.




Plus N augmente, plus les impulsions en sortie se font rares, produisant ainsi un signal plus grave.

## AUDC1-4

Chaque canal possède aussi un registre de commande associé. Il modifie l'intensité sonore et le contenu de distorsion selon l'arrangement ci-dessous :

| Numéro de bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | distorsion | | | Intens. seult. | intensité | | | |

### INTENSITE SONORE

Les 4 bits de poids faible contiennent une valeur numérique donnant en proportion directe l'intensité sonore. La valeur 0 éteint totalement le son tandis que 15 correspond au maximum pour ce canal. La somme des 4 intensités ne doit pas dépasser 32, sous peine de surmoduler la sortie audio. Le résultat sonore serait alors de très médiocre qualité.

### DISTORSION

Les 3 bits de poids fort modifient la distorsion du son. Cela permet la création d'effets sonores spéciaux. L'utilisation de ce paramètre permet de synthétiser une variété presque infinie de sons, depuis des ronronnements jusqu'à des sons d'accompagnement en passant par toute sorte de cliquetis, sifflets, bruits de moteurs, etc.

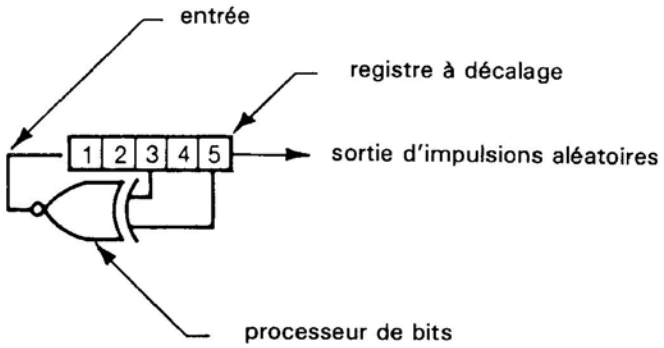
La distorsion dont nous parlons ici n'est pas à prendre dans le sens habituel de ce terme. Par exemple, la distorsion d'intermodulation et la distorsion harmonique constituent des critères de qualité pour les appareils de reproduction haute fidélité. Ces types de distorsion se rapportent à une dégénérescence du signal, la forme du signal étant légèrement modifiée en raison des erreurs dans les circuits électroniques. La distorsion de l'ordinateur ne modifie pas le signal (il s'agit toujours d'impulsions), mais elle supprime plutôt certaines impulsions du signal. Un terme plus adéquat pour désigner cette distorsion pourrait être «bruit».



ATARINSIDE

Avant que vous compreniez pleinement ce qu'est la distorsion, vous devez savoir ce qu'est un compteur polynomial (nous parlerons désormais de polycompteur). Les polycompteurs sont employés dans l'ordinateur ATARI comme une source d'impulsions aléatoires utilisées dans un générateur de bruit. Les polycompteurs utilisent un registre à décalage fonctionnant à 1,79 MHz. Leurs contenus sont ensuite traités et renvoyés sur l'entrée des polycompteurs elle-même. Cela produit une séquence de bits semi-aléatoires à la sortie du registre à décalage.

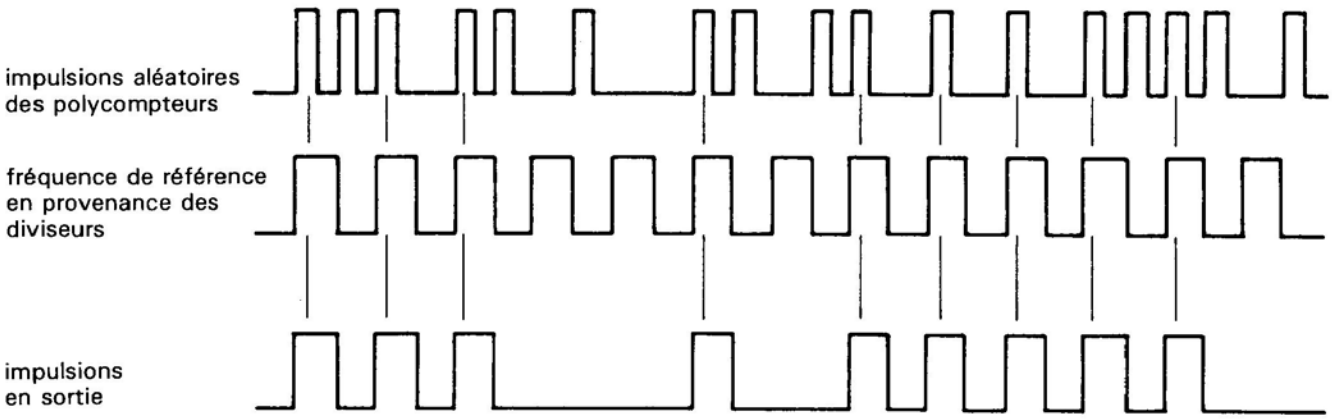
Par exemple, dans le dessin ci-dessous, le bit 5 sera extrait du registre à décalage pour former l'impulsion de sortie et le bit 1 sera donné par une fonction des bits 3 et 5.




Le processeur de bit reçoit les valeurs de certains bits du registre à décalage (bits 3 et 5 dans l'exemple ci-dessus) et les traite pour en sortir une nouvelle valeur qui sera envoyée à l'entrée du registre.

Ces polycompteurs ne sont pas totalement aléatoires car ils répètent leur suite de bits après un temps relativement cours. Comme vous vous en doutez, cette fréquence de répétition dépend du nombre de bits formant le polycompteur : plus le polycompteur sera long, plus la période de répétition de la suite d'impulsions sera longue.

Dans l'ordinateur ATARI, la distorsion est obtenue en utilisant ces impulsions alétoires en provenance des polycompteurs dans un registre de sélection. Ce circuit est en fait un comparateur numérique. Les seules impulsions parvenant à sa sortie sont celles correspondant à la coïncidence entre une fréquence de référence et les impulsions aléatoires. De cette manière, des impulsions de la fréquence de référence sont supprimées dans le signal d'une manière aléatoire. En voici un exemple sur le dessin ci-dessous :




Puisque des impulsions sont supprimées dans le signal de référence, la note n'aura plus le même timbre. Voilà comment une distorsion est introduite dans un canal son.

Puisque les polycompteurs répètent leur suite de bits, la forme d'onde en sortie se répète périodiquement. La note que l'on entendra sera donc transformée d'une manière cyclique. Il est ainsi possible de programmer un bourdonnement, un bruit de moteur, ou tout autre bruit répétitif.

L'ordinateur ATARI est équipé de trois polycompteurs de longueur différente, pouvant être combinés de différentes manières pour produire des effets sonores variés. Les plus petits polycompteurs (4 et 5 bits de long) ont un cycle de répétition suffisamment court pour créer un bourdonnement à variation rapide. le grand polycompteur (17 bits de long) possède un cycle de répétition si long que la fréquence de répétition ne peut être discernée. Le polycompteur de 17 bits peut être utilisé pour générer des explosions, et toutes sortes de bruits où l'on a besoin d'une suite aléatoire de crépitements. Il est suffisamment irrégulier pour générer un bruit blanc (signal dans lequel toutes les fréquences existent simultanément avec la même amplitude).

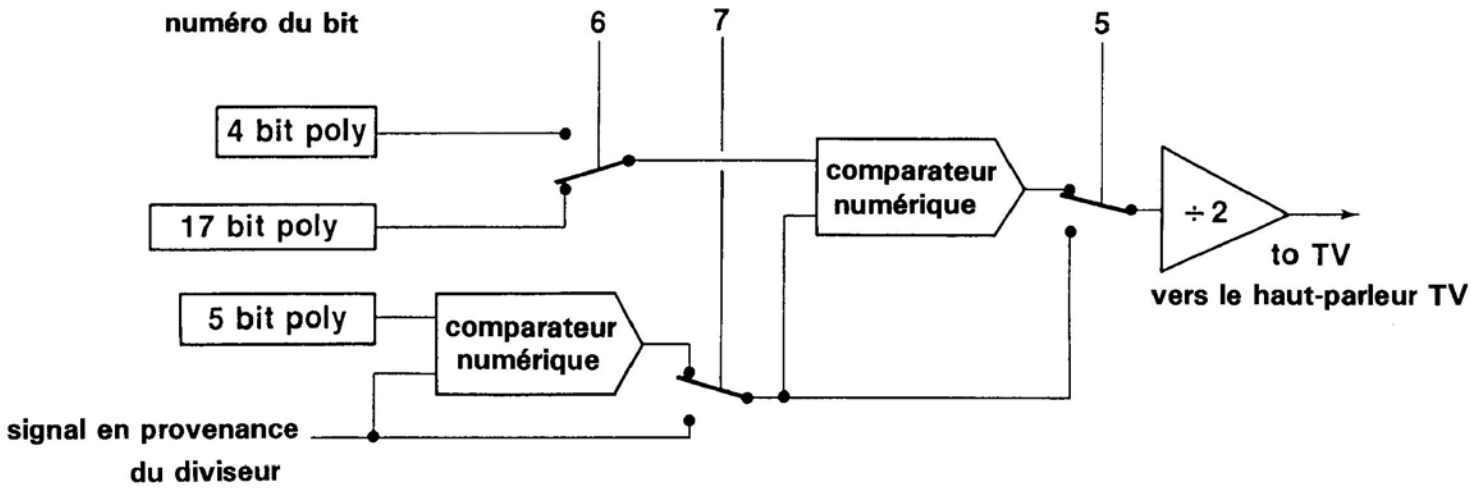
Chaque canal audio offre six combinaisons distinctes des trois polycompteurs.


ATARINSIDE

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | | | | | | utilisation des registres 5 bits et 17 bits, puis division par 2 |
| 0 | X | 1 | | | | | | utilisation du registre 5 bits, puis division par 2 |
| 0 | 1 | 0 | | | | | | utilisation des registres 5 bits puis 4 bits, et division par 2 |
| 1 | 0 | 0 | | | | | | utilisation du registre 17 bits et division par 2 |
| 1 | X | 1 | | | | | | division par 2 (pas de polycompteur) |
| 1 | 1 | 0 | | | | | | utilisation du registre 4 bits, division par 2 |

«X» signifie que l'état de ce bit est quelconque.

Les trois bits de poids fort des registres AUDC1-4 contrôle trois interrupteurs placés dans le circuit audio comme indiqué ci-dessous :




Chaque combinaison des polycompteurs donne un son particulier. De plus, le résultat sonore peut grandement varier en fonction de la fréquence. Pour cette raison, il ne faut pas hésiter à faire de nombreux essais afin d'obtenir le résultat escompté. Nous vous donnons dans le tableau ci-dessous quelques indications pouvant vous servir comme point de départ.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | basses fréquences | fréquences moyennes | | hautes fréquences |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | | | | | | compteur geiger | feu violent | air pulsé | vapeur |
| 0 | X | 1 | | | | | | pistolet | voiture au ralenti | moteur électrique | transformateur |
| 0 | 1 | 0 | | | | | | feu calme | auto peinant | | |
| 1 | 0 | 0 | | | | | | début de choc | | interférences radio | chute d'eau |
| 1 | X | 1 | | | | | | son pur | | | |
| 1 | 1 | 0 | | | | | | avion | tondeuse à gazon | rasoir électrique | |

**SONS OBTENUS PAR «INTENSITE SEULEMENT»**

Le bit 4 de AUDC1-4 spécifie le mode «intensité seulement». Quand ce bit est à 1, les valeurs des bits 0 à 3 du registre sont envoyées directement au haut-parleur du téléviseur. le signal n'est pas modulé avec la fréquence indiquée par les registres AUDF1-4.

Pour pleinement comprendre l'utilité de ce mode de fonctionnement, vous devez savoir comment fonctionne un haut-parleur : tout haut-parleur électro-dynamique possède un cône qui se déplace en avant et en arrière. La position du cône est liée à la tension reçue de l'ordinateur. Lorsque la tension croît, le cône avance, et lorsqu'elle décroît, il recule. L'air est ainsi mis en mouvement et lorsque ce phénomène se répète suffisamment vite (plus de 20 fois par seconde), l'oreille humaine le détecte.



ATARINSIDE

Le fait d'envoyer une seule impulsion sur un haut-parleur suffit à provoquer un léger «clic» audible. En voici un exemple :

POKE 53761,31 : POKE 53761,16

Une suite d'impulsions provoque un son continu. Plus les impulsions sont envoyées rapidement, plus le son est aigu.

Il faut comprendre que dans le mode «intensité seulement», on envoie une tension vers le haut-parleur, mais cette tension restera constante tant que le programme ne la changera pas. le programme doit moduler cette valeur suffisamment souvent pour créer un son. Essayez maintenant les deux instructions suivantes en écoutant soigneusement ce qui se passe pour chacune :

POKE 53761,31
POKE 53761,31

La première fois, vous entendez un clic comme prévu. le cône s'est déplacé. Mais la seconde fois, vous n'entendez rien. Le cône ayant déjà atteint sa position, il ne se déplace pas et donc ne produit aucun son. Maintenant essayez cela :

POKE 53761,16
POKE 53761,16

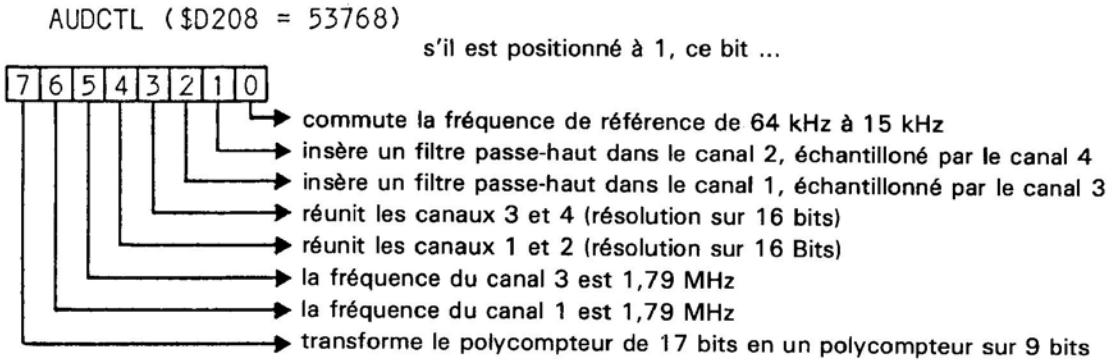
Comme précédemment, vous entendez quelque chose sur la première instruction, mais pas sur la seconde.

Ainsi, le bit d'intensité seulement donne au programme un contrôle complet de la position du cône du haut-parleur. Vous pouvez le placer à 16 positions différentes.

Par exemple, pour simuler une onde triangulaire (comme celle produite par les cuivres), vous pouvez envoyer une intensité de 8 suivie par les valeurs 9, 10, 11, 10, 9, 8, 7, 6, 5, 6, 7 et à nouveau 8, et répéter cette séquence très rapidement. Par cette technique, il est théoriquement possible de synthétiser n'importe quel type d'onde. On peut même aller jusqu'à synthétiser la voix humaine. Il est par contre indispensable dans ce cas de travailler en Assembleur. Nous reviendrons plus tard sur l'utilisation de ce bit.

# AUDCTL

En plus des octets de commande indépendants pour chaque canal (AUDC1-4), il existe un registre d'option (AUDCTL) influençant le fonctionnement des 4 voix. Chaque bit de ce registre réalise une fonction particulière.




### HORLOGE

Avant d'expliquer en détail les options AUDCTL, nous devons donner quelques explications sur les circuits d'horloge. En général, une horloge est une suite continue d'impulsions servant à synchroniser des millions d'opérations internes s'effectuant chaque seconde dans l'ordinateur. Chaque impulsion de l'horloge centrale provoque l'avancement d'un pas élémentaire dans toutes les opérations en cours. Vous vous souvenez que nous avons vu précédemment le principe du diviseur par N. Ce circuit délivre une impulsion en sortie lorsque N impulsions ont été comptées en entrée. Vous pouvez vous demander d'où vient la suite d'impulsions reçues par le diviseur. L'horloge principale du système fonctionne à 1,79 MHz ; elle peut être utilisée pour fournir ces impulsions. Mais il existe d'autres horloges secondaires pouvant être utilisées à sa place. Le registre AUDCTL vous permet de choisir l'horloge que vous désirez utiliser pour le diviseur par N. En modifiant la fréquence d'entrée du diviseur, vous pouvez modifier considérablement la fréquence en sortie.

Par exemple, imaginons que vous utilisiez une fréquence à 15 kHz et que le registre fréquence soit programmé pour diviser par 8. La fréquence de sortie se situe aux alentours de 2 kHz. Si vous modifiez la fréquence de référence pour passer à 64 kHz sans modifier le diviseur, vous obtiendrez en sortie une fréquence de 8 kHz. La fréquence de sortie est donc égale à la fréquence d'entrée divisée par N.



ATARI INSIDE

En positionnant à 1 le bit D0 de AUDCTL, vous modifiez la fréquence de référence. Tous les canaux utilisant une fréquence de 64 kHz en entrée utiliseront après cette modification une fréquence de 15 kHz. De la même manière, en positionnant les bits 5 ou 6, vous fournissez aux canaux 3 ou 1, une fréquence de 1,79 MHz. Cela produit une note plus haute :

```
SOUND 0,255,10,8        canal 1 activé, note très basse
POKE 53768,64           bit 6 de AUDCTL à 1
```

**REGISTRES FREQUENCE DE 16 BITS**

Les 8 bits de résolution des registres fréquence semblent satisfaisants pour obtenir avec une précision suffisante n'importe quelle fréquence. Toutefois, cette résolution est insuffisante dans certains cas. Voyons par exemple ce qui se passe lorsque nous tapons cette ligne d'instruction :

```
FOR I=255 TO 0 STEP -1 : SOUND 0,I,10,8 : NEXT I
```

La tonalité monte doucement au début mais au fur et à mesure que la fréquence augmente, le pas est de plus en plus grand, ce qui s'entend nettement. En effet, lorsqu'on divise 15 kHz par 255, on obtient un résultat très proche de 15 kHz divisé par 254 ; mais 15 kHz est très loin de 15 kHz divisé par 1. La solution consiste à utiliser un plus grand diviseur, donnant une meilleure résolution. Cela est déjà prévu dans le POKEY.

Les bits 3 ou 4 du registre AUDCTL permettent la réunion de 2 canaux, produisant un seul canal possédant une gamme de fréquence élargie. Normalement, le diviseur de chaque canal peut varier de 0 à 255. En réunissant les 2 canaux, la gamme s'étend de 0 à 65535 (diviseur sur 16 bits). Dans ce mode, il est possible d'abaisser la fréquence de sortie à moins de 1 Hz. Le programme ci-dessous utilise deux canaux dans le mode 16 bits et deux commandes à molette pour modifier la fréquence. Connectez la paire de commande au port 1, tapez ce programme puis RUN :

```
10 SOUND 0,0,0,0                          Initialisation
20 POKE 53768,80                          Le canal 1 reçoit une horloge de 1,79 MHz et il est couplé avec le cana
30 POKE 53761,160:POKE 53763,168          Couple le canal 1, met en service le canal 2 (sons purs)
40 POKE 53760,PADDLE(0):POKE 53762,PADDLE(1
                                          Utilisation des commandes à molette pour modifier les diviseurs
50 GOTO 40
```

La commande de droite modifie la fréquence avec un pas important tandis que la commande de gauche utilise un pas plus petit. La commande de gauche modifie la fréquence à l'intérieur d'un pas déterminé par la commande de droite.

Ce programme commence par positionner à 1 les bits 4 et 6 du registre AUDCTL, ce qui signifie «envoyer 1,79 MHz sur le canal 1 et réunir le canal 1 au canal 2». Dès ce moment, les deux registres diviseurs sont couplés pour former un registre 16 bits. Puis le volume sonore du canal 1 est mis à 0 car nous ne désirons pas écouter la sortie du canal 1 mais plutôt la sortie du résultat canal 1/canal 2. Le diviseur du canal 1 constitue l'octet de poids faible dans le diviseur 16 bits. Il modifie donc la fréquence par petites étapes. Par exemple, si l'on place la valeur 1 dans le registre fréquence du canal 1, le diviseur 16 bits vaut 1. Si par contre on place la valeur 1 dans le diviseur du canal 2, le diviseur 16 bits vaut 256. Enfin, mettre un 1 simultanément dans les 2 diviseurs provoque une division par 257.

Le bit 3 de AUDCTL couple de la même manière les canaux 3 et 4.

Les instructions suivantes donnent quelques aspects intéressants d'un son sur 16 bits.

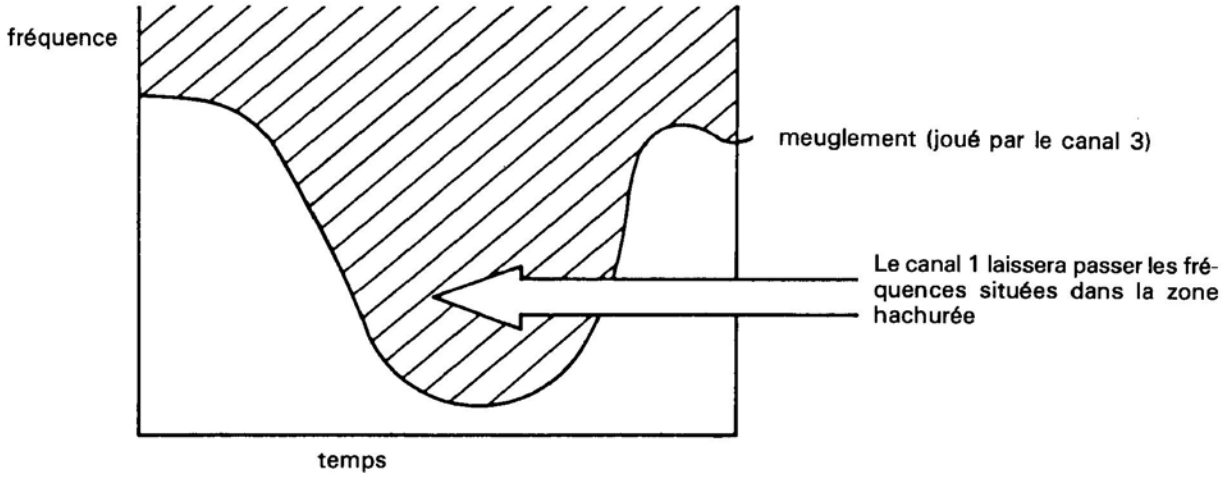
```
SOUND 0,0,0,0
POKE 53768,24
POKE 53761,168
POKE 53763,168
POKE 53765,168
POKE 53767,168
POKE 53760,240:REM Essayez dans ces 4 instructions d'autres valeurs
POKE 53764,252
POKE 53762,28
POKE 53766,49
```



ATARINSIDE

**FILTRES PASSE-HAUT**

Les bits 1 et 2 de AUDCTL commandent l'insertion dans le circuit de filtres passe-haut respectivement dans les canaux 2 et 1. Un filtre passe-haut ne laisse passer en sortie que les fréquences hautes. Dans le cas de ces filtres, les hautes fréquences sont définies comme étant les fréquences plus hautes que la sortie de n'importe quel autre canal sélectionné par la combinaison de bits dans AUDCTL. Par exemple, si le canal 3 est programmé pour produire un meuglement et si le bit 2 de AUDCTL est positionné, seuls les sons de fréquence supérieures au meuglement seront entendus sur le canal 1.




Ce filtre est programmable en temps réel, si bien que les effets peuvent être modifiés en cours d'exécution. Cela donne de très grandes possibilités d'effets. Essayez par exemple :

```
SOUND 0,0,0,0
POKE 53768,4
POKE 53761,168:POKE 53765,168
POKE 53760,254:POKE 53764,127
```

**CONVERSION DU POLYNOME SUR 9 BITS**

Le bit 7 du registre AUDCTL transforme le compteur polynomial de 17 bits en un registre à décalage sur 9 bits. Cela raccourcit le cycle avec lequel la génération aléatoire d'impulsions se répète. Cela rend la distorsion plus répétitive lors d'une écoute. Faites l'essai suivant et écoutez soigneusement :

```
SOUND 0,80,8,8
POKE 53768,128
```

Utilise le registre sur 17 bits
Passe le registre sur 9 bits

# TECHNIQUES LOGICIELLES DE GENERATION DE SONS

Il existe deux manières principales pour utiliser les circuits audio de l'ordinateur ATARI : une manière statique et une manière dynamique. La génération statique est la plus simple des deux ; le programme positionne quelques-uns des registres vus précédemment, ce qui produit un son ; puis il exécute d'autres tâches puis revient couper le son. La génération dynamique est plus difficile ; l'ordinateur modifie constamment les registres audio durant l'exécution du programme. Par exemple :

Son statique :
```
SOUND 0,120,8,8
```

Son dynamique :
```
FOR X=0 TO 255 : SOUND 0,X,8,8 : NEXT X
```



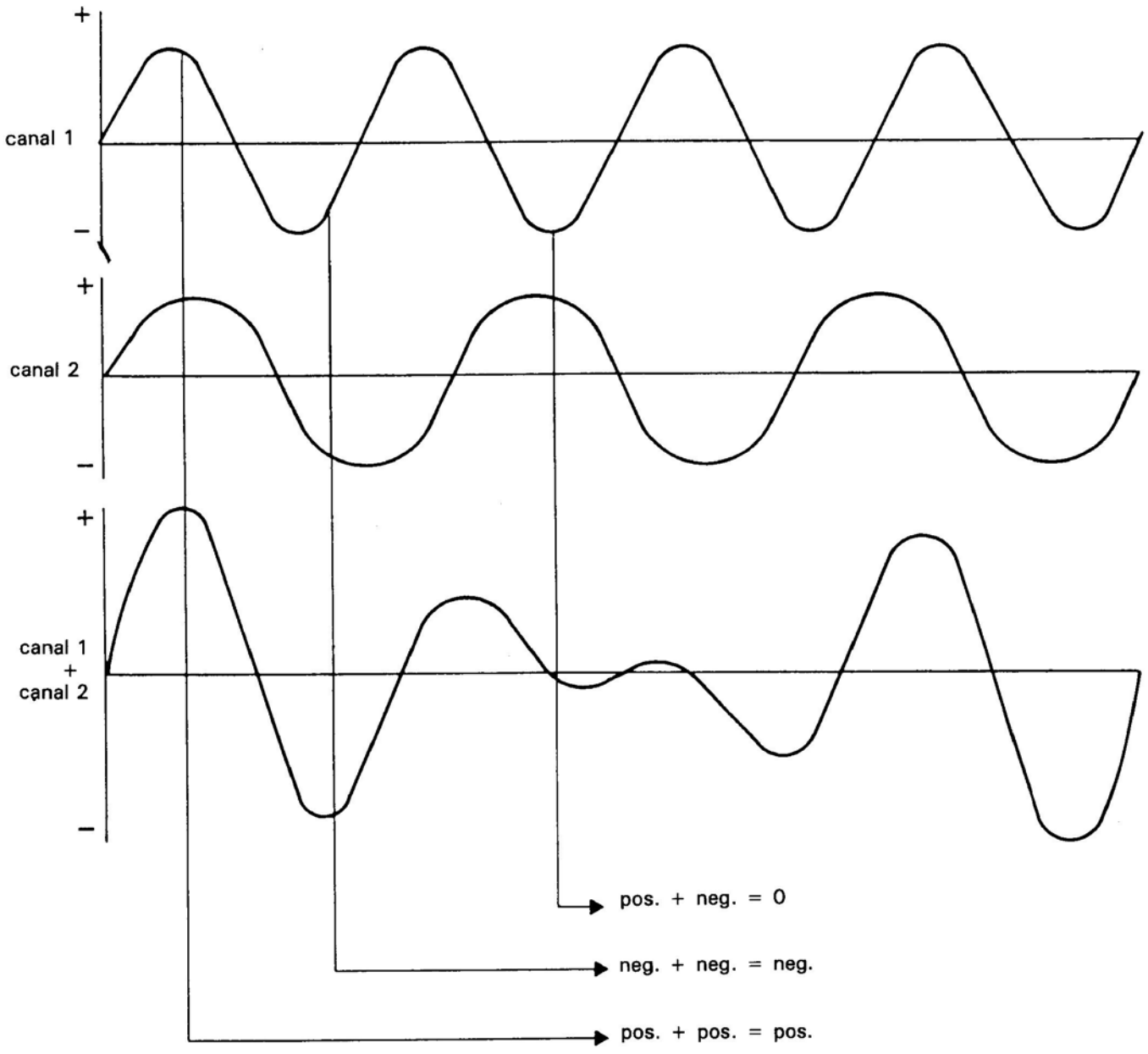
ATARIINSIDE

**SON STATIQUE**

Le son statique se limite normalement au bip, sirènes diverses, clic, etc. Il existe des exceptions. Nous en avons vues au paragraphe sur les filtres passe-haut et les registres fréquence 16 bits. Une autre manière d'obtenir des effets intéressants est d'utiliser une interférence comme dans cet exemple :

```
SOUND 0,255,10,8
SOUND 1,254,10,8
```

Cet effet curieux obtenu est le résultat de deux ondes légèrement hors phase. Reportez-vous à la figure ci-dessous. Les deux canaux jouent deux fréquences légèrement différentes. Vous entendez leur somme, c'est-à-dire la forme d'onde du bas du dessin.




Ce dessin montre qu'à certains instants, les deux signaux s'ajoutent, et qu'en d'autres, ils s'annulent. Lorsqu'il y a coïncidence exacte entre les maximum de chaque onde, le volume sonore résultant est bien égal à la somme des deux volumes sonores. Vous pouvez étendre ce principe à 3 ou 4 canaux fonctionnant simultanément. Cela permet de créer des effets très intéressants.

Plus la différence entre les fréquences des deux canaux est faible, plus le battement est lent. C'est par ce principe qu'on accorde un piano. Lorsque le battement est nul, l'accord est parfait. Pour comprendre cela, faites-en l'expérience sur un graphique similaire à la figure précédente et essayez les instructions suivantes :

```
SOUND 0,255,10,8
SOUND 1,254,10,8
SOUND 1,253,10,8
SOUND 1,252,10,8
```

Au fur et à mesure que l'écart augmente, le battement est plus rapide, la période de répétition décroît.



ATARINSIDE

## SON DYNAMIQUE

Des effets sonores plus complexes requièrent normalement l'utilisation de techniques dynamiques. Trois possibilités s'offrent au programmeur : sons en Basic, interruptions 60 Hz, et sons en langage machine.

## SONS EN BASIC

Le Basic ne permet pas de supporter tous les effets sonores possibles. Comme vous l'avez probablement remarqué, l'instruction SOUND efface toute programmation particulière du registre AUDCTL. Cela oblige le programmeur à utiliser plutôt une suite d'instructions POKE.

En outre, le Basic est limité par sa vitesse d'exécution. Si le programme ne sert pas uniquement à une génération de sons, le microprocesseur n'aura pas suffisamment de temps pour faire mieux qu'un son statique ou un son dynamique lent. Une alternative consiste à suspendre temporairement toutes les autres opérations durant la production d'un son.

Un autre problème peut surgir pour interprêter une mélodie simultanément sur plusieurs canaux. Si les quatre canaux sont utilisés, l'écart de temps entre la première instruction et la quatrième devient suffisamment important pour être perçu clairement. Le programme suivant présente une solution à ce problème :

```
10 SOUND 0,0,0,0:DIM SIMUL$(16)
20 RESTORE 9999:X=1
25 READ Q:IF Q<>-1 THEN SIMUL$(X)=CHR$(Q):X=X+1:GOTO 25
27 RESTORE 100
30 READ F1,C1,F2,C2,F3,C3,F4,C4
40 IF F1=-1 THEN END
50 X=USR(ADR(SIMUL$),F1,C1,F2,C2,F3,C3,F4,C4)
55 FOR X=0 TO 150:NEXT X
60 GOTO 30
100 DATA 182,168,0,0,0,0,0,0
110 DATA 162,168,182,166,0,0,0,0
120 DATA 144,168,162,166,35,166,0,0
130 DATA 128,168,144,166,40,166,35,166
140 DATA 121,168,128,166,45,166,40,166
150 DATA 108,168,121,166,47,166,45,166
160 DATA 96,168,108,166,53,166,47,166
170 DATA 91,168,96,166,60,166,53,166
999 DATA -1,0,0,0,0,0,0,0
9000 REM
9010 REM
9020 REM this data contains the machine lang. program,
9030 REM and is read into SIMUL$
9999 DATA 104,133,203,162,0,104,104,157,0,210,232,228,203,208,246,96,-1
```

Dans ce programme, SIMUL$ représente un petit langage machine qui commande les 4 registres sons très rapidement. Un programme Basic utilisant SIMUL$ peut rapidement manipuler les 4 voies. N'importe quel programme peut appeler SIMUL$ en plaçant la valeur des registres comme argument dans la fonction USR. L'ordre des paramètres est la valeur du registre fréquence en premier suivi par la valeur du registre commande, du canal 1 au canal 4.

Pour augmenter encore la vitesse, vous pouvez n'utiliser que 1 ou 2 canaux. Il vous suffit simplement de passer le nombre de paramètres voulus seulement (2 paramètres par canal).

SIMUL$ présente encore un avantage substantiel par rapport à une simple programmation Basic. Comme nous l'avons vu précédemment, le registre AUDCTL est repositionné, après chaque exécution d'une instruction SOUND. Si vous utilisez SIMUL$, vous n'utilisez pas l'instruction SOUND et donc AUDCTL reste programmé selon vos désirs.

Il existe une autre méthode générant des sons en Basic, mais elle est impraticable. Cette méthode utilise le bit intensité seulement d'un canal quelconque. Tapez l'exemple suivant :

```
SOUND 0,0,0,0
10 POKE 53761,16 : POKE 53761,31 : GOTO 10
```

Ce programme positionne le bit D4 (intensité seulement) du canal 1 et module l'intensité alternativement de 0 à 15 aussi vite que peut le faire le Basic. Vous constatez que le résultat s'apparente seulement à un ronflement.

## INTERRUPTION 60 Hz (50 Hz EN EUROPE)

Cette technique est probablement la plus pratique et la plus utile de toutes les méthodes offertes au programmeur d'un ordinateur ATARI.

ATARI INSIDE

Très précisément, tous les 60e de seconde, les circuits vidéo génèrent automatiquement une interruption. Lorsque cela se produit, le 6502 quitte temporairement le programme principal et vient exécuter quelques sous-programmes particuliers. C'est à ce moment qu'il lit les manettes de jeu, qu'il transfère le contenu des mémoires cache dans les registres de l'ANTIC et du CTIA, qu'il repositionne certains compteurs, etc.

Une particularité intéressante réside dans le fait qu'avant de revenir au programme principal, le 6502 peut exécuter un programme utilisateur, par exemple votre programme générateur de son. C'est le moment idéal pour cela, car ce programme s'exécutera à intervalle de temps parfaitement régulier et parce que le 6502 ne sera pas perturbé à ce stade par le programme principal. Plus extraordinaire encore est la souplesse de cette technique. Ecrit en langage machine, le programme d'effets sonores peut alors s'exécuter quel que soit le langage utilisé pour le programme principal : Basic, Assembleur, Forth, etc. Il ne nécessite en fait que très peu de modifications pour fonctionner avec un autre programme. Il peut donc être conçu à part, sans crainte de voir lors de son implantation, ses temporisations se modifier.

Une gestion de table donne une grande facilité de programmation. Cela correspond à écrire en mémoire les suites de données devant être transférées dans les registres sons. Ces données représentent les valeurs des registres fréquence et éventuellement celles des registres de commande. Le sous-programme lit simplement quelques octets et les transfère dans les registres correspondants. Par cette méthode, les notes peuvent changer jusqu'à 60 fois par seconde sur chaque canal, ce qui est suffisant pour la majorité des applications.

Lorsqu'un tel programme a été écrit et placé en mémoire (par exemple à l'adresse $600), vous devez l'intégrer dans le programme de Blanking vertical.

Les adresses mémoire $224, $225 contiennent l'adresse d'un petit programme appelé XITVBL (Exit Vertical Blank Interrupt). XITVBL est exécuté lorsque le traitement de l'interruption est terminé et son rôle est de passer la main au programme principal. La procédure pour insérer votre programme est la suivante :

1. Placez votre programme dans la mémoire.
2. Terminez-le par l'instruction JMP $E462 (c'est l'adresse de XITVBL).
3. Chargez en Assembleur le registre X du 6502 avec le poids fort de l'adresse de départ de votre programme (un 6 dans notre exemple).
4. Chargez le registre Y avec le poids faible de l'adresse (0 dans notre exemple).
5. Placez la valeur 7 dans l'accumulateur.
6. Effectuez un JSR $E45C. Cela recopiera l'adresse $600 dans les cases mémoire $224, $225.

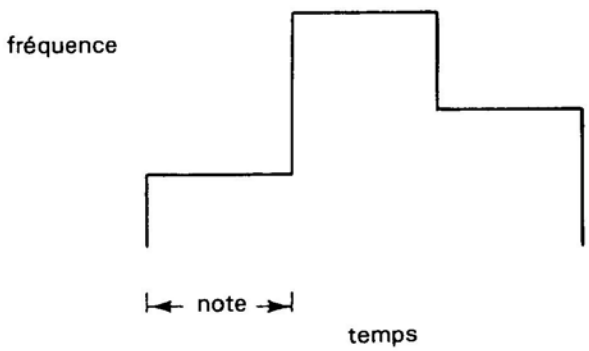
Les étapes 3 à 6 sont nécessaires pour modifier la valeur du pointeur en $224, $225 sans erreur. Le sous programme que vous appelez à l'étape 6 s'appelle SETVBV (Set Vertical Blank Vectors). Lorsque cela est terminé, votre système fonctionne ainsi : lorsqu'une interruption de Blanking Vertical se produit,

1. Le sous-programme d'interruption de l'ordinateur est exécuté.
2. Puis le 6502 exécute votre programme dont il trouve l'adresse de début en $224, $225.
3. Votre programme s'exécute.
4. Votre programme enchaîne sur XITVBL.
5. XITVBL renvoie le 6502 sur le programme principal.

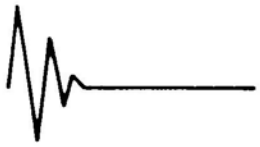
Si vous trouvez cela un peu complexe, vous trouverez dans le catalogue APX un programme appelé INSOMNIA («GENERATEUR D'EFFETS SONORES» si vous désirez la version française). Il permet la création et la modification des données tout en vous permettant de les écouter. Ce programme comporte également un générateur de sons par interruptions gérant une table et compatible avec n'importe quel langage.

**GENERATION D'EFFETS SONORES EN CODE MACHINE**

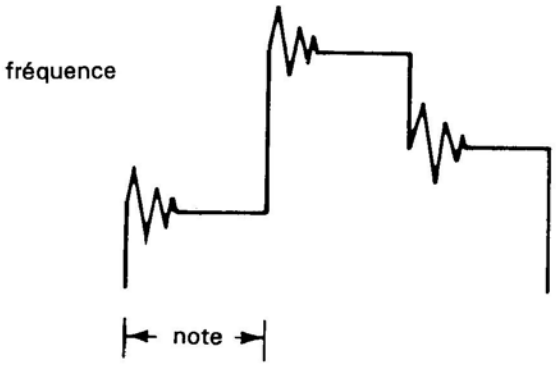
Le contrôle direct des registres audio par le microprocesseur ouvre de nouvelles portes dans le domaine du son. Cette technique consiste à écrire un programme gérant une table de données ; cela ressemble au fonctionnement par interruptions que nous avons vu au paragraphe précédent, mais maintenant, c'est le programme principal lui-même qui est chargé de ce travail. Il est ainsi possible d'obtenir des sons de qualité supérieure. Considérons par exemple la sortie d'un signal obtenu par le procédé d'interruption à 60 Hz :



Puisque le temps alloué au traitement des sons devient maintenant plus important, nous pouvons modifier les paramètres de chaque note très rapidement. Il est ainsi possible de simuler le timbre d'un instrument. Par exemple, supposons que chaque fois que nous appuyons sur une touche, le piano produise une suite de fréquences comparables à la figure ci-dessous :



Pour simuler un piano, l'idée serait de modifier très rapidement l'enveloppe de la note pendant son temps d'exécution. Par exemple, pour que les trois notes représentées précédemment simulent un piano, il faudrait obtenir une variation de fréquence ressemblant à ceci :




Nous obtenons la même mélodie mais le timbre est enrichi. Malheureusement, nous devons consacrer beaucoup de temps à ce travail. Les registres ne sont plus modifiés à chaque note mais plutôt 100 fois par note.

**SONS PAR INTENSITE SEULEMENT**

Dans les pages précédentes, nous avons pu expérimenter les bits «intensité seulement» des registres AUDC1-4 mais nous avons constaté qu'il était pratiquement impossible de les utiliser en Basic. Cela car le Basic est trop lent.

Ce bit autorise pourtant une programmation très fine des sons. Une véritable génération d'onde est maintenant possible. Envisageons la reproduction d'un son de piano. Nous allons donner un résultat plus ressemblant mais malheureusement, les 4 bits disponibles pour l'intensité ne sont pas suffisants pour obtenir un résultat de très haute qualité. Vous constaterez toutefois que la qualité est cependant surprenante. Le programme suivant exploite l'utilisation du bit d'intensité seulement. Le programme démarre à l'adresse $4000.



ATARINSIDE

```
                    0100 ;
                    0110 ; VONLY            Bob Fraser 7-23-81
                    0120 ;
                    0130 ;
                    0140 ; programme de test du bit «intensité seulement»
                    0150 ;
                    0160 ;
                    0170 ;
                    0180 ;
D208                0190 AUDCTL =     $D208
D200                0200 AUDF1  =     $D200
D201                0210 AUDC1  =     $D201
D20F                0220 SKCTL  =     $D20F
                    0230 ;
                    0240 ;
0000                0250         *=   $B0
00B0 01             0260 TEMPO   .BYTE 1
00B1 00             0270 MSC     .BYTE 0
                    0280 ;
                    0290 ;
                    0300 ;
00B2                0310         *=   $4000
4000 A900           0320         LDA  #0
4002 8D08D2         0330         STA  AUDCTL
4005 A903           0340         LDA  #3
4007 8D0FD2         0350         STA  SKCTL
400A A200           0360         LDX  #0
                    0370 ;
400C A900           0380         LDA  #0
400E 8D0ED4         0390         STA  $D40E   suppression des interruptions de Blanking vertical
4011 8D0ED2         0400         STA  $D20E   suppression des interruptions
4014 8D00D4         0410         STA  $D400   suppression du DMA
                    0420 ;
                    0430 ;
                    0440 ;
4017 BD5240         0450 L00     LDA  DTAB,X
401A 85B1           0460         STA  MSC
                    0470 ;
401C BD3640         0480         LDA  VTAB,X
401F A4B0           0490 L0      LDY  TEMPO
4021 8D01D2         0500         STA  AUDC1
4024 88             0510 L1      DEY
4025 D0FD           0520         BNE  L1
                    0530 ;
                    0540 ; dec most sig ctr
4027 C6B1           0550         DEC  MSC
4029 D0F4           0560         BNE  L0
                    0570 ;
                    0580 ;
                    0590 ; new note
                    0600 ;
```



ATARINSIDE

```
402B E8      0610          INX
402C EC3540  0620          CPX  NC
402F D0E6    0630          BNE  LOO
             0640 ;
             0650 ; compare le pointeur de note
4031 A200    0660          LDX  #0
4033 F0E2    0670          BEQ  LOO
             0680 ;
             0690 ;
4035 1C      0700 NC       .BYTE 28          compteur de note
             0710 ;
             0720 ; table des intensités
             0730 VTAB
4036 18      0740          .BYTE 24,25,26,27,28,29,30,31
4037 19
4038 1A
4039 1B
403A 1C
403B 1D
403C 1E
403D 1F
403E 1E      0750          .BYTE 30,29,28,27,26,25,24
403F 1D
4040 1C
4041 1B
4042 1A
4043 19
4044 18
4045 17      0760          .BYTE 23,22,21,20,19,18,17
4046 16
4047 15
4048 14
4049 13
404A 12
404B 11
404C 12      0770          .BYTE 18,19,20,21,22,23
404D 13
404E 14
404F 15
4050 16
4051 17
             0780 ;
             0790 ; table des durées
             0800 DTAB
4052 01      0810          .BYTE 1,1,1,2,2,2,3,6
4053 01
4054 01
4055 02
4056 02
4057 02
4058 03
4059 06
```



ATARI INSIDE

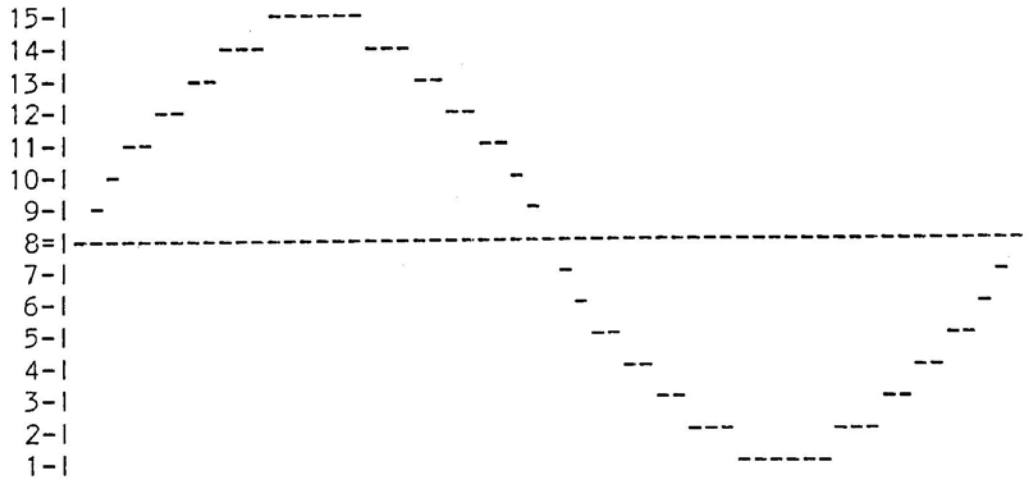
```
405A 03      0820        .BYTE 3,2,2,2,1,1,1
405B 02
405C 02
405D 02
405E 01
405F 01
4060 01
4061 01      0830        .BYTE 1,1,2,2,2,3,6
4062 01
4063 02
4064 02
4065 02
4066 03
4067 06
4068 03      0840        .BYTE 3,2,2,2,1,1
4069 02
406A 02
406B 02
406C 01
406D 01
```

La vitesse n'est plus un problème. Le signal se compose de presque 60 parties différentes et le programme peut être utilisé pour produire un son jusqu'à 10 kHz.

Supprimez les lignes 400-410 et essayez à nouveau ce programme. Les sons deviennent hachés car pendant l'exécution des interruptions, les sons cessent.

La ligne 420 supprime le DMA. Voila pourquoi l'écran se remplit d'une couleur homogène. Cela permet d'accéler le microprocesseur et lui permet de travailler avec une fréquence régulière, le DMA ne venant pas le perturber.

Dans ce programme de démonstration, le son créé a une représentation sinusoïdale. Le signal est remarquablement pur. Une représentation graphique des données donne :




Ce chapitre a passé en revue les différents aspects techniques de la génération de sons avec un ordinateur ATARI. Le programmeur doit aussi comprendre l'intérêt des effets sonores dans son logiciel.

Les auteurs des dessins animés ont compris depuis longtemps l'importance d'une musique de fond. Les récentes avantures de l'espace imaginées par Georges Lucas en donnent des exemples excellents. Lorsque les soldats de l'Empire entrent dans la salle, vous avez immédiatement peur car la musique d'accompagnement devient menaçante. Vous êtes par contre tout heureux lorsque le héros sauve la princesse tandis qu'une musique douce passe en fond. De même, les films d'horreur deviennent très impressionnants grâce à leur musique savamment étudiée même si l'action est en fait banale.



ATARINSIDE

SPACE INVADERS crée une angoisse chez son joueur avec ses bruits de pas qui s'amplifient et résonnent. Quand un Zylon de STAR RAIDERS tire sur vous une torpille photonique, vous appuyez désespérément sur la manette pour éviter l'impact. Au fur et à mesure qu'il s'approche de vous, le temps se ralentit et le bruit des explosions augmente. Juste avant l'impact, vous baissez subitement la tête et vous vous agrippez à votre siège.

Les sons impressionnants affectent notre subconscient et notre état d'esprit. L'image requiert l'attention de l'utilisateur. Si nous cessons de regarder l'image sur l'écran, elle cesse de nous émouvoir. Les sons, par contre, vont plus loin en offrant au programmeur un chemin direct vers le cerveau de l'utilisateur, en évitant toute interprêtation par la pensée. L'impact est donc encore plus important.



ATARINSIDE

# 8 - LE SYSTEME D'EXPLOITATION

## INTRODUCTION

Chaque ordinateur ATARI possède un système d'exploitation : de 10 ko sur les ATARI 400/800, de 16 ko sur les 600XL/800XL. L'importance de ce logiciel est souvent sous-estimée. Sans lui, vous posséderiez un bon nombre de circuits, renfermant une énorme puissance, mais totalement inutilisables. Cette situation se retrouve sur tout ordinateur. A l'origine, un ordinateur se compose de circuits interconnectés les uns aux autres. A la mise sous tension, le microprocesseur doit exécuter un certain nombre de tâches avant de penser traiter le programme de l'utilisateur. Ces tâches sont toujours les mêmes (effacement de la mémoire, initialisation de tous les registres, mise en place par défaut d'une image vidéo, scrutation du clavier, etc) et si tout programmeur devait repartir d'une machine vierge, que de mois de programmation seraient perdus! Ces tâches standards ont en fait été répertoriées, programmées puis mis à demeure dans l'ordinateur. Inutile ainsi de ré-inventer la roue. Ce noyau logiciel porte un nom particulier selon les constructeurs : système d'exploitation, contrôleur, moniteur système, etc. Pour les ordinateurs ATARI, nous parlerons de système d'exploitation, abrégé en SE.

Afin de pouvoir l'étudier plus facilement, commençons par décomposer le système d'exploitation en différentes parties. Pour cela, faisons l'inventaire du matériel (c'est-à-dire des circuits) composant un ordinateur ATARI :

Microprocesseur 6502
Mémoire vive (de taille variable)
ANTIC
CTIA
POKEY
PIA

Par l'intermédiaire de ces circuits, le système d'exploitation inter-agit avec un grand nombre de circuits extérieurs, comme ceux du téléviseur, des unités de disquette, du clavier, etc. Dans cette introduction, nous allons établir la liste des grandes parties du SE.

### LE MONITEUR

Le moniteur du SE est un programme servant lors de la mise en route de l'ordinateur ou lorsqu'on appuie sur la touche SYSTEM RESET. Par ce programme, le SE prend (ou reprend) le contrôle de l'ordinateur. Il le garde jusqu'à ce que le programmeur en décide autrement. Le moniteur gère la mémoire, initialise le sous-système des entrées-sorties, met en place les vecteurs du système (sorte d'aiguillages logiciels placés en mémoire et par lesquels on accède à certains sous-programmes ; la majorité de ces vecteurs peut être modifiée par le programmeur afin de dérouter le 6502 vers de nouveaux sous-programmes), et sélectionne la marche à suivre lorsque l'initialisation est terminée en fonction de l'environement (chargement d'une disquette ou non, d'une cassette ou non, etc).

### LE PROGRAMME DE GESTION DES INTERRUPTIONS

L'ordinateur utilise les interruptions classiques du 6502, avec quelques améliorations augmentant la souplesse du fonctionnement. Les interruptions sont générées par de nombreux événements, comme l'enfoncement d'une touche au clavier, la touche BREAK, certaines opérations du bus série d'entrées/sorties, les compteurs de temps, et le début de la période de Blanking vertical.

### LES VECTEURS DU SYSTEME

Ces aiguillages logiciels permettent au programmeur l'accès au programme constituant le SE, ou autorisent une certaine personnalisation du SE pour des besoins spécifiques. Leur utilisation est la plus fréquente lors des appels des sous-programmes d'entrées/sorties, l'initialisation des compteurs et les déroutements vers des programmes spécifiques. Les vecteurs existent sous deux formes : ceux placés en mémoire morte (dans le SE lui-même) qui contiennent une instruction JMP suivie de l'adresse d'un sous-programme du SE ; ils ne peuvent pas être modifiés. Ceux placés en mémoire vive lors de l'initialisation et qui sont modifiables. Les emplacements de ces vecteurs dans la mémoire sont garantis rester les mêmes dans les futures versions du SE. Par contre, un programme n'utilisant pas ces vecteurs mais accédant directement au sous-programme du SE, a de fortes chances de ne pas pouvoir fonctionner sur les futures versions du SE.

### SOUS-SYSTEME D'ENTREES/SORTIES

Par cet ensemble, le SE permet au programmeur d'accéder aux périphériques de l'ordinateur. Ces sous-programmes forment le lien entre les commandes de haut niveau et les logiciels de gestion des périphériques (que nous abrègeront en LGP dans la suite de ce document ; ce terme est équivalent au mot anglais Handler). Ce sont dans la phase finale les LGP qui contrôlent les circuits électroniques d'entrées/sorties.



ATARINSIDE

### PROGRAMMATION «TEMPS REEL»

L'ordinateur ATARI est conçu pour permettre tout synchronisme entre le programme et l'image vidéo. de plus, l'ordinateur ATARI est équipé d'un certain nombre de compteurs, intégrés dans le POKEY. On peut également mettre en place des compteurs logiciels. Les compteurs électroniques sont des circuits que l'on initialise avec une certaine valeur. Puis le registre contenant cette valeur est décrémenté au rythme déterminé par une horloge issue de l'horloge système. Le décomptage s'arrête lorsque le contenu passe à zéro. Les durées qu'il est ainsi possible d'obtenir s'étalent d'une demi-microseconde à plusieurs secondes. Des compteurs logiciels sont en fait des boucles de programme synchronisées sur la fréquence de Blanking vertical (60 ou 50 Hz), et servant à des tâches très diverses comme par exemple la gestion du bus série ou la synthèse sonore.

### JEU DE CARACTERES EN MEMOIRE MORTE

L'ordinateur est équipé d'un jeu de caractères placé en mémoire morte, utilisé lors de la mise sous tension. Mais il est tout à fait possible de l'inhiber et d'utiliser un jeu de caractères défini par l'utilisateur.

### PROGRAMMES DE CALCUL EN VIRGULE FLOTTANTE

Cet ensemble de programmes étend les possibilités arithmétiques du système. Dans tous les calculs, l'arithmétique décimale codée binaire (DCB ou BCD en anglais) est utilisée pour les fonctions de base (+, −, *, /), l'exponentiation et le logarythme, ainsi que pour les conversions ATASCII/DCB et DCB/ATASCII.

## LE MONITEUR

Le moniteur est cette portion du SE utilisée lors de la mise sous tension du système et d'une action sur la touche SYSTEM RESET. Il donne au SE le contrôle de l'ordinateur et permet d'initialiser correctement chaque paramètre avant de redonner partiellement le contrôle au programme d'application.

Le programme de mise sous tension (encore appelé démarrage à froid ou Coldstart) est exécuté soit lorsqu'on bascule l'interrupteur POWER sur ON, soit lorsque le programme utilisateur passe par le vecteur COLDSV ($E477). Les points importants à mémoriser sont :

1. TOUTE la mémoire vive est effacée sauf les 16 premières adresses ($0000 à $000F).

2. Le système tente de charger une cassette et une disquette. BOOT? ($0009) est un drapeau indiquant le succès ou non de ces tentatives. Bit 0 = 1 pour un chargement réussi de cassette, bit 1 = 1 pour un chargement réussi de disquette.

3. COLDST ($0244) est un drapeau indiquant au moniteur quel type de procédure est en cours. COLDST = 0 indique que SYSTEM RESET a été enfoncée, alors que COLDST <> 0 indique une mise sous tension. Ce drapeau peut servir pour protéger d'une certaine manière un programme : si COLDST reçoit durant l'exécution du programme d'application, une valeur différente de 0, et si la touche SYSTEM RESET est enfoncée, l'ordinateur recommencera la procédure de mise sous tension, évitant ainsi à l'utilisateur de prendre le contrôle du programme d'application.

Appuyer sur SYSTEM RESET provoque un «démarrage à chaud» (ou WARMSTART). Les points importants à mémoriser dans ce cas sont :

1. Les vecteurs en mémoire vive sont initialisés par des valeurs stockées en mémoire morte. Si vous désirez dérouter un vecteur, vous devez penser à l'action que peut avoir SYSTEM RESET. Reportez-vous au paragraphe «Gestion de la mémoire» ci-dessous pour quelques suggestions quant à la manière de procéder.

2. MEMLO, MEMTOP, APPMHI, RAMSIZ et RAMTOP sont ré-initialisés durant un SYSTEM RESET. Si vous voulez modifier ces pointeurs pour réserver de la place pour des programmes Assembleur, vous devez prendre quelques précautions. Le programme de la figure 3 vous montre comment réaliser cette tâche.

Les pages suivantes présentent un organigramme détaillé des séquences de démarrage à froid et à chaud.


ATARINSIDE

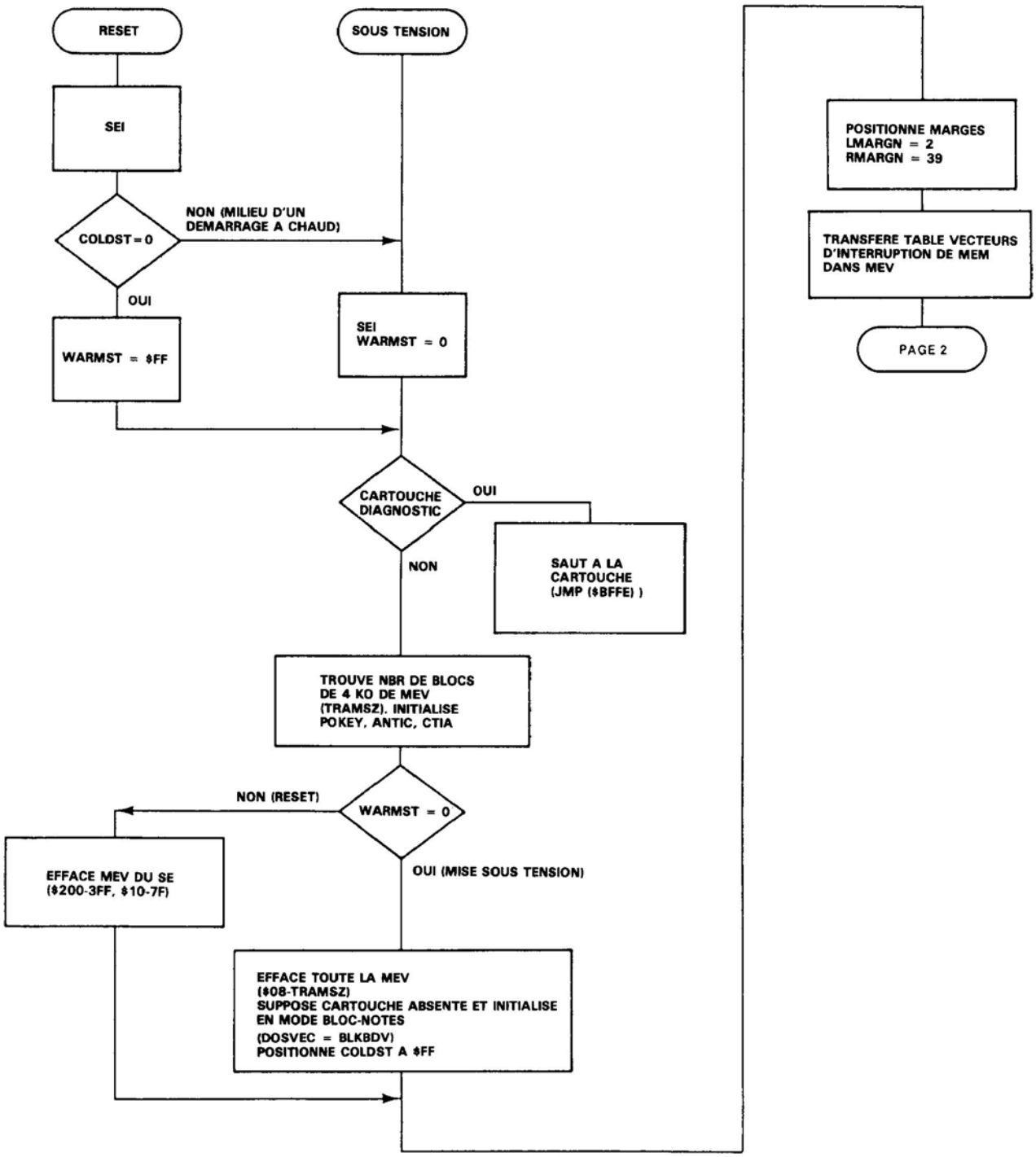



Figure 8-1 - Initialisation du système



ATARINSIDE

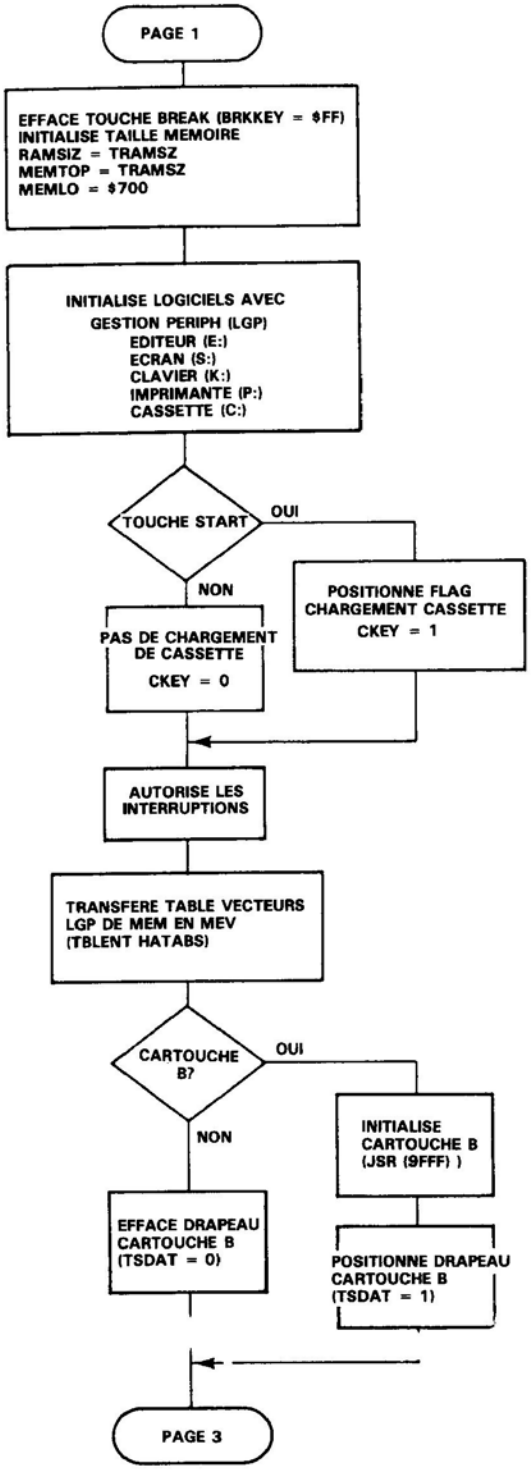



Figure 8-1.2 - Initialisation du système

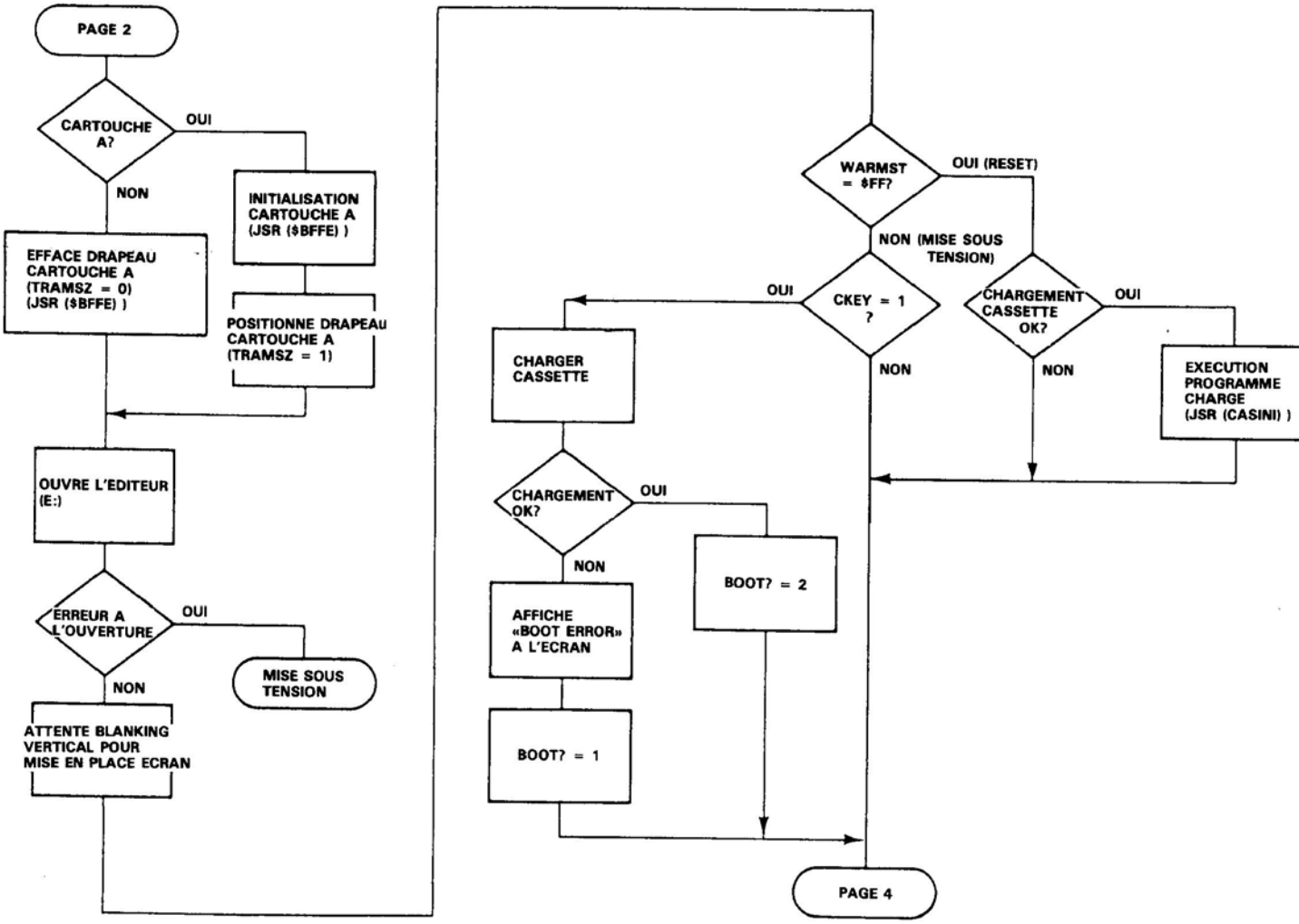



Figure 8-1.3 - Initialisation du système



ATARINSIDE

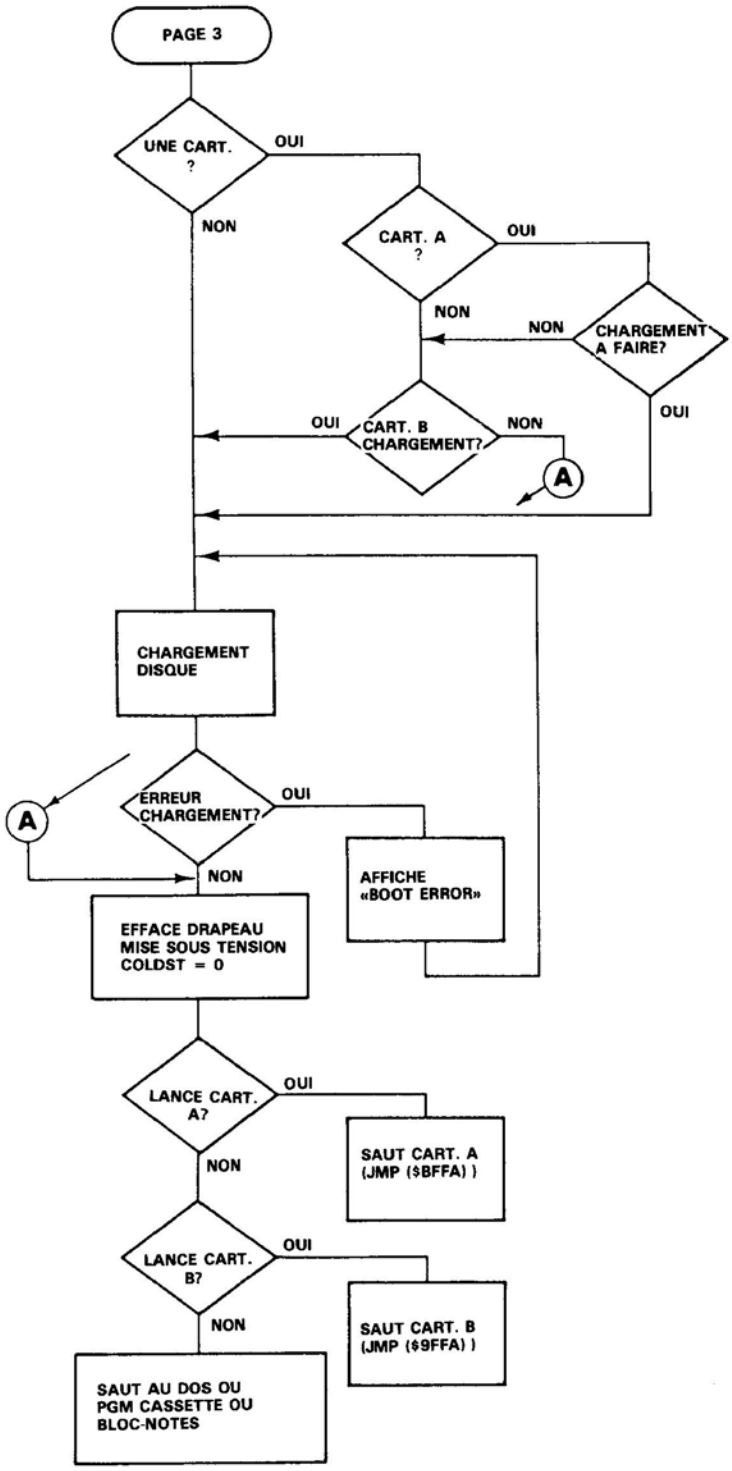



Figure 8-1.3 - Initialisation du système



ATARINSIDE

# GESTION DE LA MEMOIRE

Le fait que le SE soit écrit pour un microprocesseur 6502 a imposé certaines décisions dans l'organisation de la mémoire. Dans un 6502, trois régions de l'espace mémoire sont remarquables. La page 0 est particulièrement importante compte tenu des modes d'adressage spécifiques la concernant. Un programme exploitant des données placées dans cette zone s'exécutera plus rapidement. De plus, certaines instructions nécessitent obligatoirement des adresses en page 0 pour s'exécuter. La page 1 constitue la pile du 6502. Enfin, les adresses $FFFA à $FFFF sont réservées aux circuits de RESET ou aux vecteurs d'interruptions.

En conséquence, la mémoire morte occupera le haut de la mémoire. Le SE s'étend de $D800 à $FFFF (A800/A400). Juste en dessous de cette zone, nous trouvons les adresses des registres des circuits spéciaux : ANTIC, CTIA, POKEY de $D000 à $D7FF.

A l'autre extrémité de la mémoire, le SE utilise la moitié de la page 0 ainsi que les pages 2 à 5. Pour un programmeur, l'espace utilisable maximum va de $0600 à $BFFF.

Lorsque le système est mis sous tension, l'une des premières actions réalisée par le SE est de déterminer la taille de la mémoire vive présente dans l'ordinateur. Cela est accompli en testant le premier octet de chaque bloc de 4 ko de mémoire, en commençant à l'adresse $1000. Le contenu de cet octet est lu, complété, et une tentative d'écriture a lieu à la même adresse. Si cette tentative réussit, le compteur temporaire est incrémenté de 4. Ce processus continu jusqu'à trouver une adresse dont le contenu ne peut être changé. Deux variables, RAMTOP et RAMSIZ contiennent le nombre de pages de MEV présentes. En outre, les pointeurs MEMLO, MEMTOP et APPMHI sont gérés par les sous-programmes du SE. Les relations existantes entre ces pointeurs sont indiquées sur la figure 8-2 représentant une cartographie de la mémoire.

MEMLO est une adresse sur deux octets que le SE utilise pour indiquer où un programme d'application peut commencer. Vous pouvez modifier MEMLO pour créer des zones réservées à des programmes en langage machine. Basic utilise la valeur dans MEMLO pour déterminer l'adresse de départ d'un programme (reportez-vous au chapitre 10 pour une étude de la structure d'un programme Basic). Si la valeur de MEMLO doit être incrémentée, cela doit être fait avant que la cartouche Basic ne prenne la main. C'est une opération délicate car MEMLO retrouve sa valeur initiale lors d'une action sur SYSTEM RESET.

Si le programme d'application est placé sur une disquette, l'utilisation d'un fichier AUTORUN.SYS peut permettre de modifier MEMLO avant le chargement du programme. Toutefois, le SED (Système d'Exploitation de la Disquette) est également initialisé durant un SYSTEM RESET via le vecteur DOSINI ($000C). Ce vecteur contient l'adresse du programme d'initialisation du SED mais c'est également le seul point où vous pouvez dérouter la séquence de démarrage à chaud. Puisque l'initialisation du SED doit pouvoir se faire sans tenir compte de ce qui se passe pour MEMLO, vous devez d'abord autoriser l'initialisation normale du SED avant de modifier DOSINI. Cela s'effectue en recopiant les deux octets de DOSINI dans une instruction JSR faisant partie du fichier AUTORUN.SYS. Juste après l'instruction JSR, placez les instructions qui modifieront MEMLO. Terminez avec une instruction RTS. DOSINI doit ensuite recevoir l'adresse de l'instruction JSR. Ainsi, lorsqu'un démarrage à chaud se produit, la nouvelle séquence est appelée et l'instruction JSR est d'abord exécutée. Puis MEMLO est repositionné selon vos désirs et l'initialisation se poursuit normalement. La figure 8-3 en donne un exemple.

Cette technique peut également être utilisée avec MEMTOP, le pointeur utilisateur du haut de la mémoire. Ce pointeur indique la plus haute adresse accessible pour un programme d'application ; elle diffère de la plus haute adresse physique car le SE se réserve de la place pour la mémoire écran et la Display List. Vous pouvez faire décroître MEMTOP pour réserver de la place à des programmes Assembleur. Toutefois, MEMTOP dépend de la taille mémoire de l'ordinateur et du mode graphique sélectionné. En conséquence, vous ne pouvez pas toujours connaître à l'avance la valeur de MEMTOP et cette incertitude vous oblige à concevoir des programmes Assembleur relogeables pour pouvoir être implantés au-dessus de MEMTOP.

APPMHI contient une adresse que le SE considèrera comme une limite inférieure autorisée de la mémoire écran. Ainsi, le LGP de l'écran ne pourra pas venir interférer avec votre programme ou vos données.

RAMSIZ, comme MEMTOP, peut aussi servir à réserver de la place en mémoire. Comme RAMSIZ se compose d'un seul octet contenant le nombre de pages mémoire disponibles, décrémenter sa valeur de 1 réserve une place de 256 octets. L'avantage d'utiliser RAMSIZ au lieu de MEMTOP est que l'espace réservé par RAMSIZ se situe au-dessus de la mémoire écran alors que MEMTOP se situe en-dessous. En conséquence, la zone réservée par RAMSIZ n'est pas perturbée par une modification de l'écran.

ATARINSIDE

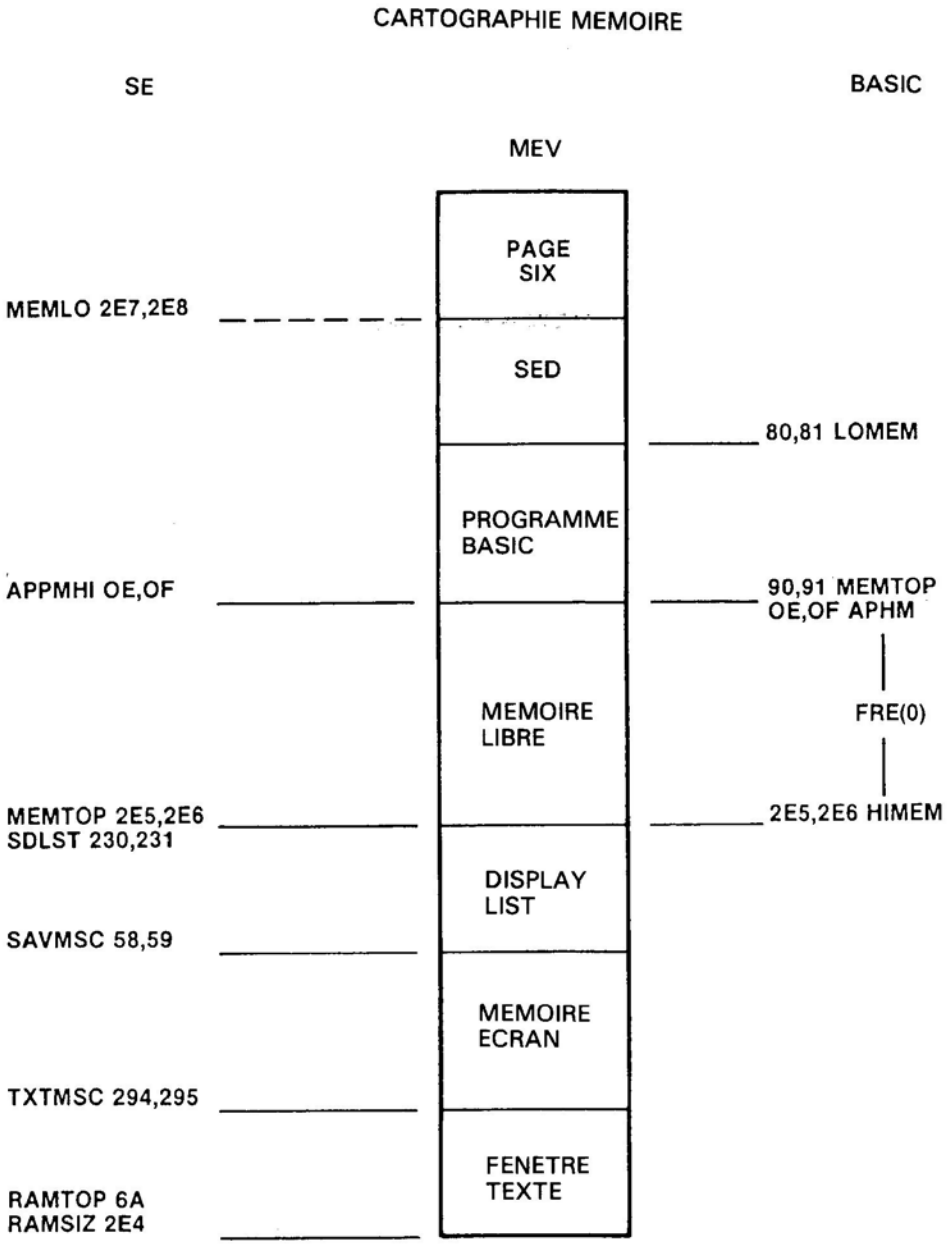



Figure 8-2 - Pointeurs SE et Basic (le SED est présent)



ATARINSIDE

```
                0010 ; MODIFIE MEMLO
                0020 ;
0600            0030 START    =    $600
000C            0040 DOSINI   =    $0C
02E7            0050 MEMLO    =    $2E7
3000            0060 NEWMEM   =    $3000      ; CECI EST LA NOUVELLE VALEUR DE MEMLO
                0065 ;
                0070 ; CE PROGRAMME RESERVE DE LA PLACE POUR DES SOUS-PROGRAMMES ASSEMBLEU
                0090 ; EN MODIFIANT LA VALEUR DE MEMLO. IL S'EXECUTE
                0100 ; COMME UN FICHIER AUTORUN.SYS ET TIENT COMPTE DE RESET.
                0120 ; MEMLO RECOIT LA VALEUR DE NEWMEM.
                0130 ;
                0140 ; CETTE PARTIE EST PERMANENTE, CAD RESIDENTE
                0150 ; LE VECTEUR DOSINI A ETE DEROUTE ET ENREGISTRE
                0160 ; DANS L'OPERANDE DE L'INSTRUCTION JSR TROJAN.
                0170 ; QUAND RESET EST APPPUYE, DOSINI POINTE VERS INITDOS,
                0180 ; JSR TROJAN APPELLE LES PROGRAMMES D'INITIALISATION DU SED,
                0185 ; MEMLO RECOIT LA NOUVELLE VALEUR ET LE MONITEUR
                0190 ; REPREND LA MAIN
0000            0200         *=   START
                0210 INITDOS
0600 200D06     0220         JSR  TROJAN    ; INITIALISE LE SED
0603 A900       0230         LDA  #NEWMEM&255
0605 8DE702     0240         STA  MEMLO
0608 A930       0250         LDA  #NEWMEM/256
060A 8DE802     0260         STA  MEMLO+1
                0270 TROJAN
060D 60         0280         RTS
                0290 ; CETTE PARTIE S'EXECUTE A LA MISE SOUS TENSION SEULEMENT ET
                0300 ; PEUT ETRE EFFACEE ENSUITE.
                0330 ; CE PROGRAMME RECOPIE LA VALEUR DE DOSINI DANS L'ARGUMENT
                0350 ; DE JSR TROJAN. PUIS PLACE L'ADRESSE D'INITDOS
                0370 ; DANS DOSINI.
                0390 GRABDOSI
060E A50C       0400         LDA  DOSINI          ; SAVE DOSINI
0610 8D0106     0410         STA  INITDOS+1
0613 A50D       0420         LDA  DOSINI+1
0615 8D0206     0430         STA  INITDOS+2
0618 A900       0440         LDA  #INITDOS&255    ; SET DOSINI
061A 850C       0450         STA  DOSINI
061C A906       0460         LDA  #INITDOS/256
061E 850D       0470         STA  DOSINI+1
0620 A500       0480         LDA  NEWMEM&255      ; SET MEMLO
0622 8DE702     0490         STA  MEMLO
0625 A930       0500         LDA  #NEWMEM/256
0627 8DE802     0510         STA  MEMLO+1
062A 60         0520         RTS
062B            0530         *=   $2E2
02E2 0E06       0540         .WORD GRABDOSI       ; SET RUN ADDRESS
02E4            0550         .END
```

Figure 8-3 - Modification de MEMLO



ATARI INSIDE

# GESTION DES INTERRUPTIONS

La possibilité de répondre sélectivement à des événements particuliers matériels et logiciels (c'est-à-dire aux interruptions), fournit une grande souplesse à tout ordinateur qui en est pourvu. Le 6502 admet deux types d'interruptions : interruptions masquables (IRQ) et interruptions non masquables (NMI). Un niveau supplémentaire d'interruption est donné par l'ANTIC, le POKEY et le PIA. Chacun de ces boîtiers est responsable d'un certain nombre d'événements pouvant provoquer des interruptions. Si une interruption particulière est autorisée au niveau de ces boîtiers, elle peut alors parvenir au 6502. L'ANTIC gère les requêtes NMI tandis que le POKEY et le PIA s'occupent des IRQ. Les interruptions suivantes sont disponibles :

| Nom (vecteur) | Type | Fonction | Utilisé par |
|---|---|---|---|
| DISPLAY LIST (VDSLST) | NMI | Cadencement graphique | Utilisateur |
| SYSTEM RESET (aucun) | NMI | Initialisation | SE |
| BLANKING VERTICAL (VVBLKI, VVBLKD) | NMI | Affichage images | SE, Utilisateur |
| Entrée série prête (VSERIN) | IRQ | Entrée série | SE |
| Sortie série prête (VSEROR) | IRQ | Sortie série | SE |
| Sortie série terminée (VSEROC) | IRQ | Sortie série | SE |
| Compteur POKEY 1 (VTIMR1) | IRQ | Circuit compteur | Utilisateur |
| Compteur POKEY 2 (VTIMR2) | IRQ | Circuit compteur | Utilisateur |
| * Compteur POKEY 4 (VTIMR4) | IRQ | Circuit compteur | Utilisateur |
| Clavier (VKEYBD) | IRQ | Touche enfoncée | SE |
| * Touche BREAK (BRKKY) | IRQ | BREAK enfoncée | SE |
| Bus série demandé (VPRCED) | IRQ | Périphérique utilise bus série | Inutilisé |
| Interruption bus série (VINTER) | IRQ | Int. demandée par périph. | Inutilisé |

*Ces vecteurs n'existent que dans la version B du système d'exploitation.

Le fait de modifier les vecteurs des interruptions demande un peu d'attention. Par exemple, si vous inhibez accidentellement l'interruption du clavier, l'ordinateur ignorera toutes les touches sauf BREAK. Bien que cela puisse être parfois utile, cela peut compliquer la mise au point de votre programme.

### LE PROGRAMME DE GESTION DES INTERRUPTIONS

Le SE possède un programme particulier pour gérer les différentes interruptions non masquables. Ce programme possède des vecteurs en MEV pour toutes les IRQ (sauf pour la touche BREAK dans la première version du SE). Les vecteurs IRQ sont initialisés durant la mise sous tension et par SYSTEM RESET. Les fonctions de ces vecteurs sont :

VIMIRQ — Vecteur IRQ immédiat. Toutes les interruptions passent par cette adresse. VIMIRQ pointe le début du programme de gestion des interruptions. Vous pouvez modifier ce vecteur si vous désirez traiter vous-même les IRQ.

VSEROR — Ce vecteur pointe normalement vers le sous-programme fournissant l'octet suivant d'une zone tampon au port de sortie série.

VSERIN — Ce vecteur pointe normalement vers le sous-programme transférant un octet du port d'entrée série vers une zone tampon.

VSEROC — Ce vecteur pointe normalement vers le sous-programme qui positionne le drapeau «transmission terminée» après que l'octet de checksum ait été envoyé.

VTIMR1 — Vecteur du compteur de temps numéro 1 initialisé pour pointer vers une séquence PLA, RTI. Utilisé lorsque le contenu du compteur passe par 0.

VTIMR2 — Idem pour compteur de temps numéro 2.

VTIMR4 — Idem pour compteur de temps numéro 4.

VKEYBD — Vecteur d'interruption en provenance du clavier. Le fait d'appuyer sur une touche (sauf BREAK) provoque cette interruption. Ce vecteur pointe normalement vers le programme de gestion de l'interruption du clavier. Vous pouvez le dérouter pour traiter le code de la touche avant qu'il soit transformé en ATASCII par le SE.

BRKKY — Vecteur de la touche BREAK. Dans la version B (et suivantes), du SE, cette interruption a son propre vecteur. Il est initialisé pour pointer vers une séquence PLA, RTI.

VPRCED — La ligne PROCEED est accessible par les périphériques sur le bus série. Cette interruption est actuellement inutilisée et pointe vers une séquence PLA, RTI.

VINTER — La ligne d'interruption est également accessible par le bus série. VINTER pointe vers une séquence PLA, RTI.



ATARINSIDE

VBREAK — Vecteur de l'instruction BRK du 6502. Chaque fois qu'une instruction BREAK ($00) est exécutée, cette interruption est générée. VBREAK peut être utilisé pour mettre en place des points d'arrêt pour un outil de déverminage. Il pointe normalement vers une séquence PLA, RTI.

Les interruptions sont autorisées ou non respectivement par les instructions CLI et SEI du 6502. Les interruptions possèdent également des bits individuels d'autorisation/inhibition dans le POKEY.

Le registre IRQEN (53774) est un registre qui, utilisé en écriture, autorise ou non certaines interruptions. Le SE utilise une mémoire cache de ce registre s'appelant POKMSK ($10) mais IRQEN n'est pas mis à jour durant chaque période de Blanking vertical. Chaque interruption est utilisée lorsque le bit correspondant est à 1 dans le registre IRQEN. Le même registre utilisé en lecture (IRQST) indique la cause de l'interruption en positionnant le ou les bits concernés à zéro.

PACTL (54018) et PBCTL (54019) sont utilisés entre autre pour autoriser et tester les interruptions du PIA. Le bit 0 de chacun de ces registres constitue l'autorisation d'interruption lorsqu'il vaut 1. Le bit 7 représente l'état de l'interruption (vaut 1 lorsqu'une interruption a eu lieue). ce bit est remis à zéro chaque fois que le registre est lu.

**UTILISATION DES INTERRUPTIONS**

La présence des vecteurs IRQ vous permet d'adapter les entrées/sorties du système à vos besoins. Actuellement, le SE ne permet pas le contrôle des entrées/sorties par les périphériques. En modifiant les 3 vecteurs du bus série, il devient possible de ré-écrire un logiciel d'entrée/sortie permettant cela. Les interruptions générées par les 3 compteurs à rebours autorisent un contrôle précis d'événements ou de boucles de logiciels. Ces compteurs sont normalement utilisés lorsque la fréquence du Blanking vertical (50 ou 60 Hz) est trop faible pour la tâche considérée. Reportez-vous au paragraphe «Programmation en temps réel» pour obtenir des renseignements complémentaires.

De nombreuses applications demandent que les programmes soient protégés contre des erreurs de manipulation. Les vecteurs IRQ peuvent être utilisés dans ce sens. L'exemple de la figure 8-4 montre comment inhiber la touche CTRL en utilisant le vecteur VKEYBD. Ce programme masque également la touche BREAK en déroutant le vecteur VIMIRQ et en ignorant l'interruption de la touche BREAK. Bien qu'écrit pour la version originale du SE, ce programme fonctionne également avec les autres versions.

Deux interruptions sont supportées par le PIA, VPRCED et VINTER. Elles sont inutilisées par le SE mais peuvent être exploitées pour de meilleurs échanges avec des périphériques futurs extérieurs.

**LE PROGRAMME DE GESTION DES NMI**

Le SE possède un programme traitant les interruptions non masquables. Contrairement aux IRQ, les NMI ne peuvent pas être inhibées au niveau du 6502. Par contre, elles peuvent l'être par l'ANTIC, sauf SYSTEM RESET.

Deux NMI, l'interruption de Display List (DLI) et celle de Blanking vertical (VBLANK) ont des vecteurs en MEV. En fait, VBLANK peut être intercepté à deux endroits : VBLANK immédiat ou VBLANK différé. Les vecteurs NMI sont :

| Nom | Vecteur |
|---|---|
| SYSTEM RESET | aucun |
| DISPLAY LIST | VDSLST ($0200) |
| BLANKINK VERTICAL IMMEDIAT | VVBLKI ($0222) |
| BLANKING VERTICAL DIFFERE | VVBLKD ($0224) |

L'interruption non masquable générée par SYSTEM RESET ne possède pas de vecteur en mémoire vive. Le fait d'enfoncer cette touche provoque toujours un saut au sous-programme de démarrage à chaud. Toutefois, le vecteur DOSINI placé en mémoire vive est utilisé durant l'exécution de ce sous-programme et peut donc être exploité pour inhiber SYSTEM RESET (reportez-vous au paragraphe concernant le moniteur).

Le vecteur des interruptions NMI des Display List est inutilisé par le SE. Reportez-vous au chapitre 5 pour obtenir davantage d'explications sur le fonctionnement et l'utilisation de ce vecteur.

**DEROULEMENT D'UNE INTERRUPTION DE BLANKING VERTICAL (VBI)**

Une interruption est générée à chaque fin d'image c'est-à-dire à chaque période de Blanking vertical. Cela constitue une ressource très intéressante pour un programmeur. Ces interruptions non masquables apparaissent donc à un intervalle de temps régulier (tous les 60e de seconde pour le standard NTSC, tous les 50e de seconde pour les standards PAL et SECAM). De plus, ces interruptions se produisent lorsque le spot est sorti de l'écran (l'image est terminée) si bien que toute modification des paramètres de l'image ayant lieu à cet instant n'apparaîront pas immédiatement à l'affichage : il faut que le spot revienne à l'écran pour l'image suivante ; cela permet des modifications synchrones et «propres».

Lorsqu'une interruption de Blanking vertical a été reconnue par le SE, le contenu des registres A, X et Y est passé dans la pile. L'exécution du programme principal est alors suspendue et le microprocesseur est dérouté via le vecteur de «Blanking vertical immédiat» (VVBLKI) situé à l'adresse $0222. Ce vecteur pointe normalement vers le sous-programme gérant l'interruption de Blanking vertical commençant à l'adresse $E45F (ordinateurs 400 et 800). Dans ce sous-programme, le SE incrémente l'horloge temps réel d'une unité (50e de seconde), décrémente les compteurs de temps, gère le mode attente (rotation des couleurs au-delà de 10 minutes écoulées sans frappe au clavier), recopie les registres cache dans les registres des circuits, et teste les données des manettes de jeu. Ce sous-programme se termine en passant par le vecteur de «Blanking vertical différé» (VVBLKD), situé à l'adresse $0224. Ce vecteur est initialisé pour pointer vers un simple retour d'interruption situé à l'adresse $E462. La figure 8-5 illustre ce processus.



ATARINSIDE

```
0010          10 POKMSK   =     $0010
D209          20 KBCODE   =     $D209
0208          30 VKEYBD   =     $0208
D20E          40 IRQEN    =     $D20E
D20E          45 IRQST    =     IRQEN
0216          46 VMIRQ    =     $0216
0000          60          *=    $600
0600 78       80 START    SEI                 INHIBE IRQ
0601 AD1602   90          LDA     VMIRQ       REMPLACE LE VECTEUR IRQ
0604 8D4D06   0100        STA     NBRK+1      AVEC LE NOTRE
0607 AD1702   0110        LDA     VMIRQ+1     TOUTES LES INTERRUPTIONS IRONT
060A 8D4E06   0120        STA     NBRK+2      A NBRK
060D A945     0130        LDA     #IRQ&255
060F 8D1602   0140        STA     VMIRQ
0612 A906     0150        LDA     #IRQ/256
0614 8D1702   0160        STA     VMIRQ+1
0617 AD0802   0200        LDA     VKEYBD      POINTE VERS REP
061B 8D4306   0210        STA     JUMP+1
061E AD0902   0220        LDA     VKEYBD+1
0621 8D4406   0230        STA     JUMP+2
0624 A939     0240        LDA     #REP&255    L'INTERRUPTION DU CLAVIER
0626 8D0802   0250        STA     VKEYBD
0629 A906     0260        LDA     #REP/256
062B 8D0902   0270        STA     VKEYBD+1
062E 58       0170        CLI                 AUTORISE LES INTERRUPTIONS
062F 60       0280        RTS
              0290        *=$639
0639 AD09D2   0300 REP    LDA     KBCODE      TOUTES LES INTERRUPTIONS DU CLAVIER
063C 2980     0310        AND     #$80        TESTE SI CTRL          VIENNENT ICI
063E F002     0320        BEQ     JUMP        NON
0640 68       0330        PLA                 SI ; ALORS IGNORE
0641 40       0340        RTI
0642 4C4206   0360 JUMP   JMP     JUMP        APPEL DE L'ANCIEN VECTEUR
0645 48       0375 IRQ    PHA                 TOUTES LES INTERRUPTIONS VIENNENT ICI
0646 AD0ED2   0380        LDA     IRQST       TESTE SI BREAK
0649 1004     0390        BPL     BREAK       SI BREAK, BRANCHEMENT
064B 68       0405        PLA                 SI NON, APPEL DE L'ANCIEN VECTEUR
064C 4C4C06   0410 NBRK   JMP     NBRK        APPEL DE L'ANCIEN VECTEUR
064F A97F     0430 BREAK  LDA     #$7F        ICI, SI BREAK
0651 8D0ED2   0440        STA     IRQST       INHIBE BREAK
0654 A510     0450        LDA     POKMSK
0656 8D0ED2   0460        STA     IRQEN
0659 68       0462        PLA
065A 40       0464        RTI                 RETOUR COMME S'IL N'Y AVAIT PAS EU BREAK
065B          0470        *=    $02E2
02E2 0006     0480        .WORD START
```

Figure 8-4 - Protection contre les touches CTRL et BREAK



ATARINSIDE

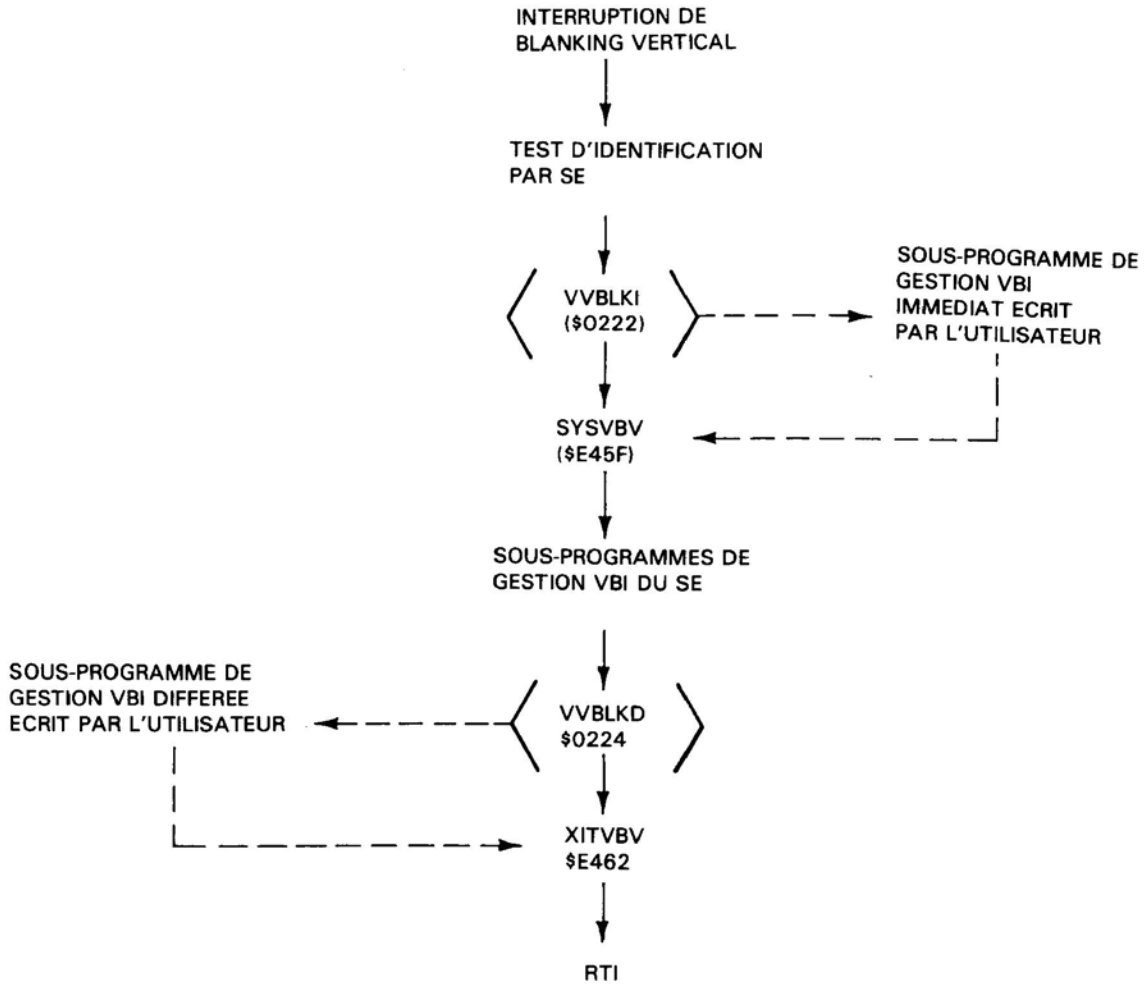



Figure 8-5 - Exécution d'une VBI

Ces deux vecteurs sont placés en mémoire vive pour permettre au programmeur de dérouter le SE à cet endroit et d'utiliser cette interruption générée à intervalle de temps régulier pour leurs propres besoins. La procédure à suivre est très simple. Tout d'abord, décidez si vous utilisez l'interruption immédiate ou différée. Généralement, la différence est faible. Mais dans certains cas, il faut choisir avec soin. Lorsque votre programme de VBI lit ou modifie des registres eux-mêmes modifiés par le programme VBI du SE, il peut être nécessaire d'écrire dans ces registres après le SE : le dernier qui écrit a raison!

Le second cas se produit lorsque votre sous-programme utilise trop longtemps le microprocesseur. Le sous-programme du SE peut alors être retardé au delà de la fin de la période de Blanking vertical. Des registres risquent alors d'être modifiés alors que le faisceau d'électrons a déjà commencé le tracé de l'image suivante. Si c'est le cas, votre programme doit utiliser le mode différé ; il est ainsi placé après le sous-programme VBI du SE (il utilise le vecteur VVBLKD). La limite de temps en mode immédiat est d'environ 2000 cycles machines. Dans le mode différé, elle est d'environ 20 000 cycles. Toutefois, la majorité de ces cycles est exécutée tandis que le faisceau d'électrons a commencé une nouvelle image. Aussi, les modifications des registres du CTIA ou de l'ANTIC doivent utiliser peu de pas de programme. N'oubliez pas que l'exécution des programmes de VBI diminue le temps disponible pour le programme principal.

Le troisième cas se produit lorsque votre propre sous-programme doit s'exécuter simultanément avec des opérations d'entrées/sorties critiques en temps comme par exemple les transferts avec l'unité de disquette ou le magnétocassette. Le sous-programme du système d'exploitation VBI est divisé en deux parties : l'une, dite critique, est la plus courte et n'effectue que les opérations indispensables. L'autre, non critique, complète normalement la VBI. Lorsqu'une opération d'entrées/sorties est en court, un drapeau est positionné (CRITIC, adresse $42, valeur différente de 0) et le SE n'exécute que la partie critique de la VBI. Si vous souhaitez que votre programme VBI s'exécute durant chaque période de Blanking vertical, vous devez donc le placer avant la routine VBI du SE, donc en mode immédiat. Toutefois, soyez sur vos gardes car vous risquez de perturber les opérations d'entrées/sorties.

Lorsque vous avez choisi entre le mode immédiat et le mode différé, vous devez placer votre programme en mémoire. Il doit se terminer en pointant vers l'adresse adéquate du SE (SYSVBV ou XITVBV). Enfin, vous devez modifier le vecteur d'appel (VVBLKI ou VVBLKD) placé en mémoire vive. Dès ce moment, le SE passera dans votre programme de VBI à chaque image. Si vous désirez ignorer totalement le programme VBI du SE, placez votre programme en mode immédiat et finissez-le par l'instruction JMP $E452.



ATARINSIDE

Un problème que l'on rencontre avec tous les microprocesseurs 8 bits apparaît lorsque l'on modifie le vecteur d'une interruption. En effet, les vecteurs sont des valeurs sur deux octets (16 bits d'adresse) ; il faut donc deux instructions de sauvegarde en mémoire pour les modifier. En conséquence, il existe une petite chance pour qu'une interruption se produise juste entre les deux instructions de sauvegarde. Le vecteur, à moitié modifié, pointe donc vers une mauvaise adresse, ce qui enverra le système dans les choux! Pour résoudre cet éventuel conflit, un petit sous-programme a été mis en place dans le SE. Il s'appelle SETVBV et commence à l'adresse $E45C. Il vous suffit de charger le registre Y du 6502 avec l'octet de poids faible de l'adresse, le registre X avec le poids fort et l'accumulateur avec la valeur 6 (pour modifier le vecteur immédiat VVBLKI) ou 7 (pour le vecteur différé VVBLKD). Exécutez ensuite un JSR SETVBV et votre programme de VBI sera correctement mis en place. Son exécution commencera au plus 1/50e (ou 1/60e selon le TV) de seconde plus tard.

Un grand nombre d'applications reste envisageable pour ces interruptions VBI. Tout d'abord, la manipulation des registres couleur peut être réalisée alors que le spot n'est pas sur l'écran. Deuxièmement, on peut en profiter pour changer d'écran (commutation d'écrans à chaque VBI pour augmenter les couleurs et les objets).

Vous pouvez mettre à profit ces interruptions pour une modification des registres son en temps réel. Vous pouvez régler précisément la durée de chaque note, un compteur étant décrémenté chaque fois qu'une VBI est exécutée. Utilisez cette technique de VBI pour modifier l'intensité sonore de chaque canal : vous transformez ainsi l'enveloppe de la note (attaque et décroissance). De même, un réglage plus fin de la fréquence et de la distorsion devient possible. Les effets sonores obtenus sont plus larges. Toutefois, la fréquence des VBI n'est pas suffisante pour exploiter correctement le mode «intensité seulement» des registres sons.

Les VBI sont également très utiles pour rester en contact avec l'utilisateur : scruter les claviers et les ports de joystick. Ce traitement nécessite peu de gestion, mais une attention constante. Une VBI est parfaite pour cela. Il devient possible de continuer un long calcul sans oublier l'utilisateur.

Enfin, une bonne gestion de VBI autorise un fonctionnement multitâches, c'est-à-dire que deux programmes indépendants peuvent tourner en mémoire. Il faut bien sûr veiller à bien séparer dans ce cas les données de chaque programme et la puissance obtenue par une telle gestion récompense bien l'auteur de ses efforts.

## VECTEURS DU SYSTEME

Une caractéristique de la puissance de n'importe quel système d'exploitation réside dans sa faculté d'adaptation. Un utilisateur peut-il facilement exploiter les programmes du SE ou les compléter par ses propres programmes?

De ce point de vue, le SE des ordinateurs ATARI est particulièrement performant. En pratique, chaque fois qu'il pouvait être intéressant d'accéder à un sous-programme du SE, le SE utilise un vecteur.

Cette souplesse est fournie par une combinaison de plusieurs mécanismes différents. Le premier est une table, placée en mémoire morte, d'instructions JMP pointant vers des sous-programmes très importants du SE. Dans les futures versions du Système d'exploitation, l'adresse de cette table ne changera pas tandis que les valeurs des pointeurs pourront être modifiées. Aussi, votre programme doit utiliser cette table comme un aiguillage vers les sous-programmes du SE. Tout accès direct rendra votre programme obsolète sur les futures versions du SE.

Le second mécanisme se compose d'un ensemble de vecteurs placés en mémoire vive enchaînant les différentes parties des programmes de gestion des interruptions. Pour modifier le traitement, ouvrez le maillon voulu en changeant la valeur d'un vecteur, de manière à ce qu'il pointe vers votre programme. Le SE initialise ces vecteurs lors d'une mise sous tension. Là également, bien que la valeur initiale de ces vecteurs puisse changer d'une version du SE à une autre, leur adresse en mémoire restera fixe.

Le troisième mécanisme est la table des pointeurs vers les LGP (Logiciels de Gestion des Périphériques) : la table des «handlers». Pour plus de détails, reporez-vous au paragraphe intitulé «Entrées/sorties centralisées», dans ce même chapitre.

| Nom | Adresse | Description |
|---|---|---|
| DISKIV | $E450 | Initialisation du LGP disque |
| DSKINV | $E453 | Vecteur du LGP disque |
| CIOV | $E456 | Vecteur du programme d'entrées/sorties centralisées (CIO) |
| SIOV | $E459 | Vecteur du programme des entrées/sorties série (SIO) |
| SETVBV | $E45C | Vecteur du programme des compteurs de temps |
| SYSVBV | $E45F | Traitement de l'interruption NMI de VBI |
| XITVBV | $E462 | Fin du traitement de VBI |
| SIOINV | $E465 | Initialisation du SIO |
| SENDEV | $E468 | PGM d'émission sur le bus série |
| INTINV | $E46B | Traitement général des interruptions |
| CIOINV | $E46E | Initialisation CIO |
| BLKBDV | $E471 | Vecteur du mode bloc-notes ou test |
| WARMSV | $E474 | Point d'entrée démarrage à chaud (SYSTEM RESET) |
| COLDSV | $E477 | Point d'entrée démarrage à froid (mise sous tension) |
| RBLOKV | $E47A | Vecteur du PGM de lecture magnétocassette (1 bloc) |
| CSOPIV | $E47D | Vecteur du PGM ouverture d'un canal en entrée pour cassette |



ATARINSIDE

Rappelons que cette table se compose de groupes de trois octets représentant des instructions JMP. Voici un exemple d'utilisation de vecteur :

```
JSR  CIOV
```

## VECTEURS EN MEMOIRE VIVE

| Nom | Adresse | Valeur | Description |
|---|---|---|---|
| | — Page 2 — | | |
| VDSLST | $0200 | $E7B3 | Vecteur NMI Interruption de Display List |
| VPRCED | $0202 | $E7B3 | Vecteur IRQ Ligne «PROCEED» du bus série (inutilisé) |
| VINTER | $0204 | $E7B3 | Vecteur IRQ Ligne «Interruption» du bus série (inutilisé) |
| VBREAK | $0206 | $E7B3 | Vecteur interruption produite par instruction BRK |
| VKEYBD | $0208 | $FFBE | Vecteur interruption du clavier |
| VSERIN | $020A | $EB11 | Vecteur IRQ Réception d'un octet sur bus série |
| VSEROR | $020C | $EA90 | Vecteur IRQ Emission d'un octet sur bus série |
| VSEROC | $020E | $EAD1 | Vecteur IRQ Emission sur bus série terminée |
| VTIMR1 | $0210 | $E7B3 | Vecteur IRQ Compteur temps numéro 1 |
| VTIMR2 | $0212 | $E7B3 | Vecteur IRQ Compteur temps numéro 2 |
| VTIMR4 | $0214 | $E7B3 | Vecteur IRQ Compteur temps numéro 4 |
| VIMIRQ | $0216 | $E6F6 | Vecteur pointant vers le PGM gestion des IRQ |
| VVBLKI | $0222 | $E7D1 | Vecteur mode immédiat de l'interruption NMI de VBI |
| VVBLKD | $0224 | $E93E | Vecteur mode différé de VBI |
| CDTMA1 | $0226 | $xxxx | JSR adresse du compteur 1 |
| CDTMA2 | $0228 | $xxxx | JSR adresse du compteur 2 |
| BRKKY | $0236 | $E754 | Vecteur de la touche BREAK (REV. B) |
| RUNVEC | $02E0 | $xxxx | Vecteur adresse RUN d'un PGM binaire |
| INIVEC | $02E2 | $xxxx | Vecteur adresse INIT d'un PGM binaire |
| | — Page 0 — | | |
| CASINI | $0002 | $xxxx | Vecteur pour initialisation PGM cassette à chargement automatique |
| DOSINI | $000C | $xxxx | Vecteur initialisation disque |
| DOSVEC | $000A | $xxxx | Vecteur d'appel du DOS |

Un x indique que la valeur de cette adresse peut changer.

Contrairement à la table de JMP placée en mémoire morte, ces routines sont réellement des adresses sur deux octets qu'on utilise par un saut indirect ; par exemple :

```
      JSR  CALL
CALL  JMP  (DOSINI)
```

## ENTREES/SORTIES CENTRALISEES : LE C.I.O.

L'un des problèmes les plus délicats à résoudre pour le concepteur d'un SE est la manière de gérer les entrées/sorties compte tenu de la grande variété de périphériques pouvant se connecter au système. Les quelques règles générales de base sont :

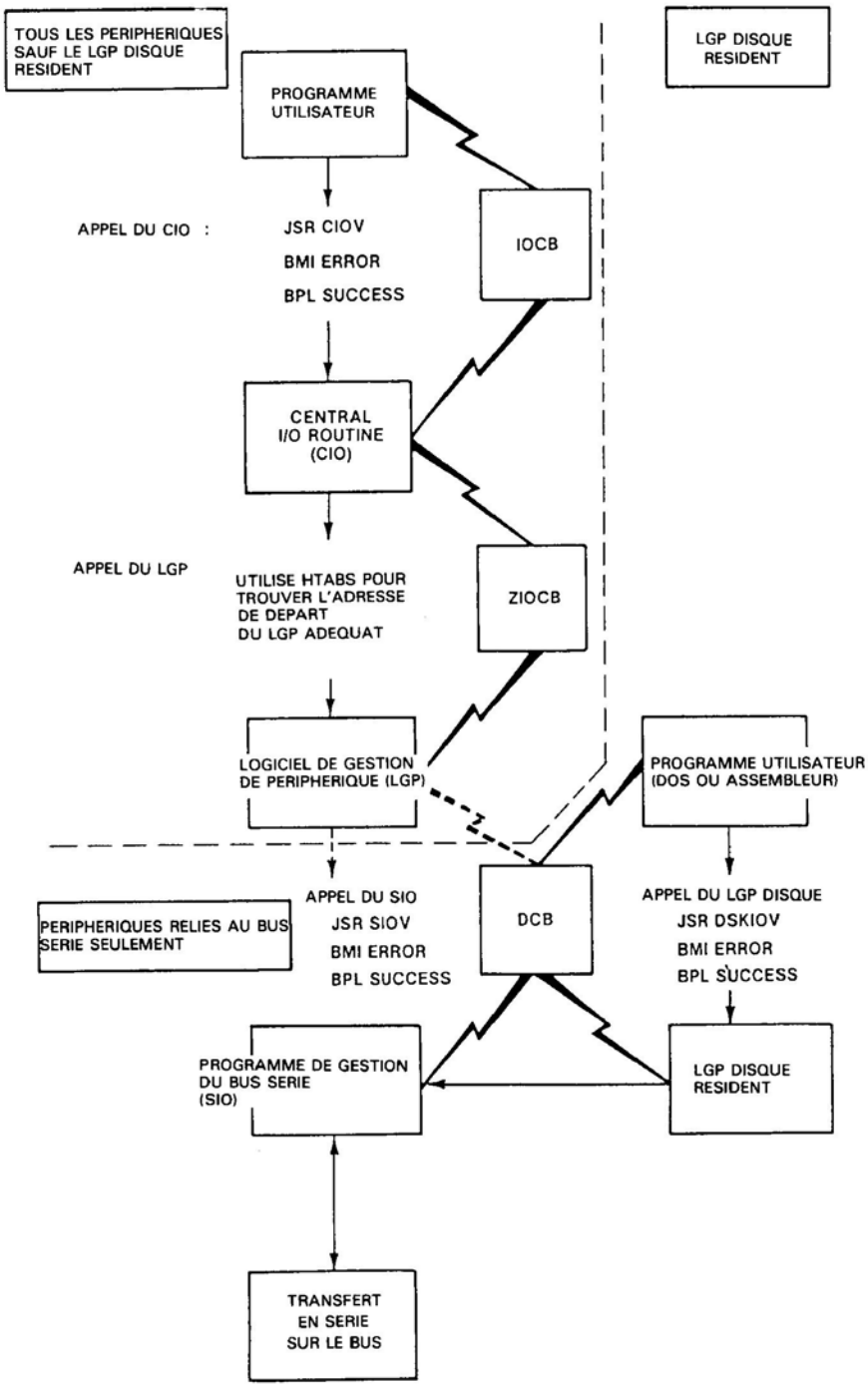
- Le transfert des données doit être indépendant du périphérique.
- La structure E/S doit supporter des transferts octet par octet, plusieurs octets à la fois, ou par enregistrement complet.
- De multiples fichiers ou périphériques doivent être accessibles simultanément.
- La gestion des erreurs doit être transparente pour les périphériques.
- L'addition de nouveaux périphériques (et de son logiciel approprié) doit être possible sans changer le SE.

Le système d'exploitation des ordinateurs ATARI a été conçu de manière à répondre à ces besoins. Il utilise un sous-système d'entrées/sorties centralisées exploitant des tables d'index. Vues par l'utilisateur, les entrées/sorties sont organisées autour de l'IOCB, bloc de commande d'entrées/sorties. Un IOCB est une table standard qui spécifie une opération entière d'entrée ou de sortie. Il existe huit opérations possibles de base. En modifiant un ou plusieurs paramètres de l'IOCB, l'utilisateur peut changer la nature de l'opération aussi bien que le périphérique cible. Ainsi, il devient très facile


ATARINSIDE

de réaliser des opérations d'e/s vers des périphériques très différents comme l'unité de disque ou l'imprimante, sans avoir à se préoccuper du fonctionnement de ces appareils. La plupart des e/s nécessite simplement le remplissage d'un IOCB avec les données appropriées, avant de passer la main au sous-système d'entrées/sorties. Le sous-système d'entrées/sorties se divise en deux parties : les programmes réalisant les opérations d'e/s et les blocs de commande du système. Les programmes comprennent le noyau central appelé CIO, les logiciels de gestion des périphériques (LGP) (il y en a 1 par périphérique possible : E:, K:, S:, P:, C:, T:, R:) et le programme de gestion du bus série (SIO). La table d'adresse des LGP (HATABS) joue un rôle majeur en chaînant le CIO aux LGP. L'interface utilisateur est le même pour tous les périphériques (procédures identiques ou très similaires vu du côté du programmeur d'application).

La figure suivante détaille les relations existant entre les programmes et les blocs de commande du système.



## LES BLOCS DE COMMANDE DU SYSTEME

Il existe quatre types de blocs :

- bloc de commande d'entrées/sorties (IOCB)
- bloc de commande d'E/S en page zéro (ZIOCB)
- bloc de commande du périphérique (DCB)
- tampon de commande (CFB)

Tous ces blocs de commande sont utilisés pour passer des paramètres caractérisant l'opération qui va avoir lieu, entre les divers programmes composant l'ensemble du système d'E/S. Reportez-vous au manuel américain «Operating System Manual» pour plus d'informations sur la structure détaillée de ces quatre types de bloc.

Les IOCBs, au nombre de huit, sont utilisés pour réaliser la communication entre les programmes utilisateur et le CIO. Un IOCB utilise 16 octets. Voici les noms mnémoniques ainsi que les adresses de ces IOCBs :

| Nom | Adresse de départ (longueur) |
|---|---|
| IOCB0 | $340 (16) |
| IOCB1 | $350 (16) |
| IOCB2 | $360 (16) |
| IOCB3 | $370 (16) |
| IOCB4 | $380 (16) |
| IOCB5 | $390 (16) |
| IOCB6 | $3A0 (16) |
| IOCB7 | $3B0 (16) |

Le tableau ci-dessous donne le contenu d'un IOCB pour quelques fonctions classiques d'E/S.

Le second type de blocs de contrôle, le ZIOCB ($0020, longueur = 16) transfère les paramètres entre le CIO et les LGP. Lorsqu'il est appelé, le CIO utilise la valeur contenue dans le registre X comme un index indiquant quel IOCB (1 parmi 8) doit être utilisé. Le CIO recopie ensuite les données du IOCB choisies vers le ZIOCB pour qu'elles soient utilisées par le LGP approprié. Le ZIOCB ne présente pas d'intérêt pour le programmeur, sauf si vous concevez votre propre LGP.

Les LGP utilisant le bus série écriront les informations de commande dans le DCB ($0300, longueur = 12). Le SIO lira les informations du DCB et y écrira le résultat des opérations effectuées (erreurs par exemple). La figure 8-10 illustre quelques fonctions classiques d'E/S ainsi que les contenus correspondant du DCB.

Le LGP résident de la disquette ne suit pas la séquence régulière Utilisateur-CIO-LGP-SIO. Il faut utiliser le DCB pour communiquer directement avec le LGP DISQUE résident. Reportez-vous au chapitre 9 pour obtenir davantage d'informations sur le LGP résident.

Le dernier type de bloc de contrôle est le CFB. Cette table de quatre octets située à l'adresse $023A est utilisée par le SIO pendant les opérations sur le bus série. Ces quatre octets contiennent le code du périphérique, le code de commande et les octets auxiliaires 1 et 2. Le tampon des données à transmettre (quel que soit le sens de transmission) est défini par deux pointeurs BUFRLO ($0032, 2) et BFENLO ($0034, 2). En général, il n'est pas recommandé d'utiliser le bus série à ce niveau. D'autres paramètres doivent être positionnés et les sous-programmes doivent être appelés d'une manière correcte. Le CIO et le SIO ont été conçus pour être facilement appelés par les programmes de l'utilisateur. Exploitez-les mais ne vous occupez pas du tampon de commande CFB.

IOCB CHART

| CALL | ICHID | ICDNO | ICCOM | ICSTA | ICBAL | ICBAH | ICPTL | ICPTH | ICBLL | ICBLH | ICAX1 | ICAX2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OPEN FILE-READ | X | X | 3 | note 1 | $80 | 06 | X | X | X | X | 4 | 0 |
| OPEN FILE-WRITE | X | X | 3 | " | $80 | 06 | X | X | X | X | 8 | note 2 |
| GET BYTES | X | X | 7 | " | 00 | 06 | X | X | $80 | 00 | X | X |
| PUT BYTES | X | X | $B | " | 00 | 06 | X | X | $80 | 00 | X | X |
| GET RECORD | X | X | 5 | " | 00 | 06 | X | X | $80 | 00 | X | X |
| PUT RECORD | X | X | 9 | " | 00 | 06 | X | X | $80 | 00 | X | X |
| CLOSE FILE | X | X | $C | " | X | X | X | X | X | X | X | X |
| STATUS | X | X | $D | " | X | X | X | X | X | X | X | X |

NOTE 1 : L'état du résultat de l'opération d'entrées/sorties est placé ici dans le registre Y au retour du CIO.
NOTE 2 : Les octets auxiliaire du IOCB sont utilisés par quelques LGP pour indiquer des modes spéciaux.
X : Ignoré, mais ne pas changer sa valeur.
NOTE GENERALE : Les définitions ci-dessus supposent :

```
*=$600
IOBUFF    .RES     80              TAMPON D'E/S UTILISATEUR
FILE      .BYTE    'D:EXEMPLE.BAS' TAMPON NOM DU FICHIER
```



ATARINSIDE

DCB CHART

| FUNCTION | NAME | LOCATION | DISK 810/815 | | | | | | PRINTER 820 |
|---|---|---|---|---|---|---|---|---|---|
| | | | READ SECTOR | | WRITE SECTOR | | PUT | FORMAT | WRITE |
| | | | 810 | 815 | 810 | 815 | | | |
| Serial Bus I.D. | DDEVIC | [$0300] | $30 | $30 | $30 | $30 | $30 | $30 | $40 |
| Device Number | DUNIT | [$0301] | 1-4 | 1-8 | 1-4 | 1-8 | 1-4 | 1-4 | 1 |
| Command Byte | DCOMND | [$0302] | $52 | $52 | $57 | $57 | $50 | $21 | $57 |
| Status | DSTATS | [$0303] | $40 | $40 | $80 | $80 | $80 | $40 | $80 |
| Buffer Address | DBUFLO | [$0304] | U | U | U | U | U | U | U |
| | DVBUFHI | [$0305] | U | U | U | U | U | U | U |
| Device Timeout | DTIMLO | [$0306] | $30 | $30 | $30 | $30 | $31 | $130 | 5 |
| Buffer Length | DBYTLO | [$0308] | $80 | 00 | $80 | 00 | $80/00 | - | $40 |
| | DBYTHI | [$0309] | $00 | 01 | $00 | 01 | $00/01 | - | $00 |
| | DAUX1 | [$030A] | 2* | 2* | 2* | 2* | - 2* | - | 1* |
| | DAUX2 | [$030B] | 2* | 2* | 2* | 2* | - 2* | - | 1* |

1* : Cet octet détermine le mode de l'imprimante (modèle ATARI A820)
2* : DAUX1 + DAUX2 donne le numéro du secteur pour une opération de lecture, d'écriture ou d'écriture avec vérification.
U : Adresse donnée par l'utilisateur
– : Non utilisé

## LE CIO

Le CIO (noyau central des entrées/sorties) doit transférer les données d'un IOCB au LGP spécifique correspondant. Puis il cède le contrôle du système au LGP. LE CIO joue également le rôle d'un pipe-line pour la plupart des entrées/sorties. Celles-ci utilisent le CIO comme un point d'entrée commun et tous les LGP se terminent par un retour au CIO. Par exemple, le Basic appèlera le CIO lors d'une instruction OPEN ou GRAPHICS. Le CIO comprend les fonctions suivantes :

| | |
|---|---|
| OPEN | Ouverture canal/périphérique |
| CLOSE | Fermeture canal/périphérique |
| GET CHARS | Lit N caractères |
| READ RECORD | Lit un enregistrement |
| PUT CHARS | Ecrit N caractères |
| WRITE RECORD | Ecrit un enregistrement |
| STATUS | Donne l'état du périphérique |
| SPECIAL | Particulier au LGP considéré (par exemple NOTE pour le FMS) |

Si vous désirez établir vos propres appels au CIO, la séquence est la suivante :

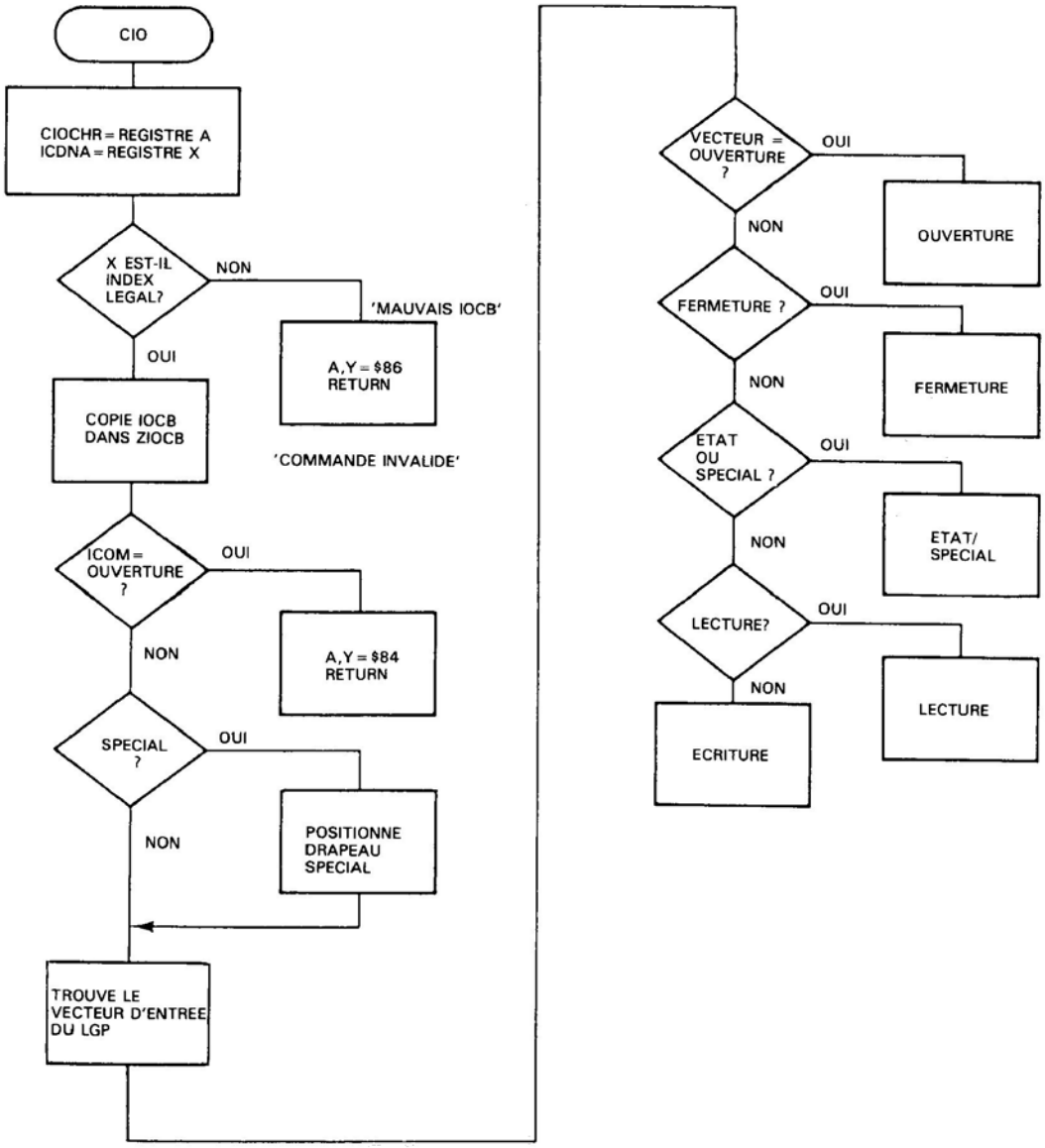
```
                        ; Vous avez déjà rempli un IOCB
LDX   #IOCBNUM          ; Donne le rang de l'IOCB (IOCB*16)
JSR   CIOV              ; Saute au CIO via le vecteur d'entrée
BMI   ERROR             ; Si le branchement a lieu au retour, alors le
                        ; code d'erreur est dans le registre Y
```

Comme celà est montré dans cet exemple, l'un des IOCB est utilisé pour donner des informations au CIO. Vous pouvez utiliser l'un quelconque des huit IOCB. Le CIO attend le numéro de l'IOCB dans le registre X, multiplié par 16. La raison de cette multiplication est que le CIO utilise X comme un véritable index et que chaque IOCB contient 16 octets. Ainsi, si vous utilisez l'IOCB n°3, vous placerez 48 dans X ($30 en hexa, soit 4 décalages à gauche dans l'accumulateur, puis passer A dans X). Au retour, le bit de signe du 6502 est positionné pour indiquer le résultat de l'opération : succès ou erreur. Si ce bit «N» est à 0, tout s'est bien passé et le registre Y contient la valeur 1. Si le bit N est à 1, une erreur s'est produite ; dans ce cas, le registre Y contient le code de l'erreur. Une instruction BNI suffit pour dérouter l'exécution vers le programme de traitement de l'erreur. Le code de l'erreur se retrouve également dans l'octet ICSTA du IOCB. Le chapitre 5 du manuel du Système d'Exploitation donne un exemple d'appel du CIO pour ouvrir un fichier, lire quelques enregistrements et fermer le fichier.

Le CIO recopie les données de l'IOCB vers le bloc en page 0 : ZIOCB. Le CIO détermine ensuite le point d'entrée du LGP correspondant au périphérique spécifié et passe la main au LGP par le vecteur adéquat. La figure 8-11 représente l'organigramme du CIO.

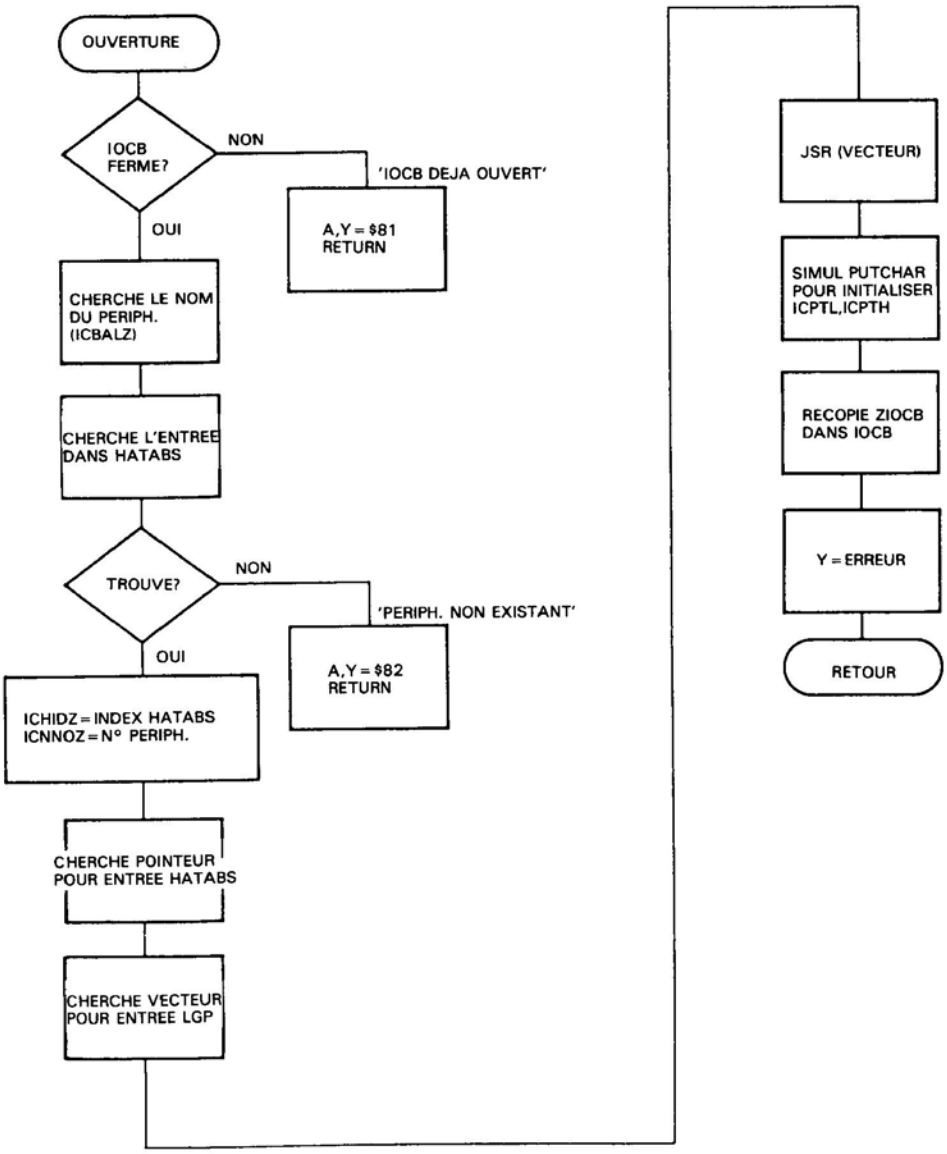


ATARINSIDE


CIO
CIOCHR = REGISTRE A
ICDNA = REGISTRE X
X EST-IL INDEX LEGAL?
NON
'MAUVAIS IOCB'
A,Y = $86
RETURN
OUI
COPIE IOCB DANS ZIOCB
'COMMANDE INVALIDE'
ICOM = OUVERTURE ?
OUI
A,Y = $84
RETURN
NON
SPECIAL ?
OUI
POSITIONNE DRAPEAU SPECIAL
NON
TROUVE LE VECTEUR D'ENTREE DU LGP
VECTEUR = OUVERTURE ?
OUI
OUVERTURE
NON
FERMETURE ?
OUI
FERMETURE
NON
ETAT OU SPECIAL ?
OUI
ETAT/ SPECIAL
NON
LECTURE?
OUI
LECTURE
NON
ECRITURE



ATARINSIDE


OUVERTURE
IOCB FERME?
NON
'IOCB DEJA OUVERT'
A,Y = $81
RETURN
OUI
CHERCHE LE NOM DU PERIPH. (ICBALZ)
CHERCHE L'ENTREE DANS HATABS
TROUVE?
NON
'PERIPH. NON EXISTANT'
A,Y = $82
RETURN
OUI
ICHIDZ = INDEX HATABS
ICNNOZ = N° PERIPH.
CHERCHE POINTEUR POUR ENTREE HATABS
CHERCHE VECTEUR POUR ENTREE LGP
JSR (VECTEUR)
SIMUL PUTCHAR POUR INITIALISER ICPTL,ICPTH
RECOPIE ZIOCB DANS IOCB
Y = ERREUR
RETOUR



ATARINSIDE

# LA TABLE D'ADRESSES DES LGP (HATABS)

Le CIO calcule l'adresse de départ du LGP de manière indirecte. Tout d'abord, une instruction d'ouverture OPEN doit obligatoirement précéder toute autre commande d'entrée/sortie. Pendant l'exécution d'une commande OPEN, le CIO recherche les caractéristiques du périphérique en cours d'ouverture. Ces caractéristiques se résument en une suite de caractères ATASCII désignée par les deux octets (adresse de son début) placés dans l'IOCB (ICBAL=poids faible, ICBAH=poids fort). Le premier élément de la chaîne doit être un caractère unique identifiant le périphérique («D» pour unité de disquette, «P» pour imprimante, «K» pour clavier, etc). Le CIO cherche alors ce caractère dans une table appelée HATABS placée en mémoire de l'adresse $031A à $033B. La figure 8-12 montre la composition de cette table. Le CIO commence sa recherche à l'adresse $033B et décrémente ensuite son pointeur 3 par 3. Cette table est initialisée à la mise en route du système par le SE. Elle n'est pas totalement utilisée, ce qui permet au programmeur de créer son propre LGP et de le désigner par une nouvelle initiale placée dans cette table. Le fait qu'elle se situe en mémoire vive autorise également une modification de son contenu. Certains périphériques la modifient d'eux-mêmes (cas du DOS lorsqu'une unité de disquette est employée, ou du module d'interface RS232). Cette table peut au total accepter 14 entrées, 5 seulement étant initialisées à la mise sous tension. Si un nouveau LGP est mis en place et s'il utilise la même initiale qu'un LGP existant, notez que le CIO le trouvera avant celui initialisé par le système, cela en raison du sens de lecture de la table : de bas en haut.

Lorsque la lettre caractérisant le périphérique a été reconnue, le CIO sait que les deux octets suivants forment une adresse à laquelle il trouvera la liste des points d'entrée (adresses) des sous-programmes formant le LGP. Il existe donc une telle liste pour chaque type de périphérique ; elle correspond aux différentes commandes acceptables par le CIO (point d'entrée du sous-programme OPEN, point d'entrée du sous-programme CLOSE, etc). La figure 8-13 donne l'organisation typique d'une telle table.

Pour trouver quel vecteur utiliser dans cette table, le CIO utilise ICCOM, l'octet de commande dans l'IOCB, comme un index. La position relative de chaque vecteur dans les tables des LGP a toujours la même signification. Par exemple, la première position pointe toujours vers le sous-programme d'ouverture (OPEN).

Vous pouvez étendre les possibilités du SE en exploitant la souplesse de HATABS. La figure 8-14 vous donne un exemple de la manière d'ajouter un «LGP inutile». Ce LGP n'effectue en effet aucun travail particulier. Ce peut être très utile dans les opérations de déverminage : plutôt qu'attendre 50 000 accès à l'unité de disquette avant de trouver une erreur, utilisez le LGP inutile à la place du LGP disque. Ainsi, les défauts des programmes peuvent être cernés plus rapidement.

```
                01 ;  HANDLER ADDRESS TABLE
E430            02 PRINTV   =      $E430
E440            03 CASETV   =      $E440
E400            04 EDITRV   =      $E400
E410            05 SCRENV   =      $E410
E420            06 KEYBDV   =      $E420
                07 ;
0000            08          *=    $031A
                09 ;
                10 HATABS
031A 50         20          .BYTE   'P'     PRINTER
031B 30E4       30          .WORD   PRINTV     ENTRY POINT TABLE
031D 43         40          .BYTE   'C'     CASSETTE
031E 40E4       50          .WORD   CASETV     ENTRY POINT TABLE
0320 45         60          .BYTE   'E'     DISPLAY EDITOR
0321 00E4       70          .WORD   EDITRV     ENTRY POINT TABLE
0323 53         80          .BYTE   'S'     SCREEN HANDLER
0324 10E4       90          .WORD   SCRENV     ENTRY POINT TABLE
0326 4B         0100        .BYTE   'K'     KEYBOARD
0327 20E4       0110        .WORD   KEYBDV     ENTRY POINT TABLE
0329 00         0120        .BYTE   0       FREE ENTRY 1 (DOS)
032A 00 00      0130        .BYTE   0,0
032C 00         0140        .BYTE   0       FREE ENTRY 2 (850 MODULE)
032D 00 00      0150        .BYTE   0,0
032F 00         0160        .BYTE   0       FREE ENTRY 3
0330 00 00      0170        .BYTE   0,0
0332 00         0180        .BYTE   0       FREE ENTRY 4
0333 00 00      0190        .BYTE   0,0
0335 00         0200        .BYTE   0       FREE ENTRY 5
0336 00 00      0210        .BYTE   0,0
0338 00         0220        .BYTE   0       FREE ENTRY 6
0339 00 00      0230        .BYTE   0,0
033B 00         0240        .BYTE   0       FREE ENTRY 7
```

Figure 8-12 - Configuration de la table d'adresse des LGP (HATABS)



ATARI INSIDE

```
        *=$PRINTV
E430 9E EE      .WORD   PHOPEN-1        OUVERTURE
E432 DB EE      .WORD   PHCLOS-1        FERMETURE
E434 9D EE      .WORD   BADST-1         LECTURE (NON REALISEE)
E436 A6 EE      .WORD   PHWRIT-1        ECRITURE
E438 80 EE      .WORD   PHSTAT-1        ETAT
E43A 9D EE      .WORD   BADST-1         SPECIAL (NON REALISE)
E43C 4C 78 EE    JMP    PHINIT         INITIALISATION
```

Figure 8-13 - Table des points d'entrée du LGP imprimante

**LES LOGICIELS DE GESTION DES PERIPHERIQUES**

Les LGP sont de deux types : résidents ou non résidents. Les résidents sont présents dans les mémoires mortes du SE et peuvent être appelés via le CIO tant que le LGP est défini dans HATABS. Les LGP non résidents doivent d'abord être chargés en mémoire vive, puis il faut adapter HATABS (déclaration d'existence) avant d'utiliser le CIO pour dialoguer avec eux. Les LGP résidents sont :

| | |
|---|---|
| (E:) | Editeur d'écran |
| (S:) | Ecran |
| (K:) | Clavier |
| (P:) | Imprimante |
| (C:) | Cassette |

Bien que les LGP non résidents ne soient pas présents dans les mémoires mortes du SE, ils peuvent être ajoutés par le SE lors des phases de mise sous tension ou de SYSTEM RESET. Vous pouvez même ajouter votre propre LGP durant l'exécution d'un programme. La figure 8-14 vous donne un exemple de l'implantation d'un nouveau LGP.

```
0000          10        *  =      $600
031A          20 HATABS    =      $031A
0600 A000     40 START     LDY    #0
0602 B91A03   60 LOOP      LDA    HATABS,Y
0605 C900     70           CMP    #0          ;    FREE ENTRY?
0607 F009     80           BEQ    FOUND
0609 C8       90           INY
060A C8       0100         INY
060B C8       0110         INY                ;    POINT TO NEXT HATABS ENTRY
060C C022     0120         CPY    #34         ;    AT END OF HATABS?
060E D0F2     0130         BNE    LOOP        ;    NO ... CONTINUE
0610 38       0140         SEC                ;    YES... INDICATE ERROR
0611 60       0150         RTS
              0160 ;
0612 A94E     0180 FOUND   LDA    #'N         ;    SET DEVICE NAME
0614 991A03   0190         STA    HATABS,Y
0617 C8       0200         INY
0618 A924     0210         LDA    #NULLTAB&255
061A 991A03   0220         STA    HATABS,Y    ;    HANDLER ADDRESS
061D C8       0230         INY
061E A906     0240         LDA    #NULLTAB/256
0620 991A03   0250         STA    HATABS,Y
0623 60       0260         RTS
              0270 ;
0624 3206     0290 NULLTAB .WORD   RTHAND-1    ;    OPEN
0626 3206     0300         .WORD   RTHAND-1    ;    CLOSE
0628 3406     0310         .WORD   NOFUNC-1    ;    READ
062A 3206     0320         .WORD   RTHAND-1    ;    WRITE
062C 3206     0330         .WORD   RTHAND-1    ;    STATUS
062E 3406     0340         .WORD   NOFUNC-1    ;    SPECIAL
0630 4C3306   0350         JMP     RTHAND      ;    INITILIZATION
              0360 ;
0633 A001     0380 RTHAND  LDY     #1          ;    SUCCESSFUL I/O FUNCTION
0635 60       0400 NOFUNC  RTS                 ;    FUNCTION NOT IMPLEMENTED
```

Figure 8-14 - LGP nul



ATARINSIDE

Les LGP exploitent les informations provenant du CIO et placées dans le ZIOCB. Ces données sont utilisées lors de l'exécution des fonctions d'entrées/sorties comme l'ouverture (OPEN), la fermeture (CLOSE), l'écriture (PUT) et la lecture (GET). Tous les LGP n'autorisent pas toutes les commandes, par exemple tenter d'écrire un caractère à destination du clavier produit l'erreur 146, «fonction non implantée». Le chapitre 5 du manuel du système d'exploitation donne la liste des fonctions acceptées par chaque LGP, ainsi que les détails de fonctionnement du LGP.

**LE LOGICIEL DE GESTION DU BUS SERIE (SIO)**

Le SIO gère le bus série de communication entre les LGP et les périphériques connectés au bus série. Il communique avec le logiciel d'appel par le bloc de contrôle du périphérique (DCB) situé à l'adresse $300 et contenant 12 octets. La figure 8-10 indique l'état du DCB pour les opérations classiques du SIO.

Pour envoyer des commandes au SIO, vous devez comprendre la structure du DCB, décrite dans le chapitre 9 du manuel du système d'exploitation. La figure 8-15 est un exemple en Assembleur d'envoi d'une ligne de texte vers une imprimante connectée sur le bus série ou au port parallèle du module d'interface.

```
0000            05              *= $3000        ADRESSE DE DEPART ARBITRAIRE
                10  ; CE PROGRAMME ENVOIE UNE LIGNE VERS L'IMPRIMANTE EN APPELANT LE SIO
E459            20 SIOV         =       $E459   VECTEUR SIO
009B            30 CR           =       $9B     FIN DE LIGNE
0040            40 PRNTID       =       $40     IDENTIFICATION IMPRIMANTE
004E            45 MODE         =       $4E     MODE NORMAL
001C            50 PTIMOT       =       $001C   ADRESSE «TEMPORISATION»
0300            60 DDEVIC       =       $300    IDENTIFICATION DU PERIPHERIQUE (DCB)
0301            70 DUNIT        =       $301    NUMERO PERIPHERIQUE (DCB)
0302            80 DCOMND       =       $302    COMMANDE SIO (DCB)
0303            90 DSTATS       =       $303    SENS DE TRANSFERT (DCB)
0304            0100 DBUFLO     =       $304    ADRESSE (POIDS FAIBLE) TAMPON (DCB)
0305            0110 DBUFHI     =       $305    ADRESSE (POIDS FORT) TAMPON (DCB)
0306            0120 DTIMLO     =       $306    «TEMPORISATION» DU SIO (DCB)
0307            0130 DTIMHI     =       $307
0308            0140 DBYTLO     =       $308    LONGUEUR DE LA ZONE TAMPON
0309            0150 DBYTHI     =       $309
030A            0160 DAUX1      =       $30A    OCTET AUXILIAIRE - MODE DE L'IMPRIMANTE
030B            0170 DAUX2      =       $30B    OCTET AUXILIAIRE - NON UTILISE
                0180 ;
3000 455841     0190 MESS       .BYTE   "EXAMPLE 12",CR
3001 4D504C
3005 452031
3009 329B
                0200 ;
300B A940       0220            LDA     #PRNTID     IDENTIFIE LE PERIPHERIQUE
300D 8D0003     0230            STA     DDEVIC
3010 A901       0240            LDA     #1          DONNE LE NUMERO DU PERIPHERIQUE
3012 8D0103     0250            STA     DUNIT
3015 A94E       0260            LDA     #MODE
3017 8D0A03     0270            STA     DAUX1       MODE NORMAL
301A A901       0275            LDA     #1
301C 8D0B03     0280            STA     DAUX2       NON UTILISE
301F 8D0703     0290            STA     DTIMHI      TEMPORISATION < 256 s
3022 A51C       0300            LDA     PTIMOT      INITIALISE TEMPORISATION (TEMPS D'ATTENTE
3024 8D0603     0310            STA     DTIMLO      D'UNE REPONSE)
3027 A900       0320            LDA     #MESS&255
3029 8D0403     0330            STA     DBUFLO      MESS EST LE TAMPON
302C A930       0340            LDA     #MESS/256
302E 8D0503     0350            STA     DBUFHI
3031 A980       0360            LDA     #$80        SENS DE TRANSFERT
3033 8D0303     0370            STA     DSTATS      ORDINATEUR → PERIPH.
3036 A957       0380            LDA     #'W         COMMANDE D'ECRITURE
3038 8D0203     0390            STA     DCOMND
303B 2059E4     0410            JSR     SIOV        APPEL DU SIO
303E 3001       0420            BMI     ERROR
3040 00         0430 GOOD       BRK
3041 00         0440 ERROR      BRK
```

Figure 8-15 - Exemple d'emploi du SIO : envoie d'une ligne à l'imprimante



ATARINSIDE

## INTERRUPTIONS DU SIO

Le SIO utilise trois interruptions masquables pour gérer les communications sur le bus série.

| IRQ | Adresse, nombre d'octets | Fonction |
|---|---|---|
| VSERIR | $020A,2 | ENTREE SERIE PRETE |
| VSEROR | $020C,2 | SORTIE SERIE LIBRE |
| VSEROC | $020E,2 | TRANSMISSION TERMINEE |

Toute exécution de programme est suspendue pendant que le SIO utilise le bus série pour communiquer. Bien que rien ne puisse se produire durant un transfert série, les entrées/sorties s'effectuent par interruption. La méthode de communication entre le SIO et le sous-programme se traitement des interruptions est appelé «méthode des sémaphores». Le programme principal est une boucle sans fin, nulle, dans lequel tourne le 6502. Une interruption se produit lorsqu'une opération élémentaire du SIO est terminée. Par exemple, pour émettre un caractère, le SIO place l'octet dans le registre à décalage du POKEY. Puis il boucle jusqu'à détecter le basculement d'un drapeau qui indiquera que tout le caractère a été transmis. La transmission en elle-même est assurée par le POKEY. A la fin du caractère, une interruption VSEROR est générée. Le programme principal du SIO sort de sa boucle et exécute un petit programme plaçant l'octet suivant dans le registre à décalage, puis il reprend sa boucle, pendant le changement d'état du drapeau. Ce processus se continue jusqu'à ce que le buffer entier ait été envoyé. Après avoir calculé la checksum, le logiciel de traitement des interruptions positionne le drapeau «transmission terminée». Le SIO sort alors définitivement de sa boucle et redonne la main au logiciel l'ayant appelé (LGP ou utilisateur).

Le fonctionnement est similaire lors de la réception d'un octet. Le POKEY génère une IRQ (VSERIR) informant le SIO qu'un octet a été reçu dans le registre à décalage d'entrée. En traitant l'interruption, le 6502 transfère l'octet dans un tampon et vérifie si la fin du tampon est atteinte. Si oui, le drapeau «transmission terminée» est positionné. Au retour de l'interruption, le SIO rendra la main au programme appelant.

Vous avez pu constater, à la lecture de ces explications, que le SIO perd beaucoup de temps en attendant que le POKEY ait terminé son travail. Les vecteurs des trois interruptions étant placés en mémoire vive, ils sont à la disposition de vos propres logiciels si vous désirez augmenter les performances d'entrées/sorties. C'est ainsi que le module d'interface ATARI A850 réalise des opérations d'entrées/sorties simultanées. Le LGP du A850 modifie les vecteurs car ce logiciel se substitue au traitement des IRQ du SIO. Le programme appelant peut ainsi continuer de s'exécuter tandis que le module d'interface transmet et reçoit des données.

## UTILISATION DU CIO DEPUIS LE BASIC

La plupart des fonctions CIO (ouverture, fermeture, etc) sont accessibles depuis le Basic par les instructions OPEN, CLOSE, GET, PUT, etc. Toutefois, le Basic n'utilise pas toutes les possibilités du CIO : celles par exemple d'effectuer une opération d'entrée/sortie sur un octet à la fois (lecture et écriture d'un caractère).

D'autre part, le traitement par buffer entier est intéressant. Par exemple, de cette manière, un programme peut directement charger dans la mémoire un fichier disque ; ou une image haute résolution dans la mémoire écran. Mais le Basic est lent dans ce genre d'opération.

Une manière d'améliorer les performances du Basic consiste à réaliser un sous-programme en Assembleur. Mais il faut trouver de la place disponible en mémoire pour ce programme, et cela peut être un gros problème. Une solution consiste à placer le programme Assembleur dans des chaînes de caractère et de l'appeler par une instruction USR (ADR(CHAINE$)). L'adresse d'une chaîne pouvant varier lors de l'édition du programme, le sous-programme Assembleur doit être relogeable, c'est-à-dire qu'il ne doit pas utiliser de références à des adresses absolues contenues dans la chaîne elle-même.

Dans la figure 8-16, nous évitons l'emploi des chaînes en utilisant la page 6 de la mémoire vive. Ainsi, le programme n'a pas à être relogeable. Les paramètres de commande sont placés (POKE) dans un IOCB : le but est de transférer un fichier objet de la disquette vers la mémoire. Ce programme Basic de la figure 8-16 peut également servir à transférer en sens inverse : de la mémoire vers la disquette, l'utilisateur devant préciser l'adresse de départ et la longueur de la zone à transférer.



ATARINSIDE

```
30  REM CE PROGRAMME CHARGE LA PAGE 6 AVEC LE FICHIER D:TEST
100 DIM FILE$(20),CIO$(7):CIO$="hhh*LVd"
106 REM CIO$ IS PLA,PLA,PLA,TAX,JMP $E456 (CIOV)
110 FILE$="D:TEST":REM _  NOM DU FICHIER
120 CMD=7:STADR=1536:GOSUB 30000
130 IF ERROR=1 THEN ? "TRANSFER COMPLETE":STOP
135 ? "ERROR # ";ERROR;" OCCURRED AT LINE # ";PEEK(186)+256*PEEK(187)
200 END
300 REM _            INITIALISATION CIO
310 REM

30000 REM
30001 REM
30002 REM CE PROGRAMME CHARGE OU SAUVEGARDE UNE ZONE MEMOIRE DEPUIS LE BASIC
30003 REM EN UTILISANT UN IOCB ET EN APPELANT LE CIO DIRECTEMENT
30004 REM
30006 REM LORS DE L'APPEL  CMD = 7 → CHARGE EN MEMOIRE
30008 REM _                CMD = 11 → SAUVEGARDE LA MEMOIRE
30009 REM _                STADR = ADRESSE DE DEPART DE LA ZONE MEMOIRE
30010 REM _                BYTES = NOMBRE D'OCTETS A TRANSFERER
30011 REM _                IOCB = LE NUMERO DE l'IOCB UTILISE
30012 REM _                FILE$ = NOM DU FICHIER DESTINATION
30013 REM
30014 REM AU RETOUR  ERROR = 1 COMMANDE REUSSIE
30018 REM _          ERROR <> 1 CODE DE L'ERREUR
30019 REM
30020 REM _    *** ADRESSE DES OCTETS DE L'IOCB ***
30022 REM
30024 IOCBX=IOCB*16:ICCOM=834+IOCBX:ICSTA=835+IOCBX
30026 ICBAL=836+IOCBX:ICBAH=837+IOCBX
30028 ICBLL=840+IOCBX:ICBLH=841+IOCBX
30029 REM
30030 AUX1=4:IF CMD=11 THEN AUX1=8
30035 TRAP 30900:OPEN #IOCB,AUX1,0,FILE$
30040 TEMP=STADR:GOSUB 30500
30090 POKE ICBAL,LOW:POKE ICBAH,HIGH
30100 TEMP=BYTES:GOSUB 30500
30130 POKE ICBLL,LOW:POKE ICBLH,HIGH
30140 POKE ICCOM,CMD:ERROR=USR(ADR(CIO$),IOCBX)
30150 ERROR=PEEK(ICSTA):RETURN
30200 REM
30300 REM _    *** CE SS-PGM RENVOIE 2 OCTETS (FORT-FAIBLE) POUR UNE QUANTITE 16 BITS
30400 REM
30500 HIGH=INT(TEMP/256):LOW=INT(TEMP-HIGH*256):RETURN
30550 REM
30600 REM *** PASSE ICI SI ERREUR ***
30900 ERROR=PEEK(195)
30920 CLOSE #IOCB:RETURN
```

Figure 8-16 - Appel direct en Basic du CIO



ATARINSIDE

# PROGRAMMATION EN TEMPS REEL

La plupart du temps, nous avons la possibilité en programmation de ne pas tenir compte du temps. Généralement, nous ne cherchons pas à évaluer le temps d'exécution d'un programme ni à mesurer précisément le temps utilisé par un sous-programme. Toutefois, il arrive parfois que le temps devienne important et joue un rôle primordial dans les performances du logiciel. Nous entrons encore dans le monde de la «programmation en temps réel». Et cela se produit souvent avec des ordinateurs ATARI, plus souvent qu'avec d'autres petits ordinateurs, car les circuits internes sont conçus pour devoir fonctionner en complet synchronisme avec un signal spécifique : le signal TV.

Afin d'obtenir des images propres, nettes, sans sautillements, les circuits des ordinateurs ATARI sont indépendants du signal TV. Malheureusement, il existe plusieurs standards dans le monde. Aux Etats-Unis, le standard est le NTSC : 60 images par seconde, 262 lignes horizontales par image et 228 périodes d'horloge par ligne. Les 262 lignes proviennent du fait que les ordinateurs ATARI ne créent pas d'image entrelacée. Le standard réel utilise 525 lignes, décomposées en deux demi-images (lignes impaires, lignes paires) transmises en alternance. D'autres pays utilisent le standard PAL. 50 images par seconde, 312 lignes par image. En conséquence, le diagramme des temps est différent entre les deux versions. Reportez-vous au chapitre 2 pour obtenir davantage de détails sur le signal TV. Dans ce qui suit, nous n'envisagerons que le cas des systèmes NTSC.

### SYNCHRONISATION SUR LE SIGNAL TV

Le 6502 est synchronisé sur le signal TV de deux manières : une grossière et une plus fine. La synchronisation grossière est fournie par le système de Blanking vertical servant à indiquer au téléviseur le changement d'image. Ce signal apparaît pour l'ordinateur comme une interruption non masquable. Pour le programmeur, cette interruption se répétant très régulièrement, présente de nombreux avantages car elle peut être utilisée pour des effets sonores, ou une programmation multitâches.

Une corrélation plus fine entre le 6502 et le signal TV est fournie par l'horloge système fonctionnant à 1,79 MHz. Cela donne une relation directe entre le temps d'exécution d'une instruction et la distance parcourue par le faisceau d'électrons sur l'écran. Par exemple, durant l'exécution de la plus courte instruction du 6502 (2 cycles), le faisceau se déplace de 4 périodes de l'horloge couleur (3,58 MHz), soit la largeur d'un caractère en mode SE 0. Cette corrélation précise permet au programmeur astucieux de produire des effets graphiques en plein milieu d'une ligne de balayage. Toutefois, l'ANTIC réalise des accès directs à la mémoire pendant lesquels il suspend le travail du 6502. En conséquence, il devient difficile de suivre un chronogramme précis, d'autant plus que la fréquence du DMA de l'ANTIC varie avec le mode graphique, la composition de la Display List, etc.

### LES COMPTEURS DE TEMPS MATERIELS

Quatre décompteurs sont intégrés dans le POKEY. Ils sont utilisés principalement pour la production d'effets sonores (voir chapitre 7). Mais ils peuvent également effectuer des mesures de temps précises car ils génèrent une interruption IRQ. Chaque compteur est associé à un registre fréquence qui mémorise la valeur initiale du compteur. Quand une opération quelconque d'écriture a lieu dans le registre STIMER ($D209), la valeur initiale est transférée dans le compteur et le compte à rebours commence. Quand la valeur passe à zéro, une interruption IRQ est générée. Il est important de noter que seuls les compteurs 1, 2 et 4 possèdent des vecteurs d'interruptions. Voici la marche à suivre pour utiliser un compteur :

1. Placez dans AUDCTL ($D208) la valeur sélectionnant la fréquence du décomptage pour le compteur choisi.

2. Positionnez la sortie d'intensité sonore à 0 pour le canal audio associé au compteur choisi (AUDC1, AUDC2, AUDC4 [$D201, $D203, $D207]).

3. Placez dans AUDF1, AUDF2, AUDF4 [$D200, $D202, $D206] le nombre d'impulsions à décompter.

4. Etablissez le programme Assembleur qui traitera l'interruption.

5. Modifiez le vecteur d'interruption correspondant pour qu'il pointe vers votre programme (défini en 4.) : VTIMR1, VTIMR2, VTIMR4 [$0210, $0212, $0214]. NOTE : Due à une erreur dans le code source, la version originale du SE n'utilisait jamais VTIMR4. Cela a été corrigé dans les versions suivantes.

6. Positionnez les bits 0, 1 ou 2 dans IRQEN et dans la mémoire cache POKMSK ($D20E et $0010) pour autoriser les interruptions des compteurs 1, 2, 4.

7. Ecrivez n'importe quelle valeur dans STIMER ($D209) pour démarrer le compte à rebours des compteurs.

Notez que le temps de réponse du 6502 à une interruption IRQ (donc à une interruption produite par un décompteur) est variable en raison du DMA de l'ANTIC et des interruptions NMI.



ATARINSIDE

## LES COMPTEURS DE TEMPS LOGICIELS

Il existe six compteurs de temps logiciels :

| Nom | Adresse, nombre d'octet | vecteur ou drapeau |
|---|---|---|
| RTCLOK | [$0012,3] | aucun |
| CDTMV1 | [$0218,2] | CDTMA1 [$0226,2] |
| CDTMV2 | [$021A,2] | CDTMA2 [$0228,2] |
| CDTMV3 | [$021C,2] | CDTMF3 [$022A,1] |
| CDTMV4 | [$021E,2] | CDTMF4 [$022C,1] |
| CDTMV5 | [$0220,2] | CDTMF5 [$022E,1] |

Tous ces compteurs sont incrémentés durant le traitement de l'interruption de Blanking vertical (VBLANK). Si cette interruption est interceptée ou inhibée, les compteurs ne seront pas mis à jour.

L'horloge temps réel (RTCLOK) et le compteur 1 (CDTMV1) sont mis à jour durant la première partie de VBLANK (mode immédiat). RTCLOK démarre à 0 (mise sous tension) et compte sur trois octets. Lorsque la valeur maximale est atteinte (16 777 216), le cycle reprend à 0. RTCLOK peut être utilisé pour former une horloge temps réel comme suggéré à la figure 8-17.

Les compteurs étant mis à jour durant VBLANK, il faut prendre quelques précautions lorsqu'on les initialise. Une interruption ne doit pas venir interférer avec l'opération d'initialisation sinon le résultat sera erronné. Pour cela, vous devez utiliser le sous-programme spécial SETVBV ($E45C) inclus dans le SE. Avant d'appeler SETVBV, vous devez positionner les registres ainsi :

X : Contient le poids fort de la valeur d'initialisation du compteur
Y : Contient le poids faible de la valeur d'initialisation du compteur
A : Contient le numéro du compteur 1-5

Exemple :

```
LDA     = 1         ; Initialise le compteur 1
LDY     = 0
LDX     = 2         ; La valeur est $200 périodes de Blanking
JSR     SETVBV      ; Initialisation
```

Les compteurs logiciels 1 à 5 utilisent 2 octets. Le SE les décrémente à chaque VBLANK, les compteurs 2 à 5 étant modifiés dans la deuxième partie du traitement de VBLANK (non critique). Diverses actions sont prise par le SE lorsque les compteurs atteignent 0.

Les compteurs 1 et 2 sont associés à des vecteurs. Lorsque le décompteur 1 ou 2 atteint 0, le SE simule un JSR via le vecteur correspondant au décompteur. La figure 8-7 donne les noms et adresses et des vecteurs.

Les compteurs 3 à 5 sont associés à des drapeaux normalement non nuls. Lorsqu'un compteur atteint 0, le SE met à zéro le drapeau correspndant. Vous testez périodiquement le drapeau pour savoir si le décomptage est terminé ou non.

Les compteurs 1 à 5 sont généraux et servent dans de nombreuses applications. Par exemple, le décompteur 1 est utilisé par le SIO dans les opérations de transmission série. Si le décompteur arrive à 0 avant qu'une opération sur le bus ne soit terminée, l'erreur «le périphérique ne répond pas» est générée. La valeur initiale du décompteur 1 varie selon le périphérique et la commande considérée. Cette procédure évite à l'ordinateur d'attendre indéfiniment la réponse d'un périphérique inexistant. Le LGP cassette utilise le compteur 3 pour déterminer le temps d'écriture ou de lecture des en-têtes sur la bande. La figure 8-18 utilise le compteur 2 pour réaliser un métronome.

Les compteurs logiciels sont généralement utilisés lorsque l'intervalle de temps recherché est supérieur à une période de VBLANK. Pour des durées inférieures, il faut exploiter les compteurs matériels ou d'autres méthodes.



ATARINSIDE

```
1 POKE 752,1
3 ? "†":REM CLEAR SCREEN (†=ESC-CTRL-CLR)
4 ? "HOUR";:INPUT HOUR:? "MINUTE ";:INPUT MIN:? "SECOND";:INPUT SEC
5 CMD=1:GOSUB 45
6 ? "†";HOUR;" : ";MIN;" : ";SEC:? " ":? " "
7 CMD=2:GOSUB 45
9 ? "";HOUR;":";MIN;":";SEC;"    ":GOTO 7

10 REM THIS IS A DEMO OF THE REAL TIME CLOCK
20 REM THIS ROUTINE ACCEPTS AN INITIAL TIME IN HOURS,MINUTES, AND SECONDS
30 REM IT SETS THE REAL TIME CLOCK TO ZERO
40 REM THE CURRENT VALUE OF RTCLOCK IS USED TO ADD TO THE INITIAL TIME,TO GE
42 THE CURRENT TIME HOUR,MIN,SEC
45 HIGH=1536:MED=1537:LOW=1538
50 REM
60 REM ******ENTRY POINT******
65 REM
70 ON CMD GOTO 100,200
95 REM
96 REM ****INITIALIZE CLOCK*****
97 REM
100 POKE 20,0:POKE 19,0:POKE 18,0
105 DIM CLOCK$(50)
106 CLOCK$=" "
107 GOSUB 300
110 IHOUR=HOUR:IMIN=MIN:ISEC=SEC:RETURN
197 REM
198 REM *******READ CLOCK*****
199 REM
200 REM
201 A=USR(ADR(CLOCK$))
210 TIME=((((PEEK(HIGH)*256)+PEEK(MED))*256)+PEEK(LOW))/59.923334
220 HOUR=INT(TIME/3600):TIME=TIME-(HOUR*3600)
230 MIN=INT(TIME/60):SEC=INT(TIME-(MIN*60))
235 SEC=SEC+ISEC:IF SEC>60 THEN SEC=SEC-60:MIN=MIN+1
236 MIN=MIN+IMIN:IF MIN>60 THEN MIN=MIN-60:HOUR=HOUR+1
237 HOUR=HOUR+IHOUR
240 HOUR=HOUR-(INT(HOUR/24))*24
250 RETURN
300 FOR J=1 TO 38:READ Z:CLOCK$(J,J)=CHR$(Z):NEXT J:RETURN
310 DATA 104,165,18,141,0,6,165,19,141,1,6,165
320 DATA 20,141,2,6,165,18,205,0,6,208,234
330 DATA 165,19,205,1,6,208,227,165,20,205,2,6,208,220,96
```

Figure 8-17 Real Time Clock

```
1 REM     PROGRAMME BASIC DE METRONOME
2 REM
3 REM
5 PRINT "†":REM                          EFFACE L'ECRAN
10 X=10:REM                              CADENCE INITIALE
20 FOR J=1 TO 10:NEXT J:REM              BOUCLE DE TEMPORISATION
50 IF STICK(0)=14 THEN X=X+1 :REM        JOYSTICK EN AVANT AUGMENTE LA VITESSE
51 IF STICK(0)=13 THEN X=X-1 :REM        JOYSTICK EN ARRIERE RALENTI LA VITESSE
52 IF X<1 THEN X=1:REM                   MAIS PAS INFERIEURE A 1
53 IF X>255 THEN X=255:REM               NI SUPERIEURE A 255
54 REM                                   AFFICHE TOP PAR MINUTE
56 ? "";INT(3600/X);" BEATS/MINUTES          "
60 POKE 0,X:REM                          ADRESSE 0 MEMORISE LA CADENCE
70 NEXT I :REM                           POUR LA TRANSMETTRE AU PROGRAMME ASSEMBLEUR CI-DESS
```

Figure 8-18 - Partie Basic du programme métronome



ATARINSIDE

```
40              *=$600
50 ;SOUS-PROGRAMME METRONOME. UTILISE $0000 POUR LA CADENCE
60 ;
70 AUDF1        =       $D200   REGISTRE FREQUENCE AUDIO 1
80 AUDC1        =       $D201   REGISTRE COMMANDE 1
90 FREQ         =       $08     VALEUR DE AUDF1
0100 VOLUME     =       $AF     VALEUR DE AUDC1
0110 OFF        =       $A0     COUPE L'INTENSITE SONORE
0120 SETVBV     =       $E45C   SOUS-PROGRAMME D'INITIALISATION DU COMPTEUR
0130 XITVBV     =       $E462
0140 CDTMV2     =       $021A   COMPTEUR 2
0150 CDTMA2     =       $0228   VECTEUR ASSOCIE AU COMPTEUR 2
0160 ZTIMER     =       $0000   VALEUR DU COMPTEUR EN PAGE 0
0170 ;
0180 START      LDA     #10
0190            STA     ZTIMER
0200 ;      DEROUTE LE VECTEUR
0220 ;
0230 INIT       LDA     #CNTINT&255
0240            STA     CDTMA2
0250            LDA     #CNTINT/256
0260            STA     CDTMA2+1
0270 ;
0280 ;      INITIALISE LA VALEUR DU COMPTEUR APRES LE VECTEUR
0290 ;
0300            LDY     ZTIMER  COMPTEUR 2 INITIALISE
0310            JSR     SETIME
0320            RTS
0340 ;      LE VECTEUR POINTE ICI POUR PRODUIRE
0380 ;      UN BIP APRES CHAQUE COMPTAGE
0390 ;
0400 CNTINT     LDA     #VOLUME
0410            STA     AUDC1
0420            LDA     #FREQ
0430            STA     AUDF1
0435            LDY     #$FF    ATTENTE
0440 DELAY      DEY
0442            BNE     DELAY
0450            STY     AUDC1
0460            JMP     INIT
0480 ;
0490 ;      SOUS-PROGRAMME POUR INITIALISER COMPTEUR
0500 ;
0520 SETIME     LDX     #0              PAS DE TEMPS > 256 PERIODE BLANKING
0530            LDA     #2              COMPTEUR 2
0540            JSR     SETVBV          INITIALISE
0550            RTS
0560            *=$2E2
0570            .WORD   START
0580            .END
```

Figure 8-19 - Sous-programme Assembleur du métronome



ATARINSIDE

# LOGICIELS DE VIRGULE FLOTTANTE

L'ensemble des logiciels de virgule flottante (FPP) donne des possibilités arithmétiques étendues au SE. Pour les ordinateurs ATARI 400 et 800, ces logiciels sont intégrés dans une mémoire morte faisant partie du module de 10 ko du système d'exploitation. Le FPP se situe aux adresses $D800-$DFFF. Il n'a pas été modifié dans la version B du SE. Les paragraphes suivants détaillent la représentation utilisée pour coder les nombres en mémoire, les programmes de calcul existants et la manière de les utiliser. Un exemple de programme en Assembleur est inclus pour illustrer l'exploitation du FPP par l'utilisateur.

REPRESENTATION INTERNE

Le FPP représente tout nombre sur 6 octets : 1 octet d'exposant et 5 octets de mantisse dans le format BCD (Décimal Codé Binaire). Cette représentation a été choisie afin de minimiser les erreurs d'arrondi qui peuvent apparaître dans les programmes mathématiques.

Le bit de poids le plus fort de l'octet exposant donne le signe de la mantisse : 0 pour positif, 1 pour négatif. Les 7 bits restants donnent l'exposant comme une puissance de 100. La valeur 64 est ajoutée à l'exposant avant qu'il soit placé dans l'octet exposant. Cela permet de décrire la gamme complète des exposants négatifs et positifs avec une valeur variant entre 0 et 127, sans plus tenir compte du bit de signe.

La mantisse est toujours normalisée, c'est-à-dire que l'octet de poids le plus fort est non nul. Toutefois, puisque la mantisse est dans le format BCD et que l'exposant représente une puissance de 100 et non de 10, la précision peut varier : 9 ou 10 chiffres. Le point décimal est sous-entendu à droite du premier octet de la mantisse ; ainsi, un exposant inférieur à 64 ($40) indique un nombre inférieur à 1.

**Exemples** (les valeurs des octets sont en hexa)

```
Nombre  : 0.02          = 2*100**-1
Format  : 3F 02 00 00 00 00    (exposant = 40-1)

Nombre  : -0.02         = -2*100**-1
Format  : BF 02 00 00 00 00    (exposant = 80+40-1)

Nombre  : 37.0          = 37*100**0
Format  : 40 37 00 00 00 00    (exposant = 40+0)

Nombre  : -460312       = -46.0312*100**2
Format  : C2 46 03 12 00 00    (exposant = 80+40+2)
```

Le nombre 0 est particulier : il est représenté par un exposant nul et une mantisse nulle. Il suffit donc de tester soit l'exposant, soit le premier octet (poids fort) de la mantisse pour détecter un 0.

La gamme dynamique des nombres pouvant être ainsi représentés va de 10E-98 à 10E+98.

**UTILISATION DE LA MEMOIRE**

Deux zones de mémoire vive sont utilisées par le FPP :
- $00D4 - $00FF en page zéro
- $057E - $05FF en page cinq.

Ces zones sont utilisées à la fois pour stocker des paramètres de commande et pour simuler plusieurs registres en virgule flottante. Les deux pseudo-registres principaux sont FR0 et FR1 (adresses $00D4 - $00D9 et $00E0 - $00E5 respectivement). Chacun de ces registres utilise six octets et mémorise un nombre dans le format vu précédemment. Un pointeur sur 16 bits est utilisé pour pointer un nombre en virgule flottante. Il s'appelle FLPTR et réside en $00FC - $00FD.

Des zones tampons doivent exister afin de permettre la réalisation des conversions ATASCII/virgule flottante. Le tampon de sortie (résultat) est appelé FBUFF ; il compte 128 octets, de l'adresse $0580 à l'adresse $05FF. Le tampon d'entrée est désigné par le pointeur 16 bits INBUFF en $00F3. Enfin, un index sur 8 bits appelé CIX et situé à l'adresse $00F2 sert comme offset dans le tampon pointé par INBUFF.

Voici comment devrait se dérouler un accès au FPP depuis un programme Assembleur. Tout d'abord, une chaîne ATASCII représentant un des nombres qui seront utilisés, est placée quelque part en mémoire. Ensuite, le pointeur INBUFF doit pointer vers le début de cette chaîne ; CIX est initialisé à 0. Le nombre est maintenant prêt à être converti dans le format virgule flottante, il suffit d'appeler le programme AFP. Le résultat est placé dans FR0 où il pourra être réutilisé pour d'autres programmes du FPP. A la fin d'un programme du FPP, le résultat en virgule flottante est toujours placé dans FR0. Le fait d'appeler FASC convertirait ce nombre en une chaîne de caractères ATASCII stockée dans LBUFF. Reportez-vous à la figure 8-21 pour un exemple de ce processus.

Pour utiliser des valeurs sur 16 bits avec le FPP, placez les deux octets composant le nombre dans les octets de poids les plus faibles de FR0 ($D4 et $D5) puis effectuez un JSR IFP, qui convertit l'entier en sa représentation virgule flottante, laissant le résultat en FR0. Le sous-programme FPI réalise l'opération inverse.

Le tableau suivant résume les fonctions disponibles, leurs adresses en mémoire morte, les pseudo-registres affectés et les temps approximatifs maximum de calcul.



ATARINSIDE

| NOMS | ADRESSES | FONCTION | RESULTAT | TEMPS MAX (microsec.) |
|---|---|---|---|---|
| AFP | D800 | ATASCII Vers virgule flottante | FR0 | 3500 |
| FASC | D8E6 | Virgule flottante vers ATASCII | LBUFF | 950 |
| IFP | D9AA | Entier vers virgule flottante | FR0 | 1330 |
| FPI | D9D2 | Virgule flottante vers entier | FR0 | 2400 |
| FSUB | DA60 | Soustraction FR0 – FR1 | FR0 | 740 |
| FADD | DA66 | Addition FR0 + FR1 | FR0 | 710 |
| FMUL | DADB | Multiplication FR0*FR1 | FR0 | 12000 |
| FDIV | DB28 | Division FR0/FR1 | FR0 | 10000 |
| FLD0R | DD89 | Chargt d'un nbre virg. flot. util. x,y | FR0 | 70 |
| FLD0P | DD8D | Chargt d'un nbre virg. flot. util. FLPTR | FR0 | 60 |
| FLD1R | DD98 | Chargt d'un nbre virg. flot. util. x,y | FR1 | 70 |
| FLD1P | DD9C | Chargt d'un nbre virg. flot. util. FLPTR | FR1 | 60 |
| FSTOR | DDA7 | Sauveg. d'un nbre virg. flot. util. x,y | FR0 | 70 |
| FSTOP | DDA8 | Sauveg. d'un nbre virg. flot. util. FLPTR | FR0 | 70 |
| FMOVE | DDB6 | Transfert FR0 vers FR1 | FR1 | 60 |
| PLYEVL | DD40 | Evaluation polynomiale | FR0 | 88300 |
| EXP | DDC0 | Exponentiation e!FR0 | FR0 | 115900 |
| EXP10 | DDCC | Elévation à la puissance 10 !FR0 | FR0 | 108800 |
| LOG | DECD | Logarithme naturel | FR0 | 136000 |
| LOG10 | DED1 | Logarithme base 10 | FR0 | 125400 |
| ZFR0 | DA44 | Mise à 0 | FR0 | 80 |
| AF1 | DA46 | Mise à 0 du registre pointé par X | varies | 80 |

Figure 8-20 - Ensemble des sous-programmes de calcul en virgule flottante



ATARINSIDE

```
0000          20          *= $4000  ; ARBITRARY STARTING POINT
DDB6          30    FMOVE  = $DDB6
DA60          40    FSUB   = $DA60
0482          50    FTEMP  = $0482
DDA7          60    FSTOR  = $DDA7
D8E6          70    FASC   = $D8E6
00F3          80    INBUFF = $00F3
D800          85    AFP    = $D800
00F2          90    CIX    = $00F2
0580          100   LBUFF  = $0580
009B          120   CR     = $9B
0009          130   PUTREC = $09
0005          140   GETREC = $05
E456          150   CIOV   = $E456
0342          160   ICCOM  = $0342
0344          170   ICBAL  = $0344
0348          180   ICBLL  = $0348
              190   ;
              200   ; FLOATING POINT DEMONSTRATION
              210   ; READS TWO NUMBERS FROM SCREEN EDITOR,
              215   ; CONVERTS THEM TO FLOATING POINT,
              220   ; SUBTRACTS THE FIRST FROM THE SECOND,
              225   ; STORES THE RESULT IN FTEMP,
              230   ; WHICH IS A USER DEFINED FP REGISTER,
              240   ; AND DISPLAYS THE RESULT.
4000 205340   260   START   JSR GETNUM       ; GET 1ST NUMBER FROM E:
4003 20B6DD   270           JSR FMOVE        ; MOVE NUMBER FROM FR0 TO FR1
4006 205340   280           JSR GETNUM       ; GET 2ND NUMBER FROM E:
4009 2060DA   290           JSR FSUB         ; FR0 <-- FR0-FR1
400C 900A     300           BCC NOERR        ; SKIP IF NO ERROR
400E A981     340           LDA #ERRMSG&255  ; IF ERR., DISPLAY MESSAGE
4010 8D4403   350           STA ICBAL
4013 A940     360           LDA #ERRMSG/256
4015 4C3940   370           JMP CONTIN
4018 A282     390   NOERR   LDX #FTEMP&255   ; STORE RESULT IN FTEMP
401A A004     400           LDY #FTEMP/256
401C 20A7DD   410           JSR FSTOR
              420   ;
              430   ; CONVERT NUMBER TO ATACSII STRING.
              440   ; FIND END OF STRING,
              445   ; CHANGE NEGATIVE NUM. TO POSITIVE,
              450   ; AND ADD CARRIAGE RETURN.
401F 20E6D8   470           JSR FASC         ; FP TO ATASCII, RESULT IN LBUFF
4022 A0FF     480           LDY #$FF
4024 C8       490   MLOOP   INY
4025 B1F3     500           LDA (INBUFF),Y   ; LOAD NEXT BYTE
4027 10FB     510           BPL MLOOP        ; IF POSITIVE, CONTINUE
4029 297F     520           AND #$7F         ; IF NOT, MASK OFF MSBIT
402B 91F3     530           STA (INBUFF),Y
402D C8       540           INY
402E A99B     550           LDA #CR          ; STORE CARRIAGE RETURN
4030 91F3     560           STA (INBUFF),Y
              570   ;
```



ATARINSIDE

```
                580  ; DISPLAY RESULT
4032 A5F3       600          LDA INBUFF       ; GET BUFFER ADDRESS
4034 8D4403     610          STA ICBAL
4037 A5F4       620          LDA INBUFF+1
4039 8D4503     630  CONTIN  STA ICBAL+1
403C A909       640          LDA #PUTREC      ; COMMAND FOR PUT RECORD
403E 8D4203     650          STA ICCOM
4041 A928       660          LDA #40          ; SET BUFFER LENGTH = 40
4043 8D4803     670          STA ICBLL
4046 A900       690          LDA #0
4048 8D4903     700          STA ICBLL+1
404B A200       710          LDX #0           ; IOCB # = 0 FOR SCREEN EDITOR
404D 2056E4     720          JSR CIOV         ; CALL CIO
4050 4C0040     730          JMP START        ; DO IT AGAIN
                740  ;
                750  ; GET ATASCII STRING FROM E:
                755  ; CONVERT TO FP, RESULT IN FR0
4053 A905       780  GETNUM  LDA #GETREC      ; GET RECORD (ENDS WITH CR)
4055 8D4203     790          STA ICCOM
4058 A980       800          LDA #LBUFF&255   ; SET BUFFER ADDRESS = LBUFF
405A 8D4403     810          STA ICBAL
405D A905       820          LDA #LBUFF/256
405F 8D4503     830          STA ICBAL+1
4062 A928       840          LDA #40          ; SET BUFFER LENGTH =40
4064 8D4803     850          STA ICBLL
4067 A900       860          LDA #0
4069 8D4903     870          STA ICBLL+1
406C A200       880          LDX #0           ; IOCB # = 0 FOR SCREEN EDITOR
406E 2056E4     890          JSR CIOV         ; CALL CIO
4071 A980       900          LDA #LBUFF&255   ; STORE BUFFER ADD. IN INBUFF
4073 85F3       910          STA INBUFF
4075 A905       920          LDA #LBUFF/256
4077 85F4       930          STA INBUFF+1
4079 A900       940          LDA #0           ; SET BUFFER INDEX = 0
407B 85F2       950          STA CIX
407D 2000D8     960          JSR AFP          ; CALL ATASCII TO FP
4080 60         970          RTS
4081 45         980  ERRMSG  .BYTE "ERROR",CR
4082 52
4083 52
4084 4F
4085 52
4086 9B
                1000 ; ROUTINE START INFO
4087            1020         * = $2E0
02E0 0040       1030         .WORD START
02E2            1040         .END
```

Figure 8-21 - Exemple d'utilisation des sous-programmes de virgule flottante



ATARINSIDE

# 9 - LE SYSTEME D'EXPLOITATION DE LA DISQUETTE (S.E.D.)

Le Système d'Exploitation de la Disquette (SED en français, DOS en anglais) est une extension du SE vous donnant accès à l'unité de disquette ATARI 810. Sans ce logiciel complémentaire, une unité de disquette est inutilisable et vous ne pouvez alors n'exploiter que le magnétocassette comme périphérique de stockage. Afin de ne pas encombrer en permanence la mémoire, le SED se charge depuis une disquette. Notez que 48 ko sont nécessaires à un bon fonctionnement du SED. Aussi, l'ordinateur A400 et le A600XL sans extension mémoire ne peuvent être utilisés avec une unité de disquette. Dans ce chapitre, nous décrivons le DOS 2.0S. Le DOS 3.0 est son adaptation aux nouvelles unités de disquette ATARI 1050, mais son fonctionnement reste très proche.

Le SED se compose de trois parties : le LGP (Logiciel de Gestion de Périphérique) «Disque» résident, le système de gestion de fichier (FMS), et un ensemble d'utilitaires (DUP). Le LGP est la seule partie intégrée dans la mémoire morte du SE. Le FMS et le DUP se chargent en mémoire depuis une disquette (la master par exemple) : le FMS, lors de la mise sous tension, le DUP lorsque vous tapez DOS au clavier ou, s'il n'y a pas de cartouche (A400/A800 seulement) ni de logiciel en «AUTORUN». Pour obtenir un maximum de renseignements sur le SED, lisez également le manuel du système d'exploitation et le manuel du SED.

## LE LGP «DISQUE» RESIDENT

Le LGP «DISQUE» est la partie la plus simple du SED. Il ne suit pas la procédure habituelle d'appel des autres LGP. Dans le DOS 2.0S, le LGP «DISQUE» résident ne sert que dans la phase d'initialisation (chargement du FMS). Après cette phase, le FMS est seul responsable des transactions avec les unités de disquette. Reportez-vous au chapitre 8 pour consulter la figure 8-8.

Le bloc de contrôle du périphérique (DCB) est utilisé pour communiquer avec le LGP. La figure 8-10 (chapitre 8) donne la structure du DCB. La séquence d'appel pour le LGP DISQUE est :

```
                ; DCB déjà préparé
JSR   DSKINV    ; vecteur vers le LGP résident
BPL   OKAY      ; branchement si succès, Y=1
                ; sinon, reg. Y = code de l'erreur = DCBSTA
```

Le LGP réalise les transferts physiques entre le 6502 placé à l'intérieur de l'ordinateur ATARI et l'autre microprocesseur placé dans l'unité de disquette A810. La communication s'établit via le bus entrées/sorties série. le LGP accepte quatre types de commande :

| | |
|---|---|
| FORMAT | Envoie une commande Format vers l'unité de disque |
| READ SECTOR | Lit le secteur spécifié |
| WRITE & VERIFY | Ecrit le secteur spécifié puis le vérifie |
| STATUS | Demande à l'unité de disque son état. |

La commande FORMAT efface toutes les pistes de la disquette et écrit un certain nombre de repères divisant la disquette en pistes et en secteurs. Aucun fichier n'est mis en place par cette commande. La partie des données de chaque secteur est remplie de zéros, la table d'occupation des secteurs (VTOC) et la table des matières (Directory) sont initialisées. Pour plus d'informations sur l'arrangement des données sur disquette, reportez-vous au manuel du système d'exploitation.

Vous devez noter que toutes les opérations d'entrées/sorties du LGP s'effectuent par secteur. Ainsi, vous pouvez lire ou écrire n'importe quel secteur de la disquette. En utilisant les commandes du FMS, il est possible de mettre en place n'importe quelle structure de fichier.

La commande STATUS lit quatre octets dans l'unité de disquette permettant de connaître l'état actuel. Ces octets sont transférés à partir de l'adresse $02EA (DVSTAT) dans la mémoire vive de l'ordinateur. Le premier octet relatif à la dernière commande reçue par le A810 et les bits sont utilisés ainsi :

| | |
|---|---|
| bit 0 | : 1 indique qu'une trame de commande invalide a été reçue |
| bit 1 | : 1 indique qu'une trame de donnée invalide a été reçue |
| bit 2 | : 1 indique qu'une commande PUT n'a pas été réalisée correctement |


ATARI INSIDE

bit 3 : 1 indique que la disquette est protégée contre toute écriture
bit 4 : 1 indique que le A810 est occupé.

Le second octet représente la copie du registre d'état du contrôleur de floppy INS1771-1 utilisé dans l'unité A810. Le troisième octet représente un nombre de secondes. Cette valeur, fournie par le A810 au LGP, indique le temps maximum nécessaire pour l'opération en cours. Le quatrième octet est inutilisé. Vous pouvez utiliser la commande STATUS pour vérifier le bon déroulement des entrées/sorties. La temporisation pour la commande STATUS étant inférieure à celle des autres commandes, vous pouvez l'utiliser pour vérifier si une unité de disquette spécifique est connectée. Si le LGP renvoie une erreur (temps maximum d'attente écoulé), vous saurez que l'unité n'est pas présente.

# LE LOGICIEL DE GESTION DE FICHIER (FMS)

Le FMS est un LGP non résident qui respecte la procédure normale d'échange avec le CIO. le FMS n'est pas présent en mémoire morte. Il réside sur disquette, dans un fichier appelé DOS.SYS.

Le FMS, comme les autres LGP, reçoit les commandes d'entrées/sorties du CIO. Le FMS utilise ensuite ses propres sous-programmes pour dialoguer avec les unités de disquette, cela en raison d'une erreur dans le système d'exploitation. Cette erreur donne un résultat incorrect lors d'une comparaison sur 16 bits des pointeurs des tampons, la comparaison étant utilisée lors des transferts par le SIO. Il a donc fallu augmenter le FMS de manière à le rendre indépendant et donc exact. Le FMS étant placé dans la mémoire vive de l'ordinateur, il est possible de le modifier quelque peu. Prenons un exemple : les circuits de l'unité de disquette sont capables d'exécuter une qutre fonction non supportée par le LGP résident : WRITE WITHOUT VERIFY (écriture sans vérification). Bien que le risque d'erreur soit légèrement supérieur, l'écriture sur disquette s'effectue plus vite. Pour obtenir ce résultat, il vous suffit de taper en Basic :

```
POKE 1913,80
```

Si vous désirez revenir au mode normal (avec vérification), tapez :

```
POKE 1913, 87
```

Le FMS est appelé en préparant un IOCB puis en passant par le CIO. Le FMS accepte quelques fonctions particulières non disponibles sur les autres LGP.

| | |
|---|---|
| FORMAT | Le FMS formate la disquette puis y écrit quelques secteurs. |
| NOTE | Le FMS donne à l'utilisateur la valeur actuelle du pointeur dans le fichier. |
| POINT | Le FMS modifie le pointeur dans le fichier. |

Le paragraphe relatif à l'accès direct donne quelques explications sur l'utilisation de NOTE et POINT.

# ENTREES/SORTIES SUR DISQUETTE

Vous réalisez les opérations standard d'entrées/sorties via le CIO. En Basic, cela revient à utiliser les instructions OPEN, CLOSE, GET, PUT et XIO. En Assembleur, vous devez préparer un IOCB puis appeler le CIO (voir chapitre 8).

Avant d'effectuer n'importe quelle opération d'entrée/sortie, vous devez d'abord ouvrir un fichier. La syntaxe Basic pour cette commande est :

```
OPEN # IOCB,ICAX1,0,"D:TEST.BAS"
```

# IOCB sélectionne l'un des 8 IOCB disponible. Evitez d'employer les IOCB 0, 6 et 7 car ils sont réservés au système d'exploitation et au Basic. L'argument ICAX1 est le code du type d'ouverture ; les bits sont utilisés de la manière suivante :

| BIT | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | x | x | x | x | W | R | D | A |

où

| | |
|---|---|
| A (Append) | = Ajout en fin de fichier |
| D (Directory) | = Le fichier à la table des matières |
| R (Read) | = Ouverture pour lecture seulement |
| W (Write) | = Ouverture pour écriture seulement |
| x | = Non utilisé |

Il est possible de positionner simultanément les bits D3 et D2 lorsqu'on désire aussi bien réaliser des accès en écriture et en lecture. Voici les combinaisons possibles :

| | |
|---|---|
| ICAX1=6 | Utilisé pour accéder à la table des matières. Un enregistrement lu donne un nom de fichier, son extension, sa taille et une indication sur son verrouillage éventuel. |


ATARINSIDE

| | |
|---|---|
| ICAX1 = 4 | Ouverture en lecture seulement. |
| ICAX1 = 8 | Ouverture en écriture seulement. Si le fichier existe déjà, il est d'abord effacé de la disquette. Le premier enregistrement commence au début du fichier. |
| ICAX1 = 9 | Ajout en écriture. Le fichier n'est pas modifié. Les enregistrements suivants se placent en fin de fichier. |
| ICAX1 = 12 | Ecriture + lecture. Le fichier n'est pas effacé mais l'écriture, comme la lecture, commence au début du fichier. |
| ICAX1 = 13 | Non prévu. |

Vous avez à votre disposition deux modes d'entrées/sorties pour transférer les données entre l'ordinateur et l'unité de disquette : enregistrement ou caractère.

Le mode caractère implique que les données dans un fichier constituent une suite séquentielle d'octets. Le SED interprête ces octets comme des valeurs sans signification particulière. En voici un exemple (en hexa) :

00 23 4F 55 FF 34 21 34 44

Le mode enregistrement signifie que le fichier se décompose en éléments logiques appelés enregistrement, chacun étant composé d'octets et se terminant par la valeur $9B (caractère de fin de ligne). En voici un exemple :

```
00  23  4F  55  FF  34  9B  21  34  44  9B
|     enregistrement 1      | enregistrement 2 |
```

Les deux modes peuvent être mélangés dans le même fichier. En fait, des enregistrements sont accessibles octet par octet par le mode caractère et un groupe de caractères peut être relu comme un enregistrement (à condition qu'il contienne le caractère $9B interprêté alors comme un délimiteur, et que la longueur du groupe ne soit pas excessive (supérieure à 120 caractères) ). La seule différence entre les deux modes réside donc dans l'interprêtation du caractère $9B : caractère de contrôle ou caractère ordinaire.

Le Basic accepte le mode enregistrement grâce à ses intructions PRINT et INPUT (écriture et lecture). Il passe en mode caractère si vous utilisez PUT et GET. Toutefois, notez que dans ce cas, vous manipulez un octet à la fois alors que le SE a la possibilité de transférer un groupe de caractères (sans que ce soit un enregistrement). Le Basic n'accepte pas tous les modes du SE. Si vous travaillez en Assembleur, vous pourrez utiliser ce mode caractère étendu en spécifiant la longueur et l'adresse du bloc de données devant être transféré. Il est envisageable que cette partie en Assembleur devienne un sous-programme d'un programme Basic, l'appel s'effectuant par l'instruction USR. La figure 8-16 constitue un exemple de ce mode particulier.

La commande XIO est une instruction générale d'entrées/sorties du Basic. Elle est décrite plus en détail quelques paragraphes ci-après.

# LE FICHIER DUP.SYS

Le fichier DUP.SYS contient un ensemble d'utilitaires que l'on utilise par l'intermédiaire d'un menu. Les commandes sont :

A. DIRECTORY (table des matières)
B. RUN CARTRIDGE (passe la main à la cartouche)
C. COPY FILES (copie de fichiers)
D. DELETE FILES (effacement de fichiers)
E. RENAME FILES (change le nom d'un fichier)
F. LOCK FILES (verrouillage de fichiers)
G. UNLOCK FILES (déverrouillage de fichiers)
H. WRITE DOS FILES (écriture du SED sur disquette)
I. FORMAT DISQUE (initialisation)
J. DUPLICATE DISK (copie de disquette)
K. SAVE BINARY FILE (sauvegarde d'un fichier binaire)
L. LOAD BINARY FILE (chargement d'un fichier binaire)
M. RUN AT ADRESS (exécution à partir de l'adresse ...)
N. WRITE MEM.SAV (création du fichier MEM.SAV)
O. DUPLICATE FILE (duplique un fichier)

Les paragraphes ci-dessous décrivent chacune de ces fonctions. Toutefois, si vous souhaitez davantage de détails, reportez-vous au manuel du SED II.

# JOCKERS

La plupart des commandes du DUP nécessitent un nom de fichier. Le DUP reconnaît deux jockers qui peuvent se substituer à des caractères dans un nom de fichier. Ces jockers sont le point d'interrogation (?) et l'étoile (*).


ATARINSIDE

Ces caractères sont utilisés dans les noms de fichier soit lorsqu'il est nécessaire de traiter plusieurs fichiers simultanément, soit lorsqu'on désire abréger la frappe.

Le point d'interrogation (?) remplace un caractère qui peut être n'importe lequel à condition qu'il existe à ce même rang dans le nom. L'astérisque remplace un groupe de caractères quelle que soit sa longueur. En voici quelques exemples :

| | |
|---|---|
| *.BAS | Tous les fichiers de l'unité 1 dont l'extension est .BAS |
| D2:*.* | Tous les fichiers de l'unité 2. |
| PRO*.BAS | Tous les fichiers de l'unité 1 dont les noms commencent par PRO et possédent l'extension BAS. |
| TEST?? | Tous les fichiers de l'unité 1 dont les noms commencent par TEST et comptent six caractères. |

A - DISK DIRECTORY (Table des matières)

C'est l'ensemble des noms des fichiers que contient la disquette. Cette commande listera les noms des fichiers, les extensions, et le nombre de secteurs que le fichier occupe sur la disquette. Une liste partielle apparaît si vous utilisez des jockers.

B - RUN CARTRIDGE (Passe la main à une cartouche)

C'est la cartouche en place qui reprend la main. Le DUP est abandonné. A partir de ce point, la suite dépend de la cartouche mise en place. Le Basic, par exemple, répondra en effaçant l'écran et en affichant READY.

Si la disquette dans l'unité 1 n'a pas été changée depuis que le DUP a été chargé, et si un fichier MEM.SAV existe sur cette disquette, alors le contenu de ce fichier est rechargé en mémoire avant que la cartouche ne prenne la direction des opérations. Ce fichier représente normalement l'ancien contenu de la partie inférieure de la mémoire, partie effacée par l'arrivée du DUP. Il assure une sauvegarde puis une restitution automatique de cette zone mémoire lorsque vous tapez respectivement DOS et B. Vous pouvez considérez que MEM.SAV et DUP.SYS alternent en mémoire. Nous vous conseillons de créer le fichier MEM.SAV sur vos disquettes afin d'éviter toute perte accidentelle de votre programme (ce qui arrive si vous tapez DOS avant de l'avoir sauvegardé).

C - COPY FILE (Copie de fichier)

La commande C copie un fichier d'une unité de disquette vers une autre unité de disquette ou vers un autre périphérique. Le fichier «origine» peut être désigné par des jockers et correspondre en fait à une série de fichiers devant être copiés. Le fichier «destination» se trouve généralement sur une disquette mais vous pouvez utiliser tout aussi bien l'écran (E:) ou l'imprimante (P:). Lorsque le second nom de fichier (destination) est suivi de l'option /A, cela indique que le premier fichier doit être ajouté au second. Cette option, très utile en Assembleur, ne peut fonctionner avec des fichiers Basic interprêtés.

D - DELETE FILE (Effacement de fichier)

Vous effacez un ou plusieurs fichiers de la disquette. Des jockers étendent les possibilités de description des fichiers. Dans un esprit de sécurité, l'ordinateur demande une confirmation après chaque nom de fichier trouvé par l'intermédiaire de jockers. Si vous êtes sûr de vous et si vous désirez rendre l'opération automatique, ajoutez l'option /N après le nom du fichier.

E - RENAME FILE (Change le nom d'un fichier)

Vous changez le nom d'un fichier de la disquette. Vous devez donner deux noms : l'ancien, puis le nouveau. Il n'est pas nécessaire de préciser dans le nouveau nom, le numéro de l'unité de disquette : c'est obligatoirement le même que l'ancien. Les jockers sont utilisés dans les deux noms (DANGER!). Par défaut, D1: est utilisé. Une erreur est signalée si le fichier n'existe pas, ou est verrouillé, ou si la disquette est protégée contre toute écriture.

F - LOCK FILE (Verrouillage de fichier)

En verrouillant un fichier, vous le protégez contre toute écriture ou effacement accidentel. Le nom d'un fichier verrouillé est précédé dans la table des matières par un astérisque. Notez toutefois que la commande I (Initialisation/Format) détruit les fichiers, qu'ils soient verrouillés ou non. Les jockers sont admis.

G - UNLOCK FILE (Déverrouillage de fichier)

Fonction inverse de la précédente, les jockers sont admis.

H - WRITE DOS FILE (Ecriture du SED sur disquette)

Cette option sert à recopier le SED sur une disquette initialisée qui devient ainsi une copie de la disquette master. Les fichiers DOS.SYS et DUP.SYS ne peuvent pas être copiés par l'option C.

I - FORMAT DISKETTE (Initialisation)

Cette option prépare une disquette pour sa future utilisation. Elle l'efface tout en déposant un certain nombre de repères délimitant les secteurs et les pistes. Si un défaut est détecté, l'unité de disquette tente une nouvelle initialisation. Au bout de plusieurs essais infructueux, la commande est abandonnée par le SED ; la disquette est alors certainement défectueuse. Attention! Tout ce qui pouvait se trouver sur la disquette est détruit, même les fichiers verrouillés. Seule la protection physique d'écriture reste efficace pour se protéger contre tout accident.

ATARINSIDE

J - DUPLICATE DISK (Copie de disquette)

Cette option recrée un double exact de la disquette d'origine, tant que cette master a été créée et gérée par le SED. Elle fonctionne avec une ou plusieurs unités. Avec une seule unité, vous intervertirez à la main les disquettes source et destination.

Le processus de copie s'effectue par groupes de secteurs. Toutefois, seuls les secteurs utilisés sont copiés.

Cette option est destructrice pour la disquette destination. Tout fichier qui s'y serait trouvé préalablement, est détruit. D'autre part, une sage précaution consiste à placer sur la disquette source une étiquette de protection d'écriture afin d'éviter toute catastrophe due à une erreur de manipulation.

K - BINARY SAVE (Sauvegarde d'un fichier binaire)

Vous sauvegardez sur disque le contenu d'une zone mémoire dans un format binaire, c'est-à-dire comme s'il s'agissait d'un fichier objet, résultat d'une opération d'assemblage. L'en-tête du fichier se compose de deux $FF, suivis de l'adresse de début de la zone mémoire, et de l'adresse de fin. Puis vient le fichier proprement dit, c'est-à-dire une suite d'octets. Vous devrez donner le nom du fichier, l'adresse de début, l'adresse de fin. En option, vous pourrez indiquer une adresse d'initialisation et une adresse d'exécution. Lorsque le fichier binaire représente un programme en code machine, la première donnera l'adresse de départ dès le chargement terminé, la seconde, l'adresse utilisée lors d'un System Reset.

L'adresse d'initialisation est placée dans le vecteur INIT ($02E0, sur deux octets) ; l'adresse d'exécution, dans le vecteur RUN ($02E2, sur deux octets).

L - BINARY LOAD (Chargement d'un fichier binaire)

Cette commande transfère un fichier binaire de la disquette dans la mémoire de l'ordinateur. Si des valeurs INIT ou RUN ont été précisées, le SE lancera l'exécution de ce fichier, considéré alors comme un programme.

M - RUN AT ADRESS (Exécution à partir de l'adresse ... )

Si le programme est chargé par l'option L ne possède pas d'adresse INIT ou RUN, vous pouvez lancer son exécution en utilisant l'option M et en connaissant l'adresse de départ. La valeur doit être donnée en hexadécimal.

N - CREATE MEM.SAV (Création du fichier MEM.SAV)

Par cette option, vous créez un fichier appelé MEM.SAV. Il sert, lorsqu'il existe, pour sauvegarder automatiquement le bas de la zone mémoire utilisateur juste avant que le fichier DUP.SYS ne soit chargé en mémoire. Inversement, en choisissant l'option B du DUP, le contenu de MEM.SAV est rappelé et reprend sa place en mémoire. MEM.SAV doit exister sur la disquette placée dans l'unité n°1. Notez que cette option accroît la fiabilité des manipulations (vous ne risquez plus de perdre un programme en ayant tapé DOS trop rapidement) mais qu'elle augmente les temps de chargement de 20 secondes environ.

O - DUPLICATE FILE (Dupliquer un fichier)

Cette option autorise la copie d'un fichier avec une seule unité de disquette. Vous pouvez également copier un programme qui aurait été conçu avec le DOS I (alors que l'option C ne l'autorise pas).

**UTILISATION DE LA COMMANDE XIO**

La commande Basic XIO est une instruction générale d'entrées/sorties réalisant un appel direct au CIO. Son format est :

XIO n° de commande, = IOCB,AUX1,AUX2,nom de fichier

Du Basic, vous pouvez ainsi réaliser des opérations accessibles normalement depuis le DUP. Voici quelques commandes possibles :

| COMMANDES | FONCTIONS |
|---|---|
| 3 | Ouverture |
| 5 | Lecture enregistrement |
| 7 | Lecture caractère |
| 9 | Ecriture enregistrement |
| 11 | Ecriture caractère |
| 12 | Fermeture |
| 13 | Requête de l'état du périphérique |
| 32 | Changement de nom |
| 33 | Effacement |
| 35 | Verrouillage |
| 36 | Déverrouillage |
| 254 | Initialisation |



ATARINSIDE

# ACCES DIRECT

L'intérêt d'une unité de disquette réside dans le fait qu'on peut y stocker des informations d'une manière séquentielle pour les relire très rapidement dans n'importe quel ordre : c'est ce qu'on appelle l'accès direct. En utilisant les commandes d'entrées/sorties disque ainsi que les instructions NOTE et POINT, vous pouvez créer et utiliser des fichiers à accès direct.

Le SED gère un pointeur de fichier pour chaque fichier ouvert. Ce pointeur lui indique le numéro du secteur et le numéro de l'octet dans le secteur correspondant au prochain emplacement qui sera lu ou écrit. NOTE permet de lire ce pointeur tandis que POINT le modifie. Le numéro du secteur est compris entre 1 et 719 inclus ; le numéro de l'octet varie de 1 à 125. La figure 9-2 donne la valeur de ces deux paramètres pour différentes positions dans le fichier. L'exemple commence arbitrairement au secteur 50.

| | A | B | C | EOL | D | E | F | EOL | G | H | I | J | K | EOL | A | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Fichier | 41 | 42 | 43 | 9B | 44 | 45 | 46 | 9B | 47 | 48 | 49 | 4A | 4B | 9B...41 | | 42 |
| Pointeur | | | | | | | | | | | | | | | | |
| Numéro de secteur | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | .50 | 50 | 50 | 50 | 50 | 50...50 | | 51 |
| Numéro d'octet | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D...7C | | 0 |

Figure 9-2 - Valeur des deux arguments du pointeur

Le fichier ci-dessus a été créé par le programme Basic suivant :

```
10 OPEN #1,8,0,"D:FILE"
20 ?#1;"ABC"
30 ?#1;"DEF"
40 ?#1;"GHIJK"
    :
    :              : REM Remplit le reste du secteur de la même manière
    :
100 ?#1;"AB"       : REM Et dépasse la capacité d'un secteur
150 CLOSE #1
```

Le secteur 51 a été atteint car nous avons dépassé la capacité d'un secteur (125 octets). Le FMS enchaîne automatiquement sur le secteur suivant.

Le compteur d'octet commence à zéro et est incrémenté à chaque caractère jusqu'à atteindre la fin du secteur : $7D (125 en décimal), cette valeur étant exclue. Le nombre maximum d'octets par secteur est donc 125 (0 à 124). Le SED utilise les trois derniers octets de chaque secteur pour le chaînage entre secteur. Quand le SED arrive à la fin d'un secteur, le compteur d'octets est remis à zéro.

Quand la commande POINT est utilisée pour positionner le pointeur du fichier, le SED vérifie que le secteur pointé appartient bien au fichier ouvert. Sinon, l'instruction POINT n'est pas exécutée.

La figure 9-3 présente un sous-programme qui peut être utilisé pour sauvegarder des enregistrements, mémoriser leurs emplacements et les relire en accès direct.



ATARINSIDE

```
1000 REM  CE PROGRAMME PERMET L'ACCES DIRECT POUR DES ENREGISTREMENTS
1001 REM  DE LONGUEUR FIXE
1002 REM
1003 REM  ... LES COMMANDES SONT
1004 REM  CMD=1     ECRITURE Nº ENREGISTREMENT
1005 REM  CMD=2     LECTURE Nº ENREGISTREMENT
1006 REM  CMD=3     MISE A JOUR Nº ENREGISTREMENT
1007 REM
1008 REM  RECORD$ EST L'ENREGISTREMENT
1009 REM  N EST SON NUMERO D'ORDRE
1010 REM  INDEX EST UN TABLEAU A DEUX DIMENSIONS (INDEX(1,RECNUM))
1015 REM  QUI MEMORISE LES VALEURS NOTE POUR TOUS ENREGISTREMENTS
1020 REM  CE PROGRAMME SUPPOSE QUE LE CANAL 1 A ETE OUVERT POUR E/S
1100 REM
1120 REM  LE PROGRAMME COMMENCE A LA LIGNE 1200
1130 REM
1200 ON CMD GOTO 2000,3000,4000
2000 REM ..........................
2100 REM ECRIT Nº ENREGISTREMENT
2200 NOTE #1,X,Y
2300 INDEX(SEC,N)=X:INDEX(BYTE,N)=Y
2400 ? #1;RECORD$:RETURN
3000 REM ..........................
3010 REM  LIT Nº ENREGISTREMENT
3020 REM
3030 X=INDEX(SEC,N):Y=INDEX(BYTE,N)
3040 POINT #1,X,Y
3050 INPUT #1;RECORD$
3060 RETURN
4000 REM ..........................
4010 REM  MISE A JOUR Nº ENREGISTREMENT
4020 REM
4040 X=INDEX(SEC,N):Y=INDEX(BYTE,N)
4050 POINT #1,X,Y
4060 ? #1;RECORD$
4070 RETURN
```

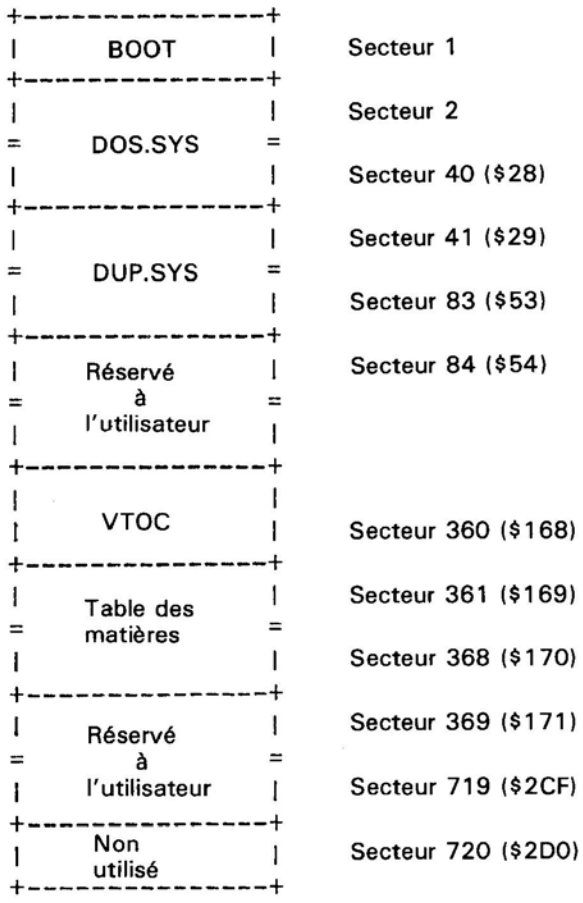
Figure 9-3 - Exemple d'utilisation de NOTE et POINT



ATARINSIDE

# UTILISATION DE LA DISQUETTE PAR LE FMS

La cartographie ci-dessous indique la manière dont la disquette est utilisée par le FMS (720 secteurs par disquette).

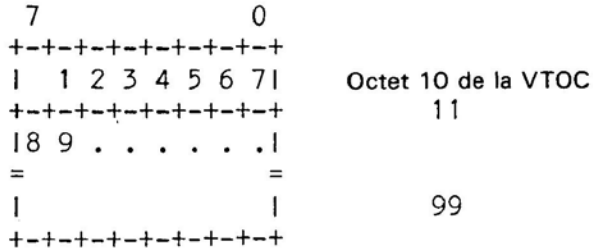



### LE FICHIER BOOT

Les trois premiers vecteurs de la disquette sont réservés au FMS. Ils contiennent les informations concernant la configuration du système et la présence du fichier DOS.SYS sur la disquette. Lorsqu'on utilise l'option H du SED sur une disquette qui vient d'être initialisée, les fichiers DOS.SYS et DUP.SYS s'enregistrent à partir du secteur 4. Mais on peut les créer à n'importe quel moment, que la disquette ait déjà des programmes ou non.

### VTOC : TABLE D'OCCUPATION DE LA DISQUETTE

Le secteur 360 est réservé par le FMS pour mémoriser les secteurs qui sont occupés et ceux qui sont libres. La VTOC étant consultée avant chaque écriture sur la disquette, le secteur 360 a été choisi car il se situe au milieu de la disquette et le temps d'accès moyen est donc minimum. La table d'occupation commence avec le 10e octet et s'étant jusqu'à l'octet 99. Chaque octet représente 8 secteurs consécutifs. Lorsqu'un bit vaut 0, le secteur correspondant est utilisé ; s'il vaut 1, le secteur est libre.



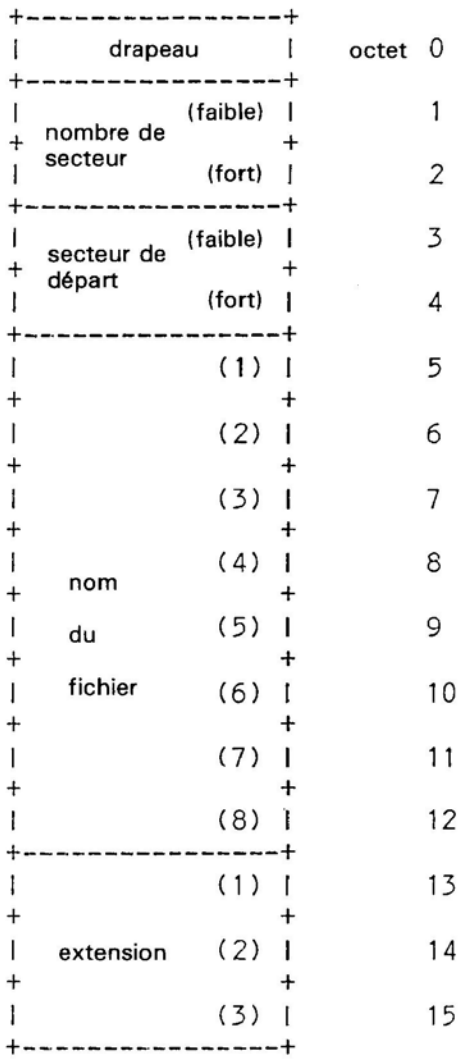



ATARINSIDE

**TABLE DES MATIERES**

Huit secteurs (361-368) sont réservés pour la table des matières de la disquette, chaque secteur pouvant enregistrer les noms de huit fichiers. Ainsi, le nombre maximum de fichiers pouvant être enregistrés sur une disquette est 64.

Chaque ligne de la table des matières comprend 16 octets. En voici la description :



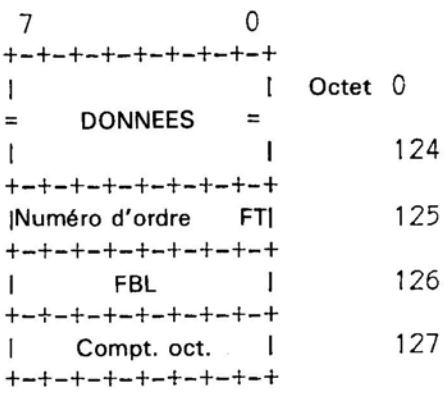

L'octet drapeau donne des renseignements sur l'état du fichier :

Bit 7 = 1 si le fichier a été effacé
Bit 6 = 1 si le fichier est utilisé
Bit 5 = 1 si le fichier est verrouillé
Bit 0 = 1 si le fichier est ouvert en écriture

ATARI INSIDE

**ORGANISATION D'UN SECTEUR**

Le format d'un secteur dans un fichier de la disquette se présente ainsi :




Le numéro d'ordre est une information que le FMS exploite pour s'assurer de l'intégrité du fichier. Cette valeur correspond à la position dans la table des matières du nom de ce fichier. S'il n'y a pas correspondance entre la position du fichier dans la table des matières et le numéro d'ordre dans le secteur, le FMS produira une erreur et arrêtera l'opération en cours.

FT et FBL représentent 10 bits (respectivement poids fort et poids faible) indiquant le numéro de secteur suivant dans le fichier. Lorsqu'on atteint la fin du fichier, le pointeur du dernier secteur vaut 0.

L'octet 127 indique le nombre d'octets utilisés dans ce secteur. Il vaut normalement 125, sauf dans le dernier secteur du fichier, à moins que l'on ait réalisé des opérations «ajout en fin de fichier».

## LE FICHIER AUTORUN.SYS

Le SED a une particularité intéressante : lorsqu'un programme s'appelant AUTORUN.SYS est présent sur la disquette à la mise sous tension du système, ce fichier est automatiquement chargé en mémoire et exécuté. Il peut s'agir de données modifiant certains paramètres du système comme par exemple les valeurs des marges ou ce peut être un programme en Assembleur s'exécutant avant que le chargement normal du SED se poursuive.

Ce fichier doit être dans un format binaire, et non pas Basic. Pour le rendre exécutable automatiquement, une valeur d'adresse doit être placée dans le vecteur INIT ($02E0, deux octets) et/ou RUN ($02E2, deux octets) par le fichier AUTORUN.SYS lui-même lors de son chargement. La différence entre ces deux vecteurs est que le code pointé par INIT sera exécuté dès que INIT est modifié, alors que le code pointé par RUN sera exécuté après que le processus de chargement soit terminé. Pour redonner la main au SED après l'exécution d'un fichier AUTORUN.SYS, terminez votre code avec l'instruction RTS.

Le fichier AUTORUN.SYS est particulièrement utile pour la mise en place de programmes Assembleur «auto-exécutables». Vous pouvez de cette manière reconfigurer le système d'exploitation et dérouter certains vecteurs avant de redonner la main au SED. Reportez-vous à la figure 8-3 : c'est un exemple d'utilisation d'un fichier AUTORUN.SYS pour modifier le pointeur MEMLO.

ATARINSIDE

# 10 - LE BASIC ATARI

## QU'EST-CE QUE LE BASIC ATARI?

Le BASIC ATARI est un langage interprêté. Cela veut dire que les programmes peuvent s'exécuter dès qu'ils ont été tapés au clavier sans nécessiter d'étapes intermédiaires de compilation et de chaînage. L'interprêteur BASIC ATARI se présente soit sous la forme d'une cartouche de mémoire morte 8ko (ATARI 400 et 800), soit comme un module intégré résident (ATARI 600XL et 800XL).

Pour utiliser efficacement le BASIC ATARI, vous devez connaître ses avantages et ses points faibles. A la fin de e chapitre, vous saurez exploiter très efficacement ce langage.

**AVANTAGES DU BASIC ATARI**

• Il est compatible avec les modes graphiques du système d'exploitation.

• Il est compatible avec les circuits de l'ordinateur et utilise très facilement les possibilités de la machine par quelques instructions comme SOUND, STICK, PADDLE.

• Il s'interface facilement avec des programmes Assembleur. La fonction USR est conçue pour cela.

• Le fait qu'il soit placé en mémoire morte le rend inaltérable par l'utilisateur. Il est compatible avec une exploitation rationnelle du SED : instructions NOTE et POINT.

• N'importe quel périphérique reconnu par le système d'exploitation peut être utilisé par un programme BASIC.

**POINTS FAIBLES DU BASIC ATARI**

• Il n'autorise pas l'emploi de nombres entiers. Tous les nombres sont représentés dans un format sur 6 octets, en notation virgule flottante.

• Les calculs mathématiques sont lents, cela étant dû principalement à la notation virgule flottante obligatoire.

• Il n'existe pas de tableau de chaîne de caractères. Seules des chaînes à une dimension peuvent être créées.

## COMMENT FONCTIONNE LE BASIC ATARI?

• Le BASIC accepte une ligne de programme en provenance de l'utilisateur puis la convertit en forme interprêtée.

• Cette ligne est ensuite placée dans la zone mémoire réservée au programme.

• Le programme est ensuite exécuté lorsque l'utilisateur tape RUN.

Le détail de ces opérations est décrit dans les quatre paragraphes suivants :

• Le processus d'interprêtation
• La structure du fichier interprêté
• Le processus d'exécution du programme
• L'interaction avec le système.

## LE PROCESSUS D'INTERPRETATION

En simplifiant quelque peu, l'interprêtation d'une ligne de BASIC s'effectue ainsi :

1. Le BASIC accepte une ligne d'information (entrée).
2. Il vérifie la syntaxe.
3. Durant cette opération, les mots BASIC sont codés.
4. La ligne finale résultante est ensuite placée dans la zone programme.
5. Si la ligne a été tapée en mode immédiat, elle est exécutée.



ATARINSIDE

Pour mieux comprendre le processus d'interprêtation, quelques termes doivent être définis :

| | |
|---|---|
| Code | Un octet contenant le code d'une commande BASIC. |
| Instruction | Une phrase complète composée éventuellement de plusieurs commandes demandant au BASIC d'exécuter une tâche particulière. Les instructions sont séparées par «:» lorsqu'il existe plusieurs instructions par ligne. |
| Ligne | Une ou plusieurs instructions précédées soit par un numéro de ligne dans la gamme 0-32767, soit par rien du tout (mode immédiat). |
| Commande | Le premier code d'une instruction indiquant au BASIC comment interprêter les codes qui suivent, jusqu'à la prochaine commande. |
| Variable | Un code qui est un pointeur indirect vers une valeur réelle ; ainsi, la valeur peut être changée sans modifier son code. |
| Constante | Une valeur numérique codée sur six octets et précédée par un code particulier unique pour toutes les constantes. Cette valeur est figée et ne se modifie pas durant l'exécution du programme. |
| Opérateur | Un quelconque des 46 codes susceptible de modifier ou de déplacer les valeurs qui les suivent. |
| Fonction | Un code qui, lorsqu'il est exécuté, renvoie une valeur au programme. |
| EOL | End Of Line. Caractère $9B indiquant la fin d'une ligne équivalent à la touche RETURN. |
| BCD | Décimal Codé Binaire. Un nombre qui utilise le mode décimal du 6502 plutôt que le mode binaire. |

Le BASIC commence le processus d'interprêtation en recevant une ligne d'information (une entrée). Celle-ci proviendra de l'un des LGP du système d'exploitation. Normalement, elle provient de l'éditeur écran ; toutefois, avec la commande ENTER, n'importe quel périphérique peut être spécifié. L'instruction INPUT produit une commande LECTURE D'ENREGISTREMENT et les données reçues sont codées en ATASCII et terminées par un EOL. Cette ligne est placée par le CIO dans le tampon d'entrée du BASIC, de l'adresse $580 à l'adresse $5FF.

Dès lors, la vérification de la syntaxe et l'interprêtation commencent. En premier, le BASIC recherche un numéro de ligne. S'il existe, ce numéro est converti en un entier sur deux octets. Si aucun numéro de ligne n'est présent, la ligne devra être exécutée immédiatement et le numéro de ligne $8000 lui est attribué. Ces deux octets constituent les deux premiers codes de la ligne interprêtée stockée dans un tampon de sortie de 256 octets et située à la fin de la mémoire vive réservée au système d'exploitation.

Le code suivant est un octet réservé pour un compteur indiquant le nombre total d'octets utilisés par cette ligne interprêtée. Sa valeur définitive est calculée à la fin du processus d'interprêtation. L'octet suivant compte les octets depuis le début de la ligne jusqu'au début de l'instruction suivante de la même ligne. Cette valeur est calculée lorsque la première instruction a été interprêtée. Nous verrons l'interêt de ces deux octets lorsque nous étudierons le processus d'exécution du programme.

Le BASIC cherche maintenant la commande de la première instruction de la ligne d'entrée. Une vérification est faite pour déterminer s'il s'agit d'une commande valide en passant en revue une liste de toutes les commandes légales placée en mémoire morte. Si une égalité est trouvée, l'octet placé alors dans la ligne interprêtée représente le rang dans la liste. Si aucune correspondance n'est trouvée, un code d'erreur de syntaxe est placé dans cet octet et le BASIC cesse l'interprêtation, copie le reste du tampon d'entrée dans le tampon de sortie, en utilisant le format ATASCII, et affiche la ligne d'erreur.

En supposant que la ligne soit correcte, la commande est suivie par l'un des septs mots suivants : une variable, une constante, un opérateur, une fonction, un guillemet, une autre instruction, ou un EOL. Le BASIC teste si le caractère suivant est numérique. S'il ne l'est pas, le BASIC compare ce caractère et les suivants avec les éléments de la table des noms de variable. Si aucune correspondance n'est trouvée, le BASIC suppose qu'il s'agit d'un nouveau nom de variable. Les caractères sont alors transférés du tampon d'entrée dans la table des noms et le bit de poids le plus fort du dernier octet est positionné à 1. Huit octets sont ensuite réservés dans la table des valeurs des variables (nous en reparlerons au paragraphe «structure du fichier interprêté»).

Le code qui est placé dans la ligne interprêtée est le numéro de variable – 1 avec le bit de poids fort positionné. Ainsi, le code de la première variable sera $80, le second sera $81, et ainsi de suite jusqu'à FF, ce qui autorise donc 128 variables différentes.

S'il s'agit d'une fonction, le code représente le rang de la fonction dans la table des fonctions et des opérateurs. Les fonctions exigent certaines séquences de paramètres : elles sont contenues dans les tables de syntaxe, et s'il n'y a pas correspndance, une erreur de syntaxe se produit.

S'il s'agit d'un opérateur, le code est donné par son rang dans la table correspondante. Les opérateurs peuvent se suivre d'une manière complexe (parenthèses imbriquées par exemple) si bien que la vérification de la syntaxe se complique.

S'il s'agit de guillemets, le BASIC considère qu'il a affaire à une chaîne de caractères et il place les valeurs $0F dans la ligne interprêtée et réserve un octet pour la longueur de la ligne. Puis les caractères sont transférés du tampon d'entrée au tampon de sortie jusqu'à ce que le BASIC lise à nouveau des guillemets ou atteigne la fin de la ligne. L'octet indiquant la longueur est alors mis à jour.


ATARINSIDE

Si les caractères placés dans le tampon d'entrée sont numériques, le BASIC les convertit sous forme d'une constante utilisant six octets dans le format BCD. Le code placé dans le tampon de sortie est $0E et il est suivi par les six octets.

Lorsqu'une virgule est lue, le code $14 est placé dans le tampon de sortie et le compteur d'octet situé au début de l'instruction est mis à jour. Puis un nouvel octet est réservé pour servir de compteur d'offset pour la commande suivante. Ainsi, en passant d'offset en offset, il sera possible de sauter rapidement de commande en commande et d'instruction en instruction.

Lorsque le caractère d'EOL est lu, le code $16 est placé dans la ligne interprêtée et le dernier offset est calculé. A ce stade, l'interprêtation est terminée et le BASIC transfère la ligne interprêtée dan la zone programme. Pour cela, il commence par rechercher une ligne ayant le même numéro. Si elle existe, il remplace l'ancienne ligne par la nouvelle. Sinon, il insère la nouvelle ligne en respectant l'ordre des numéros de ligne. Dans tous les cas, les données placées après la ligne traitée seront déplacées en conséquence dans la mémoire.

Le BASIC teste maintenant si la ligne interprêtée est en mode immédiat. Si oui, cette ligne est exécutée conformément aux méthodes décrites dans le processus d'interprêtation ; si non, le BASIC attend une nouvelle ligne d'entrée.

Si la longueur de la ligne interprêtée excède 256 octets, le message ERROR 14 (ligne trop longue) est envoyé à l'écran et le BASIC attend une nouvelle ligne.

Voici un exemple d'une ligne telle qu'elle serait tapée au clavier et, sa correspondance interprêtée :

10 LET X=1 : PRINT X

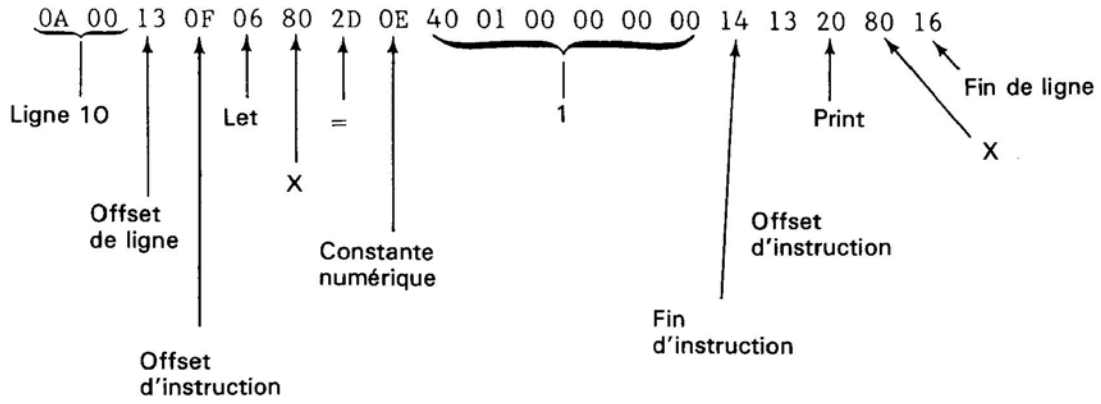



Figure 10-1 - Exemple d'interprêtation

| COMMANDE | | OPERATEUR | | FONCTION | |
|---|---|---|---|---|---|
| HEX | DEC | HEX | DEC | HEX | DEC |
| 00 | 0 REM | 0E | 14 [NUM CONST] | 3D | 61 STR$ |
| 01 | 1 DATA | 0F | 15 [STR CONST] | 3E | 62 CHR$ |
| 02 | 2 INPUT | 10 | 16 [NOT USED] | 3F | 63 USR |
| 03 | 3 COLOR | 11 | 17 [NOT USED] | 40 | 64 ASC |
| 04 | 4 LIST | 12 | 18 , | 41 | 65 VAL |
| 05 | 5 ENTER | 13 | 19 $ | 42 | 66 LEN |
| 06 | 6 LET | 14 | 20 : [STMT END] | 43 | 67 ADR |
| 07 | 7 IF | 15 | 21 ; | 44 | 68 ATN |
| 08 | 8 FOR | 16 | 22 [LINE END] | 45 | 69 COS |
| 09 | 9 NEXT | 17 | 23 GOTO | 46 | 70 PEEK |
| 0A | 10 GOTO | 18 | 24 GOSUB | 47 | 71 SIN |
| 0B | 11 GO TO | 19 | 25 TO | 48 | 72 RND |
| 0C | 12 GOSUB | 1A | 26 STEP | 49 | 73 FRE |
| 0D | 13 TRAP | 1B | 27 THEN | 4A | 74 EXP |
| 0E | 14 BYE | 1C | 28 # | 4B | 75 LOG |
| 0F | 15 CONT | 1D | 29 <= [NUMERICS] | 4C | 76 CLOG |
| 10 | 16 COM | 1E | 30 <> | 4D | 77 SQR |
| 11 | 17 CLOSE | 1F | 31 >= | 4E | 78 SGN |
| 12 | 18 CLR | 20 | 32 < | 4F | 79 ABS |
| 13 | 19 DEG | 21 | 33 > | 50 | 80 INT |
| 14 | 20 DIM | 22 | 34 = | 51 | 81 PADDLE |
| 15 | 21 END | 23 | 35 | 52 | 82 STICK |
| 16 | 22 NEW | 24 | 36 * | 53 | 83 PTRIG |
| 17 | 23 OPEN | 25 | 37 + | 54 | 84 STRIG |
| 18 | 24 LOAD | 26 | 38 - | | |
| 19 | 25 SAVE | 27 | 39 / | | |
| 1A | 26 STATUS | 28 | 40 NOT | | |
| 1B | 27 NOTE | 29 | 41 OR | | |
| 1C | 28 POINT | 2A | 42 AND | | |
| 1D | 29 X10 | 2B | 43 ( | | |
| 1E | 30 ON | 2C | 44 ) | | |
| 1F | 31 POKE | 2D | 45 = [ARITHM ASSIGN] | | |
| 20 | 32 PRINT | 2E | 46 = [STRING ASSIGN] | | |
| 21 | 33 RAD | 2F | 47 <= [STRINGS] | | |
| 22 | 34 READ | 30 | 48 <> | | |
| 23 | 35 RESTORE | 31 | 49 >= | | |
| 24 | 36 RETURN | 32 | 50 < | | |
| 25 | 37 RUN | 33 | 51 > | | |
| 26 | 38 STOP | 34 | 52 = | | |
| 27 | 39 POP | 35 | 53 + [UNARY] | | |
| 28 | 40 ? | 36 | 54 - | | |
| 29 | 41 GET | 37 | 55 ( [STRING LEFT PAREN] | | |
| 2A | 42 PUT | 38 | 56 ( [ARRAY LEFT PAREN] | | |
| 2B | 43 GRAPHICS | 39 | 57 ( [DIM ARRAY LEFT PAREN] | | |
| 2C | 44 PLOT | 3A | 58 ( [FUN LEFT PAREN] | | |
| 2D | 45 POSITION | 3B | 59 ( [DIM STR LEFT PAREN] | | |
| 2E | 46 DOS | 3C | 60 , [ARRAY COMMA] | | |
| 2F | 47 DRAWTO | | | | |
| 30 | 48 SETCOLOR | | | | |
| 31 | 49 LOCATE | | | | |
| 32 | 50 SOUND | | | | |
| 33 | 51 LPRINT | | | | |
| 34 | 52 CSAVE | | | | |
| 35 | 53 CLOAD | | | | |
| 36 | 54 [IMPLIED LET] | | | | |
| 37 | 55 ERROR - [SYNTAX] | | | | |



ATARINSIDE

# LA STRUCTURE DU FICHIER INTERPRETE

Un fichier interprêté se compose de deux parties principales :
1. un groupe de pointeurs en page zéro,
2. le fichier lui-même.

Les pointeurs en page zéro sont des valeurs sur deux octets qui pointent vers diverses parties du fichier. Il existe neuf pointeurs situés aux adresses $80 à $91. En voici la liste :

| Pointeur (hexa) | Partie du fichier interprêté (blocs contigus) |
|---|---|
| LOMEM 80,81 | Tampon de sortie. C'est le tampon de 256 octets utilisé par le BASIC pour stocker la ligne interprêtée au fur et à mesure de sa construction. Ce tampon se situe à la fin de la mémoire vive allouée au système d'exploitation. |
| VNTP 82,83 | Table des noms de variable. C'est la liste de tous les noms de variable qui ont été utilisés par le programme. Les noms sont enregistrés sous forme de caractères ATASCII et dans leur ordre d'entrée. Trois types de noms existent :<br>1. variable scalaire ; le bit de poids fort est à 1 pour le dernier caractère du nom.<br>2. variable chaîne de caractères ; le dernier caractère est un «$» avec D7 = 1.<br>3. variable tableau ; le dernier caractère est un «(» avec D7 = 1. |
| VNTD 84,85 | Fin de la table des noms de variables. Le BASIC utilise ce pointeur pour indiquer la fin de la table des noms. Il pointe normalement vers un octet nul quand il existe moins de 128 variables. Lorsque 128 variables sont présentes, ce pointeur indique l'adresse du dernier octet du dernier nom de variable. |
| VVTP 86,87 | Table des valeurs des variables. Cette table contient les valeurs actuelles des variables. Pour chaque variable déclarée dans la table des noms, huit octets sont réservés dans cette table des valeurs. Les informations pour chaque type de variable sont : |

| N° de l'octet | 1 | 2 | 3 4 | 5 6 | 7 8 |
|---|---|---|---|---|---|
| scalaire | 00 | n° var | constante sur 6 octets | | |
| tableau (dimens.) | 41 | n° var | offset depuis STARP ($8C, $8D) | DIM + 1 | DIM + 1 |
| (non dimens.) | 40 | | | | |
| chaîne (dimens.) | 81 | n° var | Offset depuis STARP ($8C, $8D) | Longueur | Dimension |
| (non dimens.) | 80 | | | | |

Une variable scalaire contient une valeur numérique. Un exemple est X = 1. Le scalaire est X et sa valeur est 1, enregistrée dans le format virgule flottante sur six octets. Un tableau est composé d'éléments numériques stockés dans une zone mémoire particulière pointée par STARP et l'offset associé. Une chaîne, composée de caractères, est placée dans la même zone et son adresse se calcule également en partant de STARP auquel on ajoute son offset.

Le premier octet de chaque valeur indique le type de la variable : 00 pour un scalaire, $40 pour un tableau, et $80 pour une chaîne. Si le tableau ou la chaîne a été dimensionné, alors le bit de poids faible du premier octet est positionné à 1.

Le second octet contient le numéro de la variable. Pour la première variable, cette valeur vaut 0, et si 128 variables sont présentes, la dernière vaut 7F.

Dans le cas d'une variable scalaire, les octets 3 à 8 représentent le nombre lui-même dans le format BCD virgule flottante. Cette valeur change évidemment en fonction du programme.

Pour les tableaux et les chaînes, les octets 3 et 4 constituent un offset partant du début de la zone mémoire réservée à cet effet (STARP) et désignant le premier octet de la variable.

Les octets 5 et 6 d'une variable tableau constituent une quantité entière sur 16 bits représentant la dimension de ce tableau. Cette quantité est égale à la valeur définie par le programme + 1. Les octets 7 et 8 représentent la seconde dimension, définie par le programme + 1.

Pour une variable chaîne de caractères, les octets 5 et 6 forment un entier sur 16 bits représentant la longueur réelle de la chaîne. Les octets 7 et 8 représentent sa dimension (jusqu'à 32767).

| | |
|---|---|
| STMTAB 88,89 | Table des instructions. C'est le programme proprement dit, sous sa forme interprêtée. Il se termine par la ligne tapée en mode immédiat. Nous avons vu précédemment le format d'une ligne interprêtée. |



ATARINSIDE

STMCUR 8A,8B

Ce pointeur est utilisé par le BASIC comme référence à des codes particuliers à l'intérieur d'une ligne de la table des instructions. Quand le BASIC attend une entrée, ce pointeur désigne le début de la ligne en mode immédiat.

STARP 8C,8D

Début de la mémoire réservée aux tableaux et aux chaînes de caractères. Les chaînes de caractères sont stockées directement dans leur format ATASCII ; une chaîne de 20 caractères utilisera 20 octets. Les tableaux utilisent six octets pour chaque élément. Un tableau à 10 éléments utilisera 60 octets. Cette zone mémoire s'étend au fur et à mesure que le programme exécute les instructions DIM.

RUNSTK 8E,8F

Pile réservée au programme. Cette pile mémorise les entrées GOSUB et FOR...NEXT. Une instruction GOSUB provoque l'empilement de 4 octets. Le premier vaut 0 et indique qu'il s'agit d'un GOSUB. Il est suivi par le numéro de la ligne de l'instruction GOSUB (valeur sur 2 octets) et par l'offset correspondant à l'instruction suivante dans cette ligne (1 octet). Cela détermine le point de reprise du programme lors de l'exécution de l'instruction RETURN.

L'instruction FOR...NEXT provoque l'empilement de 16 octets. La première quantité empilée est la limite fixée par la variable compteur. La seconde quantité représente le pas. Chacune de ces quantités est présentée dans le format virgule flottante. Le 13e octet est le rang de la variable compteur avec le bit de poids fort positionné à 1. Les octets 15 et 16 représentent le numéro de ligne, et le 16e l'offset pour l'instruction FOR.

MEMTOP 90,91

Pointeur du haut de la mémoire vive. Il indique la fin de la zone mémoire utilisée par le programme utilisateur. Le programme peut grossir tant que ce programme n'atteint pas le début de la Display List. Notez que ce pointeur n'est pas le même que celui de la variable du système d'exploitation MEMTOP. La fonction FRE calcule le nombre d'octets libres en mémoire vive en soustrayant MEMTOP de HIMEM ($2E5, $2E6)

# LE PROCESSUS D'EXECUTION DU PROGRAMME

L'exécution d'une ligne de code est un processus qui commence à la lecture des codes créés durant l'interprêtation. Chaque code possède un sens particulier qui entraîne l'exécution d'une suite spécifique d'opérations. Le BASIC lit donc un code à la fois et le traite. Le code est un index dans une table de sauts renvoyant vers le sous-programme adéquat. Ainsi, le code de PRINT pointe indirectement vers le programme d'affichage et d'impression. Ensuite, le BASIC passe au code suivant. Le pointeur qui sert à passer en revue les codes les uns après les autres est appelé STMCUR, aux adresses $8A, $8B.

La première ligne de code qui est exécutée est celle du mode immédiat. Il s'agit généralement d'un RUN ou d'un GOTO. Dans le cas de l'instruction RUN, le BASIC lit la première ligne de code de la table des instructions. Puis il exécute les lignes dans l'ordre croissant des numéros.

Si l'instruction GOTO est rencontrée, il faut d'abord trouver la ligne destination. Pour cela, l'interprêteur BASIC reprend la table des instructions à son début.

L'adresse de la première ligne est mémorisée dans le pointeur STMTAB situé aux adresses $88 et $89. Cette adresse est maintenant placée dans un pointeur temporaire. Les deux premiers octets de la première ligne constituent son numéro de ligne, qui est comparé au numéro recherché. Si le numéro de ligne recherché est plus grand, le BASIC lit le troisième octet de la première ligne : c'est l'offset de ligne. Il l'ajoute au pointeur temporaire qui pointe maintenant vers la seconde ligne. A nouveau les deux premiers octets de cette nouvelle ligne sont comparés au numéro recherché et le cycle se poursuit jusqu'à trouver l'égalité. A ce moment, le contenu du pointeur temporaire est recopié dans STMCUR et le BASIC exécute les codes de cette ligne. S'il n'y a jamais correspondance, une ERROR 12 est générée.

L'instruction GOSUB nécessite un traitement légèrement plus complexe que le GOTO. Le processus pour trouver la ligne destination reste le même, mais avant que le BASIC n'exécute le code de cette ligne, il place quatre octets dans la pile réservée au programme. Le premier contient 0 et indique qu'il s'agit d'un GOSUB. Les deux suivants représente le numéro de la ligne contenant l'instruction GOSUB. Et enfin, le dernier octet contient l'offset dans cette ligne compté depuis le début de la ligne jusqu'à l'instruction GOSUB. Le BASIC exécute ensuite la nouvelle ligne de code et lorsque l'instruction RETURN est trouvée, ces quatre octets sont dépilés. Le BASIC revient à l'instruction suivant le GOSUB d'origine.

L'instruction FOR utilise 16 octets dans la pile. Les six premiers octets constituent la limite que la variable peut atteindre, dans un format BCD sur six octets. Les six octets suivants forment le pas, dans un même format. Puis le BASIC place le numéro de la variable (rang de la variable dans la table des noms) avec D7 à 1. Enfin, il enregistre le numéro de la ligne contenant l'instruction FOR ainsi que l'offset de cette instruction dans la ligne, puis le reste de la ligne est exécuté.

Quand le BASIC trouve la commande NEXT, il examine le dernier groupe d'octets dans la pile. Il vérifie que la variable utilisée par l'instruction NEXT est la même que celle placée dans la pile ; puis il vérifie si le compteur a atteint ou dépassé la limite. Si non, le BASIC utilise numéro de ligne et offset pour reprendre l'exécution à partir de l'instruction FOR. Si oui, les 16 octets sont effacés de la pile et l'exécution continue à partir de l'instruction NEXT.

Quand une expression est évaluée, les opérateurs sont placés dans une pile spéciale puis dépilés un par un au fur et à mesure de leur évaluation. L'ordre dans lequel les opérateurs sont empilés peut être implicite, le BASIC se réfère alors à une table de précédence figée en mémoire morte, ou explicitement établie par l'utilisation de parenthèses.



ATARINSIDE

Le fait d'appuyer sur la touche BREAK à n'importe quel moment provoque le positionnement d'un drapeau par le système d'exploitation. Le BASIC teste ce drapeau après l'exécution de chaque code. Si le drapeau est à 1, l'interprêteur mémorise le numéro de la ligne en cours, imprime le message «STOPPED AT LINE xxxx», efface le drapeau et attend une intervention de l'utilisateur. Celui-ci peut taper CONT et l'exécution du programme reprend alors à la ligne suivante.

# L'INTERACTION AVEC LE SYSTEME

Le Basic communique avec le système d'exploitation principalement en appelant le CIO. La liste ci-dessous donne les instructions d'entrées/sorties BASIC et la préparation des IOCB correspondants.

| BASIC | SYSTEME D'EXPLOITATION |
|---|---|
| OPEN = 1,12,0,"E:" | IOCB = 1<br>Commande = 3 (OPEN)<br>AUX1 = 12 (Entrée/sortie)<br>AUX2 = 0<br>Adresse du tampon = ADR("E:") |
| GET = 1,X | IOCB = 1<br>Commande = 7 (GET)<br>Longueur du tampon = 0<br>Caractère retourné dans l'accumulateur |
| PUT = 1,X | IOCB = 1<br>Commande = 11 (PUT)<br>Longueur du tampon = 0<br>Le caractère se trouve dans l'accumulateur |
| INPUT = 1,A$ | IOCB = 1<br>Commande = 5 (GET RECORD)<br>Longueur du tampon = longueur de A$ (inférieure ou égale à 120)<br>Adresse du tampon = tampon d'entrée du BASIC |
| PRINT = 1,A$ | IOCB = 1<br>Le BASIC utilise un vecteur spécial dans l'IOCB pour contrôler directement le LGP (Logiciel de Gestion des Périphériques) |
| XIO 18, = 6,12,0,"S:" | IOCB = 6<br>Commande = 18 (Remplissage de zone)<br>AUX1 = 12<br>AUX2 = 0 |

SAVE/LOAD : quand un programme BASIC interprêté est dirigé vers un périphérique dans le but d'être sauvegardé, deux blocs d'information sont écrits. Le premier bloc se compose de sept des neuf pointeurs placés en page zéro que le BASIC utilise pour gérer le fichier interprêté. Ce bloc commence à LOMEM ($80, $81) et finit à STARP ($8C, $8D). Une modification est faite sur ces pointeurs avant qu'ils soient enregistrés : la valeur de LOMEM est soustraite de chacun de ces pointeurs. Ainsi, les deux premiers octets écrits seront 0,0.

Le second bloc d'information écrit se compose de :

1. la table des noms de variables
2. la table des valeurs des variables
3. le programme interprêté
4. la ligne en mode immédiat.

Quand ce programme est chargé dans la mémoire, le BASIC lit la valeur de la variable du système d'exploitation MEMLO ($2E7, $2E8), et ajoute cette valeur à chacun des pointeurs lus. Le résultat est replacé en page zéro ($80, $8F) et MEMTOP ($90, $91) est initialisé par la valeur de STARP.

Puis 256 octets sont réservés en mémoire au-dessus de la valeur de MEMLO : c'est le tampon de sortie du BASIC. Puis le programme interprêté composé des quatre parties vues précédemment est chargé en mémoire, immédiatement après le tampon de sortie.



ATARINSIDE

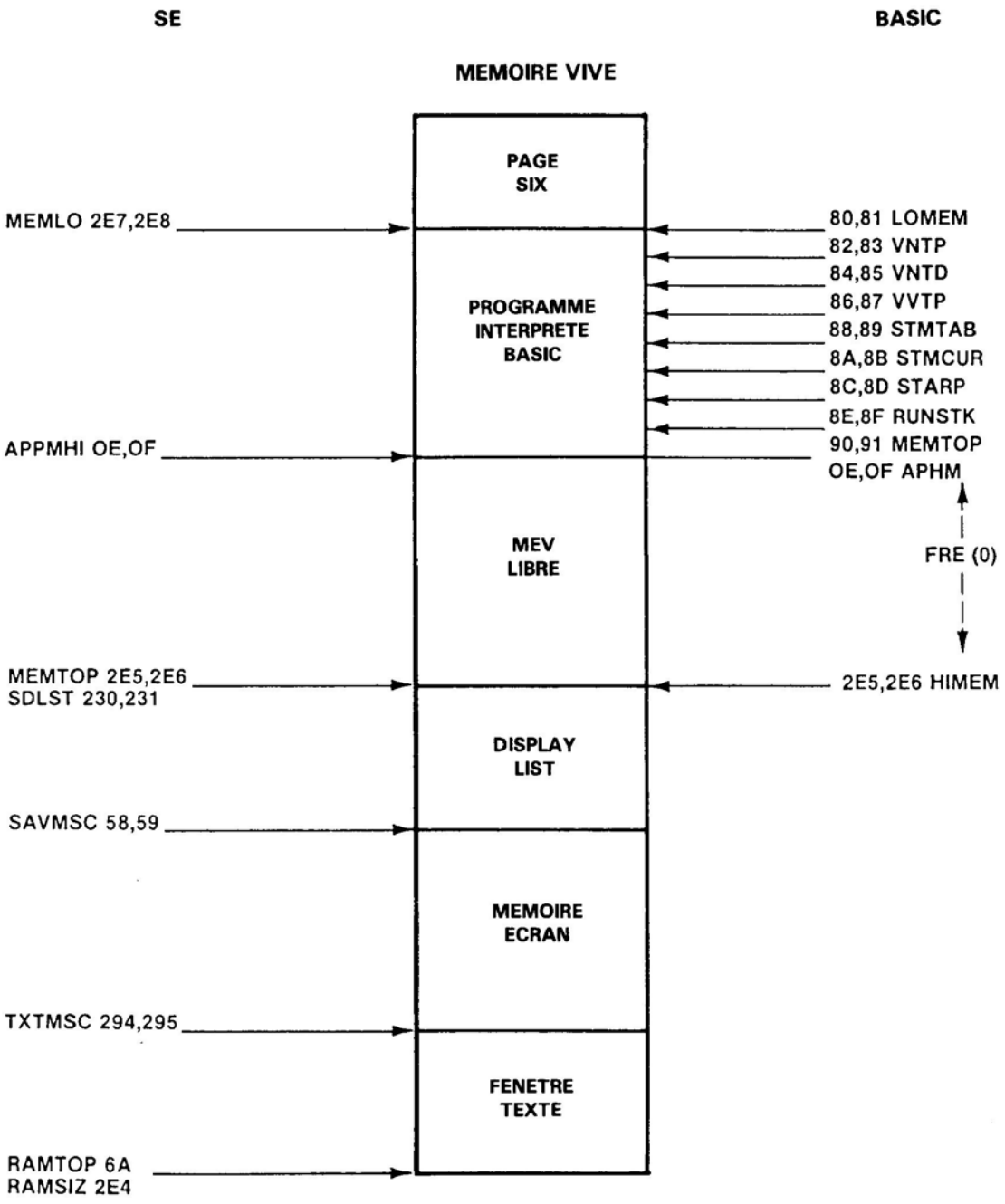



Figure 7-2 - Les pointeurs du BASIC et du Système d'Exploitation (le SED n'est pas présent)

# AMELIORATION DES PERFORMANCES D'UN PROGRAMME

Les performances d'un programme peuvent être améliorées de deux manières. Premièrement, le temps d'exécution peut être réduit (le programme tourne plus vite) ; deuxièmement, on peut chercher à réduire la taille de la mémoire utilisée. La liste ci-dessous peut vous guider dans ce travail. Les méthodes sont classées dans un ordre décroissant d'efficacité ; c'est-à-dire que les méthodes du début de la liste auront plus d'effet que les dernières.

### AMELIORATION DE LA VITESSE

1. Reprogrammez. Le BASIC n'étant pas un langage structuré, il est facile d'écrire des lignes inefficaces. Après plusieurs manipulations, arrangements et corrections, le programme peut devenir très lourd. Il est donc conseillé de reprogrammer, notamment les sous-programmes utilisés souvent. Le temps passé à cette tâche peut être long mais cela en vaut la peine.



ATARINSIDE

2. Vérifiez la logique des algorithmes. Vérifiez que la procédure suivie pour résoudre un problème spécifique est bien la plus efficace possible.

3. Placez les sous-programmes fréquemment appelés ainsi que les boucles FOR...NEXT au tout début du programme. Lorsqu'il recherche un numéro de ligne, le BASIC relit le programme depuis son début : un sous-programme placé dans les dernières lignes aura donc un plus long temps d'accès.

4. Si un sous-programme est fréquemment appelé depuis une boucle FOR...NEXT, placez-le directement dans cette boucle. La vitesse d'exécution augmente car vous supprimez les manipulations de la pile.

5. Lorsque plusieurs boucles sont imbriquées, arrangez-vous pour que la boucle utilisée le plus souvent soit la boucle centrale (ou intérieure). De cette manière, la pile sera manipulée moins de fois.

6. Simplifiez les calculs à l'intérieur des boucles. Par exemple, si vous multipliez une constante par un pointeur, transformez cette opération en une addition de constante.

7. Placez plusieurs instructions par ligne. Une boucle sur une ligne s'exécute plus rapidement que la même boucle sur deux ou plusieurs lignes.

8. Supprimez l'affichage à l'écran. En bloquant l'accès en DMA réalisé par l'ANTIC, vous gagnez 30% du temps d'exécution. Cela est intéressant tant que l'affichage n'est pas nécessaire. Pour cela, tapez POKE 559,0.

9. Utilisez un mode graphique plus rapide ou une Display List plus courte. Pour afficher un message court, utilisez par exemple le mode GRAPHICS 2 plutôt que le mode 0. Vous pouvez gagner jusqu'à 25% du temps d'exécution.

10. Utilisez des sous-programmes en Assembleur et appelez-les par la fonction USR.

**DIMINUTION DE L'ESPACE MEMOIRE OCCUPE**

1. Reprogrammez. Comme nous l'avons vu précédemment, le fait de restructurer le programme le rendra plus efficace. Cela économise également la mémoire.

2. Supprimez les instructions REM. Chaque caractère d'une chaîne de remarque occupe un octet.

3. Remplacez une constante utilisée plus de deux fois par une variable (pensez notamment aux structures : FOR I = ... TO ..., SOUND X,X,X,X, ...). Le BASIC utilise 7 octets (6 pour le nombre, 1 pour le code) pour une constante et seulement un pour une variable ; ainsi, 6 octets sont économisés à chaque remplacement.

4. Initialisez les variables par une instruction READ/DATA. En effet, les valeurs sont enregistrées comme des caractères ATASCII dans le programme, 1 octet par caractère et non pas 7 octets par constante.

5. Définissez des variables comme des combinaisons de variable. Ainsi, au lieu d'écrire Z1 = 1:Z2 = 2:Z3 = 3, écrivez Z1 = 1:Z2 = 2:Z3 = Z1 + Z2.

6. Placez les numéros de ligne fréquemment utilisés (dans les GOTO et les GOSUB) dans des variables. Si par exemple la ligne 100 est appelée 50 fois, près de 300 octets sont économisés en définissant Z100 = 100 puis en écrivant GOTO Z100 ou GOSUB Z100.

7. Minimisez le nombre de variables utilisées. Chaque nouvelle ligne dans la table des valeurs des variables utilise 8 octets plus quelques uns pour son nom. Utilisez plutôt des tableaux.

8. Nettoyez de temps en temps les tables des valeurs et des noms. En effet, un nom de variable et sa dernière valeur ne sont pas effacés des tables, même lorsque toutes les références à cette variable sont supprimées du programme. Pour cela, sauvegardez le programme par une instruction LIST, puis tapez NEW, et rechargez le programme par ENTER.

9. Choisissez des noms de variable aussi courts que possibles. Chaque nom est stocké dans la table des noms et chaque caractère occupe un octet dans cette table. Plus les noms sont courts, plus courte est la table.

10. Si un texte est répété plusieurs fois à l'écran, placez-le dans une variable chaîne de caractères.

11. Initialisez les chaînes de caractère en utilisant le signe = et en plaçant la donnée entre guillemets. N'utilisez pas une instruction READ couplée à la fonction CHR$.

12. Placez plusieurs instructions par ligne. Trois octets sont économisés chaque fois que deux lignes sont regroupées.

13. Lorsqu'un sous-programme n'est utilisé qu'une fois, insérez-le directement dans le programme sans utiliser GOSUB...RETURN.

14. Si vous manipulez des valeurs numériques entières inférieures à 256, transformez-les en caractères. Vous utilisez un octet au lieu de six.

15. Remplacez les instructions SETCOLOR par des POKE. Vous économiserez 8 octets.

16. Utilisez les caractères de contrôle du curseur plutôt que les instructions POSITION. Vous remplacez 15 octets par un seul.


ATARINSIDE

17. Supprimez des lignes sous le contrôle du programme lui-même lorsque certaines parties deviennent inutiles. Voyez le paragraphe suivant.

18. En modifiant la valeur de STARP, il est possible de sauvegarder et de rappeler les informations relatives aux tableaux et aux chaînes de caractères.

19. De petits programmes Assembleur peuvent être placés dans des chaînes de caractères. Ainsi : X$ = ''hhh▯LV▯'' : X = USR(ADR(X$),16).

20. Découpez un grand programme en plusieurs parties indépendantes et chaînez-les.

# TECHNIQUES DE PROGRAMMATION AVANCEES

Une bonne cannaissance du BASIC ATARI rend possible l'écriture de programmes intéressants.

**1er exemple : initialisation de chaînes** : Ce programme place la même valeur dans tous les octets d'une chaîne, quelle que soit sa longueur. Le BASIC copie le premier octet de la chaîne source dans le premier octet de la chaîne destination, puis le second, le troisième, et ainsi de suite. En mélangeant chaîne destination et chaîne source, on obtient une initialisation très efficace et très rapide.

**2e exemple : suppression de lignes** : En utilisant une possibilité du système d'exploitation, un programme peut se supprimer ou se modifier des lignes lors de son exécution. L'éditeur d'écran peut en effet accepter des données depuis l'écran, sans intervention de l'utilisateur. Il suffit de placer la ligne sur l'écran, d'arrêter le programme et de générer un RETURN. Cette ligne (ou ces lignes) est alors acceptée comme si vous veniez de la taper.

**3e exemple : Players-Missiles et chaînes de caractères** : Voici une manière rapide de déplacer un Player-Missile. Une chaîne dimensionnée représente le Player. Il suffit de modifier sa valeur d'offset associée dans la table des chaînes de caractères pour que cette chaîne soit stockée dans la mémoire réservée aux P-M. Le fait de modifier le contenu de cette chaîne revient à modifier le P-M à la vitesse de l'Assembleur.

```
10 REM STRING INITIALIZATION
20 DIM A$(1000)
30 A$(1)="A":A$(1000)="A"
40 A$(2)=A$
```

```
10 REM DELETE LINE EXAMPLE
20 GRAPHICS 0:POSITION 2,4
30 ? 70:? 80:? 90:? "CONT"
40 POSITION 2,0
50 POKE 842,13:STOP
60 POKE 842,12
70 REM THESE LINES
80 REM WILL BE
90 REM DELETED
```



ATARINSIDE

```
100 REM PLAYER/MISSILE EXAMPLE
110 DIM A$(512),B$(20)
120 X=X+1:READ A:IF A<>-1 THEN B$(X,X)=CHR$(A):GOTO 120
130 DATA 0,255,129,129,129,129,129,129,129,129,255,0,-1
2000 POKE 559,62:POKE 704,88
2020 I=PEEK(106)-16:POKE 54279,I
2030 POKE 53277,3:POKE 710,224
2040 VTAB=PEEK(134)+PEEK(135)*256
2050 ATAB=PEEK(140)+PEEK(141)*256
2060 OFFS=I*256+1024-ATAB
2070 HI=INT(OFFS/256):LO=OFFS-HI*256
2090 POKE VTAB+2,LO:POKE VTAB+3,HI
3000 Y=60:Z=100:V=1:H=1
4000 A$(Y,Y+11)=B$:POKE 53248,Z
4010 Y=Y+V:Z=Z+H
4020 IF Y>213 OR Y<33 THEN V=-V
4030 IF Z>206 OR Z<49 THEN H=-H
4420 GOTO 4000
```



ATARINSIDE

# ANNEXE A
# UTILISATION DE LA MEMOIRE

Le programmeur peut rencontrer de sérieuses difficultés lorsqu'il désire utiliser certaines parties de la mémoire car il interfère avec le système d'exploitation, la cartouche et le SED. Cela ne lui laisse que peu de possibilités. Toutefois, une bonne préparation du travail résout généralement les problèmes.

## PAGE ZERO

La partie de la mémoire la plus importante est sans nul doute la page zéro. Elle est absolument essentielle pour les pointeurs. Certains types d'adressages du 6502 ont besoin d'adresses en page zéro et ils permettent des manipulations très rapides de données. En conséquence, il faut que le programmeur connaisse le nombre d'octets dont il dispose dans cette page, ainsi que leurs adresses. Cet additif ne décrit pas l'utilisation de chaque octet de la page zéro. Il définit simplement les adresses utilisables.

La moitié inférieure de la page zéro (adresses $00 à $80) est réservée au système d'exploitation. Ces 128 octets sont nécessaires pour le bon déroulement des tâches exécutées par le SE. Toutefois, la plupart des programmes peuvent ne pas faire appel à toutes les possibilités du SE, ce qui libère certaines adresses. En particulier, les 43 octets allant de $50 à $7A sont utilisés par l'éditeur d'écran et son LGP. Si votre programme possède son propre logiciel de gestion d'écran, cette zone devient libre. Un raisonnement similaire peut s'appliquer à d'autres adresses tant que vous n'utilisez pas la fonction correspondante du SE.

Malheureusement, il existe un problème important. Le Département Logiciel d'ATARI améliore en permanence les performances du système d'exploitation et modifie les programmes le composant. Ainsi, il existe à l'heure actuelle quatre versions du SE. Elles fonctionnent presque de la même manière mais il est fort probable que ces 43 octets ne seront pas toujours utilisés de la même manière. Aussi, tout logiciel qui utiliserait des octets de la moitié inférieure de la page zéro risque de ne plus fonctionner correctement sur des versions plus récentes du SE. Si vous concevez un logiciel devant être commercialisé, n'utilisez jamais les 128 premiers octets de la page zéro.

Les 128 octets de la partie supérieure de la page zéro sont utilisés par la cartouche. Si aucune cartouche n'est en place, ils sont tous libres. Si une cartouche est en place, quelques octets restent encore disponibles. La cartouche BASIC laisse sept octets pour le programmeur, de $CB à $D1. Si le programmeur a besoin de davantage de place, la seule solution consiste à utiliser la zone allant de $D4 à $FF, zone réservée normalement aux calculs en virgule flottante. Cela sous-entend bien sûr que le programmeur n'utilise pas des sous-programmes mathématiques du SE. Si la cartouche BASIC est en place, le BASIC utilise ces routines pour la conversion des valeurs des constantes dans le format BCD. Si vous utilisez ces adresses dans un programme Assembleur de gestion des interruptions, il peut y avoir conflit si l'interruption se produit tandis que le Basic exécute une conversion ou un calcul.

Le programmeur travaillant en Basic est généralement peu intéressé par les performances de vitesse que l'on peut atteindre grâce à la page zéro. Si la vitesse est vraiment sa préoccupation première, il doit utiliser le langage Assembleur. Un code objet ne nécessite aucune cartouche et en conséquence, il peut donc utiliser les 128 octets de la moitié supérieure de la page zéro. Si toutefois il est nécessaire d'utiliser la cartouche Assembleur/Editeur, seuls 32 octets ($B0 à $CF) restent disponibles. Si vous utilisez seulement le Debug de cette cartouche, 30 octets de plus sont disponibles : $A4, $A5, $AD, $AE, $DB à $E5, $EA à $F1, $F5, $F6, $F9 à $FB, $FE, $FF. Si ces octets sont utilisés, il ne faudra pas revenir ensuite à l'Assembleur ou à l'Editeur. De plus, le mini-assembleur de Debug ne doit pas être utilisé.

## ADRESSAGE ABSOLU

Un autre problème rencontré par le programmeur réside dans l'utilisation du SED. Il est en effet intéressant d'écrire des programmes qui fonctionnent aussi bien en version cassette sur 16 k qu'en version disquette sur 48 k. Malheureusement, de tels systèmes ont très peu de mémoire en commun car le SED utilise une grande partie de la mémoire disponible sur les systèmes 16 k. Plusieurs solutions existent. L'une consiste à produire deux versions différentes du programme, une version disquette et une cassette. Les adresses d'origine seront donc différentes. Un utilisateur qui passera d'un système 16 k à un système 48 k devra donc continuer d'utiliser son magnétocassette.

Il existe une autre solution. La page 6 est commune à tous les systèmes. Les variables et les vecteurs doivent être placés dans cette page 6. Une table des matières des adresses des sous-programmes peut être calculée et placée à cet endroit. Le programme utilise ainsi ces pointeurs pour trouver l'adresse de départ de chaque sous-programme. Si cette technique est intéressante à utiliser pour des programmes de taille moyenne (1 à 2 k), elle est difficile à mettre en place pour de grands programmes Assembleur.



ATARINSIDE

# ANNEXE B
# CONCEPTION DE LOGICIELS

Les ordinateurs ATARI sont des ordinateurs domestiques. Ils ont été étudiés pour être très simple d'emploi. Les circuits sont conçus pour résister aux erreurs de manipulation. Le logiciel écrit pour cet ordinateur doit donc respecter les mêmes concepts. L'utilisateur moyen n'est pas familier des conventions et des traditions du monde informatique. Une fois qu'il a compris un programme, il l'utilisera bien la plupart du temps. Parfois, il manquera de soin et fera des erreurs. C'est la responsabilité du programmeur de tenter de protéger l'utilisateur contre ses propres maladresses.

Une bonne conception des logiciels dans le domaine du grand public est malheureusement encore rare. Les programmes les plus tordus sont souvent écrits par des programmeurs amateurs mais il arrive de trouver des logiciels, écrits par de grandes sociétés, qui ne respectent aucune structure.

Bien concevoir un logiciel est un art, pas une science. Cela demande certes un bon niveau technique, mais aussi de la réflexion et de l'intelligence. Il s'agit de notions purement subjectives et cet additif étant le travail d'un seul auteur, il ne constitue pas un guide absolu. Pour pouvoir approfondir ce thème, il aurait fallu écrire un ouvrage entier. Mais une présentation trop complète de tous les points de vue aurait pu décontenancer le lecteur. J'ai donc choisi la tâche plus facile de présenter seulement ma propre manière de voir, envisageant également les objections les plus sérieuses. Le résultat est suffisamment impartial pour satisfaire les lecteurs les plus académiques.

## L'ORDINATEUR CONSIDERE COMME UN ETRE SENSIBLE

Une manière instructive d'étudier le problème de la conception humaine consiste à considérer le programmeur comme un sorcier capable de communiquer avec un esprit que nous appellerons homo informaticus, habitant les entrailles de l'ordinateur. Cette créature a l'aspect d'un être humain mais possède des traits intellectuels, et notamment la faculté de traiter et d'organiser l'information. L'utilisateur du programme rentre en relation avec cet homo informaticus. Ces deux êtres pensent différemment. Les cheminements de la pensée humaine sont associatifs, intégrateurs et diffus tandis que les processus de pensée du programme sont directs, analytiques et spécifiques. Ces différences sont complémentaires et productives car l'homo informaticus réalise des opérations que l'être humain ne peut pas faire. Malheureusement, ces différences créent aussi une barrière entre l'utilisateur et l'homo informaticus. Ils ont beaucoup de choses à se dire car ils sont très différents. Mais, parce qu'ils sont très différents, ils communiquent mal. Le problème central d'une bonne programmation consiste à établir les meilleures communications possibles entre l'utilisateur et l'homo informaticus. De nombreux programmeurs dépensent de grands efforts dans l'amélioration de la puissance et des performances de leurs programmes. Cela produit simplement un être intelligent sans yeux pour voir ni bouche pour parler.

Les cerveaux des ordinateurs personnels ont atteint des puissances qui les rendent capable d'exécuter des programmes suffisamment intelligents pour répondre au besoin de l'utilisateur moyen. Le premier facteur de limitation n'est pas en fait la fréquence de l'horloge ou la taille de la mémoire. C'est plutôt la minceur du lien de communication entre l'utilisateur et l'ordinateur. Chacun peut exécuter rapidement certaines tâches ; seul le problème de communication diminue l'efficacité de l'ensemble.

## COMMUNICATION ENTRE L'HOMME ET LA MACHINE

Comment pouvons-nous élargir le pipe-line de la communication? Considérons le langage avec lequel ces deux êtres communiquent. Comme tout langage, un langage homme-machine est restreint par les moyens physiques d'expression disponibles par les interlocuteurs. En raison des différences physiques existant entre l'ordinateur et l'être humain, leur mode d'expression sont sensiblement différents. Cela nous conduit à utiliser un langage qui n'est pas bidirectionnel (comme le sont les langages humains). Nous utilisons une procédure à deux voies : une voie d'entrée et une voie de sortie. Commençons par examiner les composants physiques de l'interface homme-machine.

## VOIE DE SORTIE (DE L'ORDINATEUR VERS L'ÊTRE HUMAIN)

Il existe deux voies de sortie pour l'ordinateur. La première est l'écran TV ; la seconde est le haut-parleur. Heureusement, elles constituent des moyens souples d'emploi autorisant une large gamme d'expression. Les autres chapitres de ce livre décrivent les possibilités de l'ordinateur vues sur un plan technique, c'est-à-dire vues par l'homo informaticus. Dans cet annexe, il est plus utile de s'en tenir au point de vue de l'être humain. De ces deux accessoires (l'écran



ATARI INSIDE

TV et le haut-parleur), l'écran constitue l'élément le plus expressif et le plus puissant. L'œil humain capte les informations d'une manière plus détaillée que l'oreille. En terme spécifique, nous dirions que sa bande passante est plus large. L'œil est sensible à trois formes d'informations visuelles : les formes, la couleur et l'animation.

**LES FORMES**

Les formes constituent un moyen idéal pour présenter une information à l'être humain. L'utilisation la plus directe des dessins consiste à décrire des objets. Un dessin vaut mieux qu'un grand discours, car il est direct, évident et immédiat.

On utilise également formes et dessins pour des symboles. Certains concepts de la pensée humaine interdisent en effet une image directe. Il en est ainsi pour l'amour, l'infini, ou la direction. Ils doivent être convertis en symboles comme un cœur, un huit horizontal, ou une flèche. Parfois, vous pouvez créer le symbole adéquat pour l'utiliser dans vos programmes. La plupart des utilisateurs le comprendront rapidement. Les symboles constituent une manière compacte d'exprimer une idée mais ils ne doivent pas être utilisés à la place d'un dessin sauf lorsqu'il faut être très concis. Un symbole est une expression indirecte ; un dessin est une expression directe.

La troisième utilisation des dessins et des formes réside dans l'écriture du texte. Une lettre est un symbole ; nous juxtaposons des lettres pour former des mots. Le langage que nous produisons est extrêmement riche dans sa puissance d'expression «si vous ne pouvez le dire, vous ne le savez pas». Cette puissance présente malheureusement un défaut : l'aspect indirect de la description. Le mot qui exprime une idée n'a pas de relation sensorielle ou émotionnelle avec l'idée. L'être humain est forcé de réaliser une gymnastique mentale intensive pour interprêter le mot. Bien sûr, nous le faisons si souvent que nous sommes devenu très habile à traduire des suites de lettres en idées. Nous ne remarquons plus l'effort. Mais le point important est que cet accès indirect supprime la communication immédiate.

Il existe une école de pensée qui soutient que le texte est supérieur au graphique pour communiquer. Le cœur de cet argument est que le texte encourage l'utilisation par le lecteur de sa propre imagination, ce qui peut être très riche. Cet argument ne me satisfait pas car si le lecteur doit utiliser son imagination, il doit fournir une information complémentaire qui n'est pas intégrée dans la communication elle-même. Un exercice d'imagination utilisant des graphiques donnerait de meilleurs résultats. Un argument plus intéressant pour le texte est que l'accès indirect permet de stocker des quantités considérables d'informations dans un petit volume. Les contraintes de n'importe quelle communication réelle rendent la compacité du texte très intéressante. Mais à mon avis, cela ne rend pas le texte supérieur au graphique. Le texte est simplement plus économique. Les graphiques nécessitent plus de place, de temps, de mémoire ou d'argent, mais ils sont plus expressifs que le texte. En fait, le choix entre des graphiques et du texte est une affaire de goût et le goût de l'acheteur est en question. Comparez la popularité de la télévision par rapport à celle de la radio, celle des films par rapport aux livres.

**COULEUR**

La couleur est un autre véhicule pour l'information. Il est moins puissant que le dessin et joue normalement un rôle secondaire. Son emploi le plus fréquent consiste à différencier des formes qui, sinon, seraient confuses. La couleur joue ainsi un rôle important en fournissant une réponse à l'utilisateur. Une couleur bien choisie permet d'identifier facilement un dessin ambigu. Par exemple, un arbre représenté comme un caractère doit être dessiné dans une grille $8 \times 8$. La définition du dessin est trop faible pour que l'on puisse facilement reconnaître un arbre. Toutefois, si vous le dessinez en vert, l'image devient plus facile à reconnaître. La couleur est également très utile pour attirer l'attention ou signaler un événement important. Les couleurs chaudes attirent l'attention. Enfin, l'esthétique y gagne lorsqu'on utilise des couleurs.

**ANIMATION**

J'utilise ici le terme d'animation pour désigner n'importe quel changement visuel. L'animation comprend le changement de couleur, le changement de dessin, le déplacement d'objets ou le défilement de l'image. Le but de l'animation consiste à mettre en valeur un processus dynamique. Elle est indispensable pour présenter avec succès des événements très actifs. Par exemple, l'animation est à la base de Star Raiders. Pouvez-vous imaginer ce que serait ce jeu sans animation? Voyez-vous ce que cela donnerait en texte pur? Mais l'animation dépasse largement le cadre du simple jeu. Elle constitue indéniablement un avantage majeur de l'ordinateur sur le papier. De plus, l'œil humain est justement organisé pour réagir rapidement à des changements intervenants dans son champ visuel. L'animation peut attirer l'attention de l'œil et augmenter la participation de l'utilisateur dans le déroulement du programme.

**SONS**

Les images, pour avoir de l'effet, doivent être regardées. Les sons, par contre, peuvent attirer l'attention de l'utilisateur même lorsqu'elle n'est pas fixée sur la machine. Les sons ont une grande valeur comme annonciateur ou avertisseur. Des actions correctes effectuées au clavier produisent un son agréable tandis qu'une erreur sera mise en évidence par un bruit agressif. Un danger sera signalé par un «pouet».

Les sons présentent une seconde utilité : produire des effets sonores réalistes. Ils augmentent l'impact du programme même lorsque l'attention de l'utilisateur est déjà occupée par ailleurs.

Le son est mal approprié pour délivrer une information précise ; la plupart des gens n'ont pas l'acuité auditive nécessaire pour distinguer entre deux fines nuances. Par contre, les gens associent facilement certaines idées à certains effets sonores. Par exemple, une suite de notes descendantes n'implique généralement pas qu'on a gagné lors d'un jeu. Un bruit d'explosion implique une idée de destruction. Une fanfare annonce l'arrivée d'un phénomène important.


ATARINSIDE

# VOIE D'ENTREE (DE L'UTILISATEUR VERS L'ORDINATEUR)

Les ordinateurs ATARI utilisent fréquemment trois organes d'entrée : le clavier, la commande à levier et les commandes à molette.

## LE CLAVIER

Le clavier constitue le périphérique d'entrée le plus puissant pour le programmeur. Il possède 50 touches immédiatement accessibles. L'utilisation des touches CTRL et SHIFT permettent de plus que doubler le nombre de codes distincts que l'utilisateur peut créer. La touche CAPS LWR et l'inversion vidéo augmentent encore ces possibilités. 125 commandes sont accessibles en une seule action. Plus de 15 000 commandes le sont par deux actions. Le clavier est donc un grand moyen d'expression. Pour cette raison, il constitue le choix préféré de la majorité des programmeurs.

Alors que les avantages du clavier sont indéniables, ses faiblesses sont rarement reconnues. La première est que trop peu de personnes savent bien l'utiliser. Les programmeurs utilisent les claviers à longueur de journée ; en conséquence, ils deviennent de rapides dactylographes. L'utilisateur moyen n'est pas si habile avec un clavier. Il peut facilement se tromper de touche. L'existence même de toutes ces touches et le fait que l'on doive appuyer sur la bonne sans risque d'erreur constitue une barrière intimidante pour la plupart des utilisateurs.

Un second point faible pour le clavier réside dans sa neutralité. Il est très difficile de rattacher directement une expression à un clavier. Un clavier ne présente pas de signification émotionnelle ou sensorielle. Or, tout le travail avec un clavier est symbolique ; les touches étant gravées avec des symboles dont le sens varie selon les circonstances. Ainsi, la touche A peut représenter la première lettre de l'alphabet, une préposition, une première proposition dans une énumération, ou un choix dans un menu. L'utilisateur a une difficulté inconsciente de réaliser ce chaînage entre le clavier et la signification des symboles, d'autant plus que ce chaînage peut changer plusieurs fois en cours de programme.

Une autre propriété du clavier que le concepteur doit garder à l'esprit est sa nature numérique. C'est-à-dire que le clavier est échantillonné par l'ordinateur à certains moments et une seule touche peut être enfoncée à la fois. Ce n'est pas un avantage dans les applications temps réel. Or, l'être humain est une créature fonctionnant en temps réel. Il faut donc que le programmeur comprenne que l'utilisation du clavier ralentira le dialogue avec l'utilisateur.

## COMMANDES A MOLETTE

Ces commandes sont les seules entrées réellement analogiques que l'ordinateur accepte. Comme telles, elles souffrent du problème commun à tous les phénomènes analogiques : l'utilisateur doit précisément positionner la molette pour obtenir un résultat répétitif. Leur résolution angulaire est mauvaise et des effets thermiques provoquent un glissement des résultats sans que l'on ait manœuvré le bouton.

La commande à molette ne représentant rien de concret, il est essentiel d'obtenir sur l'écran un écho de la position du bouton : position d'une croix, modification d'une valeur, etc. L'être humain peut jouer le rôle d'une contre réaction en asservissant la position de la molette au résultat apparaissant sur l'écran.

Les molettes sont particulièrement utiles pour modifier des valeurs unidimensionnelles. L'utilisateur comprend immédiatement que la molette permet de passer en revue différentes valeurs et lorsque la bonne est atteinte, il suffit d'appuyer sur le bouton pour valider la sélection. Ensuite, l'utilisateur est sensible au fait qu'en un mouvement rapide de deux doigts, il couvre une large gamme de valeurs.

On peut réaliser beaucoup de choses avec une molette. On peut par exemple choisir une option dans un menu. Selon ce principe, on peut faire défiler les lettres de l'alphabet et choisir la lettre désirée en appuyant sur le bouton. Cette méthode ne permet pas d'atteindre des vitesses importantes mais elle est très utile pour des enfants et l'idée pourrait être appliquée à d'autres problèmes.

## COMMANDES A LEVIER

Les commandes à levier constituent le plus simple moyen d'entrer des données dans l'ordinateur. Elles sont résistantes et peuvent être employées dans des conditions sévères. Elles contiennent seulement cinq interrupteurs. Pour cette raison, les commandes à levier sont sous-estimées. Mais lorsqu'elles sont couplées à un curseur, une commande à levier peut adresser n'importe quel point de l'écran et une sélection sera validée par le bouton rouge. Avec un dessin d'écran bien défini, la commande à levier étend les possiblités de commande. Je l'ai utilisé pour commander un réacteur nucléaire («Centrale nucléaire») et un jeu de guerre («Front de l'Est 1941»).

La clé du succès de l'utilisation de cette commande à levier réside plus dans la durée de fermeture d'un contact que dans le choix du contact fermé. En maintenant le contact pendant un temps plus ou moins long, l'utilisateur déplace un curseur plus ou moins loin. Cela nécessite une vitesse de déplacement du curseur constante. Mais si le curseur se déplace trop rapidement, il deviendra difficile de la positionner correctement là où l'utilisateur le désire. Si le curseur se déplace trop doucement, l'utilisateur s'impatientera lorsqu'il faudra traverser tout l'écran. La solution consiste à concevoir un curseur à deux vitesses : lente au début, plus rapide lorsque le contact a été maintenu pendant un certain temps, par exemple trois secondes.

Le grand intérêt de la commande à levier, c'est sa bonne prise en main. Vous exprimez sans intermédiaire votre sentiment et le curseur se déplace immédiatement en conséquence. Pour piquer, vous poussez le levier vers l'avant comme le manche à balai d'un avion. Pour vous déplacer vers la gauche, vous appuyez à gauche. Il est d'ailleurs étonnant de voir comme le mouvement du corps accompagne le mouvement de la main dans les jeux d'action, même pour les joueurs novices. Ce qui prouve que le joueur adopte immédiatement cette commande et qu'il n'effectue aucun transcodage



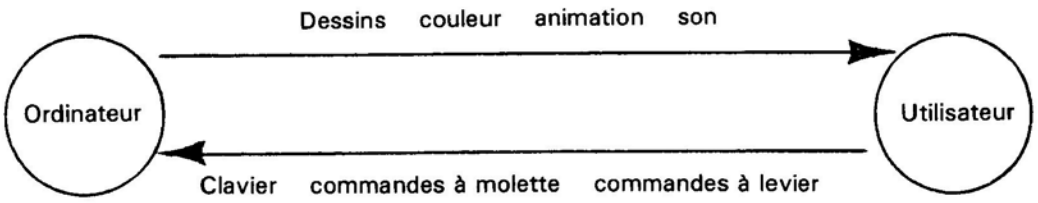
ATARINSIDE

entre le mouvement qu'il veut donner au curseur et l'action que ses doigts doivent exécuter. Il est impossible d'obtenir le même résultat avec un clavier.

Les commandes à levier ont leurs limitations. Bien qu'il soit possible de déplacer le levier en diagonale, ses directions ne sont pas suffisamment distinctes pour effectuer facilement un déplacement en diagonale. Aussi vaut-il mieux éviter ce genre de déplacement.

## RESUME DES ELEMENTS DE COMMUNICATION

Nous venons de voir un certain nombre d'éléments permettant l'établissement d'une interaction entre l'ordinateur et l'utilisateur. Il s'agit de :




## CONSTRUIRE UN LANGAGE

Comment devons-nous assembler ces éléments dans un langage effectif? Nous devons d'abord déterminer les caractéristiques principales que nous attendons d'un bon langage homme-machine, c'est-à-dire d'un bon programme :

- son aspect complet
- sa facilité d'emploi
- sa prévision des égarements

### L'ASPECT COMPLET

Le langage homme-machine doit complètement exprimer toutes les idées devant être communiquées entre les deux parties. Par contre, il ne doit pas exprimer les idées internes à chaque processus de pensée. Par exemple, le dialogue dans Star Raiders doit exprimer tous les concepts relatifs à la commande du vaisseau et à la situation du combat. Il ne doit pas exprimer l'anxiété du joueur ou les intentions de mouvement des Zylons ; ces concepts, bien que se rapportant au but du jeu, ne doivent pas être échangés entre l'utilisateur et l'ordinateur.

La nécessité d'échanges complets entre l'homme et la machine pour une tâche donnée semble être reconnue intuitivement par le programmeur. Des problèmes apparaissent toutefois lorsque le programmeur doit ajouter des fonctions à un programme, fonctions qui ne peuvent être supportées par le langage que le programmeur a créé, même si leurs mises en place ne posent pas de difficultés majeures. Par exemple, vous utilisez une commande à levier pour déplacer le curseur dans les quatre directions. Et vous désirez ajouter une fonction faisant apparaître une cible sur l'écran et vous avez auparavant interdit l'usage du clavier. Techniquement, ce n'est pas difficile de faire apparaître une cible. Le problème réside dans l'implantation de la nouvelle commande, donc dans l'extension du langage de dialogue.

### LA FACILITE D'EMPLOI

Tout nouveau langage est difficile à étudier. Et aucun utilisateur n'a beaucoup de temps pour apprendre un langage surchargé. Le dialogue s'établissant entre l'homme et la machine doit être simple et efficace, direct. Il doit être le plus proche possible de ce que l'utilisateur connaît déjà. Enfin, il doit être évident. Par exemple, CTRL X est une commande bien obscure. Que veut-elle dire? Peut être quelque chose doit-il être détruit, X sous-entendant l'élimination ou la négation? Peut être quelque chose doit-il être examiné, purgé ou abandonné? Si aucune de ces possibilités n'est la bonne, alors la commande est trop indirecte. Elle est difficile à retenir et le mode d'emploi du programme doit rester à proximité. C'est inacceptable. Les claviers sont réputés pour créer ce genre de problèmes.

### LA PREVISION DES EGAREMENTS

Voici le point provoquant le plus de problèmes. Le concept est mieux expliqué avec une analogie. Supposons que l'utilisateur se situe à un point A d'un programme et veuille se rendre à un point B. Un programme mal conçu se transforme en une corde raide tendue entre les points A et B. L'utilisateur qui sait exactement ce qu'il a à faire et comment le faire, réussira sans problèmes. Mais le plus souvent, il se trompera et s'égarera. Certains auteurs de logiciels cherchent à aider l'utilisateur en lui disant ce qu'il faut faire et ne pas faire dans le mode d'emploi ou à l'écran. J'ai vu plusieurs programmes qui plaçaient des signes sous l'erreur écrite à l'écran, si bien qu'on pouvait au moins voir où résidait la faute. D'autres programmes établissent une meilleure protection en filtrant les caractères tapés au clavier et en n'autorisant que les légaux. Cela est beaucoup plus agréable mais la programmation devient nettement plus complexe car il faut qu'en aucun cas, les protections ne se transforment en dragon. Enfin, certains programmes utilisent des messages de préventions cybillins. Ils deviennent ainsi totalement inutiles. Le programme idéal se compare à une voie de chemin de fer. Il existe un seul chemin mais ce chemin mène obligatoirement au succès.


ATARINSIDE

Afin de prévenir et d'éviter les égarements de l'utilisateur, il est nécessaire de réduire le nombre d'options à chaque menu et de supprimer les possibilités marginales et inutiles. Une bonne étude ne doit pas produire une accumulation d'accessoires et de gadgets sur une architecture de base ; elle exige du programmeur une grande clairvoyance dans les choix à effectuer.

Cette thèse se heurte très souvent aux programmeurs eux-mêmes. Le programmeur désire toujours une liberté totale de commandement sur l'ordinateur. Ainsi, lorsqu'un programme est quelque peu restrictif, le programmeur se rebiffe : pourquoi se priver des possibilités de la puissance de cet outil merveilleux qu'est l'ordinateur?

La réponse réside dans la différence entre l'utilisateur et le programmeur. Le programmeur consacre sa vie à l'ordinateur ; l'utilisateur est accidentellement en contact avec la machine. Le programmeur passe tellement de temps sur la machine qu'il est intéressant pour lui de l'étudier à fond dans ses moindres détails. L'utilisateur, par contre, n'a pas de temps à perdre avec la machine. Il désire aller du point A au point B aussi rapidement que possible. Les subtilités qui plaisent au programmeur ne l'intéressent pas. Si vous êtes un programmeur, vous pouvez ne pas être passionné par les exigences de l'utilisateur, mais si vous souhaitez préserver votre gagne-pain, vous devez plutôt en tenir compte.

La protection de l'utilisateur s'obtient en créant des entrées/sorties n'admettant pas de valeurs illégales. Ce but est très difficile à atteindre avec un clavier car il y a souvent une combinaison de touches que vous n'avez pas prévue. Une commande à levier est meilleure dans ce sens car vous n'avez pas beaucoup de possibilités avec elle. Il est ainsi plus facile d'exclure les mauvaises commandes. L'idéal est atteint quand tous les choix sont exprimés avec une commande à levier : l'utilisateur ne peut pas faire d'erreur car il n'en a même plus la possibilité.

Mais un programme bien conçu va plus loin que cela. Filtrer les entrées constitue une amélioration mais elle est insuffisante à elle-seule : le résultat ne sera pas toujours fonctionnel. Par exemple, supposons que vous interdisiez l'emploi de la touche M sur le clavier, simplement parce qu'elle ne sert à rien. L'utilisateur voit cette touche au clavier, et il peut s'imaginer qu'en l'enfonçant, il provoquera une réponse de l'ordinateur. Mais comme vous avez inhibé cette touche, rien ne se produit. L'utilisateur perd alors plus de temps à essayer de comprendre pourquoi il ne se passe rien plutôt qu'à corriger son erreur. Cette perte de temps a été prévue par le programmeur qui imagine, et il a raison, l'utilisateur appuyant sur M par inadvertance. Curieusement, le programmeur croit bien faire en éliminant la cause (inhibition de M) mais les conséquences empirent (l'utilisateur ne comprend pas pourquoi sa machine ne répond plus). Dans ce cas présent, il suffirait simplement qu'un message adéquat apparaisse à l'écran («ce choix n'est pas valide») pour que le problème soit résolu.

On retire toujours de nombreux avantages d'un programme bien conçu. Les lignes de programme sont moins nombreuses, l'exécution est plus rapide, l'utilisateur apprend plus vite à se servir de sa machine et il a moins de problèmes avec elle.

Mais il est vrai que cela demande davantage d'efforts au programmeur. Le langage servant au dialogue entre l'utilisateur et la machine doit être soigneusement analysé afin de réduire le vocabulaire nécessaire au strict minimum. La simplicité des écrans augmente, et même s'il faut supprimer quelques astuces ou quelques gadgets, l'utilisateur y gagnera.

# CONCLUSIONS

L'étude des dialogues homme-machine constitue la phase la plus délicate dans l'élaboration d'un logiciel. Le concepteur doit soigneusement connaître les possibilités de sa machine ainsi que les besoins de l'utilisateur. Il doit précisément définir les informations pouvant s'échanger entre ces deux partenaires. Une clarté maximale doit être recherchée dans la présentation des informations et elle doit être préférée à la quantité.

# QUELQUES PROBLEMES CLASSIQUES

Après avoir développé quelques concepts théoriques, nous allons nous tourner maintenant vers une étude de problèmes pratiques. Cette liste n'est pas exhaustive mais elle couvre la plupart des cas standards.

### TEMPS D'ATTENTE

Beaucoup de programmes calculent longuement. En vérité, la plupart des programmes ont besoin, à un moment donné ou à un autre de plusieurs secondes pour réaliser des calculs. Malheureusement, à cet instant, le dialogue entre l'ordinateur et l'utilisateur cesse. L'utilisateur est abandonné à son triste sort devant un écran désespérément inactif : l'ordinateur ne répond plus. Toute action sur les touches reste sans effet. Ceci dénote une grave lacune dans la conception du programme. Abandonner l'utilisateur est impardonnable.

### PROCESSUS SEPARES

Le meilleur moyen de résoudre ce problème consiste à séparer le processus de saisie du processus de calcul. L'utilisateur devrait être capable de taper des données pendant que l'ordinateur calcule. Ceci est techniquement possible ; en utilisant les interruptions de Blanking vertical, le programmeur peut faire exécuter deux programmes indépendants simultanément (multi-tâches). Cette technique est utilisée dans «Front de l'Est 1941». Malheureusement, beaucoup de phénomènes se déroulent de manière séquentielle dans la nature. En conséquence, le programme principal a besoin des données de l'utilisateur à des instants précis. Ce qui peut rendre très délicate la recherche d'indépendance entre les deux programmes. Parfois, il est possible de réaliser des calculs anticipés qui seront peaufinés lorsque l'utilisateur aura entré sa donnée.



ATARINSIDE

### AUGMENTATION DE LA VITESSE D'EXECUTION

Une autre solution consiste à réduire le temps d'attente, donc à améliorer la vitesse d'exécution. Les parties critiques peuvent être reprogrammées afin d'être optimisées. Il vaut mieux placer une boucle se répétant de nombreuses fois à l'intérieur d'une boucle se répétant peu de fois que l'inverse. Le programmeur peut remplacer des sous-programmes Basic par des programmes Assembleur. Les résultats seront sensibles pour des graphiques ou des animations, et, mais dans une moindre mesure, pour les calculs. En supprimant provisoirement les interruptions de Blanking vertical (l'écran devient d'une couleur uniforme), vous récupérerez un nombre non négligeable de cycles machines pour le programme principal. Les modes graphiques les plus économes en mémoire occuperont l'ANTIC le moins longtemps. Dans le même ordre d'idées, raccourcir la Display List diminue le nombre de cycles DMA, au bénéfice du 6502.

### AVERTIR L'UTILISATEUR

La troisième solution consiste à occuper l'utilisateur durant les calculs. Un compte à rebours en est un exemple. Lorsque la valeur affichée atteint 0, le dialogue avec l'utilisateur reprend. Un message complémentaire peut préparer l'utilisateur ; par exemple «je dois calculer pendant 3 minutes». Tout cela ne constitue que des pis aller car il serait préférable de ne jamais avoir à attendre.

## S'ACCOMODER DES MAUVAISES DONNEES DE L'UTILISATEUR

Les logiciels bien conçus éliminent le problème des erreurs de données en supprimant la possibilité même qu'elles existent. Comme je le mentionnais précédemment, il est possible d'atteindre ce résultat en utilisant par exemple une commande à levier. Mais bien sûr certaines applications exigent un clavier (traitement de texte par exemple). De plus, que se passe-t-il lorsqu'une erreur a été faite? Il est impératif que toutes les erreurs possibles aient été prévues afin que l'utilisateur ne trouve jamais de faille : il ne manquerait pas d'y tomber et de se perdre.

### SIGNALEZ L'ERREUR ET SUGGEREZ UNE SOLUTION

L'approche la plus souhaitable dans une situation d'erreur consiste à indiquer à l'utilisateur l'erreur qu'il a faite, dans un langage très clair. Il faut ensuite lui suggérer une réponse correcte. Trois éléments doivent constituer la réponse de l'utilisateur. Premièrement, la donnée de l'utilisateur doit réapparaître à l'écran afin de situer la zone de l'erreur. Deuxièmement, le ou les caractères plus particulièrement illégaux doivent être clairement mis en évidence afin que l'utilisateur comprenne ce qui ne va pas. Troisièmement, un exemple de réponse correcte doit s'afficher afin que l'utilisateur, après plusieurs tentatives infructueuses, n'ait pas l'impression d'être devant un mur de briques. Par exemple, nous pourrions imaginer la réponse suivante de l'ordinateur : «vous avez tapé CTRL A qui est une demande d'impression. Mais vous n'avez pas fini votre page. Je vous suggère de la terminer en tapant CTRL X.».

A l'évidence, cette méthode est très exigeante en taille mémoire et en temps de programmation. Mais c'est l'idéal et on peut même envisager par ce biais l'amorce d'un véritable dialogue en langage naturel entre l'homme et la machine. Heureusement, il existe des méthodes moins dispendieuses quoique moins satisfaisantes.

### MASQUAGE DES MAUVAISES TOUCHES

Une solution couramment employée consiste à masquer les mauvaises frappes au clavier. Si l'utilisateur se trompe de touche, rien ne se produit. Le programme attend ce qu'il doit attendre. Si cette solution protège bien le programme et l'intégrité des fichiers contre toute maladresse, elle n'est malheureusement pas explicite pour l'utilisateur. Si celui-ci a effectivement enfoncé une touche pensant qu'une certaine action en résulterait, le fait de ne pas avoir pris en compte cette touche ne corrige pas l'erreur de l'utilisateur. Il peut seulement penser que quelque chose s'est détraqué dans son ordinateur.

Une variante de ce schéma consiste à ajouter un son désagréable lorsqu'une mauvaise touche a été enfoncée. des messages généraux d'avertissement sont également envoyés vers l'écran. mais il ne faut pas en abuser afin de pouvoir établir une hiérarchie dans les niveaux d'erreur. Enfin, ne programmez pas des messages comme ceux que l'on peut voir dans certains programmes d'amateurs : vous devez rester poli.

## MESSAGES D'ERREUR

Le message d'erreur peut se résumer à quelques mots indiquant simplement que l'utilisateur s'est trompé ; ce qui reste général mais donc imprécis. Dans d'autres cas, le message d'erreur est codé, ce qui ne renseigne pas rapidement l'utilisateur. Les erreurs de langage Basic en sont un exemple ; en fait, cette pratique se justifie lorsque de nombreuses erreurs doivent être différenciées et que le programme doit tenir dans une petite zone de mémoire.

Dans la plupart des cas, le programmeur choisit de sacrifier la simplicité d'exploitation à la puissance technique. Nous avons vu précédemment qu'il ne fallait pas aller trop loin dans cette voie.

### PROTECTION VS PUISSANCE

Lorsque l'utilisateur s'est habitué à son programme, il peut être gêné par les protections elles-mêmes, comme par exemple la traditionnelle question «En êtes-vous sûr?». Une bonne solution consisterait à pouvoir modifier le taux de protec-


ATARINSIDE

tion d'un programme. Par exemple, à la mise sous tension, toutes les protections sont en place. Mais l'utilisateur chevronné sait qu'en tapant une suite particulière de touches, il pourra inhiber ce genre de questions et pourra ainsi gagner du temps. Cette suite de touches ne sera indiquée que dans la documentation, ce qui obligera justement l'utilisateur à un certain approfondissement.

# TECHNIQUE DES MENUS ET DES SELECTIONS

Les menus sont ces écrans représentant des listes d'options parmi lesquelles vous devez généralement en choisir une. Certains programmes n'utilisent pas de menu mais attendent qu'une commande leur soit transmise par le clavier ; ce qui sous-entend la mémorisation d'un catalogue de mots, ce qui n'est pas à la portée du débutant. Par contre, le professionnel aime bien cette interaction directe et c'est pourquoi on trouve ce principe sur les logiciels destinés au programmeur.

### TAILLE DU MENU

Combien de choix peut-on placer sur un menu? La limite supérieure est déterminée par la taille de l'écran. Mais en Basic, mode texte, il serait possible de présenter jusqu'à 48 choix. Cela fait beaucoup trop. Je pense plutôt qu'il faut se limiter à 7 choix. Vous pouvez ainsi aérer la présentation, mettre un titre en relief, etc.

### MENUS MULTIPLES

Un programme important nécessite bien souvent plusieurs menus s'enchaînant les uns les autres. Il faut que ces menus soient organisés d'une manière claire et structurée. Il est très facile de perdre quelqu'un dans un labyrinthe de menus. Une manière consiste à établir un menu principal, puis des menus secondaires, équipés chacun d'un choix particulier permettant un retour immédiat au menu principal. Une autre solution consiste à disposer les menus dans une structure hiérachisée. Il est très utile dans ce cas d'assister le programmeur par des effets sonores et visuels (couleurs). Par exemple, la fréquence de la note peut être associée à la position dans la hiérarchie.

### METHODES DE SELECTION

Une fois que l'utilisateur a fait son choix, comment peut-il le faire connaître à l'ordinateur? La manière la plus simple est de commencer chaque ligne de menu par une lettre ou un chiffre ; l'utilisateur fait son choix en appuyant sur la touche correspondante du clavier. Avec cette solution, il est très facile d'éliminer les mauvaises réponses. Mais il existe d'autres méthodes. Par exemple, une flèche se déplace de ligne en ligne chaque fois que l'on appuie sur SELECT. La flèche peut être remplacée par la ligne ré-écrite en inversion vidéo. Arrivée en bas de l'écran, la flèche se repositionne automatiquement au début du menu. Lorsque la flèche est placée sur la ligne voulue, il suffit d'appuyer sur START. Les commandes à levier et à molette se révèlent très pratiques lorsqu'elles sont utilisées dans ce but. Dans ce cas, on déplace un curseur avec le levier et l'on convient de son choix en appuyant sur le bouton rouge. On peut envisager également des menus se présentant comme des pages de texte en défilement fin, la commande à levier déplaçant la fenêtre dans le sens voulu.

# MANUEL OU TEXTE A L'ECRAN?

Les menus, les messages d'erreur, les messages ordinaires utilisent beaucoup de place en mémoire. On peut être tenté de reporter ces indications dans un manuel de référence mais agir ainsi entraîne une réduction notable de la qualité du logiciel, sur le plan de la facilité de mise en œuvre. Avec des programmes stockés sur disquette, le problème est moins crucial car il est possible de stocker une bonne partie du mode d'emploi sur la disquette. Idéalement, vu du côté de l'utilisateur, le manuel devrait être inclus dans le programme ou au moins sur la disquette. Malheureusement, des considérations techniques et économiques s'opposent à cela. En fait, chaque technologie doit être utilisée à bon escient. Bien que l'ordinateur puisse présenter des pages statiques de texte, il convient mieux à des animations ou à des calculs intenses. Par contre, du papier et de l'encre présentent des informations statiques plus clairement qu'un ordinateur et à meilleur marché. Aussi, je préfère reporter les informations statiques dans un manuel, quitte à ce que le programme renvoie l'utilisateur au manuel. Mais j'inclus toujours dans le programme les informations importantes.

# MESURES DU SUCCES

Comment un programmeur peut-il déterminer la qualité de conception de son logiciel? Le premier élément de réponse réside dans la longueur du manuel. Plus il est long, plus votre programme risque d'être farfelu. Un programme bien conçu nécessite peu d'explications. Mais n'en concluez pas qu'il faille bacler la documentation. Le manuel peut être long pour décrire les programmes en détail, mais il ne faut pas que ces détails soient indispensables au fonctionnement du programme.

Il est intéressant de mesurer le temps d'adaptation pour un nouvel utilisateur. Si, par exemple, dans le cas d'un traitement de texte, il faut passer plusieurs heures avant de sortir une lettre, le programme n'est à l'évidence, pas bien conçu.

Un troisième élément consiste à évaluer la réflexion nécessaire à l'utilisateur pour qu'il exploite le programme. Plus le programme est bien conçu, moins l'utilisateur doit réfléchir. Cela ne veut pas dire que l'utilisateur ne doive jamais penser, mais il doit penser à ce qu'il veut faire plutôt qu'au mécanisme du programme et à ce qu'il doit faire. L'être humain et l'homo informaticus se rejoignent et travaillent dans une symbiose mentale totale.


ATARINSIDE

# ANNEXE C
# LA CASSETTE ATARI

Ce chapitre sur le magnétocassette ATARI se divise en deux parties :

• **Comment fonctionne le magnétocassette** : informations sur le matériel et sur le logiciel.

• **Applications de la cassette** : comment mélanger les informations audio et numériques pour produire un programme complet.

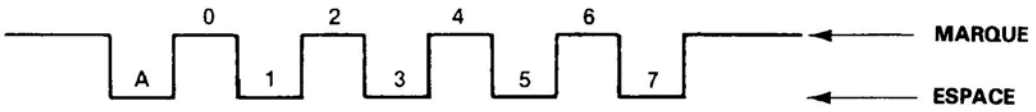
# COMMENT FONCTIONNE LE MAGNETOCASSETTE

## STRUCTURE D'UN ENEGISTREMENT

### DEFINITION D'UN OCTET

Le SE écrit des fichiers sous forme de blocs de longueur fixe à la vitesse de 600 bauds (bits par seconde). Une transmission série asynchrone est utilisée dans les opérations de lecture et d'écriture entre les ordinateurs ATARI et le magnétocassette. Le POKEY reconnaît chaque octet de donnée dans cet ordre : un bit START (espace), 8 bits de donnée (0 = espace, 1 = marque), puis un bit STOP (marque). Un octet est transmis avec le bit de poids le plus faible (D0) en premier.

La fréquence utilisée pour représenter le signal marque est 5327 Hz. Pour un espace, cette fréquence est 3995 Hz. Voici un exemple de transmission :



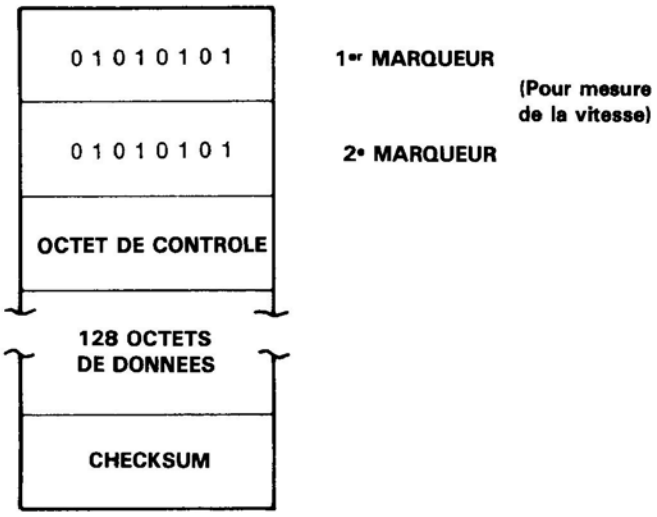

A = bit START (espace)
0-7 = bits de données
B = bit STOP (marque)



ATARINSIDE

## DEFINITION D'UN ENREGISTREMENT

Un enregistrement contient 132 octets répartis de la manière suivante : deux caractères marqueur pour la mesure de la vitesse, un octet de contrôle, 128 octets de données, et l'octet de checksum.




## MARQUEUR

Chaque marqueur est constitué par le caractère $55. En comprenant les bits START et STOP, chaque marqueur est long de 10 bits. Idéalement, il ne devrait pas y avoir de bande vierge entre les marqueurs et les données qui le suivent.

## MESURE DE LA VITESSE

Ces deux caractères marqueur servent à ajuster la vitesse de transmission.

La fréquence de transmission en réception (vue de l'ordinateur) est fixée à une valeur nominale de 600 bauds. Mais le SIO peut modifier cette valeur pour tenir compte des variations de vitesse du moteur, de la bande, etc. Lorsque la vraie vitesse de transmission a été calculée, les circuits de l'ordinateur sont ajustés en conséquence. Des vitesses de réception comprises entre 318 et 1407 Bauds peuvent ainsi être théoriquement générées.

Le SE vérifie la vitesse de la bande de la manière suivante : le logiciel scrute en permanence l'état du bit de réception dans le POKEY. Il attend un bit à 0 (START) qui indique le début d'un caractère. Quand il en trouve un, le SE stocke dans sa mémoire le contenu du registre VCOUNT de l'ANTIC (compteur de lignes sur l'écran). En continuant d'examiner directement l'état du bit reçu, le SE compte 20 bits (soit la fin des deux marqueurs), puis en utilisant VCOUNT et le compteur d'image, le SE détermine le temps écoulé. Il en déduit la vitesse de transmission. Cette opération est réalisée pour chaque enregistrement.

## OCTET DE CONTROLE

L'octet de contrôle représente l'une de ces trois valeurs :

- $FC indique que l'enregistrement est complet (128 octets de données).

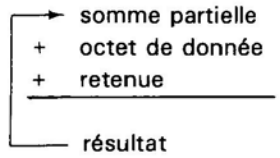
- $FA indique que l'enregistrement est partiel ; moins de 128 octets sont réellement utilisables. Cela ne peut se produire qu'à la fin d'un fichier, juste avant l'enregistrement de fin de fichier. Le nombre réel d'octets utilisés (1 à 127) est placé dans le dernier octet de données situé avant la checksum.

- $FE indique que l'enregistrement constitue la fin du fichier et la zone des données comprend 128 octets nuls (00).

## CHECKSUM

La checksum est calculée et vérifiée par le SIO mais elle n'est pas mémorisée dans le tampon d'entrées/sorties cassette CASBUF ($03FD).

La checksum représente simplement l'addition sur un seul octet de tous les autres octets de l'enregistrement, y compris les deux marqueurs. Lorsqu'un octet est additionné à la somme, le bit de retenu est également additionné.



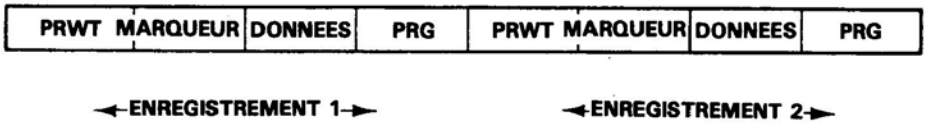



ATARINSIDE

# CHRONOGRAMMES

### SEPARATEUR D'ENREGISTREMENTS

Comme nous l'avons vu précédemment, chaque enregistrement se compose de 132 octets de données incluant l'octet de checksum. Afin de pouvoir distinguer un enregistrement d'un autre, le logiciel de gestion de la cassette ajoute un signal de pré-enregistrement (PRWT) et un signal de fin d'enregistrement (PRG). PRWT et PRG sont des signaux de fréquence pure (signal marque). Le séparateur d'enregistrement se compose du PRG du bloc précédent suivi du PRWT du bloc suivant. Voici l'organisation finale sur la bande :




### MODE COURT ET MODE NORMAL

Les longueurs de PRWT et de PRG dépendent du mode d'écriture défini lors de l'ouverture du canal. Il existe deux modes : le mode normal et le mode court.

Quand un fichier est ouvert, le bit le plus significatif de AUX2 indique le mode choisi. Toutes les opérations ultérieures d'écriture ou de lecture s'effectueront conformément au mode choisi. Lorsque D7 de AUX2 vaut 0, il indique un mode normal. Lorsqu'il vaut 1, il indique un mode court (mode continu).

### MODE NORMAL

Ce mode est utilisé pour permettre d'intercaler un traitement des données entre les lectures de deux enregistrements ; il est ainsi possible d'arrêter, si nécessaire, la bande après chaque enregistrement. Si l'ordinateur exécute son traitement suffisamment vite, la lecture suivante commencera si rapidement que le magnétocassette vera seulement une courte impulsion passer sur la ligne de commande.

### MODE COURT

Dans ce mode, la bande n'est pas arrêtée entre les blocs, que ce soit lors d'un enregistrement ou lors d'une lecture.

En lecture, le programme doit générer une commande READ pour chaque enregistrement avant qu'il passe devant la tête de lecture. L'emploi habituel de ce mode est réservé aux programmes Basic interprêtés, le Basic n'ayant qu'à placer les octets de données en mémoire vive. Cela correspond aux commandes Basic CSAVE et CLOAD.

Bien sûr, il n'est pas possible d'enregistrer un fichier en mode court pour le relire en mode long, car le système perdrait systématiquement le début de l'enregistrement suivant.

### CHRONOGRAMMES

Les temps attribués à chaque séparateur d'enregistrement sont :

| | | |
|---|---|---|
| PRWT | MODE NORMAL | = 3 s de signal marque |
| PRWT | MODE COURT | = 0,25 s de signal marque |
| PRG | MODE NORMAL | = jusqu'à 1 s de signaux quelconques |
| PRG | MODE COURT | = de 0 à n s de signaux quelconques, où n dépend du programme utilisateur d'écriture. |

Chaque enregistrement est écrit de la manière suivante : lorsque le moteur a démarré, PRWT est écrit avec la durée du signal indiquée ci-dessus. Puis l'enregistrement lui-même est transmis et enfin PRG est écrit. Le moteur s'arrête en mode normal, mais il continue de fonctionner en mode court (écriture de marques).

Notez que pour le mode normal, la bande contient une section de données inconnues en raison de l'arrêt et du redémarrage du moteur (jusqu'à une seconde, mais cela dépend du magnétocassette). Ces données inconnues peuvent être par exemple des données d'un programme précédemment enregistré sur cette bande.

### ECHO SONORE

Lors d'une opération de lecture, vous entendez dans le haut-parleur du téléviseur un son composé des signaux marque et espace. Vous constaterez rapidement que cet écho constitue un bon témoin de la qualité du transfert. Lorsque le son devient «bizarre», une erreur se produit généralement.



ATARINSIDE

# STRUCTURE D'UN FICHIER

Un fichier se compose des trois éléments suivants :

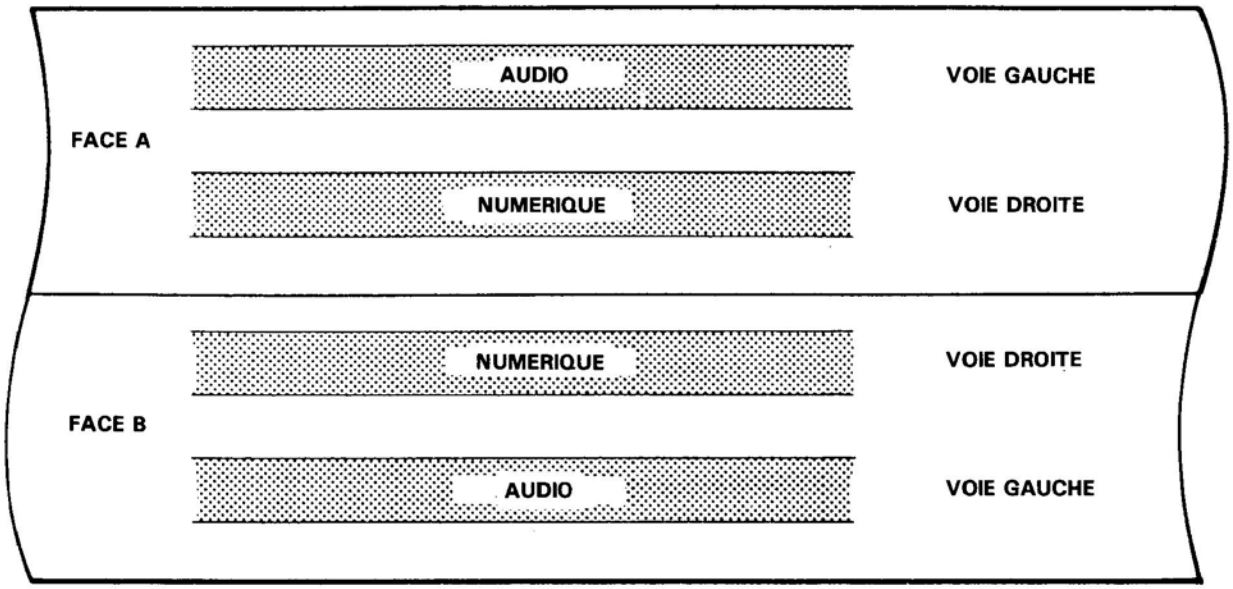
- un en-tête de 20 secondes de signal marque.
- un nombre quelconque d'enregistrements.
- un enregistrement de fin de fichier.

Lorsque le fichier est ouvert en écriture, le SE commence par écrire un en-tête de 20 secondes puis la main est redonnée au programme d'appel, mais la bande tourne toujours et le SIO continue d'écrire les marques.

Le compteur de temps servant à détecter l'absence de réponse d'un périphérique est initialisé à 35 secondes. Si ce temps s'est écoulé avant que le premier enregistrement ne soit écrit, la bande s'arrêtera, laissant un espace entre l'en-tête et le premier enregistrement.

# STRUCTURE DE LA BANDE

La bande est divisée en quatre pistes, deux pistes étant simultanément utilisées pour chaque côté de la cassette. Les deux pistes n'ont pas la même signification : l'une est numérique, l'autre est audio.




La bande utilisée est une bande standard défilant à 4,75 cm/s. Notez que le magnétocassette ATARI possède une tête de lecture stéréo.

# CHARGEMENT AUTOMATIQUE D'UNE CASSETTE

Il est possible de charger automatiquement un programme depuis une cassette lors de la mise sous tension de l'ordinateur. L'initialisation du système réalise un certain nombre de fonctions comme par exemple la remise à zéro de tous les registres matériels, l'effacement de la mémoire, l'initialisation de vecteurs, etc. Lorsque les différents LGP résidents ont été mis en place, et si la touche START a été enfoncée, le drapeau CKEY ($004A) est positionné. Une tentative de chargement de programme en cassette est alors effectué.

Pour obtenir un chargement automatique, il faut réunir simu tanément les conditions suivantes :

1. L'utilisateur doit appuyer sur la touche START lors de a mise sou tension de l'ordinateur.

2. Un fichier prévu pour être chargé automatiquement doit être mis sur la cassette et la touche PLAY doit être enfoncée.

3. Le fichier a dû être enregistré en mode court.

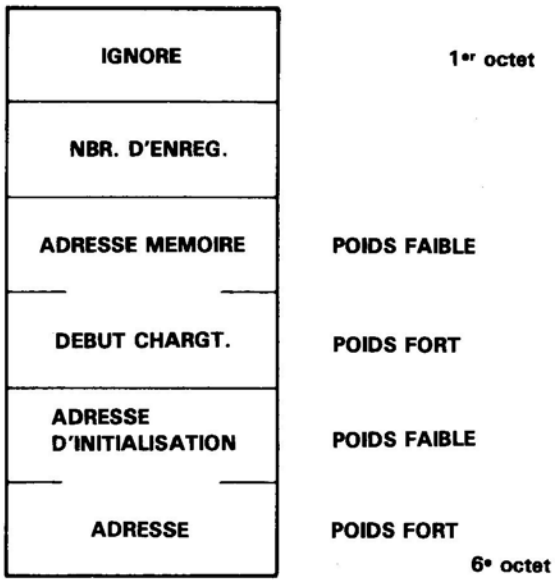
4. Quand le bip sonore se fait entendre, l'opérateur doit appuyer sur une touche du clavier.


ATARINSIDE

Si toutes ces conditions sont réalisées, le SE transfèrera le fichier en mémoire et passera le contrôle à ce logiciel. Le langage utilisé pour le programme est donc l'Assembleur. Le chargement automatique se décompose de la manière suivante :

1. Lecture du premier enregistrement de la bande.

2. Extraction des six premiers octets utilisés ainsi :




Le premier octet n'est pas utilisé.

Le deuxième octet contient le nombre d'enregistrements de 128 octets composant le fichier. Ce nombre peut varier de 1 à 255, 0 signifiant 256 (soit 32768 octets au maximum). Les octets 3 et 4 représentent l'adresse à partir de laquelle doit être chargé le programme.

Les octets 5 et 6 représentent l'adresse de début d'exécution de ce programme. Lorsque le chargement sera terminé, le fait d'appuyer sur SYSTEM RESET renverra également l'exécution à cette adresse.

Lorsque cette étape est terminée, le programme de chargement automatique aura :

- mémorisé le nombre d'enregistrements à lire.
- mémorisé l'adresse de chargement.
- transféré l'adresse d'initialisation dans CASINI ($02, $03).

3. Transfert de l'enregistrement venant d'être lu vers l'adresse de chargement.

4. Lecture des enregistrements restants et transfert direct dans la zone mémoire tampon.

5. JSR vers l'adresse de chargement + 6 où un processus de chargement en plusieurs étapes pourra continuer. Le bit de retenu indique le succès de l'opération (carry = 1 = erreur, carry = 0 = succès) lors du retour.

6. JSR indirect via CASINI pour l'initialisation de l'application. L'application devrait placer son adresse de départ dans le vecteur DOSVEC ($OA, $OB) durant l'initialisation, puis exécuter un RTS.

7. JMP indirect via DOSVEC pour donner définitivement la main à l'application.

Le fait d'appuyer sur SYSTEM RESET lorsque le chargement est terminé provoque la répétition des étapes 6 et 7.

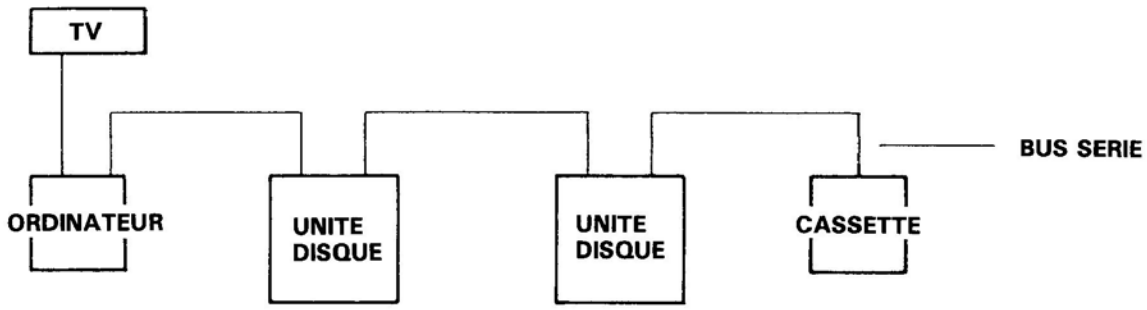


ATARINSIDE

# APPLICATIONS DE LA CASSETTE

## MISE EN PLACE D'UN MAGNETOCASSETTE

La plupart des périphériques possèdent deux connecteurs identiques : l'un est l'entrée du bus série, l'autre une extension de ce même bus. En utilisant ces connecteurs, on peut connecter en cascade les périphériques en suivant le diagramme ci-dessous :




Toutefois, le magnétocassette ne se conforme pas à ce protocole car il ne possède pas de connecteur d'extension. En conséquence, il doit être le dernier maillon de la chaîne. Cela interdit de connecter plus d'un magnétocassette au système. Le système ne peut pas détecter l'absence ou la présence d'un magnétocassette.

Lorsqu'il faut ouvrir un fichier cassette en lecture ou en écriture, utilisez les instructions suivantes :

- lecture (du magnétocassette vers l'ordinateur) : quand le canal est ouvert en lecture, le haut-parleur de l'ordinateur produit un seul bip sonore. Si le magnétocassette est prêt (mis sous tension, câble série connecté, bande bobinée au début du fichier), l'utilisateur doit enfoncer la touche PLAY du magnétocassette puis une touche quelconque du clavier de l'ordinateur.

- enregistrement (de l'ordinateur vers le magnétocassette) : dans ce cas, le haut-parleur produit deux bips sonores. L'utilisateur doit alors enfoncer les touches PLAY et RECORD du magnétocassette puis une touche du clavier.

## TRANSFERTS DE PROGRAMMES NUMERIQUES

Les techniques ci-dessous transfèrent des données numériques directement entre l'ordinateur et le magnétocassette ATARI.

**EN BASIC :**

FORMAT : CSAVE
100 CSAVE

Cette commande est généralement utilisée en mode direct pour sauvegarder le programme interprêté placé dans la mémoire vive sur une cassette.

FORMAT : CLOAD
100 CLOAD

Cette commande peut être utilisée en mode direct ou en mode programme pour transférer des programmes de la cassette dans la mémoire de l'ordinateur (version interprêtée).

**EN ASSEMBLEUR :**

Programme source

FORMAT : LIST = C : [,xx,yy]

Cette commande écrit sur cassette le code source d'un programme en Assembleur. L'option ,xx,yy permet de ne sauvegarder qu'une partie du programme, de la ligne xx à la ligne yy. Si ces numéros de ligne ne sont pas précisés, tout le programme est enregistré.



ATARINSIDE

FORMAT : ENTER = C :

Cette commande lit un code source depuis une cassette.

Programme objet :

FORMAT : SAVE = C : <xxxx,yyyy>

Le contenu d'une zone mémoire, s'étendant de l'adresse xxxx à l'adresse yyyy est sauvegardé sur la cassette.

FORMAT : LOAD = C :

Cette commande recharge en mémoire les données précédemment sauvegardées par SAVE : il est inutile de préciser les adresses de chargement car il s'agit obligatoirement des mêmes que celles utilisées lors de l'enregistrement.

## TRANSFERTS DE PROGRAMMES NUMERIQUES AVEC UN FOND SONORE

Cette technique d'enregistrement permet simplement la reproduction par le haut-parleur du téléviseur d'une musique pré-enregistrée. Il n'est pas possible de commander par programme cette musique. L'intérêt consiste à supprimer la monotonie des longues opérations de chargement.

Etape n°1 — Dans ce concept, il faut pouvoir enregistrer dans un premier temps le programme numérique et dans un deuxième temps, le programme musical. Or, il n'est pas possible sur un magnétocassette de n'enregistrer qu'une piste à la fois sur les deux composant une face. Il faut donc obligatoirement passer par un magnétophone professionnel télécommandable par l'ordinateur. Cette technique n'est donc pas à la portée de l'amateur.

Sur la machine professionnelle, il faut enregistrer la partie audio sur la voie de gauche (canal 1) et la piste numérique sur la voie de droite (canal 2). Dans cette première étape, vous devez enregistrer le programme numérique.

Etape n°2 — Vous rembobinez la bande à son début, puis vous enregistrez sur la voie 1 seulement la partie audio.

## PROGRAMMES NUMERIQUES, AUDIO, MARQUES DE SYNCHRO, ET GESTION DE L'ECRAN

Il n'existe pas de possibilité pour un programme de détecter et d'interprêter un segment audio lorsque la bande défile. Pour permettre d'établir une synchronisation entre l'exécution du programme et la bande sonore que l'on entend, il est possible d'utiliser la piste numérique pour enregistrer des tops de synchronisation.

Etape n°1 — Prenons l'exemple d'un programme audiovisuel dont le script serait :
(MUSIQUE) AUJOURD'HUI JE VAIS VOUS RACONTER UNE FABULEUSE HISTOIRE «LA PRINCESSE ET LA GRENOUILLE». C'EST UNE BELLE HISTOIRE, AUSSI, NE PARTEZ PAS./

(MUSIQUE) AVANT DE COMMENCER, JE VOUDRAIS CONNAITRE VOTRE NOM. S'IL VOUS PLAIT, TAPEZ-LE ET APPUYEZ SUR RETURN. (PAUSE).

(MUSIQUE) ET MAINTENANT, VOICI L'HISTOIRE. IL Y A BIEN LONGTEMPS, VIVAIT UNE TRES BELLE PRINCESSE DANS UN GRAND CHATEAU, ET SON NOM ETAIT YYYY./

(MUSIQUE) PAR UNE BELLE JOURNEE D'ETE, LA PRINCESSE MARCHAIT LE LONG D'UNE RIVIERE.../

REMARQUES :

- «/» veut dire que le programme attend un top de synchronisation. Il faut que le narrateur s'arrête pendant environ 1/2 seconde à cet endroit.

- «PAUSE» indique que le narrateur doit s'arrêter une bonne seconde pour permettre l'arrêt puis le redémarrage du magnétocassette. Chaque segment audio doit durer au moins 10 à 30 secondes car des tops de synchronisation trop rapprochés peuvent perturber l'ordinateur.

Etape n°2 — Il est nécessaire de faire un plan détaillé des relations entre les écrans et le commentaire. Le tableau de la page suivante en donne un exemple.

Etape n°3 — Le programmeur peut ensuite rédiger le programme qui pourrait ressembler à cela :



ATARINSIDE

```
 10 REM PROGRAM "FROG" TO DEMONSTRATE SYNCHRONIZATION
 20 REM OF AUDIO WITH DIGITAL FOR THE CASSETTE SYSTEM
 30 REM
 40 DIM IN$(20)
 50 POKE 54018,52:REM TURN ON MOTOR
 60 GRAPHICS 1
 70 PRINT #6;"THE PRINCESS AND THE FROG":PRINT #6;....:REM
    SET UP THE SCREEN FOR EVENT 2.
 80 GOSUB 1000:REM CHECK SYNC MARK, MAKE SURE THE INTRODUCTION
    IS SAID.
100 POSITION X,Y:PRINT #6;"YOUR NAME?":REM FOR EVENT 4
105 GOSUB 1000:REM EVENT 5
110 POKE 54018,60:REM STOP MOTOR FOR USER INPUT
120 INPUT IN$:REM WAIT FOR THE USER'S NAME
130 POKE 54018,52
135 PRINT #6,CHR$(125):REM CLEAR THE SCREEN
140 POSITION X,Y:PRINT #6;IN$:PRINT #6;....:REM DISPLAY
    SCREEN FOR EVENT 10
150 GODUB 1000:REM MAKE SURE SPEECH FOR EVENT 10 IS FINISHED
160 PRINT #6;....:REM READY FOR EVENT 12
```



ATARINSIDE

| EVENEMENT | AUDIO | ECRAN | ATTENTE SYNCHRO | MOTEUR |
|---|---|---|---|---|
| 1 | | | | |
| 2 | AUJOURD'HUI JE VAIS... | LA PRINCESSE ET LA GRENOUILLE<br>GRAPHIQUE | | |
| 3 | | | OUI | |
| 4 | AVANT DE... | LA PRINCESSE ET LA GRENOUILLE<br>VOTRE NOM? | | |
| 5 | | | OUI | |
| 6 | | | | ARRET |
| 7 | | ATTENDRE JUSQU'A 1 ENTREE RECONNUE | | |
| 8 | | | | |
| 9 | | EFFACE L'ECRAN | | DEPART |
| 10 | ET MAINTENANT... | XXXX<br>GRAPHIQUE | | |
| 11 | | | OUI | |
| 12 | PAR UNE BELLE... | GRAPHIQUE | | |
| 13 | | | | |

TOP DE SYNCHRONISATION : sur la bande, un signal marque est constamment enregistré sur la piste numérique et un top de synchronisation est représenté par un signal espace. Comme mentionné précédemment, le signal marque est à 5327 Hz, le signal espace à 3995 Hz. Le programme de détection peut être :

```
1000 IF INT (PEEK(53775)/32 + 0.5) = INT (PEEK(53775)/32) THEN RETURN : REM TESTE LE 5e
     BIT DE CHAQUE OCTET RECU. SI C'EST UN ZERO, ALORS SYNCHRO.

1010 GOTO 1000
```

COMMANDE DU MOTEUR : il est possible par programme d'arrêter ou de redémarrer le moteur du magnétocassette :

```
MARCHE : POKE 54018,52
ARRET  : POKE 54018,60
```

Etape n°4 — Le programmeur doit estimer le temps d'exécution du programme et le temps du script. Si le programme n'attend pas assez longtemps un top de synchronisation ou si l'ensemble du programme avec la partie audio ne tient pas sur une face de cassette, il faut revoir le problème.

Etape n°5 — Sauvegarder le programme sur une bande master, par exemple MASTER1.

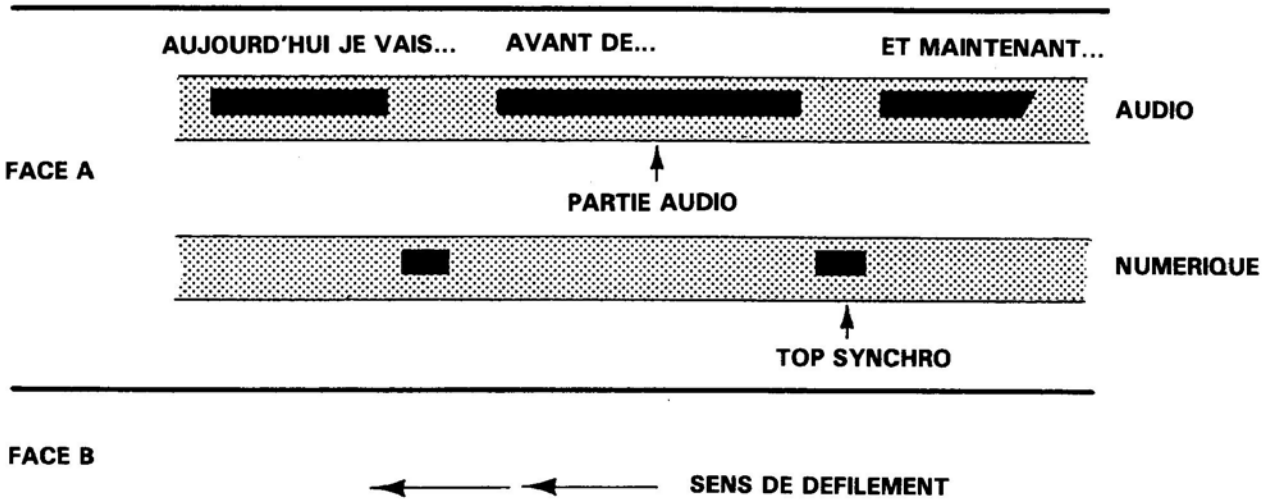


ATARINSIDE

Etape n°6 Enregistrement de la voix sur une autre bande master : MASTER2.

Etape n°7 Production d'une bande MASTER3 mélangeant les bandes MASTER1 et MASTER2. Trois machines professionnelles sont nécessaires à ce stade. Faire deux copies de MASTER3.

Etape n°8 Charger le programme générateur de tops de synchronisation dans l'ordinateur ATARI.




```
 10 REM PUSH "START" CONSOL KEY TO
 20 REM ADD THE SYNC MARK ONTO THE TAPE
 30 REM
 40 REM
 50 IO=53760 : CONSOLE=53279 : CASS=54018
100 FOR I=0 TO 8
110 READ J : POKE IO+I,J
120 NEXT I
125 REM THE FOR LOOP SETS THE AUDIO FREQUENCY & CHANNEL
130 DATA 5,160,7,160,5,160,7,160,0
140 REM
150 REM I/O IS SETUP; NOW START THE CASSETTE
160 POKE CASS, 52
200 POKE CONSOLE,8
210 IF PEEK(CONSOLE)   7 THEN 230:REM CONSOLE=7 MEANS WRITE
    MARK,
220 POKE IO+15,11: GOTO 200: REM CONSOLER KEYS NOT PRESSED
230 POKE IO+15,128+11: GOTO 200: REM IF CONSOLE   7 WRITE
    "SPACE"
```

Etape n°9 Monter les deux copies de la MASTER3 sur deux magnétophones et rembobiner les bandes jusqu'à la fin du programme numérique (début de l'audio). Configurer une machine en enregistrement et une en lecture. Relier un ordinateur ATARI à la machine en enregistrement.

Etape n°10 Taper RUN pour lancer le programme de synchronisation. Puis démarrer simultanément les deux machines, l'une en enregistrement et l'autre en lecture. Ecoutez l'audio et appuyez sur la touche START chaque fois que cela est indiqué dans le script.

Etape n°11 Le résultat final est une bande prête pour la production de masse.



ATARINSIDE

# INHIBITION DE LA TOUCHE BREAK

Nous suggérons au programmeur d'inhiber la touche BREAK. Cela évitera l'arrêt intempestif du magnétocassette lors du chargement d'un programme. Il vous suffit de placer le programme suivant dans le programme d'origine et de l'appeler chaque fois qu'il y aura modification du mode graphique ou une ouverture d'un canal pour l'écran.

```
4000 X = PEEK(16) : IF X>128 THEN 4020
4010 POKE 16,X-128 : POKE 53774,X-128
4020 RETURN
```

# PRODUCTION DE MASSE

Toutes les bandes master ATARI sont enregistrées à 19 cm/s sur une bande 4 pistes 1/4 de pouce. La bande master fournie au duplicateur est appelée master source.

Le duplicateur produira une master de travail en partant de la master source. Le produit final avant duplication se situera à la troisième génération depuis l'ordinateur.


### PRODUCTION EN MASSE

La master intermédiaire est recommandée pour le duplicateur car la master de travail peut être détruite ou abîmée par un emploi intensif. La master source doit être réservée à un emploi exceptionnel, lorsqu'on ne peut plus faire autrement. La master intermédiaire permet de regénérer une master de travail.

### PRODUCTION DE MASSE DES CASSETTES

La master de travail est copiée pour produire une master de duplication. Celle-ci est placée dans un loop bin faisant office de réservoir pour la bande qui y est placée en vrac. Le début et la fin de la bande sont collés entre eux et la boucle est lue très rapidement (32 ou 64 fois la vitesse normale).

A la machine mère sont connectées des machines esclaves utilisant de la bande cassette en bobine. Lorsque le ruban d'amorce de la boucle passe devant la tête de lecture de la machine mère, un bip est généré sur les machines esclaves. Ce bip sépare ainsi deux répétitions de la bande master. Chaque bande de chaque machine esclave contient à la fin de l'opération de duplication plusieurs fois le programme d'origine, par exemple 100 fois. Chaque plateau passe ensuite sur une bobineuse qui charge la bande dans les cassettes en la coupant correctement grâce à la détection du bip sonore.

### CONTROLE DE LA QUALITE

Chaque fois qu'une production de masse est réalisée, des échantillons doivent être prélevés et vérifiés avant que la production soit approuvée et commercialisée. Cela s'effectue normalement en prenant la première et la dernière cassette produite. ATARI doit recevoir au moins 10 échantillons de chaque production pour chaque master produite.



ATARINSIDE

# ANNEXE D
# ABERRATIONS CHROMATIQUES

Ce chapitre explique comment obtenir plusieurs couleurs à l'écran dans un mode graphique qui normalement n'en autorise qu'une.

Les modes de l'ANTIC avec lesquels il est possible de produire des aberrations chromatiques sont les modes 2, 3 et 15. Le mode 2 de l'ANTIC correspond au mode Basic 0, le mode 15 de l'ANTIC est le mode 8 du Basic et le mode 3 de l'ANTIC n'a pas de mode correspondant en Basic. Chacun de ces modes a une résolution d'une demi période d'horloge de large par une ligne de balayage de haut. Ces modes autorisent normalement une couleur et deux luminances. En exploitant les possibilités d'aberrations chromatiques, quatre couleurs différentes peuvent être affichées à l'écran pour chacun de ces modes.

Le terme d'aberration chromatique correspond au fait qu'un point élémentaire de l'écran présente une couleur différente que celle qui lui est assignée. En voici un exemple simple :

```
10 GRAPHICS 8
20 COLOR 1
30 POKE 710,0
40 PLOT 60,60
50 PLOT 63,60
60 GOTO 60
```

Ces instructions afficheront deux points sur un fond noir, toutefois, chaque point aura une couleur différente. Pour comprendre la cause de cette erreur, il faut savoir que les informations affichées sur l'écran du téléviseur composent un signal modulé.

Les deux parties principales de ce signal sont la luminance, ou la luminosité, et la couleur. L'information de luminance est la plus importante. Elle assure la compatibilité avec les téléviseurs noir et blanc et fournit les différents tops de synchronisation. Le signal couleur ne produit que l'information couleur et il est combiné avec l'information de luminance.

La luminance d'un point de l'écran est directement liée à l'amplitude du signal luminance pour ce point.

L'information couleur est donnée par un signal modulé en phase (toute cette théorie s'applique aux standards NTSC et PAL, mais pas SECAM). Une modulation de phase utilise une fréquence d'oscillation modifiée très légèrement par rapport à une fréquence de référence. A l'écart de temps entre ces deux fréquences (une fraction de la période du signal de référence) correspond une couleur.

La fréquence de référence couleur se situe à 3,579 MHz, ce qui détermine 160 points de couleur sur une ligne visible de balayage (il en existe en fait 228, mais 68 sont perdues par les tops de synchronisation et les dépassements de balayage).

Lè terme d'horloge couleur désigne en fait un cycle élémentaire d'information couleur. Le mode GRAPHICS 7 est un exemple d'une résolution couleur maximale, chaque point de l'écran pouvant recevoir une couleur différente (il existe toutefois des limitations dues au microprocesseur).

ATARI propose également un mode haute résolution (GRAPHICS 8) qui permet d'adresser 320 points sur une ligne horizontale. La fréquence du signal de luminance se trouve ainsi modulée jusqu'à 7,16 MHz, soit deux fois la fréquence couleur.

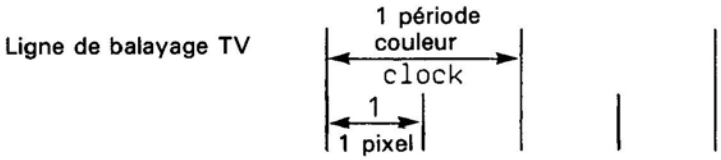
Puisque les deux signaux sont théoriquement indépendants, on devrait pouvoir déterminer une couleur de fond (c'est-à-dire une information de phase «au repos», en l'absence de modulation) et une modification de la luminance point par point. C'est d'ailleurs le principe de base de fonctionnement du mode Basic 8, la couleur de fond provenant du registre couleur 2 et les luminances des registres couleur 1 et 2.

Le problème est qu'en pratique les signaux de couleur et de luminance ne sont pas indépendants. Ils font partie d'un même signal modulé et leur démodulation entraîne une certaine interaction. Ainsi, chaque fois que le système de luminance change, il produit une modification de phase dans le signal couleur et donc une modification de la couleur elle-même. Lorsque la luminance reste constante durant une ou plusieurs périodes du signal couleur, aucun problème ne se pose. Mais si la luminance est modifiée au milieu d'une période du signal couleur, le signal couleur sera légèrement perturbé.

Puisque la luminance peut être modifiée deux fois plus rapidement que l'information couleur, cela produit deux couleurs fausses (première demi période du signal couleur, ou seconde demi période du signal couleur) qui peuvent être combinées pour fournir au total quatre nouvelles couleurs, dont deux durant toute une période du signal couleur. En voici une illustration :



ATARINSIDE




luminance
0 = éteint
1 = allumé

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |

couleur A sur 1/2 période horloge couleur
couleur B sur 1/2 période horloge couleur
couleur C sur 1 période horloge couleur
couleur D sur 1 période horloge couleur

Notez que ces pixels nécessitent une période d'horloge couleur de distance et ainsi la résolution horizontale descend à 160.

Les couleurs A à D sont différentes d'un téléviseur à un autre car elles dépendent grandement des circuits du téléviseur et de ses réglages. Elles ne peuvent être décrites comme des couleurs absolues comme par exemple le rouge ou le bleu mais elles sont bien distinstes les unes des autres et il est tout à fait possible de les utiliser.

Pour illustrer cela, voici un exemple. Ce programme trace des lignes dans chacune de ces quatre couleurs artificielles et remplit des zones en utilisant trois de ces couleurs (notez que l'affichage de nombreux pixels du type C ou D proches les uns des autres donne le même résultat : une ligne de luminance constante avec la couleur du fond).

La commande POKE 87,7 demande au SE de fonctionner en mode 7, ce qui permet l'emploi de masques sur 2 bits pour positionner facilement les bits voulus dans la mémoire écran. Pour produire la couleur A, utilisez COLOR 1, puis COLOR 2 pour la couleur B, et COLOR 3 pour la couleur C. La couleur D est produite en affichant des points COLOR 1 à gauche de COLOR 2.

```
10   GRAPHICS 8:POKE 87,7:POKE 710,0:POKE 709,14
20   COLOR 1:PLOT 10,5:DRAWTO 10,70
30   PLOT 40,5:DRAWTO 40,70
40   COLOR 2:PLOT 20,5:DRAWTO 20,70
50   PLOT 41,5:DRAWTO 41,70
60   COLOR 3:PLOT 30,5:DRAWTO 30,70
70   FOR X=1 TO 3:COLOR X:POKE 765,X
80   PLOT X*25+60,5:DRAWTO X*25+60,70
90   DRAWTO X*25+40,70:POSITION X*25+40,5
100  XIO 18,#6,12,0,"S:"
110  NEXT X
```


ATARINSIDE

# ANNEXE E
# GTIA

Le GTIA est un nouveau circuit d'affichage qui a remplacé le CTIA. Il reprend les mêmes possibilités du CTIA auquel il en ajoute 3 : 3 nouvelles manières d'interprêter les informations provenant de l'ANTIC. L'ANTIC, quant à lui, ne nécessite pas un nouveau mode pour parler au GTIA ; il utilise simplement le mode haute résolution $F. Tous les programmes fonctionnant sur le CTIA fonctionnent sur le GTIA. Par contre, l'inverse n'est pas toujours vrai.

Le CTIA est conçu pour générer l'image vidéo ; il produit le dessin de l'écran, les Players et les Missiles ; il détecte les recouvrements ou les collisions entre des objets de l'écran. Le CTIA interprête les données fournies par l'ANTIC de 14 manières différentes : 6 modes texte et 8 modes graphiques. Dans un affichage statique, il utilisera 4 registres couleur pour définir couleur et luminance de chaque point de l'écran. Le GTIA étend cela aux 9 registres couleur, en utilisant 9 couleurs et 9 luminances sur l'écran, ou 16 couleurs avec une même luminance ou 16 luminances d'une même couleur.

Les trois nouveaux modes graphiques du GTIA sont simplement trois nouvelles interprêtations du mode haute résolution $F de l'ANTIC. Ces trois modes affectent l'image «statique», les Players Missiles pouvant toujours être ajoutés pour introduire de nouvelles couleurs et de nouvelles luminances. Toutes les couleurs et les luminances peuvent toujours être changées en cours d'image par des interruptions de Display List. Le GTIA utilise 4 bits de donnée de l'ANTIC pour chaque pixel. Chaque pixel est large de 2 périodes d'horloge couleur et haut d'une ligne de balayage. En conséquence, le point graphique élémentaire est donc quatre fois plus large que haut. La résolution est de 80 points horizontalement et de 192 points verticalement. Chaque ligne de balayage nécessite donc 320 bits, soit 40 octets de mémoire, le même nombre d'octets utilisés dans le mode ANTIC $F. Un programme qui utiliserait les modes propres au GTIA doit donc attribuer au moins 8 ko de mémoire vive pour l'affichage.

Les modes du GTIA sont sélectionnés par le registre de priorité PRIOR. PRIOR est à l'adresse $D01B avec son registre cache associé à l'adresse $026F. Les bits D6 et D7 sont les bits de commande. Lorsqu'aucun d'eux n'est positionné, le GTIA opère comme le CTIA. Lorsque D7 vaut 0 et D6 vaut 1, le mode 9 est sélectionné, ce qui donne 16 luminances différentes de la même couleur. Souvenez-vous que la donnée fournie par l'ANTIC est de 4 bits de large, ce qui autorise bien 16 valeurs différentes. les Players et les Missiles restent disponibles pour introduire de nouvelles couleurs. Quand D7 est à 1 et D6 à 0, le mode 10 est spécifié. Ce mode autorise 9 couleurs en utilisant les 4 registres couleur standards, les 4 registres couleur des Players Missiles et le registre de couleur de fond. Lorsque les Players Missiles sont utilisés avec ce mode, les 4 registres couleur PM sont bien sûr également utilisés pour eux. Lorsque D7 est à 1 et D6 à 1, on sélectionne le mode 11. Cela donne 16 couleurs avec la même luminance. Les Players et les Missiles redeviennent indépendants.

PRIOR

| D7 | D6 | | | | | | |
|---|---|---|---|---|---|---|---|

| D7 | D6 | OPTION | |
|---|---|---|---|
| 0 | 0 | Modes standards (modes CTIA) | (0-8) |
| 0 | 1 | 1 couleur, 16 luminances | (mode 9) |
| 1 | 0 | 9 couleurs et luminances | (mode 10) |
| 1 | 1 | 16 couleurs, 1 luminance | (mode 11) |

Pour utiliser ces modes depuis le Basic, il vous suffit d'employer les commandes GRAPHICS 9, GRAPHICS 10 et GRAPHICS 11 respectivement pour les modes 9, 10 et 11. En Assembleur, la procédure est tout à fait similaire pour ouvrir l'écran. Si vous construisez votre propre Display List, vous devez alors positionner vous-même les bits 6 et 7 de PRIOR comme indiqué dans le tableau ci-dessus.



ATARINSIDE

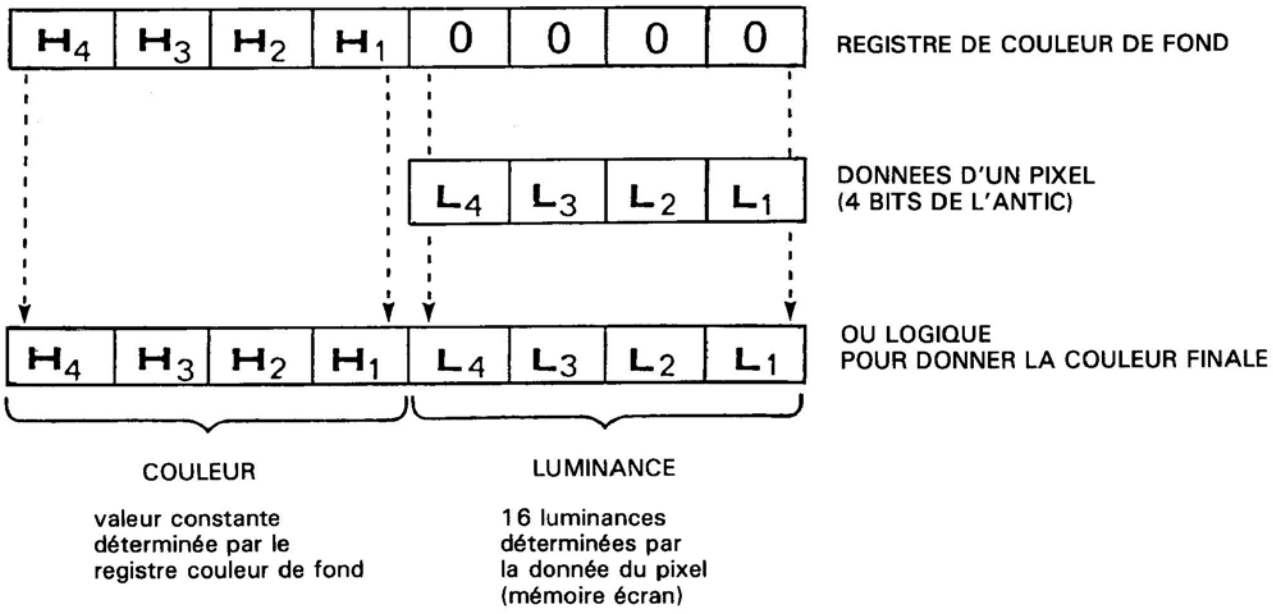
Le mode 9 produit 16 luminances différentes pour la même couleur. L'ANTIC fournit les 4 bits nécessaires pour définir la luminance à chaque point. Le registre couleur de fond détermine la couleur. En Basic, la couleur est déterminée par l'instruction :

SETCOLOR 4,COULEUR,0

COULEUR pouvant prendre n'importe quelle valeur entre 0 et 15. Il est nécessaire de mettre la luminance à 0 car la donnée en provenance de l'ANTIC est combinée par un OU logique avec les 4 bits de poids faible (donc la luminance) du registre de couleur de fond. Si ces 4 bits n'étaient pas à 0, la donnée de l'ANTIC serait modifiée. La commande SETCOLOR est ensuite utilisée pour sélectionner la luminance voulue. COLOR peut varier de 0 à 15. Ainsi, un programme Basic pourrait se présenter de cette manière :

```
10 GRAPHICS 9 : REM POUR SPECIFIER LE MODE 9
20 SETCOLOR 4,12,0 : REM INITIALISE COULEUR DU REGISTRE DE FOND
30 FOR I=0 TO 15
40 COLOR I
50 PLOT 4,I+10 : REM AFFICHAGE DES DIFFERENTES LUMINANCES
60 NEXT I
70 GOTO 70
```

En Assembleur, utilisez le registre cache du registre couleur de fond $2C8 en codant la couleur dans les 4 bits de poids forts. Si des appels au CIO sont utilisés, placez la donnée du pixel dans le registre ATACHR situé à l'adresse $2FB. Cela permet de modifier la luminance en utilisant les valeurs hexa $0 à $F. Si vous gérez vous-même votre propre mémoire écran, vous devez placer la donnée du pixel directement dans la moitié supérieure ou inférieure de l'octet de la mémoire écran.




Le mode 11 est tout à fait similaire au mode 9 mais il lui est symétrique : 16 couleurs avec une même luminance. Pour chaque point graphique, l'ANTIC fournit une donnée sur 4 bits provenant de la mémoire écran. Cette donnée détermine la couleur. La luminance commune est donnée par les 4 bits de poids faible du registre couleur de fond tandis que les 4 bits de poids fort sont forcés à 0. La commande Basic devient :

SETCOLOR 4,0,LUMINANCE

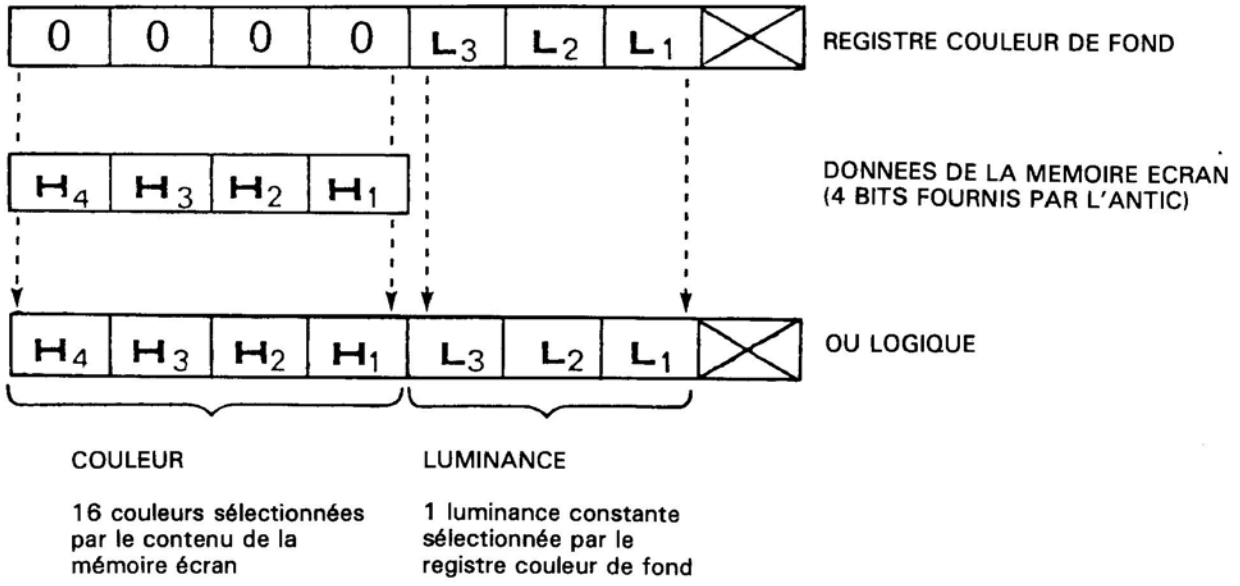
Comme dans tous les autres modes graphiques sauf dans le mode 9, le premier bit de la luminance n'est pas utilisé. Si bien que seuls 8 luminances différentes sont possibles. La commande COLOR détermine la couleur de chaque point graphique. Un OU logique est également réalisé entre le registre couleur de fond et les 4 bits de donnée fournis par l'ANTIC depuis la mémoire écran. Voici un programme d'exemple :

```
10 GRAPHICS 11 : REM SPECIFIE MODE 11
20 SETCOLOR 4,0,12 : REM POUR INITIALISER LA LUMINANCE REGISTRE COULEUR DE FOND
30 FOR I=0 TO 15 : REM MODIFIE LA TEINTE
40 COLOR I : REM PAR L'INSTRUCTION COLOR
50 PLOT 4,I+10
60 NEXT I
70 GOTO 30
```



ATARINSIDE

En Assembleur, placez la valeur désirée pour la luminance dans les 4 bits de poids faibles du registre cache $2C8 (couleur de fond). Cette valeur doit donc être comprise entre 0 et $F. Si des appels au CIO doivent être exécutés, placez la donnée du pixel dans ATACHR, en $2FB. Vous choisissez ainsi la teinte avec une valeur comprise entre 0 et $F.




Formation d'une teinte en mode 11

Le mode 10 autorise l'emploi simultané des 9 registres couleur. Chaque registre contient une information complète : couleur + luminance. La donnée contenue dans la mémoire écran sélectionne le registre utilisé. En Basic, la commande SETCOLOR est utilisée de manière classique pour le registre couleur de fond et pour les 4 registres image. Vous pouvez remplacer l'instruction SETCOLOR par l'instruction POKE, ces 5 registres se situant aux adresses $708 à $712. Vous devez obligatoirement utiliser l'instruction POKE pour modifier le contenu des registres couleur des Players-Missiles (adresses $704 à $707). La commande COLOR est utilisée pour choisir le registre couleur désiré. Les seules valeurs utilisables vont de 0 à 8. Puisque l'ANTIC fournit 4 bits par point graphique, cela autorise la sélection de un registre parmi 16. Toutefois, les valeurs comprises entre 9 et 15 sélectionnent 1 des 9 registres déjà existants. Un programme Basic utilisant le mode 10 devra donc inclure :

1. Une commande GRAPHICS 10 pour spécifier le mode 10.
2. Un jeu d'instruction POKE (et/ou SETCOLOR) pour positionner les registres couleur.
3. Une commande COLOR pour choisir le registre couleur.

En Assembleur, placer la donnée correspondant au point graphique dans ATACHR ($2FB) ou directement dans la mémoire écran comme dans les modes 9 et 11. Dans ce mode, n'oubliez pas que la donnée doit être comprise entre 0 et 8.

| VALEUR DE L'INSTRUCTION COLOR | REGISTRE COULEUR UTILISE | MEMOIRE CACHE |
|---|---|---|
| 0 | D012 | 2C0 |
| 1 | D013 | 2C1 |
| 2 | D014 | 2C2 |
| 3 | D015 | 2C3 |
| 4 | D016 | 2C4 |
| 5 | D017 | 2C5 |
| 6 | D018 | 2C6 |
| 7 | D019 | 2C7 |
| 8 | D01A | 2C8 |

Registres utilisés en mode 10



ATARINSIDE

Tous les logiciels fonctionnant sur un CTIA tourneront correctement sur un GTIA. Les modes graphiques propres au GTIA sont parfaitement compatibles avec le système d'exploitation et avec le Basic. De nombreuses couleurs sont disponibles simultanément. Jusqu'à 16 nuances peuvent être utilisées sur une même ligne, indépendemment du microprocesseur. Cela est donc plus puissant que les interruptions de Display List qui donnent au maximum 12 couleurs par ligne. Des impressions de relief par variation de luminance sont possibles en mode 9. Par contre, il existe quelques désavantages. Les nouveaux modes du GTIA ne sont pas des modes texte, ils n'autorisent pas non plus de texte dans le bas de l'écran ; il faut donc créer sa propre Display List pour cela. Le point graphique dans les modes 9, 10 et 11 est quatre fois plus long que haut. Il est donc difficile de représenter des lignes courbes. Chaque point graphique nécessite 4 bits pour être pleinement déterminé. Chacun de ces modes nécessite donc 8 ko de mémoire vive. les programmes conçus pour le GTIA produiront sur le CTIA des dessins reconnaissables mais peu colorés. Il n'est pas possible de déterminer par programme si un GTIA est installé dans l'ordinateur ou non. Enfin pour terminer, les aberrations chromatiques produites par un GTIA ne sont pas identiques à celles générées par un CTIA.


ATARINSIDE

# GLOSSAIRE

**$**

Ce symbole placé devant un nombre indique que ce dernier est représenté en notation hexadécimale (base 16).

## A

**ABERRATION CHROMATIQUE**

Manière astucieuse d'utiliser les défauts de décodage couleur des téléviseurs PAL ou NTSC pour créer de nouvelles couleurs. Cela autorise 4 couleurs en mode Basic 8.

**ANTIC**

Il s'agit d'un microprocesseur indépendant, conçu spécialement par ATARI pour l'affichage TV. L'ANTIC est programmable par l'utilisateur. Il possède un jeu d'instructions, un programme (la Display List), et des données (la mémoire écran).

**ATTENTE (MODE)**

Cette possibilité du système d'exploitation produit une rotation automatique des couleurs après qu'aucune touche n'ait été enfoncée pendant 10 minutes. Ainsi, les différents luminophores de l'écran (de couleurs différentes) sont successivement allumés puis éteints, ce qui évite leur usure prématurée et l'apparition d'une image fantome.

## B

**BCD**

Acronyme pour Binary Coded Decimal, soit décimal codé binaire. Il s'agit d'un système de numérotation dans lequel chaque nombre devient une suite de chiffres décimaux. Chaque chiffre est ensuite codé en binaire, sur 4 bits. Dans l'ordinateur ATARI, deux chiffres sont donc enregistrés sur un octet. Un nombre est toujours représenté en virgule flottante, sur 6 octets.

**BLANKING HORIZONTAL**

Sur l'écran TV, retour du faisceau d'électrons au début de la ligne vidéo suivante.

**BLANKING VERTICAL**

Lorsque le faisceau a tracé toutes les lignes d'une image, il se situe dans le coin inférieur droit de l'écran. Il faut le remonter rapidement dans le coin supérieur gauche de l'écran : c'est la période de Blanking Vertical, qui sépare deux images successives.

**BORD DE L'ECRAN**

En mode Basic 0, il s'agit de la zone située sur le périmètre de l'écran, au delà de la zone du fond de l'écran. Elle est noire à la mise sous tension, mais peut recevoir n'importe quelle couleur.

**BRKKEY**

Drapeau positionné par le système d'exploitation lorsque la touche BREAK est enfoncée. Sa valeur normale est $FF. Si elle est différente, la touche BREAK a été enfoncée.

**BUFFER D'ENTREE**

Voir TAMPON D'ENTREE.



ATARINSIDE

# C

**CACHE**

Voir REGISTRE CACHE.

**CARACTERES GRAPHIQUES**

Caractères obtenus en appuyant simultanément sur la touche CTRL et sur une touche alphabétique. Ils permettent la réalisation de cadres et ornements divers. Ce terme désigne également des caractères alphanumériques ordinaires qui auraient été redéfinis de manière à former sur l'écran de nouveaux graphiques tout en restant manipulables comme des caractères.

**CHBAS**

Nom mnémonique donné au registre cache dont le contenu indique à l'ANTIC où se trouve le début de la table des caractères. L'adresse de CHBAS est 756.

**CHECKSUM**

Il s'agit d'un octet dont la valeur est déterminée par la somme avec la retenue de tous les octets composant un enregistrement sur disquette ou sur cassette.

**CIO**

Acronyme pour l'ensemble des sous-programmes du système d'exploitation responsable du paasage des commandes du programme utilisateur au logiciel de gestion associé au périphérique voulu (Central Input Output system). C'est en fait le point d'entrée commun pour la plupart des opérations d'entrées/sorties du système d'exploitation.

**COLDSTART**

Voir DEMARRAGE A FROID.

**COLLISION**

Cela se produit lorsque deux registres couleur différents devraient être utilisés pour un même point de l'écran. Ce qui se produit en pratique lorsqu'un Player ou un Missile passe sur une portion de l'image ne correspondant pas à sa propre couleur. Au maximum, 60 collisions sont détectables, certaines n'étant pas utilisables. Chacune peut être détectée indépendamment des autres, et à chacune est associé un drapeau sur 1 bit. D'où l'emploi de 16 registres placés dans le CTIA (seuls les 4 bits de poids faibles sont utilisés dans chaque registre ; certains bits sont inutilisés).

**COMMANDE**

Partie élémentaire d'une instruction Basic interprêtée, codée sur un octet. Indique au Basic la manière dont il faut interprêter les autres codes de l'instruction. Ainsi, par exemple, dans PRINT,A : PRINT est la commande, et de cette commande dépend le sens que l'on attribue à la virgule et à la lettre A.

**COMPTEUR D'OCTETS**

Compteur mémorisant la position du pointeur utilisé pour les opérations de lecture ou d'écriture dans un fichier sur disquette. Ce pointeur est utilisé à l'intérieur d'un secteur. Sa valeur varie entre 0 et 124. On peut lire ou modifier sa valeur respectivement par les instructions Basic NOTE et POINT (qui modifient également le pointeur de secteur).

**COMPTEURS DE TEMPS**

Il s'agit de registres initialisés à un certain instant, puis décrémentés très régulièrement, indépendamment les uns des autres (soit par logiciel, soit par une circuiterie adéquate programmable). Lorsque le compteur atteint 0, soit une interruption IRQ est générée, soit le logiciel le détecte par lui-même.

**COMPTEURS DE TEMPS MATERIELS**

(Voir COMPTEURS DE TEMPS). Ces compteurs de temps sont placés dans le POKEY. Le programmeur peut modifier la fréquence de décomptage de ces registres.

**CONSTANTE**

Valeur codée BCD sur 6 octets représentant un nombre. Cette valeur reste inchangée durant l'exécution d'un programme. Après avoir été interprêtée par le Basic, cette constante est précédée d'un 7e octet (drapeau particulier au langage Basic). 12 et 3,141592 sont des constantes.

**CTIA**

Circuit conçu spécialement par ATARI et responsable de la production de l'image elle-même. Le CTIA est normalement contrôlé par l'ANTIC.



ATARINSIDE

**COULEUR**

Encore appelée dans ce manuel teinte ou nuance. Correspond en fait à l'ensemble couleur + luminance défini par les 8 bits d'un registre couleur. 128 teintes différentes peuvent être codées dans un registre couleur.

# D

**DCB**

Décimal Codé Binaire. Voir BCD.

**DEFILEMENT FIN**

Par opposition à un défilement grossier, un défilement fin provoque le déplacement de l'image vidéo point élémentaire par point élémentaire, sans sauts brusques.

**DEFILEMENT GROSSIER**

Procédé consistant à modifier les octets opérandes (c'est-à-dire l'adresse, ou encore l'argument) placés après l'octet de commande LMS dans une Display List. Cela produit un défilement horizontal ou vertical de l'image octet par octet (c'est-à-dire caractère par caractère en modes texte. S'oppose à défilement fin (voir ce mot).

**DEFILEMENT HORIZONTAL**

Déplacement de l'image de la gauche vers la droite ou inversement. Cela permet de voir davantage d'informations qu'avec une image statique. Le défilement peut être fin ou grossier.

**DEFILEMENT VERTICAL**

Défilement de l'image du haut vers le bas et inversement. Le défilement peut être fin ou grossier.

**DEMARRAGE A CHAUD**

Ensemble des opérations s'exécutant lorsque l'utilisateur appuie sur SYSTEM RESET.

**DEMMARRAGE A FROID**

Ce terme décrit l'ensemble des opérations se produisant uniquement lors de la mise sous tension de l'ordinateur ; notamment, toute la mémoire vive est effacée.

**DEVICE HANDLER**

Traduit par Logiciel de Gestion de Périphérique (LGP). Voir ce mot.

**DISPLAY LIST**

Cette liste d'octets constitue le programme de l'ANTIC. Elle est générée lorsque le programme exécute une instruction GRAPHICS. Le programmeur peut la modifier lui-même. La Display List indique l'emplacement de la mémoire écran, le mode graphique utilisé, ligne d'affichage par ligne d'affichage, et un certain nombre d'options (interruptions, défilements fin et grossier, etc).

**DLI**

Acronyme de Display List Interrupt. Voir INTERRUPTION DE DISPLAY LIST.

**DMA**

Acronyme de Direct Memory Access, ou accès direct en mémoire. Désigne l'opération réalisée par l'ANTIC lorsqu'il lit la Display List ou la mémoire écran, et consistant à arrêter temporairement le 6502, à s'approprier les bus du système pour réaliser ses propres opérations, avant de redonner le contrôle au 6502.

**DOS**

Disk Operating System. Voir SED.

**DUP OU DUP.SYS**

Disk Utility Package. Voir SED.



ATARINSIDE

# E

**ENREGISTREMENT**

Lors d'une opération d'entrée/sortie, c'est une suite de caractères se terminant par le caractère spécial $9B. Dans un fichier, c'est la quantité minimale d'information traitée à la fois (par exemple, le nom d'un objet).

**EOL**

Caractère $9B indiquant la fin d'une ligne logique ou la fin d'un enregistrement logique sur cassette ou sur disquette. Est généralement généré par la touche RETURN.

**E/S**

Acronyme pour entrée/sortie. Une entrée correspond à la réception de caractères par l'ordinateur. Ces caractères viennent du clavier ou d'un périphérique. Une sortie correspond à l'émission de caractères par l'ordinateur (typiquement, l'écran est un périphérique de sortie).

# F

**FENETRE TEXTE**

Ce sont les 4 lignes de texte apparaissant dans le bas de l'écran dans les modes Basic 1 à 8.

**FICHIER**

Terme général désignant un ensemble cohérent d'octets stockés sur cassette ou sur disquette. Un fichier peut être un ensemble de données (comme un répertoire d'adresses par exemple) ou un programme.

**FICHIER CASSETTE A CHARGEMENT AUTOMATIQUE**

Fichier binaire représentant le code objet d'un programme Assembleur se chargeant automatiquement dans l'ordinateur lorsque vous appuyez sur la touche START lors de la mise sous tension.

**FMS**

Partie du SED (voir ce mot) se chargeant dans la mémoire de l'ordinateur à sa mise sous tension.

**FOND DE L'ECRAN**

La zone sur l'écran TV où les Players-Missiles, les différents objets et le texte apparaissent. le fond de l'écran possède son propre registre couleur (qui est initialisé avec la couleur bleue à la mise sous tension de l'ordinateur).

**FORMAT OU FORMATAGE**

Voir INITIALISATION.

# G

**GTIA**

CTIA amélioré autorisant les modes Basic 9, 10 et 11 (variantes du mode $F de l'ANTIC).

# H

**HANDLER**

Voir LGP.

**HATABS**

Abréviation de HAndler TABleS. Le CIO utilise cette table pour trouver l'adresse de début du LGP associé à un périphérique, lui-même désigné par une lettre. La HATABS se compose donc de triplet d'octet : lettre d'identification du périphérique (1 octet), adresse du LGP associé (2 octets). La HATABS accepte 14 triplets au maximum. 5 sont initialisés à la mise sous tension de l'ordinateur, les 9 autres étant libres. La HATABS commence à l'adresse $031A.

**HORLOGE SYSTEME**

Horloge principale de laquelle dépendent tous les autres signaux de l'ordinateur. Sa fréquence est 3,59 MHz.



ATARINSIDE

# I

### IMAGE LARGE, NORMALE, ETROITE

En temps normal, une ligne de balayage se compose de 160 périodes d'horloge couleur. Mais par programme, cette longueur de ligne peut être diminuée jusqu'à 128 (image étroite) ou augmentée à 192 périodes d'horloge couleur. Cela modifie en conséquence le nombre de caractères disponibles par ligne dans la mémoire écran (mais pas à l'affichage).

### INITIALISATION OU INIT

Action effectuée par le DUP et permettant de préparer une disquette vierge pour son emploi par l'unité de disquette A850 ou A1050. Si la disquette contenait déjà des fichiers, ceux ci sont systématiquement effacés.

### INSTRUCTION

En Basic, une instruction est l'ensemble commande-opérateur-constante-variable définissant une opération de base pour l'ordinateur. Lorsque plusieurs instructions sont placées sur la même ligne, elles doivent être séparées par :.

### INTERPRETATION

Cette phase transforme les frappes au clavier en codes directement utilisables par la cartouche Basic.

### INTERRUPTION DE DISPLAY LIST

Instruction particulière de la Display List interrompant le fonctionnement normal du 6502 pour lui permettre de modifier certains paramètres de l'image alors même qu'elle est en cours de tracé.

### I/O

Acronyme pour Input/Output. Voir E/S.

### IOCB

Input/Output Control Block. Ensemble de 16 octets servant à transférer les commandes entre le programme et le CIO. Huit IOCB différents sont utilisables simultanément.

### IRQ

Interruption du 6502 ; masquable.

# J

### JEUX DE CARACTERES

L'ordinateur ATARI autorise la redéfinition par le programmeur de l'ensemble des caractères définis à la mise sous tension. L'utilisateur peut même définir plusieurs groupes de caractères (on parle alors de jeux de caractères ou de polices), et il lui suffit de placer le poids fort de l'adresse de départ de la table des caractères dans CHBAS.

# L

### LIGNE

En Basic, une ligne se compose de 120 caractères au maximum, y compris l'éventuel numéro de ligne ; soit en pratique 3 lignes de texte à l'écran.

### LIGNE D'AFFICHAGE

Ensemble de 1 à 10 lignes de balayage formant une ligne cohérente d'information. Ainsi, en mode GRAPHICS 0 par exemple, une ligne de texte est une ligne d'affichage ; elle comprend 8 lignes de balayage.

### LIGNE DE BALAYAGE (HORIZONTAL)

Unité de mesure de la hauteur de l'image vidéo correspondant à la hauteur d'une ligne tracée par un seul passage du faisceau d'électrons. Un caractère en mode GRAPHICS 0 est composé de 8 lignes de balayage.

### LGP

Acronyme de Logiciel de Gestion de Périphérique. Chaque LGP est une sous-partie du Système d'Exploitation, dont le but est d'effectuer l'interface logiciel entre le périphérique auquel il correspond et le système d'exploitation. les LGP sont normalement appelés et gérés par l'organe centralisateur responsable de toutes les opérations d'E/S : le CIO. Les LGP résidents dans la mémoire morte de l'ordinateur sont ceux de l'éditeur, de l'écran, du clavier, de l'imprimante et



ATARINSIDE

du magnétocassette. Certains LGP se chargent en mémoire vive : le LGP disquette (composé du FMS et du DUP), le LGP du module d'interface, et les LGP spéciaux définis par le programmeur.

**LGP RESIDENT**

Ce LGP constitue une amorce du dialogue avec la disquette. De lui-même, il est insuffisant, mais il permet le chargement du DOS à la mise sous tension. Il est inclus dans le SE en MEM.

**LUMINANCE**

Synonyme de brillance. La luminance est déterminée par les 4 bits de poids faibles de chaque registre couleur.

# M

**MARQUE**

Signal à 5327 Hz servant lors d'une opération d'E/S avec le magnétocassette.

**MARQUEUR**

Octet dont la valeur est $55. Deux marqueurs sont enregistrés au début de chaque bloc sur cassette afin de permettre l'ajustage de la vitesse de transmission lors de la relecture.

**MEM**

MEmoire Morte. Cette mémoire reçoit un logiciel lors de sa fabrication et ne le perds pas lorsqu'elle est hors tension. Ce type de mémoire est utilisé dans les cartouches de jeux et de langages.

**MEMOIRE ECRAN**

Partie de la mémoire recevant les données devant être affichées sur l'écran TV. Sa taille varie en fonction du mode graphique utilisé.

**MEV**

MEmoire Vive. C'est la mémoire effacée lors de la mise sous tension de l'ordinateur.

**MISSILE**

Mobile de 2 bits de large. 4 missiles sont définissables simultanément.

**MODE CARACTERE**

Mode d'affichage de l'ANTIC dans lequel les données de la mémoire écran sont interprêtées comme des caractères (utilisation de la table générateur de caractères en provenance de la mémoire morte. L'ANTIC autorise 6 modes caractère, 3 d'entre eux sont accessibles depuis le Basic (GR.0, GR.1, et GR.2).

**MODE COURT**

La bande du magnétocassette n'est pas arrêtée entre deux blocs de données consécutifs. Convient pour les programmes Basic interprêtés (instructions CSAVE et CLOAD).

**MODES D'AFFICHAGE**

Méthodes d'interprêtation des octets placés dans la mémoire écran. L'ANTIC autorise 15 modes d'affichage différents, le Basic seulement 9.

**MODE DIFFERE**

Voir MODE PROGRAMME.

**MODE IMMEDIAT**

Une ligne d'instructions Basic non précédée d'un numéro de ligne. Le Basic exécute immédiatement cette ligne dès que l'on a appuyé sur RETURN.

**MODE NORMAL**

La bande du magnétocassette s'arrête entre 2 blocs de données consécutifs. Opposé à MODE COURT. Convient pour des fichiers.



ATARINSIDE

**MODE PROGRAMME**

En Basic, se dit d'une instruction (ou d'un groupe d'instructions) précédée d'un numéro de ligne. L'exécution aura lieue après avoir tapé RUN en mode immédiat.

**MONITEUR**

Programme utilisé lors de la mise sous tension de l'ordinateur ou lorsqu'on appuie sur SYSTEM RESET.

# N

**NMI**

Interruption non masquable pour le 6502.

# O

**OCTET DE COMMANDE**

Encore appelé octet de contrôle. Il est utilisé dans l'en-tête de chaque enregistrement sur cassette et peut avoir 3 significations distinctes.

# P

**PAGE 0**

Adresses allant de 0 à 255. Certains modes d'adressage du 6502 exploitent astucieusement ces adresses.

**PERIODE D'HORLOGE COULEUR**

Unité standard de mesure de distance horizontale sur une image TV. Une ligne horizontale se compose de 228 périodes d'horloge couleur mais 160 seulement sont visibles avec une image de largeur normale (la largeur de l'image peut être programmée : étroite, normale ou large).

**PIA**

Périphal Interface Adaptor. Circuit électronique à haute densité. Utilisé dans les ordinateurs ATARI notamment pour contrôler les manettes de jeux.

**PILE DU PROGRAMME**

Zone mémoire particulière utilisée par le Basic lors de ses calculs, de ses boucles et de ses sous-programmes.

**PIXEL**

Voir POINT ELEMENTAIRE.

**PLAYER**

Encore appelé «lutin» ou «sprike» ou «mobile». C'est une image indépendante de la mémoire écran, et pouvant être très facilement déplacée sur l'écran. Un Player est défini par 128 ou 256 octets selon la résolution choisie. Un octet représente 8 points élémentaires de large sur l'écran. Cette largeur peut être doublée ou quadruplée.

**POINT ELEMENTAIRE**

Le plus petit point «allumable» sur l'écran. Un caractère en mode Basic 0 est défini dans un rectangle de $8 \times 8$ points élémentaires.

**POINT GRAPHIQUE**

Unité d'affichage (par exemple c'est un caractère en mode Basic 0, ou un petit rectangle de couleur en mode Basic 7) composée de plusieurs points élémentaires.

**POINTEUR**

Registre ou ensemble de registres indiquant à l'ordinateur où il en est dans sa tâche (par exemple quel secteur sur quelle piste de la disquette a-t-il lu en dernier?).



ATARINSIDE

**POKEY**

Circuit électronique spécial conçu par ATARI et responsable du bus série, des effets sonores, de la scrutation du clavier et de la génération de nombres aléatoires. Le POKEY contrôle également les interruptions masquables et décode les commandes à molette.

**PRIORITE**

Lorsqu'un point de l'écran doit recevoir deux couleurs différentes (ce qui se produit inévitablement avec les Players Missiles), l'ordinateur doit savoir quelle couleur choisir : celle du Player, celle de l'image statique, ou celle du Missile? Le programmeur doit donc définir quel objet est prioritaire sur le texte. Cela s'effectue grâce à un registre spécial.

**PROGRAMME**

Suite d'instructions formant un tout cohérent destiné à réaliser une tâche précise (par exemple, la gestion d'un répertoire d'adresses. ou un jeu).

# R

**REGISTRE CACHE**

Registre placé au début de la mémoire vive et dont le contenu est recopié dans les circuits électroniques (ANTIC, PIA, POKEY, CTIA) de l'ordinateur à la fin de chaque image (soit 50 fois par seconde en Europe). Le programmeur doit normalement utiliser ces registres plutôt qu'écrire directement dans les registres des circuits (les problèmes de synchronisation sont ainsi évités).

**REGISTRE COULEUR**

Un registre du CTIA (auquel est associé un registre cache en mémoire vive) utilisé pour déterminer la couleur d'un élément de l'écran. Neuf registres couleur sont disponibles dans l'ordinateur ATARI.

**REGISTRE DE POSITION HORIZONTALE**

Registre placé en MEV et dont le contenu détermine la position horizontale du Player ou du Missile auquel il est associé. Cette valeur est mesurée en périodes d'horloge couleur.

**RESOLUTION (SUR 1 LIGNE, SUR 2 LIGNES)**

Dans un Player-Missile, 1 octet (8 bits) définit un rectangle de 8 points élémentaires de large (en simple largeur) et de 1 point élémentaire de haut si la résolution est d'une ligne de balayage. Un octet définit donc 8 points élémentaires. Si la résolution est sur 2 lignes, le même rectangle sera répété immédiatement sous le premier. Un octet définit alors 16 points élémentaires sur l'écran. Dans le premier cas, il faut 256 octets pour définir le Player sur toute sa hauteur et le dessin est très fin. Dans le second cas, 128 octets suffisent mais la résolution est plus grossière.

# S

**SE**

Abréviation de Système d'Exploitation.

**SECTEUR**

La zone utilisable d'une disquette est divisée en 40 pistes concentriques, chaque piste contenant 18 secteurs, 1 secteur contenant 128 octets dont 3 réservés au DOS. Le secteur représente l'unité logique de transfert entre l'ordinateur et l'unité de disquette.

**SED**

Système d'Exploitation de la Disquette. Extension du SE permettant à l'utilisateur d'exploiter une unité de disquette. Le SED n'est pas totalement résident en mémoire morte et il doit être chargé à la mise sous tension de l'ordinateur. Le SED se compose de deux parties appelées DOS.SYS et DUP.SYS sur la disquette. La première (DOS.SYS) est encore appelée FMS (File Management Subsystem) dans ce manuel et se charge à la mise sous tension de l'ordinateur. Elle complète l'amorce de SED figée en MEM. La seconde (DUP.SYS) ne se charge en mémoire que lorsque vous désirez effectuer certaines opérations particulières (effacements ou copies de fichiers par exemple) ; elle se charge lorsque vous tapez «DOS» en Basic ou en Assembleur (attention aux confusions!).

**SEPARATEUR D'ENREGISTREMENT**

Dans le processus d'enregistrement sur magnétocassette, c'est l'ensemble des deux signaux de pré-enregistrement et de fin d'enregistrement, séparant 2 blocs consécutifs de données.



ATARINSIDE

**SIGNAL COULEUR**

Ce signal dont la fréquence de référence se situe à 3,579 MHz contient uniquement l'information couleur de l'image TV. Il est mélangé à un autre signal plus important composé de l'image noir et blanc et des tops de synchronisation. A ces deux signaux est encore ajouté le son et l'ensemble est envoyé vers le téléviseur.

**SIO**

Serial Input/Output. Sous-ensemble du SE responsable des liaisons sur le bus série.

# T

**TABLE DES NOMS DE VARIABLES**

Tous les noms de variables sont regroupés dans une table placée en tête du programme Basic. A chaque variable est associé un pointeur indiquant où se trouve le contenu de la variable dans la mémoire.

**TAMPON D'ENTREE**

Zone tampon placée en mémoire vive de l'adresse $580 à l'adresse $5FF. Les caractères tapés au clavier ou en provenance d'un périphérique transitent par ce tampon.

**TOP DE SYNCHRONISATION**

Signal enregistré sur la piste numérique d'une cassette et pouvant être décodé par le programme. Il peut ainsi exister un bon synchronisme entre la piste audio et le déroulement du programme.

# V

**VARIABLE**

Sorte de boîte qui contient une valeur numérique pouvant être modifiée en cours de programme. On accède au contenu de cette boîte en l'appelant par son nom (devant commencer par une lettre). Exemple : A, XYZ, ATARI sont des noms de variables valides. Dans l'expression A=2, A est une variable recevant la valeur 2.

**VECTEUR**

Deux octets consécutifs en mémoire formant une adresse sur 16 bits. Le 6502 utilise cette quantité par un adressage indirect pour trouver une adresse particulière (début d'un sous-programme ou scrutation d'une table).

# W

**WARMSTART**

Voir DEMARRAGE A CHAUD.



ATARINSIDE

## ANTIC MODES

| ANTIC MODE # | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OS MODE # | 0 | - | - | - | 1 | 2 | 3 | 4 | 5 | 6 | - | 7 | - | 8 |
| DISPLAY TYPE | CHAR | CHAR | CHAR | CHAR | CHAR | CHAR | MAP | MAP | MAP | MAP | MAP | MAP | MAP | MAP |
| SCAN LINES | 8 | 10 | 8 | 16 | 8 | 16 | 8 | 4 | 4 | 2 | 1 | 2 | 1 | 1 |
| COLOR CLOCKS | 4 | 4 | 4 | 4 | 8 | 8 | 4 | 2 | 2 | 1 | 1 | 1 | 1 | 1/2 |
| PIXELS/LINE | 40 | 40 | 40 | 40 | 20 | 20 | 40 | 80 | 80 | 160 | 160 | 160 | 160 | 320 |
| BYTES/LINE | 40 | 40 | 40 | 40 | 20 | 20 | 10 | 10 | 20 | 20 | 20 | 40 | 40 | 40 |
| # OF COLORS | 2.5 | 2.5 | 5 | 5 | 5 | 5 | 4 | 2 | 4 | 2 | 2 | 4 | 4 | 2 |
| BYTES/SCREEN | 960 | 800 | 960 | 480 | 480 | 240 | 240 | 480 | 960 | 1920 | 3840 | 3840 | 7680 | 7680 |

HIGH ORDER INSTRUCTION BITS:
D7: display list interrupt
D6: load memory scan register
D5: vertical scroll enable
D4: horizontal scroll enable

SPECIAL INSTRUCTION CODES:

| | |
|---|---|
| 00-70 | blank 1 through 7 scan lines |
| 01 | JMP: jump (over 1K boundary) |
| 41 | JVB: jump and wait for vblank |

## IOCB STRUCTURE

| RELATIVE ADDRESS | LABEL | DESCRIPTION | SET BY |
|---|---|---|---|
| 0 | ICHID | handler ID | CIO on OPEN |
| 1 | ICDNO | device number | CIO on OPEN |
| 2 | ICCMD | I/O command byte | user |
| 3 | ICSTA | status (errors) | CIO on return |
| 4 | ICBAL | buffer address (low) | user |
| 5 | ICBAH | buffer address (high) | user |
| 6 | ICPTL | put character pointer | CIO |
| 7 | ICPTH | put character pointer | CIO |
| 8 | ICBLL | buffer length/byte count | user/CIO |
| 9 | ICBLH | buffer length/byte count | user/CIO |
| A | ICAX1 | D2=read, D3=write on OPEN | user |
| B | ICAX2 | auxilliary information | user |
| C-F | ICAX3-ICAX6 | auxilliary information | CIO |

## USEFUL OS EQUATES

| LABEL | ADDRESS | DESCRIPTION |
|---|---|---|
| PAGE ZERO | | |
| DOSVEC | 0A | DOS vector |
| DOSINI | 0C | warm start address |
| RTCLOK | 12 | real-time clock, 3 bytes |
| ATTRACT | 4D | attract mode flag |
| DRKMSK | 4E | dark mask for attract |
| COLRSH | 4F | scrambles color for attract |
| SAVMSC | 58 | points to beginning of screen data |
| PAGE TWO | | |
| VDSLST | 200 | display list interrupt vector |
| VPRCED | 202 | proceed line IRQ vector |
| VINTER | 204 | interrupt line IRQ vector |
| VBREAK | 206 | software break (00) IRQ vector |
| VKEYBD | 208 | keyboard IRQ vector |
| VVBLKI | 222 | immediate vblank interrupt vector |
| VVBLKD | 224 | deferred vblank interrupt vector |
| ROM VECTORS | | |
| EDITRV | E400 | E: device handler |
| SCRENV | E410 | S: device handler |
| KEYBDV | E420 | K: device handler |
| PRINTV | E430 | P: device handler |
| CASETV | E440 | C: device handler |
| DISKIV | E450 | D: initialization |
| DISKINV | E453 | disk interface |
| CIOV | E456 | central input-output vector |
| SIOV | E459 | serial input-output vector |
| SETVBV | E45C | routine for setting vectors |
| SYSVBV | E45F | vertical blank routine vector |
| XITVBV | E462 | exit vertical blank routines |
| SIOINV | E465 | serial input-output initialization |
| SENDEV | E468 | send enable routine |
| INTINV | E46B | interrupt handler initialization |
| CIOINV | E46E | CIO initialization |
| BLKBDV | E471 | blackboard mode (memo pad) |
| WARMSV | E474 | warm start entry point |
| COLDSV | E477 | cold start entry point |
| RBLOKV | E47A | cassette read block vector |
| CSOPIV | E47D | cassette open for input vector |

## BIT ASSIGNMENTS FOR SELECTED HARDWARE REGISTERS

| REGISTER | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| AUDCX | distortion | | | volume only bit | volume | | | |
| AUDCTL | 17-bit poly changes into 9-bit poly | clock Ch 1 w/ 1.79 MHz not 64 KHz | clock Ch 3 w/ 1.79 MHz not 64 KHz | clock Ch 2 w/ Ch 1 not 64 KHz | clock Ch 4 w/ Ch 3 not 64 KHz | hi-pass filter in Channel 1 | hi-pass filter in Channel 3 | use 15 KHz not 64 KHz |
| CHACTL | | | | | | vertical reflect | video invert | video blank |
| CONSOL | | | | | speaker | option | select | start |
| DMACTL | | | display list instruction DMA enable | 1=1-line PM 0=2-line PM | enable player DMA | enable missile DMA | playfield | width |
| GRACTL | | | | | | latch TRIG0-3 | enable players | enable missiles |
| IRQEN (IRQST) | BREAK key | other key | serial input data ready | serial output data needed | SOTF | timer 4 | timer 2 | timer 1 |
| NMIEN | display list interrupt | vertical blank interrupt | | | | | | |
| NMIST | " | " | SYSTEM RESET | | | | | |
| PRIOR | GTIA mode selection | | multiple color players | fifth player enable | PF0-PF1 P0-P3 PF2-PF3 | PF0-PF3 P0-P3 --- | P0-P1 PF0-PF3 P2-P3 | P0-P3 PF0-PF3 --- |
| PORTA,B | right | left | back | forward | right | left | back | forward |

# PROGRAMMER'S CARD

| Hardware Register | | | | OS Shadow | | |
|---|---|---|---|---|---|---|
| | | Address | | | Address | |
| Name | Description | Hex | Dec | Name | Hex | Dec |
| ALLPOT | Read 8 line Pot Port State | D208 | 53768 | | | |
| AUDC1 | Audio Channel 1 Control | D201 | 53761 | | | |
| AUDC2 | Audio Channel 2 Control | D203 | 53763 | | | |
| AUDC3 | Audio Channel 3 Control | D205 | 53765 | | | |
| AUDC4 | Audio Channel 4 Control | D207 | 53767 | | | |
| AUDCTL | Audio Control | D208 | 53768 | | | |
| AUDF1 | Audio Channel 1 Frequency | D200 | 53760 | | | |
| AUDF2 | Audio Channel 2 Frequency | D202 | 53762 | | | |
| AUDF3 | Audio Channel 3 Frequency | D204 | 53764 | | | |
| AUDF4 | Audio Channel 4 Frequency | D206 | 53766 | | | |
| CHACTL | Character Control | D401 | 54273 | CHART | 2F3 | 755 |
| CHBASE | Character base address | D409 | 54281 | CHBAS | 2F4 | 756 |
| COLBK | Color-Luminance of Background | D01A | 53274 | COLOR4 | 2C8 | 712 |
| COLPF0 | Color Luminance of Playfield 0 | D016 | 53270 | COLOR0 | 2C4 | 708 |
| COLPF1 | Color Luminance of Playfield 1 | D017 | 53271 | COLOR1 | 2C5 | 709 |
| COLPF2 | Color Luminance of Playfield 2 | D018 | 53272 | COLOR2 | 2C6 | 710 |
| COLPF3 | Color Luminance of Playfield 3 | D019 | 53273 | COLOR3 | 2C7 | 711 |
| COLPM0 | Color Luminance of Player-Missile 0 | D012 | 53266 | PCOLR0 | 2C0 | 704 |
| COLPM1 | Color Luminance of Player-Missile 1 | D013 | 53267 | PCOLR1 | 2C1 | 705 |
| COLPM2 | Color Luminance of Player-Missile 2 | D014 | 53268 | PCOLR2 | 2C2 | 706 |
| COLPM3 | Color Luminance of Player-Missile 3 | D015 | 53269 | PCOLR3 | 2C3 | 707 |
| CONSOL | Console Switch Port | D01F | 53279 | Set to 8 during VBLANK | | |
| DLISTH | Display List Pointer (high byte) | D403 | 54275 | SDLSTH | 231 | 561 |
| DLISTL | Display List Pointer (low byte) | D402 | 54274 | SDLSTL | 230 | 560 |
| DMACTL | Direct Memory Access (DMA) Control | D400 | 54272 | SDMCTL | 22F | 559 |
| GRACTL | Graphic Control | D01D | 53277 | | | |
| GRAFM | Graphics for all Missiles | D011 | 53265 | | | |
| GRAFP0 | Graphics for Player 0 | D00D | 53261 | | | |
| GRAFP1 | Graphics for Player 1 | D00E | 53262 | | | |
| GRAFP2 | Graphics for Player 2 | D00F | 53263 | | | |
| GRAFP3 | Graphics for Player 3 | D010 | 53264 | | | |
| HITCLR | Colission Clear | D01E | 53278 | | | |
| HPOSM0 | Horizontal Position of Missile 0 | D004 | 53252 | | | |
| HPOSM1 | Horizontal Position of Missile 1 | D005 | 53253 | | | |
| HPOSM2 | Horizontal Position of Missile 2 | D006 | 53254 | | | |
| HPOSM3 | Horizontal Position of Missile 3 | D007 | 53255 | | | |
| HPOSP0 | Horizontal Position of Player 0 | D000 | 53248 | | | |
| HPOSP1 | Horizontal Position of Player 1 | D001 | 53249 | | | |
| HPOSP2 | Horizontal Position of Player 2 | D002 | 53250 | | | |
| HPOSP3 | Horizontal Position of Player 3 | D003 | 53251 | | | |
| HSCROL | Horizontal Scroll | D404 | 54276 | | | |
| IRQEN | Interrupt Request (IRQ) Enable | D20E | 53774 | POKMSK | 10 | 16 |
| IRQST | IRQ Status | D20E | 53774 | | | |
| KBCODE | Keyboard Code | D209 | 53769 | CH | 2FC | 764 |
| M0PF | Missile 0 to Playfield Collisions | D000 | 53248 | | | |
| M0PL | Missile 0 to Player Collisions | D008 | 53256 | | | |
| M1PF | Missile 1 to Playfield Collisions | D001 | 53249 | | | |
| M1PL | Missile 1 to Player Collisions | D009 | 53257 | | | |
| M2PF | Missile 2 to Playfield Collisions | D002 | 53250 | | | |
| M2PL | Missile 2 to Player Collisions | D00A | 53258 | | | |

| Hardware Register | | | | OS Shadow | | |
|---|---|---|---|---|---|---|
| | | Address | | | Address | |
| Name | Description | Hex | Dec | Name | Hex | Dec |
| M3PF | Missile 3 to Playfield Collisions | D003 | 53251 | | | |
| M3PL | Missile 3 to Player | D00B | 53259 | | | |
| NMIEN | Non-Maskable Interrupt (NMI) Enable | D40E | 54286 | Set to $40 by IRQ code | | |
| NMIRES | NMI reset | D40F | 54287 | written to by NMI code | | |
| NMIST | NMI Status | D40F | 54287 | read by NMI code | | |
| POPF | Player 0 to Playfield Collisions | D004 | 53252 | | | |
| POPL | Player 0 to Player Collisions | D00C | 53260 | | | |
| P1PF | Player 1 to Playfield Collisions | D005 | 53253 | | | |
| P1PL | Player 1 to Player Collisions | D00D | 53261 | | | |
| P2PF | Player 2 to Playfield Collisions | D006 | 53254 | | | |
| P2PL | Player 2 to Player Collisions | D00E | 53262 | | | |
| P3PF | Player 3 to Playfield Collisions | D007 | 53255 | | | |
| P3PL | Player 3 to Player Collisions | D00F | 53263 | | | |
| PACTL | Port A Control | D302 | 54018 | Set to $3C by IRQ Code | | |
| PAL | PAL/NTSC indictor | D014 | 53268 | | | |
| PBCTL | Port B Control | D303 | 54019 | Set to $3C by IRQ Code | | |
| PENH | Light Pen Horizontal Position | D40C | 54284 | LPENH | 234 | 564 |
| PENV | Light Pen Vertical Position | D40D | 54285 | LPENV | 235 | 565 |
| PMBASE | Player Missile Base Address | D407 | 54279 | | | |
| PORTA | Port A | D300 | 54016 | STICK0,1 | 278,279 | 632,633 |
| PORTB | Port A | D301 | 54017 | STICK2,3 | 27A,27B | 634,635 |
| POT0 | Pot 0 | D200 | 53760 | PADDL0 | 270, | 624 |
| POT1 | Pot 1 | D201 | 53761 | PADDL1 | 271 | 625 |
| POT2 | Pot 2 | D202 | 53762 | PADDL2 | 272 | 626 |
| POT3 | Pot 3 | D203 | 53763 | PADDL3 | 273 | 627 |
| POT4 | Pot 4 | D204 | 53764 | PADDL4 | 274 | 628 |
| POT5 | Pot 5 | D205 | 53765 | PADDL5 | 275 | 629 |
| POT6 | Pot 6 | D206 | 53766 | PADDL6 | 276 | 630 |
| POT7 | Pot 7 (right paddle controller) | D207 | 53767 | PADDL7 | 277 | 631 |
| POTGO | Start POT Scan Sequence | D20B | 53771 | WRITTEN DURING VBLANK | | |
| PRIOR | Priority Select | D01B | 53275 | GPRIOR | 26F | 623 |
| RANDOM | Random number generator | D20A | 53770 | | | |
| SERIN | Serial Port Input | D20E | 53774 | | | |
| SEROUT | Serial Port output | D20D | 53773 | | | |
| SIZEM | Sizes for all missiles | D00C | 53260 | | | |
| SIZEP0 | Size of Player 0 | D008 | 53256 | | | |
| SIZEP1 | Size of Player 1 | D009 | 53257 | | | |
| SIZEP2 | Size of Player 2 | D00A | 53258 | | | |
| SIZEP3 | Size of Player 3 | D00B | 53259 | | | |
| SKCTL | Serial Port Control | D20F | 53775 | SSKCTL | 232 | 562 |
| SKREST | Reset Serial Port Status (SKSTAT) | D20A | 53770 | | | |
| SKSTAT | Serial Port Status | D20F | 53775 | | | |
| STIMER | Start Timer | D209 | 53769 | | | |
| TRIG0 | Joystick Controller Trigger 0 | D010 | 53264 | STRIG0 | 284 | 644 |
| TRIG1 | Joystick Controller Trigger 1 | D011 | 53265 | STRIGL | 285 | 645 |
| TRIG2 | Joystick Controller Trigger 2 | D012 | 53266 | STRIG2 | 286 | 646 |
| TRIG3T | Joystick Controller Trigger 3 | D013 | 53267 | STRIG3 | 287 | 647 |
| VCOUNT | Vertical Line Counter | D40B | 54283 | | | |
| VDELAY | Verical Delay | D01C | 53276 | | | |
| VSCROL | Vertical Scroll | D405 | 54277 | | | |
| WSYNC | Wait for Horizontal Sync | D40A | 54282 | Used by keyboard click routine | | |

ATARINSIDE

# DE RE ATARI®

ANNO DOMINI MCMLXXXI

Le DE RE ATARI est un livre de référence indispensable pour tous les programmes avancés en BASIC ATARI et en ASSEMBLEUR.

Rédigé par des ingénieurs d'ATARI, il permet de comprendre la structure interne des Ordinateurs ATARI, LE BASIC ATARI, ainsi que le système d'exploitation en mémoire morte. Il sert ainsi d'outil d'initiation aux techniques avancées de programmation des Ordinateurs ATARI.

Ce guide traite en profondeur tout un ensemble de sujets aussi fondamentaux que les Player Missiles (objets graphiques), les modes multi-graphiques, le scrolling (effet de rouleau), l'appel des routines en mémoire morte, les interruptions graphiques, etc... Il contient également plusieurs routines et programmes en BASIC et en ASSEMBLEUR.

Le DE RE ATARI est l'outil de base qui permet de programmer avec plus de facilité votre Ordinateur ATARI.

ATARI

P E C E ATARI 9 ... rue Georges Enesco 94008 - CRETEIL CEDEX

ISBN 2-905299-00-2

PRIX : 259 F

ATARINSIDE