pom's

La revue francophone des utilisateurs de l'Apple

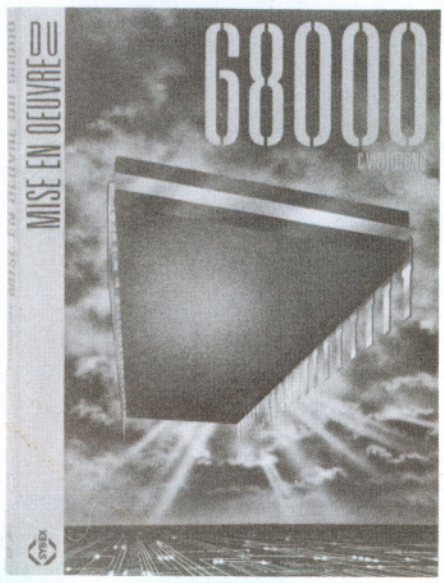
Le Macintosh est un ordinateur ! (voir cahier Mac)

L'Apple en multitâche

Un désassembleur en Applesoft et WPL

CALLS à gogo

Initiation à l'assembleur (6)



NUMERO 16 • PRIX 40 F

ISSN : 0294-6068

M2366-16-40 F

APPLE

SANS MAUX DE TETE

avec les livres P.S.I.

P.S.I.

La pratique de l'Apple II
tome 1
par Nicole Bréaud-Pouliquen
Cet ouvrage présente les spécificités du Basic Applesoft à partir d'une description du matériel et du logiciel du système Apple. Les techniques de programmation, de composition et d'animation de dessins et graphiques colorés y sont expliquées à l'aide d'exemples illustratifs et d'exercices résolus.
128 pages - 85,00 FF

La pratique de l'Apple II
tome 2
par Nicole Bréaud-Pouliquen
Ce second volume de la pratique de l'Apple II est consacré au système d'exploitation disque, à la gestion des fichiers, à l'impression et aux imprimantes, à la carte horloge Appleclock. De nombreux exemples de programmes illustrent les fonctions et les commandes décrites.
120 pages - 85,00 FF

La pratique de l'Apple II
tome 3
par Nicole Bréaud-Pouliquen et Daniel-Jean David
Ce volume est une initiation à la programmation en langage machine 6502 dont le jeu d'instructions est expliqué et utilisé. L'assembleur symbolique et ses logiciels connexes y sont décrits. L'interaction avec le Basic et avec le système y est etudiée.
176 pages - 95,00 FF

Clefs pour l'Apple II
par Nicole Bréaud-Pouliquen
Un livre de référence, dans la collection "Mémentos", destiné à se trouver en permanence à côté de l'ordinateur Apple II. Son but est l'accès rapide à l'information : syntaxe des commandes, codes caractères, messages d'erreurs, codes machine, adresses utiles. Il comporte également un recueil de 25 "trucs" utiles, les "Comment...?".
144 pages - 105,00 FF

L'Apple et ses fichiers
par Jacques Boisgontier
Pour apprendre progressivement la programmation des applications utilisant les fichiers, l'ouvrage commence par une présentation concise et illustrée des commandes du Système d'Exploitation Disque et des instructions du Basic Applesoft. Les instructions des fichiers séquentiels et à accès direct sont ensuite décrites ainsi que leur utilisation. Des méthodes pratiques, souvent mal connues, montrent comment utiliser au mieux des fichiers à accès direct : accès indexé, liste inverse. Une vingtaine de programmes illustrent l'utilisation de ces techniques.
le livre : 176 pages - 95,00 FF
La disquette d'accompagnement : 210,00 FF
Pour Apple II, II plus, IIe - Dos 3.3 - version 48 K ou plus.

Multiplan pour Apple II plus et //e
par Hervé Thiriez
Multiplan est un progiciel qui permet de gérer plusieurs tableaux simultanément ; cet ouvrage sera pour les possesseurs d'ordinateurs Apple II Plus ou //e un véritable guide d'utilisation de Multiplan grâce à des exemples progressifs et à de nombreux cas d'application (gestion de portefeuilles, de copropriété, feuille de paie, impôts, tableaux de bord, etc.).
Le livre : 216 pages - 105,00 FF
La disquette d'accompagnement : 210,00 FF
Pour Apple II plus, IIe - Dos 3.3 - version 64 K ou plus (Disquette maîtresse Multiplan indispensable).

Visicalc sur Apple
par Hervé Thiriez
Après une présentation progressive du modèle Visicalc, l'ouvrage étudie de nombreux cas d'application : feuille d'impôt, gestion de copropriété, paye, facturation..., permettant d'introduire les différentes instructions et astuces d'utilisation.
Le livre : 176 pages - 95,00 FF
La disquette d'accompagnement : 210,00 FF
Pour Apple II, II plus, IIe - Dos 3.3 - version 48 K ou plus (Disquette maîtresse Visicalc indispensable).

Pascal UCSD sur Apple II
par Jacques Rouault et Patrice Girard
L'ordinateur Apple II, le langage Pascal et le système d'exploitation UCSD forment à eux seuls le plus petit ensemble de micro-informatique professionnelle. Une première partie de cet ouvrage est consacrée à l'étude de ces trois éléments. Sont ensuite abordés les programmes de mise en route. Enfin les types et instructions Pascal UCSD sont étudiés en détail, ce qui permettra au lecteur de se rendre compte de la richesse mais aussi de la facilité d'emploi du Pascal.
232 pages - 120,00 FF

Gestion de fichiers et de périphériques pour Apple II / Pascal
par Hervé Haut
Ce livre propose un moyen rapide et facile pour gérer les fichiers et les périphériques sur Apple. De la gestion de bibliothèque à l'utilisation des périphériques, l'auteur propose un ensemble de programmes utilitaires, écrits en Pascal. Un exposé des techniques les plus élaborées permettra d'atteindre le niveau suffisant pour utiliser les méthodes de programmation mises en œuvre.
176 pages - 100,00 FF

Les bases de données sur Apple II
par Michel Keller
L'objet de cet ouvrage est d'aider le lecteur à faire un choix parmi les nombreux logiciels existants sur Apple. Quatre de ces logiciels sont selectionnés ici : PFS et PFS/Report - DB Master - CX BASE 200 - DBASE II. Pour chacun on trouve une description détaillée du logiciel lui-même et e ses procédures de mise en route, de création de fichier, de saisie des données, de maintenance et d'édition. L'auteur termine cette étude par l'exposé des avantages et des inconvénients inhérents à chaque logiciel.
144 pages - 90,00 FF

Microbook : base de données pour Apple II
par Ted Lewis
Microbook est un système de gestion de données et un outil de développement de programmes conçu pour transformer l'Apple II en un outil de classement de fichiers, de collecte et de recherche d'informations, et de traitement de données. C'est un ensemble de programmes écrits en Pascal qui fait de l'ordinateur un véritable bibliothécaire.
Les emplois de Microbook sont multiples : il peut être utilisé pour quasiment toutes les applications où le stockage et la recherche de données est importante. Les programmes étant très longs, il est recommandé d'acheter également la disquette d'accompagnement, proposée en deux versions : l'une compilée, l'autre code source.
Le livre : 248 pages - 145,00 FF

LES DISQUETTES D'ACCOMPAGNEMENT :

Microbook : base de données pour Apple II
par Ted Lewis
volume MICRO :
Apple II, II plus, //e et //c avec système Pascal disquette simple face, compilée, donc non modifiable.
Prix : 210,00 FF

SPÉCIAL PROGRAMMES

Microbook : base de données pour Apple II
par Ted Lewis
volume SOURCE et volume UNIT :
Apple II, II plus, //e et //c, avec système Pascal, disquette 2 faces enregistrées.
Prix : 310,00 FF

Le système Pascal ainsi que les consignes d'utilisation des programmes contenus dans le livre sont indispensables. **Cette disquette existe en deux versions : l'une, double face, destinée aux passionnés de la programmation en Pascal, qui pourront directement programmer dans ce langage, et développer et étendre à leur gré les facultés du système. La seconde, compilée, simple face, destinée aux personnes qui ne connaissent pas la programmation en Pascal ou qui n'ont pas le temps de programmer elles-mêmes, et qui désirent utiliser directement les multiples possibilités de Microbook.**
Tous les programmes proposés dans le livre sont présents dans chacune des disquettes, exceptés les ordres pour imprimante.

POM 1

P.S.I.

Au Maroc
SMER DIFFUSION
3, rue Ghazza
Rabat
MAROC
Tél. : (7) 237.25

Au Canada
SCE Inc.
65, avenue Hillside
Montréal (Westmount)
Québec H3Z1W1 - CANADA
Tél. : (514) 935.13.14

P.S.I. DIFFUSION
BP 86 - 77402 Lagny-S/Marne Cedex
FRANCE
Téléphone (6) 006.44.35
P.S.I. BENELUX
5, avenue de la Ferme Rose
1180 Bruxelles
BELGIQUE
Téléphone (2) 345.08.50
P.S.I. SUISSE
Case postale
Route neuve 1
1701 Fribourg
SUISSE
Tél. : (037) 23.18.28
CCP 17.56.84

Table de conversions en Francs belges et Francs suisses

85,00 FF -	612 FB - 26,90 FS
90,00 FF -	648 FB - 28,40 FS
95,00 FF -	684 FB - 29,90 FS
100,00 FF -	720 FB - 31,50 FS
105,00 FF -	756 FB - 33,00 FS
120,00 FF -	864 FB - 37,60 FS
145,00 FF -	1044 FB - 45,20 FS
DISQUETTES	
210,00 FF -	1620 FB - 66,10 FS
310,00 FF -	2390 FB - 97,00 FS

Envoyer ce bon accompagné de votre reglement à P.S.I. DIFFUSION ou pour la Belgique et le Luxembourg à P.S.I. BENELUX ou pour la Suisse à P.S.I. SUISSE

DESIGNATION	NOMBRE	PRIX
	TOTAL	

par avion : ajouter 8 FF (75 FB) par livre
Pour la SUISSE frais de port sur tout envoi : 1.50 FS

Signature (obligatoire pour paiement par carte de crédit)

Paiement par cheque joint ☐ Paiement en FF par carte bleue VISA (à P.S.I. DIFFUSION uniquement) paiement supérieur à 50 FF

☐ Je désire recevoir le catalogue P.S.I. gratuit ;

N ____ Date d'expiration ____

NOM ____ PRENOM ____

rue ____ n° ____

Code postal ____ Ville ____

AG94PH

pom's n°16

JANVIER-FEVRIER 1985

Sommaire

Les annonceurs

Éditions MEV - 49, rue Lamartine - 78000 Versailles

Directeur de la publication : Hervé Thiriez. Imprimerie Rosay, 94300 Vincennes. Imprimé en France. Dépôt légal : 1er trimestre 1985.

Editorial

L'environnement Apple est en évolution constante : nous nous sommes à peine habitués au //c, au Macintosh et au Mac 512 Ko que l'on voit poindre à l'horizon le fameux Apple //x et la "norme //c".
En effet, comme vous pourrez le voir dans la rubrique Micro-Informations, l'intention d'Apple est d'établir le //c comme norme de la famille 6502, avec un modèle haut de gamme, le //x.

Les possesseurs de //c ont une occasion de se réjouir : des essais systématiques réalisés à Pom's nous ont montré que tout ce qui a été publié, depuis le numéro 1, fonctionne sur le //c. En ce qui concerne les programmes écrits en Pascal UCSD, il est bon de vérifer toutefois, avant toute utilisation, que votre version du système n'est pas trop ancienne pour "booter" correctement sur le //c (reconnaissance des 80 colonnes).

Malgré des retards généraux par rapport aux dates de sortie annoncées au Sicob, nous commençons à voir apparaître de nombreux logiciels pour le Macintosh, avec un bon pourcentage de produits français, dont certains n'ont rien à envier aux créations américaines. Par contre, l'arrivée des périphériques de toutes sortes n'est pas particulièrement rapide.

Si vous possédez uniquement un Macintosh, voici quelques précisions concernant les abonnements à Pom's. Seul l'abonnement sans disquette peut vous intéresser, puisque les disquettes comprises dans les abonnements sont destinées aux Apples II et //. Il existe par ailleurs des disquettes d'accompagnement Macintosh, vendues séparément et identifiées comme telles dans les bons de commande.
La disquette Mac No 1 est désormais complète. Elle comprend tous les programmes originaux publiés dans les numéros 14 - 15 - 16, plusieurs polices de caractères (Cairo, Los Angeles, Mos Eisley, Manhattan, Hollywood), Copy Disk avec un seul drive et le programme "Localizer".
La disquette No 2, qui s'ajoutera au catalogue Mac, commencera donc à partir du Pom's 17.

Vos contributions concernant le Macintosh sont, malheureusement, encore trop peu nombreuses. Nous les recevrons et testerons avec plaisir.
A ce propos, et pour allécher les intéressés, signalons que les capacités du nouveau Mac 512 Ko sont plus en rapport avec les performances du processeur 68000 que celles de son petit frère. Il permet, par exemple, de disposer de quelques 389 Ko avec le Basic et de constituer, si le coeur vous en dit, des tableaux de variables de plus de 64 Ko...

Pour en revenir à la "ligne 6502", certains lecteurs semblent redouter que nous délaissions l'Apple II pour le //e. Bien heureusement il n'en est rien. Nous nous efforçons toujours de publier des programmes compatibles Apple II - //e ou //c (mis à part les problèmes éventuels liés à l'utilisation des touches particulières du clavier //e - //c). L'Apple II est toujours d'actualité : n'ayez crainte nous ne vous laisserons pas tomber !

Hervé Thiriez

Photo de couverture : "Mise en œuvre du 68000" aux éditions Sybex.

Ont collaboré à ce numéro : Alexandre Avrane - Jean-Luc Bazanegue - Alexandre Duback - Jean-Michel Gourévitch - J. Gunther - Roland Jost - Thierry Han - Guy d'Herbemont - Yvan Kœnig - Jérôme Leclerc - Gérard Michel - Patrice Neveu - Eric Pascual - Jean-François Rabasse - François Sermier - Marianne Sutz. **Rédacteurs :** Alexandre Avrane - Gérard Michel. **Directeur de la publication, rédacteur en chef :** Hervé Thiriez. **Dessins :** Laurent Bidot.
Siège sociale et abonnements : Editions MEV - 64-70, rue des Chantiers - 78000 Versailles - Tél. : (3) 951.24.43.
Rédaction et courrier des lecteurs : 59, bd de Glatigny - 78000 Versailles.
Régie publicitaire : Force 7 - Anne Jourdan - 5 place du Colonel Fabien - 75010 Paris - Tél. : (1) 240.22.01.
Diffusion N.M.P.P. : Sophie Marnez - Tél. : (1) 240.22.01.
Composition : Télécompo - 13-15, avenue du Petit Parc - 94300 Vincennes - Tél 328.18.63.
Impression : Rosay - 47, avenue de Paris - 94300 Vincennes - Tél. : 328.18.63.


Macintosh
MacTell
4 ACHATS PAR CO
5 INFORMATIONS ANNUAIRE ELECTRONIQUE
→ 6 MIEUX CONNAITRE TELETEL
→ 7 ANNUAIRE DES SERVICES
ou un nom .....
mode d'emploi
recherche terminée
ENVOI
GUIDE
C/FIN
Retour
Suite
1 2 3
4 5 6
7 8 9
* 0 #
Sommaire
Guide
Inverser
Répétition
iregistrer
un an après APPLETELL (1000 cartes installées en 8 mois - Quatre Pommes d'Or)
Le Minitel automatique sur Mac
• procédures au to ma tiques d'interrogation des serveurs
• enregistrement et impression des écrans
1 600 F H.T.
Frédéric Lévy
HELLO INFORMATIQUE - Tél.: (1) 523.30.34
ANNUAIRE ELECTRONIQUE - CALVADOS - COMPUSERVE -
THE SOURCE - QUESTEL - DOW JONES - MISSIVE - KIOSQUE -
(tous les serveurs ASCII et vidéotex)

Pot-Pourri ProDOS

Alexandre Avrane

Les informations sur ProDOS commencent à devenir abondantes, ainsi d'ailleurs que les lettres des lecteurs de Pom's demandant plus d'information sur le nouveau système d'exploitation de l'Apple II.

Je vous propose donc aujourd'hui de compléter et détailler mon article paru dans le Pom's 13. Trois principaux sujets seront abordés, les indications fournies par la lecture de l'article précédent sur ProDOS étant supposées connues :
1-divers compléments, donnés sans ordre prémédité, concernant les questions les plus fréquentes;
2-un programme de conversion de fichiers à accès direct DOS 3.3 / ProDOS;
3-la méthode d'initialisation des utilitaires système tournant dans un environnement ProDOS (il ne s'agit plus, comme par le passé, de créer des programmes égoïstes s'ignorant mutuellement, avec les problèmes de contention mémoire qui en résultent).

La documentation ProDOS

Apple diffuse quatre principaux manuels sur ProDOS :

1. Le "ProDOS User's Manual" explique l'utilisation des différents utilitaires livrés avec la disquette système ProDOS (Filer, Convert, etc.).
2. "Basic Programming with ProDOS" est le manuel de référence des commandes accessibles sous Applesoft, telles que CATALOG, READ ou CREATE. De nombreux lecteurs nous ont écrit pour nous signaler qu'ils n'avaient pas reçu ce manuel, pourtant nécessaire à l'utilisation de ProDOS (et tout particulièrement indispensable aux nouveaux utilisateurs de l'Apple //c qui ignorent tout du DOS 3.3). Ce manuel est apparemment commercialisé séparément, avec la disquette contenant APA (Applesoft Programmer's Assistant) qui offre des utilitaires de renumérotation et fusion de programmes.
3. "The ProDOS Technical Reference Manual" est destiné aux programmeurs en assembleur désirant créer des modules appelant directement ProDOS. Bien entendu, le fonctionnement interne de ProDOS n'est pas expliqué; seuls les points d'entrée, et la manière de les appeler, sont étudiés.
4. Enfin, "ProDOS Assembler Tools" explique le fonctionnement de l'assembleur 6502 (commercialisé séparément), ainsi que celui de l'utilitaire de mise au point Bugbyter (voir Pom's 13).

Ces manuels sont naturellement disponibles en français. On peut s'étonner que seul le premier soit livré avec ProDOS. En fait, comme cela avait été auparavant le cas avec le Dos Tool-Kit, Apple préfère proposer des produits complémentaires plutôt que de les fournir systématiquement.

Démarrage d'une disquette ProDOS

Voici, donné plus en détail, le mécanisme du "boot" sous ProDOS :

1. Lors du "boot", lancé par l'allumage de l'Apple ou des commandes telles que PR#6, 6<Ctrl-P> ou C600G, le processeur 6502 commence par exécuter la routine contenue dans la ROM de la carte du contrôleur d'interface (s'il s'agit du slot 6, elle est située en $C600 - $C6FF). Cette routine ignore tout de la disquette qu'elle va charger; il peut s'agir d'une disquette DOS 3.3, ProDOS, Pascal, CP/M, Forth ou même d'un programme protégé. Elle charge un ou plusieurs secteurs de la piste 0; pour ProDOS, les quatre premiers secteurs, c'est-à-dire les deux premiers blocs, sont placés en mémoire à partir de l'adresse $0800. Enfin, le processeur 6502 effectue un saut en $0801.

2. Premier travail à effectuer : "Où suis-je ?". Le programme commençant en $0801, extrait d'une disquette ProDOS, peut avoir été chargé par un Apple II ou un Apple ///. Après examen des ROMs du moniteur, celui-ci charge alors à l'adresse $2000, soit le fichier ProDOS pour un Apple II, soit le fichier SOS pour un Apple ///. S'il ne peut trouver le fichier recherché, le programme s'arrête avec un message d'erreur. A l'avenir, nous ne nous occuperons que de ProDOS sur un Apple II.

3. Le fichier ProDOS contient le M.L.I. (Machine Language Interface) de ProDOS, c'est-à-dire son noyau (ou Kernel). A l'origine du développement de ProDOS, celui-ci devait pouvoir fonctionner sur un Apple II de 48k en se plaçant sous l'adresse $C000, ainsi que sur un //e de 64k en se plaçant sur la carte langage. Quelles qu'en soient les raisons, une seule version sur carte langage a été commercialisée, et cette troisième étape reloge donc ProDOS sur l'extension de 16k et place HIMEM (l'adresse de la plus haute mémoire disponible) à $BF00. Petite digression avant de poursuivre : il doit être possible de placer ProDOS sur une extension mémoire située ailleurs que sur le slot 0 (cela permettrait l'utilisation du Basic Integer sous ProDOS). Si un lecteur a réalisé une telle adaptation...

4. ProDOS mis en place, il faut continuer le "boot". Le MLI va donc charger et exécuter le premier fichier qu'il trouve sur le catalogue principal de la disquette dont le nom est de la forme XXX.SYSTEM. Dans la majorité des cas, il s'agit de BASIC.SYSTEM qui permet l'interface entre un programme Applesoft et ProDOS. Un autre programme qui répond à cette convention peut être appelé, par exemple un programme professionnel de traitement de texte en langage machine. Si aucun fichier correct n'est trouvé, on s'arrête là avec un message d'erreur.

5. Tout fichier système se charge à l'adresse $2000 et doit assurer lui-même son transfert à une adresse définitive, dépendant de la position précédente de HIMEM. Ainsi le BASIC.SYSTEM se reloge à partir de l'adresse $9600 et fixe HIMEM à cette valeur.

6. S'il s'agit de BASIC.SYSTEM, il recherche, toujours sur le catalogue principal, un fichier s'appelant STARTUP. Une fois trouvé, celui-ci est chargé et exécuté, quel que soit son type (Applesoft, assembleur ou EXEC).

Voilà qui achève le "boot" d'une disquette ProDOS; il faut retenir principalement que l'ordre des fichiers système sur le catalogue principal est d'importance capitale.
Petite parenthèse avant de poursuivre : il serait bon de pouvoir formater une disquette ProDOS sans devoir obligatoirement utiliser l'utilitaire système Filer. Si un lecteur a réussi à séparer la routine de formatage de l'ensemble du programme, qu'il nous fasse signe...

Compléments sur les commandes ProDOS

Tout d'abord une petite rectification par rapport à l'article de Pom's 13 : la commande CATALOG fournit un affichage sur 80 colonnes, et CAT sur 40 colonnes, et non l'inverse.

Voici, dans un désordre savamment dosé, quelques compléments :

– CHAIN permet, comme RUN, l'utilisation du paramètre @ pour désigner le numéro de la première ligne d'exécution. Attention : un tableau défini dans le premier programme ne peut être redimensionné dans le second.

– EXEC comporte le paramètre F (field) permettant de spécifier le numéro de champ initial de l'exécution. Il remplace le paramètre R (record) du DOS, fort peu utilisé d'ailleurs.

– BLOAD et BRUN comportent les paramètres B, E et L : B (beginning), E (end) ou L (length) permettent de choisir le chargement d'une portion du fichier, au lieu de son intégralité (l'utilisation de E ou L est exclusive).

– BLOAD et BSAVE peuvent charger ou sauvegarder un fichier de type non-binaire, si on utilise le paramètre T (type). Ce même paramètre est utilisable par OPEN et APPEND pour ouvrir un fichier non-texte. De plus le paramètre L de BSAVE est étendu jusqu'à $FFFF.

– READ et WRITE peuvent utiliser les paramètres F (field) et B (byte) pour débuter leur action plus loin que la position courante. Cette extension rend inutile la commande APPEND qui est néanmoins conservée pour préserver la compatibilité avec les programmes écrits pour le DOS.

– Les routines en assembleur spécifiées par les commandes IN# A$2000 ou PR# A$300, qui interceptent les entrées/sorties vers un slot donné, doivent commencer par l'instruction 6502 CLD (CLear Decimal), afin de permettre à ProDOS de vérifier leur existence; il s'agit d'un contrôle supplémentaire de sécurité qui évitera certains blocages clavier / écran.

– APPEND peut être utilisé désormais dans un fichier à accès direct.

Malgré son allocation dynamique de l'espace disque, ProDOS possède la même limitation que le DOS 3.3 : tout fichier re-sauvé avec une taille inférieure à sa taille initiale conserve l'intégralité de ses blocs. Il sera donc encore nécessaire d'effectuer régulièrement un test de compression sur les fichiers.
Nous verrons dans un prochain numéro la méthode à utiliser pour rajouter de nouvelles instructions à ProDOS ainsi que, plus généralement, la programmation système qui doit suivre un certain nombre de règles.

Quelques précisions sur le chemin d'accès d'un fichier.
Pour accéder à un fichier sur un volume, ProDOS concatène trois informations :
– le PREFIX, dont la valeur correspond initialement au nom du volume de démarrage,
– le chemin d'accès réduit éventuellement précisé dans la commande,
– le nom du fichier.
Ainsi pour un PREFIX /DISK.POMS/N01/, si l'on émet l'instruction : LOAD UTIL / DIRECT. CONVERT, ProDOS ira chercher : /DISK. POMS / NO1 / UTIL / DIRECT. CONVERT.
Nota : si les paramètres Slot et Drive sont également précisés mais incompatibles, ils seront ignorés.

Voici, pour terminer l'étude des commandes ProDOS / BASIC.SYSTEM, leur syntaxe : (pn = pathname = chemin d'accès réduit + nom de fichier)

APPEND pn [,Ll] [,Ss] [,Dd]
BLOAD pn [,Aa] [,Bb] [,Ll/Ee] [,Tt] [,Ss] [,Dd]
BRUN pn [,Aa] [,Bb] [,Ll/Ee] [,Ss] [,Dd]
BSAVE pn ,Aa ,Ll/Ee [,Bb] [,Tt] [,Ss] [,Dd]
CAT p
CATALOG p
CHAIN pn [,@n] [,Ss] [,Dd]
CLOSE [pn]
CREATE pn [,Tt] [,Ss] [,Dd]
DELETE pn [,Ss] [,Dd]
EXEC pn [,Ff] [,Ss] [,Dd]
FLUSH [pn]
IN# [s] [,] [Aa]
LOAD pn [,Ss] [,Dd]
LOCK pn [,Ss] [,Dd]
OPEN pn [,Ll] [,Ss] [,Dd]
POSITION pn [,Ff] [,Bb]
PREFIX p [,Ss] [,Dd]
PR# [s] [,] [Aa]
READ pn [,Rr] [,Ff] [,Bb]
RENAME pn1,pn2 [,Ss] [,Dd]
RESTORE pn [,Ss] [,Dd]
RUN pn [,@n] [,Ss] [,Dd]
SAVE pn [,Ss] [,Dd]
STORE pn [,Ss] [,Dd]
UNLOCK pn [,Ss] [,Dd]
WRITE pn [,Rr] [,Ff] [,Bb]
– pn [,Ss] [,Dd]

Note: dans tous les cas, le paramètre V (volume) du DOS 3.3 est autorisé mais ignoré.

Et les codes erreurs possibles (accessibles par PEEK(222) si ONERR est actif) :

02 Range error
03 No device connected (ROM non trouvée sur le slot spécifié)
04 Write protected
05 End of data
06 Path not found
07 Path not found
08 I/O error
09 Disk full
10 File locked
11 Invalid option
12 No buffers available (maximum de 8 fichiers ouverts contre 16 sur DOS 3.3)
13 File type mismatch
14 Program too large
15 Not direct command
16 Syntax error (code identique au Basic, seul manque le '?' initial)
17 Directory full (catalogue principal uniquement)
18 File not open
19 Duplicate filename (avec CREATE ou RENAME)
20 File busy (si CAT, CATALOG, DELETE ou RENAME d'un fichier ouvert)
21 File(s) still open (CLOSE impératif avant la fin de programme)

Modifications des commandes Applesoft

Trois points essentiels méritent d'être soulignés :

– Il est possible d'utiliser la commande HIMEM, mais de manière à ce que celui-ci se trouve sur une page mémoire de 256 ($100) octets. En effet, ProDOS alloue dynamiquement les buffers de fichiers en examinant la valeur actuelle de HIMEM.

– L'instruction TRACE de l'Applesoft fonctionne sans problème sous ProDOS (en fait celui-ci utilise le TRACE pour contrôler le programme en cours !).

– Etrangement, la commande FRE qui permettait un nettoyage très rapide de la mémoire semble avoir disparu de la version commercialisée de ProDOS. De même, deux améliorations importantes de l'Applesoft, qui étaient prévues dans les spécifications originelles de ProDOS, ont disparu : d'une part la commande INPUT devait pouvoir accepter la saisie des caractères de ponctuation telles que la virgule ou les deux-points; d'autre part ProDOS devait, à l'interception d'une instruction HGR ou HGR2, libérer l'espace mémoire correspondant pour les programmes Basic. Apparemment ces routines ont dépassé le quota d'octets qui leur avait été alloué, d'où leur suppression pure et simple.

Utilisation avec carte 80 colonnes étendue

Caractéristique encore mal connue de ProDOS, celui-ci détecte automatiquement la carte 80 colonnes étendue. Qu'en fait-il ? Il crée un disque virtuel dans l'espace mémoire auxiliaire de 61K, accessible par le préfixe /RAM. Le disque créé contient 128 blocs de 512 octets arrangés ainsi :

Blocs 0-1 : non disponibles
Bloc 2 : catalogue principal pour 12 fichiers
Bloc 3 : carte du volume
Blocs 4-7 : non disponibles
Blocs 8-127 : sous-catalogues et fichiers.

Ce volume, considéré de manière interne comme situé au Slot 3, Drive 2, ne peut cependant être référencé que par son préfixe. Bien entendu, les informations qui y sont contenues disparaissent lorsque se retire le courant électrique !

Conversion DOS/ProDOS de fichiers à accès direct

L'utilitaire Convert de la disquette système ProDOS ne permet pas de convertir automatiquement les fichiers à accès direct. Deux difficultés majeures empêchent, en effet, une conversion automatique :
1. Il est nécessaire de spécifier la longueur d'un enregistrement. Cette valeur, (bien que mémorisée de manière interne par ProDOS), doit impérativement être indiquée au système d'exploitation afin qu'il reconnaisse un fichier à accès direct.
2. Les informations du fichier peuvent être clairsemées (sparse file ou "fichier creux") : certains enregistrements ont été écrits, d'autres ne l'ont jamais été et contiennent donc des octets à zéro binaire qui provoquent, à la moindre tentative de lecture, un irrémédiable "End of Data Error".

Quelques rappels sur les fichiers à accès direct peuvent être utiles :

1. Un fichier à accès direct est un ensemble d'enregistrements de même longueur. Ceux qui ont déjà été écrits se terminent par un caractère retour-chariot : un enregistrement de longueur n contient n-1 caractères significatifs suivis d'un Return. Certains fichiers à accès direct ont cependant une structure moins orthodoxe et contiennent des enregistrements à n caractères significatifs : il est alors nécessaire de les lire par une série de GETs et non par un INPUT.

2. Chaque enregistrement (record) est composé de 1 à plusieurs champs (field). Ceux-ci peuvent être de longueur variable, à la condition que la somme de leur longueur soit constante. Les champs sont séparés entre eux, soit par un retour-chariot, soit par une virgule, selon l'humeur du programmeur.

3. Sous DOS 3.3, le nombre de secteurs d'index d'un fichier ne peut dépasser la taille d'une disquette; le programme :

```
100 PRINT D$"OPEN FICHDIR,L"512
110 PRINT D$"DELETE FICHDIR"
110 PRINT D$"WRITE FICHDIR,R"200*122
120 PRINT "A"
130 PRINT D$"CLOSE FICHDIR"
```

sature quasi-complètement les 140 kilo-octets disponibles ... après plus de 3 minutes d'exécution. Le même programme lancé sous ProDOS n'alloue que 6 blocs (3 kilo-octets), avec un gain comparable en vitesse.
Sous ProDOS, un fichier de 16 méga-octets, peut donc résider sur une disquette de 140K, pourvu que seuls quelques enregistrements aient été écrits.
Deux remarques :
– Sous ProDOS, il est nécessaire d'ajouter la ligne : 105 PRINT D$"CLOSE FICHDIR" car tout fichier ouvert doit dorénavant être explicitement fermé.
– Le programme CONVERT se plante lamentablement à la lecture du catalogue d'une disquette Dos 3.3 contenant FICHDIR ! Prudence donc.

4. En règle générale, l'enregistrement numéro zéro est utilisé de manière particulière; il contient souvent le nombre d'enregistrements du fichier.

Méthodes de conversion

Revenons à notre objectif de conversion de fichiers directs. Trois méthodes peuvent être envisagées :

1. Conversion du fichier direct en fichier séquentiel, puis conversion de ce dernier par Convert, enfin reconversion du résultat en fichier direct. D'où un total de 3 à 4 disquettes dont une 1 à 2 intermédiaires, donc beaucoup de manipulations et beaucoup de temps, plus la nécessité que tous les enregistrements aient été écrits au préalable, ne serait-ce que par un simple Return. C'est la méthode préconisée jusqu'à présent.

2. Re-formatage logique de la disquette DOS 3.3 en disquette ProDOS (ou vice-versa), sans altérer le contenu du ou des fichiers. C'est la méthode la plus rapide mais également la plus hasardeuse et complexe, compte tenu des systèmes différents d'allocation des blocs et secteurs.

3. Lecture et écriture simultanée de chaque enregistrement par l'utilisation conjointe en mémoire des deux systèmes d'exploitation. Cette méthode n'utilise aucune disquette intermédiaire, n'impose qu'un seul accès à chaque enregistrement, et est relativement facile à programmer en Applesoft, lorsque les problèmes de cohabitation mémoire sont résolus.

Cette dernière méthode a été retenue pour le programme CQFD (pour Conversion Qualifiée de Fichiers Directs !), qui impose la disponibilité de deux lecteurs de disquettes (ou un lecteur et un disque dur, pourquoi pas ?), ce qui ne devrait pas être une contrainte importante, les fichiers à accès direct étant généralement utilisés dans un environnement quasi-professionnel. En revanche, les enregistrements peuvent ne pas avoir été initialisés, les erreurs "End of Data" étant interceptées et gérées par le programme.

Utilisation

Une fois mis en place, CQFD convertit automatiquement tout fichier direct, selon différents paramètres modifiables, et propose des valeurs par défaut : – nom des fichiers origine et destination;
– enregistrements terminés ou non par un Return;
– longueur totale d'un enregistrement (y compris l'éventuel Return);
– nombre de champs séparés par un Return dans chaque enregistrement;
– numéro de début et fin des enregistrements à convertir;
– et, bien sûr, le sens du transfert : DOS vers ProDOS ou l'inverse.

L'utilisateur a la possibilité de donner des commandes directes DOS 3.3 ou ProDOS sans sortir du programme afin, par exemple, de modifier les numéros de slot et drive du Dos ou le Prefix ProDOS. D'autre part, afin de minimiser le temps d'exécution, les enregistrements ne sont pas lus et écrits un par un, mais par bloc de plusieurs dizaines à plusieurs milliers selon la capacité mémoire disponible.

"CQFD" se compose de trois parties :
– le programme Applesoft CQFD;
– le module utilitaire en assembleur CQFD.B (source CQFD.B.S en Big Mac);
– le fichier CQFD.DOS.16K (voir plus loin).
Tous trois doivent être placés sur une disquette ProDOS (ils peuvent être transférés sans problème par Convert). Après avoir "booté" sous ProDOS, on lance le programme par un simple RUN CQFD.

Dix messages d'erreur peuvent être générés par le programme; il est bon de les détailler :

– Fichier non précisé : il faut saisir les paramètres de conversion avant de demander le transfert.
– Fin brutale de fichier : la fin du fichier a été rencontrée. Les enregistrements lus à zéro binaire ne provoquent pas cette erreur; en revanche, la lecture d'un fichier tellement clairsemé (creux) que certains blocs de données n'ont même pas été alloués par le système d'exploitation engendrera cette erreur. Il est alors nécessaire de définir une plage moins étendue d'enregistrements.
– Erreur Système : une (improbable ?) erreur dûe au programme.

Les autres messages d'erreur s'expliquent par eux-mêmes. L'interruption par CTRL-C est gérée par le programme; en revanche, le Reset agit comme à l'habitude.

Démonstration du programme

Deux fichiers directs de démonstration (présents sur la disquette d'accompagnement) peuvent être utilisés pour observer le fonctionnement du programme :
– CQFD.TEST1 contient quinze enregistrements (0 à 14) de longueur unitaire 24 contenant chacun trois champs de 7 caractères suivis d'un Return.
– CQFD.TEST2 contient quatre enregistrements, numérotés 1, 19, 33 et 73 de longueur unitaire 8, contenant un champ dont les 8 caractères sont significatifs.

Pour vérifier le bon fonctionnement, il suffit de convertir un fichier dans un sens puis dans l'autre (sous un nom différent !) puis de comparer le fichier original avec le fichier résultat, soit par un des utilitaires disponibles (le Filer de ProDOS par exemple), soit par la commande TDUMP (Pom's 12 p.28).

Fonctionnement interne

Le programme repose sur la présence simultanée en mémoire de ProDOS et du DOS 3.3, d'où un premier problème : où les placer ? ProDOS se loge, dans sa version commercialisée, sur la partie haute de la mémoire. En revanche le DOS est relogeable et une disquette Master peut le placer en 16K, 32K ou 48K, selon la configuration mémoire de l'Apple.

A l'évidence, la version 16K s'imposait; néanmoins comment obtenir une version assemblée en 16K lorsque l'on ne possède que des Apple de 48K ou plus ? Le petit programme CQFD.DOS.16K.CREATE, listé plus loin, génère le fichier CQFD.DOS.16K qui est l'image instantanée du DOS issu d'une disquette Master avant qu'il ne se reloge en 48K et charge le programme de HELLO. Ce programme n'est destiné qu'aux lecteurs de Pom's qui n'ont pas la disquette de ce numéro. Je n'en détaillerai pas le fonctionnement car il est bien peu orthodoxe...

Bref, vous avez les 3 fichiers CQFD, CQFD.B et CQFD.DOS.16K que vous avez transférés vers une disquette ProDOS et vous lancez le programme CQFD. Celui-ci charge les deux autres fichiers (lignes 6000) et fixe LOMEM et HIMEM (ligne 150) afin d'obtenir l'allocation mémoire suivante :
$0000-$07FF : mémoire partagée (fichier CQFD.B en $0300)
$0800-$1AA5 : programme Applesoft
$1AA6-$1CFF : buffer Dos 3.3
$1D00-$3FFF : Dos 3.3 en version 16K
$4000-$95FF : variables Applesoft
$9600-$97FF : buffer de fichier ProDOS
$9800-$FFFF : ProDOS + Applesoft + Moniteur sur Rom et carte langage

Le DOS 3.3 est démarré à froid pour qu'il s'initialise sans effacer le programme puis on lui fait comprendre que, pour une fois, les variables du Basic se situeront au-dessus de lui. Simultanément, on indique à ProDOS que les routines d'entrée / sortie seront gérées par le Dos (ligne 6500). Maintenant qu'ils ont fait connaissance, on obtient la hiérarchie suivante pour l'interception des entrées / sorties :
1-en premier le DOS
2-puis ProDOS
3-qui renvoie à l'Applesoft
4-et finalement c'est le Moniteur qui travaille !

Un petit POKE permet de définir CHR$ (2) comme caractère de réveil du DOS, CHR$ (4) restant dirigé vers ProDOS (ligne 3000). Tout serait parfait dans notre petite famille... malheureusement, à chaque erreur, les deux systèmes d'exploitation se montrent insupportablement égoïstes et tentent de se reconnecter en tête de la hiérarchie; or ce ne sont pas les erreurs qui manquent, ne seraient-ce que les "End of Data" à la lecture des enregistrements vides ! Il faut alors patiemment reconstruire la pyramide; c'est le rôle du module CQFD.B qui comprend plusieurs parties :
– redirection des entrées / sorties entre DOS et ProDOS;
– initialisation à froid du DOS sans perte ni arrêt du programme;
– routine fixant le bug du traitement d'erreur de l'Applesoft;
– routines de saisie générale : & INPUT accepte les virgules et autres caractères insolites; & GET permet la lecture de plusieurs caractères et leur concaténation pour former une variable Applesoft (syntaxe : & GET <nom variable>, <longueur variable>);
– interception des erreurs de lecture du DOS car celui-ci (bug que j'ignorais !) "oublie" la longueur des enregistrements d'un fichier direct après un "End of Data".

Afin d'accélérer le programme de transfert, les enregistrements sont lus et écrits par bloc. La taille du bloc (déterminée en ligne 753) permet une utilisation optimale de la mémoire vive disponible (compte tenu des besoins de ProDOS pour ses buffers).
En corollaire le programme utilise un système d'allocation et libération dynamiques pour le tableau A$(M,N) contenant tous les champs de tous les enregistrements du bloc en cours : quelques PEEKs et POKEs (lignes 1010, 1070, 1237, 1240) suffisent, pourvu que le tableau soit la dernière variable Applesoft déclarée dans le programme.

Quelques remarques générales avant de terminer :

1. Le programme peut être modifié dans la mesure où il ne dépasse pas la dizaine d'octets restant disponibles avant le buffer du DOS (on peut éliminer les remarques); néanmoins, attention à certaines instructions d'apparence anodine (telles que l'INPUT de la ligne 6060) qui sont néanmoins nécessaires pour l'initialisation et l'équilibre du système.

2. L'allocation de l'espace disque est moins efficace sous DOS 3.3; vous pouvez donc obtenir des erreurs "DISK FULL" en convertissant de ProDOS vers DOS, et à plus forte raison à partir d'un gros fichier sur disque dur.

3. En sortant du programme, vous avez la possibilité de conserver les deux systèmes d'exploitation actifs. La hiérarchie décrite ci-dessus reste valable mais les interceptions ne sont pas prévues pour une utilisation directe des commandes : essayez (c'est sans risque) et comparez ! Pour activer un système ou l'autre, allez sous le Moniteur et tapez sur la même ligne :
- 0 [ctrl-P] 0 [ctrl-K] BE00G pour ProDOS
- 0 [ctrl-P] 0 [ctrl-K] 1DBFG pour DOS

Deux observations :
1) sous DOS ou l'Applesoft, l'appui sur la touche Return provoque un double saut de ligne, sous ProDOS un seul saut est généré.
2) N'oubliez pas que les commandes PR#, IN#, LOAD et SAVE sont à la fois des instructions DOS, ProDOS et Applesoft... lequel des trois réagit en premier ?

■

```
1   REM ** CQFD.DOS.16K.CREATE **
2   REM AA/251084/V1
10  D$ =  CHR$ (4)
20  F$ = "CQFD.DOS.16K"
100   IF  PEEK (104) = 8 THEN  POKE 104,64
      : POKE 16384,0: PRINT D$"RUN CQFD.
      DOS.16K.CREATE"
110   TEXT : HOME : PRINT "GENERATION DOS
      3.3 SUR ENVIRONNEMENT 16K"
120   PRINT : PRINT : INPUT "INSEREZ UNE D
```

```
       ISQUETTE MAITRE (AVEC DOS  RELOGEA
       BLE) EN SLOT 6, DRIVE 1 -->";Z$
130 A$ = "300:A0 0 B9 0 C6 99 0 96 88 D0
       F7 A9 60 8D F8 96 20 0 96 A9 60 8D
        4A 8 20 1 8 A9 60 8D 47 37 4C EA
       3": GOSUB 700
135 A$ = "230<300.3CFM": GOSUB 700
150  CALL 560
155  POKE 49384,0
160  HOME : PRINT "INSEREZ UNE DISQUETTE
       RECEPTRICE DU","FICHIER "F$" EN SL
       OT 6, DRIVE 1:";: INPUT "";Z$
170  PRINT D$"BSAVE"F$",A$1D00,L$2300,S6,
       D1,V0"
180  PRINT "FICHIER GENERE.": PRINT D$"FP
       "
700 A$ = A$ + " ND9C6G": FOR I = 1 TO  LE
       N (A$): POKE 511 + I, ASC ( MID$ (
       A$,I,1)) + 128: NEXT : POKE 72,0:
       CALL  - 144: RETURN
```

```
1  REM          CQFD
2  REM (C) 1984 Alexandre Avrane
3  REM 25/oct/84
4  REM Version 8.4
100  NOTRACE : ONERR  GOTO 8500
115  IF  PEEK (175) + 256 *  PEEK (176) >
       6810 THEN  FLASH : PRINT "PROGRAM
       ME TROP GRAND!": NORMAL : END
117  IF  PEEK (55) <  > 183 THEN  FLASH :
       PRINT "VOUS N'ETES PAS SOUS PRODO
       S!": NORMAL : END
120  PRINT  CHR$ (4)"CLOSE": POKE 116,150
       : GOSUB 5000: IF  PEEK (768) <  >
       216 THEN  GOSUB 6000
150  GOSUB 6500: POKE 105,0: POKE 106,64:
       POKE 115,0: POKE 116,150: CLEAR
170  GOSUB 3000
180  GOSUB 2000
190 J = 0: GET A$: FOR I = 1 TO  LEN (T$)
       :J = J + (A$ =  MID$ (T$,I,1)): NE
       XT : IF  NOT J THEN 190
499  REM GESTION DU MENU
500  PRINT : ON A$ = "T" GOTO 1000: ON A$
       = "F" GOTO 700: IF A$ = "C" THEN
       X$(0) =  CHR$ (2) + "CATALOG,S" +
       STR$ (DS) + ",D" +  STR$ (DD):X$(
       1) =  CHR$ (4) + "CAT" + PF$: TEXT
       : HOME : PRINT X$(S): GET Z$
530  IF A$ = "S" THEN S = 1 - S
540  IF A$ = "P" THEN  TEXT : HOME : PRIN
       T "Commande ProDOS:": PRINT "->";:
       &  INPUT Z$: IF Z$ <  > "" THEN
       PRINT  CHR$ (4)Z$: GET Z$: GOTO 54
       0
550  IF A$ = "D" THEN  TEXT : HOME : PRIN
       T "Commande Dos 3.3:": PRINT "->";
       : &  INPUT Z$: IF Z$ <  > "" THEN
       PRINT  CHR$ (2)Z$: GET Z$: GOTO 5
       50
555 DS =  PEEK (14313) / 16:DD =  PEEK (1
       4314): IF A$ <  > "X" THEN 180
570  PRINT : PRINT  CHR$ (2)"CLOSE": PRIN
       T  CHR$ (4)"CLOSE": HOME : PRINT :
       PRINT "P: Retour sous Applesoft +
       ProDOS","D: Retour sous Applesoft
       + Dos 3.3","!: Retour sous Apples
       oft + ProDOS+Dos"
580  GET Z$: IF Z$ <  >  CHR$ (27) AND Z$
       <  > "D" AND Z$ <  > "P" AND Z$ <
       > "!" THEN 580
585  ON Z$ =  CHR$ (27) GOTO 1: TEXT : HO
       ME : IF Z$ = "D" THEN  PRINT  CHR$
       (2)"FP"
600  IF Z$ = "P" THEN  PRINT  CHR$ (4)"PR
       #0": PRINT  CHR$ (4)"IN#0"
610  END
699  REM SELECTION DU FICHIER
700  VTAB 13: CALL  - 958: PRINT : POKE 3
       4,12: POKE 35,22: PRINT : PRINT "N
       om du fichier origine:":X$ = F1$:
       GOSUB 7000: IF Z$ <  > "" THEN F1$
       = Z$
720 F2$ = F1$: PRINT : PRINT "Nom du fich
       ier destination:":X$ = F2$: GOSUB
       7000: IF Z$ <  > "" THEN F2$ = Z$
725  PRINT : PRINT "Enregistrements termi
       n(s","par un retour-chariot (o/n)
        0"B$(1);: GOSUB 7100:LX = Z: IF
       NOT LX THEN LF = 1
730  PRINT : PRINT "Longueur totale d'un
       enregistrement:":X$ =  STR$ (LR):
       GOSUB 7000: IF Z AND Z < K64 THEN
       LR = Z
750  IF LX THEN  PRINT : PRINT "Dans chaq
       ue enregistrement,","nombre de cha
       mps s(par(s","par un retour-chario
       t:":X$ =  STR$ (LF): GOSUB 7000: I
       F Z AND Z < K64 THEN LF = Z
753 B% = ( FRE (0) - 2048) / (LR + 3 * LF
        + 9)
755  IF LR + 3 * LF >  FRE (0) THEN  PRIN
       T  CHR$ (7) CHR$ (7): INVERSE : PR
       INT "TAILLE D'1 ENRGT > MEMOIRE DI
       SPONIBLE";: NORMAL : GET Z$: GOTO
       700
760  PRINT : PRINT "No premier enrgt:":X$
        =  STR$ (R1): GOSUB 7000: IF Z >
        - 1 AND Z < M16 THEN R1 = Z
770 R2 =  INT (M16 / LR): PRINT : PRINT "
       No dernier enrgt:":X$ =  STR$ (R2)
       : GOSUB 7000: IF Z >  = R1 AND Z <
        M16 THEN R2 = Z
780  GOTO 180
999  REM CONVERSION
1000  HOME : IF F1$ = "" THEN ER = 2: GOS
       UB 8000: PRINT : GOTO 180
1005 I1 = R1:I2 = R1:K = 0:T = 0:ER = 0
1010  POKE 7, PEEK (109): POKE 8, PEEK (1
       10): DIM A$(B% - 1,LF - 1)
1030  REM LECTURE
1040  PRINT D1$"OPEN"F1$",L"LR: CALL 928
1070  POKE 3, PEEK (111): POKE 4, PEEK (1
       12)
1075  FOR J = 0 TO B% - 1
1080 : PRINT D1$"READ"F1$",R"I1
1090 : PRINT "Lecture #"I1;
1100 : FOR K = 0 TO LF - 1
1110 : IF LX THEN  &  INPUT A$(J,K): GOTO
        1130
1120 : &  GET A$(J,0),LR
1125  IF S THEN  VTAB  PEEK (37)
1130  IF  LEN (A$(J,K)) THEN  IF  NOT  AS
       C ( LEFT$ (A$(J,K),1)) THEN A$(J,K
       ) = ""
1135  HTAB 10 +  LEN ( STR$ (I1)): CALL
       - 868: PRINT ,A$(J,K);: IF  NOT  L
       EN (A$(J,K)) THEN  INVERSE : PRINT
       ,"VIDE";: NORMAL :K = LF - 1
1140 : PRINT : NEXT :I1 = I1 + 1: IF I1 >
        R2 THEN J = B%
1150 : PRINT D1$: NEXT : PRINT D1$"CLOSE"
       F1$
1160  REM ECRITURE
1165  PRINT "Ecriture #"I2"-"I1 - 1: PRIN
```

```
      T D2$"OPEN"F2$",L"LR
1170  FOR J = 0 TO B% - 1
1175 : IF  NOT  LEN (A$(J,0)) THEN 1220
1180 : PRINT D2$"WRITE"F2$",R"I2
1190 : FOR K = 0 TO LF - 1
1200 :: PRINT A$(J,K);
1205 :: IF K < LF - 1 THEN  PRINT
1210 : NEXT : IF LX THEN  PRINT
1220 :I2 = I2 + 1: IF I2 > R2 THEN J = B%
1230 : PRINT D2$: NEXT
1235  PRINT D2$"CLOSE"F2$
1237  POKE 109, PEEK (7): POKE 110, PEEK
      (8)
1240  POKE 111, PEEK (3): POKE 112, PEEK
      (4): IF I1 <  = R2 THEN 1010
1280  FLASH : PRINT : PRINT  SPC( 5)"***
      CONVERSION TERMINEE ***" SPC( 6) C
      HR$ (7): NORMAL : WAIT 49152,128:
      GOTO 180
1999  REM MENU
2000  GOSUB 5000: VTAB 1: PRINT "CONVERSI
      ON QUALIFIEE DE FICHIERS DIRECTS":
       PRINT  SPC( 7)"(C) 1984 ALEXANDRE
       AVRANE" SPC( 8): POKE 33,38: POKE
       32,1: POKE 34,5: POKE 35,23: HOME

2030  PRINT C$(1)"PREFIX": INPUT PF$
2040 D1$ = C$(S):D2$ = C$(1 - S):A$ = F1$
      : IF A$ THEN A$ = A$ + ",L" +  STR
      $ (LR)
2060  PRINT  SPC( 8)S$(S): PRINT : PRINT
      " Dos 3.3: S"DS",D"DD: PRINT " Pro
      DOS : "PF$: PRINT " Fichier: "A$:
      FOR I = 1 TO 38: PRINT "_";: NEXT
2080  PRINT : PRINT " P: Donne une comman
      de ProDOS": PRINT " D: Donne une c
      ommande Dos 3.3": PRINT " S: Inver
      se le sens du transfert"
2090  PRINT " C: Catalogue volume origine
      ": PRINT " F: S(lection du fichier
      ": PRINT " T: Transfert & conversi
      on": PRINT " X: Sortie"
2100  PRINT : HTAB 17: PRINT "-> <-";: HT
      AB 19: RETURN
2999  REM INITIALISATIONS
3000  DIM C$(1):C$(0) =  CHR$ (2):C$(1) =
       CHR$ (4): POKE 10930,130: REM ct
      rl-B pour DOS
3020 T$ = "PDSCFTX": DIM X$(1):S = 0: REM
       sens initial
3050  DIM S$(1):S$(0) = "DOS 3.3 ----> PR
      ODOS":S$(1) = "PRODOS ----> DOS 3.
      3":M16 = 16777215:K64 = 65536:LR =
       1:LF = 1
3060 DS =  PEEK (48700):DD = 3 -  PEEK (4
      8701): REM sur autre drive par d(f
      aut
3070  DIM B$(30):B$(1) =  CHR$ (8): FOR I
       = 2 TO 30:B$(I) = B$(I - 1) +  CH
      R$ (8): NEXT
3080  DIM PE$(1):PE$(0) = "01225833745906
      0900":PE$(1) = "018758337459060994
      "
3090  DIM ER$(9): FOR I = 0 TO 9: READ ER
      $(I): NEXT
3100  DATA *** ERREUR SYSTEME ***,FICHIER
       NON PRECISE,PARAMETRES DE TRANSFE
      RT INVALIDES,VOLUME OU FICHIER INC
      ONNU,DISQUE OU DIRECTORY PLEIN
3110  DATA FICHIER OU VOLUME VERROUILLE,T
      YPE DE FICHIER INVALIDE,ERREUR D'E
      NTREE/SORTIE,FIN BRUTALE DU FICHIE
      R,COMMANDE; PREFIXE OU FICHIER INV
      ALIDE
4999 - REM MODULES UTILITAIRES
5000  TEXT : COLOR= 2: FOR I = 47 TO 0 ST
      EP  - 1: HLIN 0,39 AT I: NEXT : RE
      TURN
6000  PRINT  CHR$ (4)"BLOAD DOS.16K,A$1D0
      0"
6010  POKE 10929,1: POKE 10930,130: REM m
      axfiles=1 & ctrl-b
6020  PRINT  CHR$ (4)"BLOAD CQFD.B,A$300"
6030  CALL 776: POKE 1014,60: POKE 1015,3
6050  GOSUB 6500
6060  TEXT : CALL  - 868: INPUT "Appuyez
      sur <Return> pour commencer:";Z$:
      RETURN
6500  PRINT  CHR$ (4)"PR#A$300": PRINT  C
      HR$ (4)"IN#A$304": RETURN
7000  PRINT "==>"X$B$( LEN (X$));: INPUT
      "";Z$:Z =  INT ( VAL (Z$)): IF Z$
      = "" THEN  VTAB  PEEK (37): HTAB 4
      : PRINT X$
7010  RETURN
7100  GET Z$: ON Z$ <  > CHR$ (13) AND Z
      $ <  > "N" AND Z$ <  > "O" AND Z$
      <  > "n" AND Z$ <  > "o" GOTO 7100
      : PRINT Z$:Z = (Z$ = "O" OR Z$ = "
      o" OR Z$ =  CHR$ (13)): RETURN
7999  REM GESTION DES ERREURS
8000  HOME : PRINT :ER =  VAL ( MID$ (PE$
      (T),ER,1))
8010  HTAB 19 -  LEN (ER$(ER)) / 2 + .5:
      FLASH : PRINT  CHR$ (7)ER$(ER) CHR
      $ (7): NORMAL : PRINT : PRINT "[Re
      turn]->continue, [Esc]->abandonne"
      ;
8020  GET Z$: IF Z$ =  CHR$ (27) THEN  PR
      INT : RUN
8030  IF Z$ =  CHR$ (13) THEN  RETURN
8040  GOTO 8020
8500  CALL 818:T = ( PEEK (55) = 183): CA
      LL 804: REM v(rifie Dos actif et r
      (tablit ProDOS
8520 ER =  PEEK (222) + 1: IF ER > 254 TH
      EN 570
8523  IF ER = 6 AND  NOT  PEEK (986) AND
       NOT T THEN 1130
8525  IF ER > 18 THEN ER = 1
8550  GOSUB 8000: PRINT : RESUME
11000  PRINT  CHR$ (4)"PR#1": PRINT  CHR$
       (27)"A" CHR$ (5): PRINT  CHR$ (9)
      "90N": LIST 0,9999: PRINT  CHR$ (4
      )"PR#0": END
```

```
1 ********************************
2 *             CQFD.B           *
3 ********************************
4
5 * Module appele par CQFD
6 * (conversion qualifiee de
7 *  fichiers directs)
8
9 * Copyright (C) 1984 A.Avrane
10 * 20/oct/84
11
12 * Toutes adresses pour dos en 16K
13
14          ORG  $300
15
16 ********************************
17 * VECTORISATION E/S PRODOS->DOS *
18 ********************************
```

```
19
20 * Ces routines doivent debuter par
    un 'CLD' pour être validees
21
22          CLD
23          JMP  $1EBD       dos 3.3 cs
    w
24          CLD
25          JMP  $1E81       dos 3.3 ks
    w
26
27 ********************************
28 * INITIALISATION PRODOS+DOS 16K *
29 ********************************
30
31          LDA  #$60        'RTS'
32          STA  $1DE7       pour retou
    r apres demarrage a froid
33          JSR  $FE89       pr#0
34          JSR  $FE93       in#0
35          JSR  $1D84       le dos s'i
    nitialise en $1D00-$3FFF
36
37          LDA  #$A0        caractere
    blanc envoye a l'affichage...
38          LDX  #0
39          LDY  #0
40          JSR  $1EBD       ...pour te
    rminer l'initialisation du dos
41
42          LDA  #$6C        'JMP' indi
    rect retablit le Dos...
43          STA  $1DE7       ...pour pe
    rmette la commande "FP"
44
45          JSR  $FE89       pr#0
46          JSR  $FE93       in#0
47          JSR  $9AAF       ProDos pre
    nd la main sur csw/ksw
48          LDA  #0
49          STA  $24         initialise
     htab pour ProDos
50          RTS
51
52 ********************************
53 * ROUTINE D'ERREURS D'APPLESOFT *
54 ********************************
55
56          PLA
57          TAY
58          PLA
59          LDX  $DF         retablit l
    e pointeur de pile
60          TXS
61          PHA
62          TYA
63          PHA
64          RTS
65
66 ********************************
67 * ROUTINE DE SAISIE PAR 'INPUT' *
68 ********************************
69
70          CMP  #$BE        token de G
    ET?
71          BNE  INPUT       non
72          JSR  $00B1       lit le car
    actere suivant
73          JSR  $DFE3       cherche ou
     cree la variable, fixe $83
74          PHA
75          TYA
76          PHA
77          JSR  $DD6C       contrôle
    qu'il s'agit d'une chaîne
78          JSR  $DEBE       verifie la
    presence de la virgule
79          JSR  $E6F8       recoit la
    longueur demandee...
80          STX  $06         ...et la s
    auvegarde
81          PLA
82          STA  $83+1       restore $8
    3=adresse du descripteur de la var
    iable
83          PLA
84          STA  $83
85          LDX  #0
86 LOOP     JSR  $FD0C       lit un car
    actere...
87          STA  $200,X      et le stoc
    ke dans le buffer d'entree
88          JSR  $FDED       on l'envoi
    e en sortie via les Dos
89          INX
90          CPX  $06         suffisamme
    nt de caracteres?
91          BCC  LOOP        pas encore
92          BCS  TRANSFER    =jmp
93
94 INPUT    LDA  #$84        token d'IN
    PUT
95          JSR  $DEC0       verifie pr
    esence du token
96          JSR  $DFE3
97          JSR  $DD6C
98
99          LDX  #$80
100         STX  $33         trompe le
    moniteur et...
101         JSR  $FD6A       ...appelle
     sa routine d'entree ligne
102         STX  $06         on conserv
    e la longueur
103
104 TRANSFER JSR  $D539      formatte l
    a chaîne pour Applesoft
105         LDA  $06         on prend l
    a longueur...
106         JSR  $E452       ...et on d
    emande de la place en memoire
107
108         LDX  #<$200
109         LDY  #>$200      le buffer
    clavier contient notre chaîne,
110         JSR  $E5E2       on la depl
    ace la ou on a trouve de la place
111
112         LDY  #0
113         LDA  $06         on place s
    a longueur...
114         STA  ($83),Y     ...dans le
     descripteur
115         INY
116         LDA  $6F         ainsi que
    son adresse de stockage
117         STA  ($83),Y
118         INY
119         LDA  $6F+1
120         STA  ($83),Y
121         RTS  ; c'est fini!
122
123 ********************************
124 * PATCH DU DOS 3.3 POUR LECTURE *
125 ********************************
126
```

```
127             LDA  #$4C       'jmp' init
    ialise le patch
128             STA  $22B7
129             LDA  #<P
130             STA  $22B7+1
131             LDA  #>P
132             STA  $22B7+2
133             LDA  $2A6C      sauvegarde
     les valeurs de longueur d'enrgt.
134             STA  $22BA
135             LDA  $2A6C+1
136             STA  $22BA+1
137             LDA  #<Q        et patch p
    our fin de fichier
138             STA  $2CB9
139             LDA  #>Q
140             STA  $2CB9+1
141             LDA  #0
142             STA  FLAG^EOF
143             RTS
144
145 P           LDA  $22BA      le dos "ou
    blie" le parametre L d'un fichier.
    ..
146             STA  $2A6C      ...a chaqu
    e erreur, même interceptee par O
    NERR
147             LDA  $22BA+1    Il faut do
    nc lui rappeler si...
148             STA  $2A6C+1    ...on ne v
    eut pas constamment reouvrir le fi
    chier
149             JMP  $22BC
150
151 FLAG^EOF DFB  0
152 Q           LDA  #1
153             STA  FLAG^EOF   indique fi
    n de fichier (et non donnee nulle)
    ...
154             JMP  $336F      ...part en
     END OF DATA intercepte par ONERR
    GOTO
155
156             END
```

```
 300.3E2

0300- D8 4C BD 1E D8 4C 81 1E
0308- A9 60 8D E7 1D 20 89 FE
0310- 20 93 FE 20 84 1D A9 A0
0318- A2 00 A0 00 20 BD 1E A9
0320- 6C 8D E7 1D 20 89 FE 20
0328- 93 FE 20 AF 9A A9 00 85
0330- 24 60 68 A8 68 A6 DF 9A
0338- 48 98 48 60 C9 BE D0 2C
0340- 20 B1 00 20 E3 DF 48 98
0348- 48 20 6C DD 20 BE DE 20
0350- F8 E6 86 06 68 85 84 68
0358- 85 83 A2 00 20 0C FD 9D
0360- 00 02 20 ED FD E8 E4 06
0368- 90 F2 B0 14 A9 84 20 C0
0370- DE 20 E3 DF 20 6C DD A2
0378- 80 86 33 20 6A FD 86 06
0380- 20 39 D5 A5 06 20 52 E4
0388- A2 00 A0 02 20 E2 E5 A0
0390- 00 A5 06 91 83 C8 A5 6F
0398- 91 83 C8 A5 70 91 83 60
03A0- A9 4C 8D B7 22 A9 C8 8D
03A8- B8 22 A9 03 8D B9 22 AD
03B0- 6C 2A 8D BA 22 AD 6D 2A
03B8- 8D BB 22 A9 DB 8D B9 2C
03C0- A9 03 8D BA 2C A9 00 8D
03C8- DA 03 60 AD BA 22 8D 6C
03D0- 2A AD BB 22 8D 6D 2A 4C
03D8- BC 22 00 A9 01 8D DA 03
03E0- 4C 6F 33
```

Gérer la date ProDOS sans carte horloge

François Sermier

Le système ProDOS, entre autres améliorations par rapport au DOS 3.3, permet de dater les fichiers. Il prévoit, pour chacun d'entre eux, une date de création et une date de modification. Cette datation se fait automatiquement si l'Apple possède une carte horloge; malheureusement, rien n'est prévu pour ceux qui n'en ont pas et rien n'est plus désagréable que d'avoir des fichiers datés de NO-DATE...
Heureusement, Apple a bien documenté l'emplacement et le format de la date du système; la date est stockée sur 2 octets, en 49040 – 49041 ($BF90 – $BF91). Une première solution consiste à mettre, soi-même, la date à jour.

Démarche à suivre

– Multiplier le numéro du mois par 32.
– Soustraire 256 si le résultat obtenu est strictement supérieur à 255 (ce qui se produit à partir du mois d'août).
– Ajouter à ce résultat le jour du mois; on obtient alors un premier nombre A.
– Multiplier l'année par 2 et ajouter 1 si on a dû soustraire 256 à A; on obtient un deuxième nombre B.
– Il n'y a plus qu'à POKEr les nombres trouvés aux bonnes adresses ou les saisir sous le moniteur (si l'on préfère convertir en hexa...).
Exemple : le calcul, pour la date du 21 décembre 1984, s'effectuera comme suit :
A = 32 * 12 = 384. Il est supérieur à 255 et devient donc
A = 384 - 256 = 128. Cette opération faite, on lui ajoute le jour.
A = 128 +21 = 149.
A étant trouvé on peut calculer B.
B = 84 * 2 + 1 = 169.
Nous arrivons à la dernière étape.
POKE 49040,149
POKE 49041,169
ou encore :
CALL-151
BF90: 95 A9
3D0G

C'est peut-être fastidieux, mais c'est efficace.

La deuxième solution consiste à employer un utilitaire, comme celui que nous vous proposons ici, qui réalise la même chose. Mieux encore, il garde en mémoire la dernière date entrée et permet de la modifier. Grâce aux flèches gauche et droite on modifie le jour, utilisées avec "Pomme Pleine" on modifie le mois; l'année est changée automatiquement (le passage de décembre à janvier entraîne une incrémentation et celui de janvier à décembre une décrémentation); enfin la date affichée est acceptée par RETURN.
En faisant le catalogue, on peut connaître la date de la dernière utilisation d'un disque (puisqu'elle est stockée dans la directory comme date de modification du programme DATE).
Le programme proposé peut être utilisé par la commande –DATE (smart RUN). Il est peut-être plus simple de l'inclure dans un programme STARTUP de manière à ce qu'il s'exécute automatiquement, en prenant garde de l'appeler sous la forme :
PRINT CHR$(4);"–DATE"
si on tient à avoir la vraie date (celle de modification de DATE). En effet, pour économiser les accès disque et la place mémoire, le programme utilise la table de paramètres du dernier programme auquel ProDOS a accédé.

Appel de ProDOS en langage machine

L'utilisation des routines ProDOS est relativement simple par rapport au DOS 3.3 du fait qu'Apple a prévu une procédure et un point d'entrée uniques par un appel au Machine Language Interface (MLI). Les seules données à fournir sont : un code d'opération et l'adresse de la table des paramètres adaptée à cette opération.
JSR $BF00
DFB CodeOp
DW Adr.Par

Dans notre problème, on pourrait utiliser les codes suivants :
$C4 : GET – FILE – INFO retournant les 15 paramètres définissant le fichier dans la directory
$C3 : SET – FILE – INFO réécrivant les dits paramètres (à l'exclusion de ceux concernant la création évidemment).

Mais pour économiser de la place mémoire, de manière à ce que le programme tienne dans la zone calme de la page 3, on peut aussi utiliser l'accès à ProDOS contenu dans la 'page globale' du Système Basic (page $BE). En effet, GOSYSTEM utilise une table prédéfinie en page globale, dans laquelle on lit ou écrit les informations souhaitées; il ne reste plus qu'à charger le code opération dans l'accumulateur et à appeler GOSYSTEM. De plus, comme on vient de charger DATE pour l'exécuter, cette table contient, avant même l'exécution de la première instruction du programme, les informations concernant le programme DATE. Grande économie de moyens !

Le programme DATE

Il se décompose en trois blocs :
- Lignes 31-45 : récupération de la date comme expliqué ci-dessus; décomposition en jour, mois et année, stockés aux adresses 9, 8, 7 ainsi que dans les registres X, Y, A.
- Lignes 48-109 : boucle principale;

– 48-69 : affichage de la date en utilisant la routine LINPRT, de l'interpréteur Basic, qui affiche en décimal le nombre hexadécimal dont l'octet de poids fort est en A et l'octet de poids faible en Y (c'est la routine qui affiche les numéros de ligne Basic).
– 70-109 : saisie d'une touche au clavier. Si "Pomme Pleine" est enfoncée, on modifie le mois, l'incrémentation ou la décrémentation se faisant par les flèches droite et gauche (pour la flèche droite, on a prévu l'affichage en 40 ou 80 colonnes : les codes utilisés sont différents). Les vérifications de date sont très simples : jour compris entre 1 et 31 quel que soit le mois. Attention au 31 février ! On quitte la boucle par RETURN.
– 112-132 : assemblage des octets de DATE, stockage dans la date du Système et dans la table de paramètres puis appel de GOSYSTEM pour écrire ces informations dans la directory.

Remarque : on ne donne pas le nom du fichier utilisé ! En cas d'erreur, l'interpréteur Basic affiche le message correspondant. Si l'erreur MLI ne correspond à aucun message Basic, on récupère une I/O ERROR.

Bibliographie :
ProDOS Technical Reference Manual (APPLE) et Beneath Apple ProDOS de D.Worth et P.Lechner (Quality Software).

■

```
SOURCE FILE: DATE2.SCE
0000:                1 **********************
      **********************
```

```
0000:                2 *
                       *
0000:                3 *     Proc(dure DATE sa
```

```
       ns carte horloge      *
0000:                4 *     mise @ jour par
       <- et ->              *
0000:                5 *
                           *
0000:                6 ***********************
       ***********************
0000:                7 *
0007:                8 AA       EQU  $7
0008:                9 MM       EQU  $8
0009:               10 JJ       EQU  $9
0000:               11 *
BE09:               12 ERROUT   EQU  $BE09
BE70:               13 GOSYSTEM EQU $BE70
BEB4:               14 SSGINFO  EQU  $BEB4
BEBE:               15 FIMDATE  EQU  $BEBE
BF90:               16 DATE     EQU  $BF90
0000:               17 *
C062:               18 PDL1     EQU  $C062
DB5C:               19 OUTDO    EQU  $DB5C
ED24:               20 LINPRT   EQU  $ED24
FC1A:               21 UP       EQU  $FC1A
FC9C:               22 CLREOL   EQU  $FC9C
FD0C:               23 RDKEY    EQU  $FD0C
FD8E:               24 CROUT    EQU  $FD8E
0000:               25 *
00C3:               26 SETINFO  EQU  $C3
0007:               27 SETPAR   EQU  $07
0000:               28 ***********************
       ******
----- NEXT OBJECT FILE NAME IS DATE2.SCE.
       OBJ0
0300:               29          ORG  $300
0300:               30 ***********************
       ******
0300:AD BF BE       31          LDA  FIMDATE+1
0303:4A             32          LSR  A
0304:85 07          33          STA  AA
0306:AD BE BE       34          LDA  FIMDATE
0309:48             35          PHA
030A:6A             36          ROR  A
030B:4A             37          LSR  A
030C:4A             38          LSR  A
030D:4A             39          LSR  A
030E:4A             40          LSR  A
030F:A8             41          TAY
0310:68             42          PLA
0311:29 1F          43          AND  #$1F
0313:AA             44          TAX
0314:               45 *
0314:               46 *
0314:86 09          47 LOOP     STX  JJ
0316:84 08          48          STY  MM
0318:20 9C FC       49          JSR  CLREOL
031B:20 8E FD       50          JSR  CROUT
031E:20 1A FC       51          JSR  UP
0321:A0 02          52          LDY  #2
0323:D0 05          53          BNE  PR2
0325:A9 2D          54 PR1      LDA  #$2D
0327:20 5C DB       55          JSR  OUTDO
032A:B6 07          56 PR2      LDX  AA,Y
032C:E0 0A          57          CPX  #10
032E:10 05          58          BPL  PR3
0330:A9 30          59          LDA  #$30
0332:20 5C DB       60          JSR  OUTDO
0335:98             61 PR3      TYA
0336:48             62          PHA
0337:A9 00          63          LDA  #0
0339:20 24 ED       64          JSR  LINPRT
033C:68             65          PLA
033D:A8             66          TAY
033E:88             67          DEY
033F:10 E4          68          BPL  PR1
0341:               69 *
0341:20 0C FD       70          JSR  RDKEY
0344:A6 09          71          LDX  JJ
0346:A4 08          72          LDY  MM
0348:C9 95          73          CMP  #$95
034A:F0 04          74          BEQ  ACCROIT
034C:C9 A0          75          CMP  #$A0

034E:D0 1F          76          BNE  DECROIT
0350:AD 62 C0       77 ACCROIT  LDA  PDL1
0353:30 07          78          BMI  ACRMOIS
0355:E8             79          INX
0356:E0 20          80          CPX  #32
0358:30 BA          81          BMI  LOOP
035A:A2 01          82          LDX  #1
035C:C8             83 ACRMOIS  INY
035D:C0 0D          84          CPY  #13
035F:30 B3          85          BMI  LOOP
0361:A0 01          86          LDY  #1
0363:E6 07          87          INC  AA
0365:A5 07          88          LDA  AA
0367:C9 64          89          CMP  #100
0369:30 A9          90          BMI  LOOP
036B:A9 00          91          LDA  #0
036D:85 07          92          STA  AA
036F:               93 *
036F:C9 88          94 DECROIT  CMP  #$88
0371:D0 17          95          BNE  ENDLOOP
0373:AD 62 C0       96          LDA  PDL1
0376:30 05          97          BMI  DECMOIS
0378:CA             98          DEX
0379:D0 99          99          BNE  LOOP
037B:A2 1F         100          LDX  #31
037D:88            101 DECMOIS  DEY
037E:D0 94         102          BNE  LOOP
0380:A0 0C         103          LDY  #12
0382:C6 07         104          DEC  AA
0384:10 8E         105          BPL  LOOP
0386:A9 63         106          LDA  #99
0388:85 07         107          STA  AA
038A:C9 8D         108 ENDLOOP  CMP  #$8D
038C:D0 86         109          BNE  LOOP
038E:              110 *
038E:              111 *
038E:98            112          TYA
038F:0A            113          ASL  A
0390:0A            114          ASL  A
0391:0A            115          ASL  A
0392:0A            116          ASL  A
0393:0A            117          ASL  A
0394:05 09         118          ORA  JJ
0396:8D 90 BF      119          STA  DATE
0399:8D BE BE      120          STA  FIMDATE
039C:A5 07         121          LDA  AA
039E:2A            122          ROL  A
039F:8D 91 BF      123          STA  DATE+1
03A2:8D BF BE      124          STA  FIMDATE+1
03A5:              125 *
03A5:A9 07         126          LDA  #SETPAR
03A7:8D B4 BE      127          STA  SSGINFO
03AA:A9 C3         128          LDA  #SETINFO
03AC:20 70 BE      129          JSR  GOSYSTEM
03AF:90 03         130          BCC  OK
03B1:20 09 BE      131          JSR  ERROUT
03B4:60            132 OK       RTS
```

```
0300- AD BF BE 4A 85 07 AD BE
0308- BE 48 6A 4A 4A 4A 4A A8
0310- 68 29 1F AA 86 09 84 08
0318- 20 9C FC 20 8E FD 20 1A
0320- FC A0 02 D0 05 A9 2D 20
0328- 5C DB B6 07 E0 0A 10 05
0330- A9 30 20 5C DB 98 48 A9
0338- 00 20 24 ED 68 A8 88 10
0340- E4 20 0C FD A6 09 A4 08
0348- C9 95 F0 04 C9 A0 D0 1F
0350- AD 62 C0 30 07 E8 E0 20
0358- 30 BA A2 01 C8 C0 0D 30
0360- B3 A0 01 E6 07 A5 07 C9
0368- 64 30 A9 A9 00 85 07 C9
0370- 88 D0 17 AD 62 C0 30 05
0378- CA D0 99 A2 1F 88 D0 94
0380- A0 0C C6 07 10 8E A9 63
0388- 85 07 C9 8D D0 86 98 0A
0390- 0A 0A 0A 0A 05 09 8D 90
0398- BF 8D BE BE A5 07 2A 8D
03A0- 91 BF 8D BF BE A9 07 8D
03A8- B4 BE A9 C3 20 70 BE 90
03B0- 03 20 09 BE 60
```

Votre Epson en mode proportionnel

Thierry Han

Les imprimantes Epson sont sûrement les plus vendues pour un système Apple II (aujourd'hui talonnées, je crois, par l'Imagewriter). Mais, à part la plus performante (et chère !) FX-80, si je suis bien informé, aucune Epson ne sait proportionner ses lettres. Si vous en possédez une, vous direz certainement : "elles font bien d'autres choses !". Certes, mais à présent, vous n'aurez plus besoin de tenir une polémique avec votre voisin qui possède une imprimante proportionnelle pour savoir laquelle est la meilleure. Grâce au programme que je vous propose ici, vous obtiendrez, moyennant une perte de 1,75K de mémoire vive, un mode proportionnel sur votre Epson graphique. Parmi ces imprimantes figurent tous les MX-type III, plus la RX-80 (que je possède). D'autres pourront éventuellement être utilisées, moyennant une modification des codes d'initialisation du mode graphique double densité. Si votre imprimante n'a pas de double densité, on peut quand même tourner la difficulté en modifiant, d'une part, les codes à imprimer, et, d'autre part, PROP.DEF pour les proportions "vertical / horizontal" en tenant compte du nombre d'aiguilles de la tête.

Grandes lignes de fonctionnement

Une grande partie des 1792 octets occupés est prise par la définition des caractères, sous une forme que j'expliquerai plus loin. Un jeu de caractères "standards" est inclus, mais le programme PROP.DEF permet de modifier ces définitions, dont l'ensemble ne devra toutefois pas dépasser l'adresse $9500, soit environ 1280 octets. Avec le jeu standard, 252 octets restent libres. Cette définition se scinde en deux parties : la partie index et la partie des codes à passer directement à l'imprimante.

Le programme lui-même ne fait que 291 octets, à partir de l'adresse $8F00. Il se compose d'une phase d'initialisation, de la routine utilisée par l'ampersand (&), la routine d'entrée des caractères à imprimer, et enfin la routine d'impression lors d'un retour chariot ou de saturation du buffer (sans retour chariot ni saut de ligne dans ce dernier cas). L'entrée des caractères pour impression proportionnelle peut se faire de deux façons : par &, ou par déviation du vecteur CSWL du DOS. La première possibilité est offerte par défaut, et la seconde peut être utilisée grâce au petit module nommé PREPROP (relogeable et prenant 59 octets), qui sauve les registres internes avant de sauter directement dans la routine d'entrée de PROP. Le mode opératoire devient alors similaire à un PR#1 du DOS.

Principe de codage

Le codage des caractères pour le module PROP est assez simple : il s'agit de codes à envoyer directement à l'imprimante. Le manuel de votre Epson donne des explications assez claires et je ne ferai donc qu'un bref résumé.
La tête d'impression est composée de 9 aiguilles disposées verticalement. Le mode graphique consiste alors à imprimer un point à la position de chaque bit à 1. On associe ainsi les bits d'un octet aux aiguilles de la tête. Comme un octet ne fait que huit bits, il est clair que l'une des aiguilles reste inactive : c'est la plus basse. On ne peut donc avoir que 8 points verticaux. Toutefois, en ayant recours à des sauts de lignes bien dosés, on peut faire arriver la tête d'impression juste en-dessous de la ligne précédente, créant ainsi une continuité verticale et une matrice plus grande. Mais pour le moment, ce n'est pas ce qui nous intéresse.
Prenons l'exemple de l'impression du code $A5, 10100101 en binaire. Ceci nous donne sur papier :

```
1 * (bit à 1 = un point)
0
1 *
0
0
1 *
0
1 *
```

Voilà pour le format primaire des octets qui forment les codes des caractères. Cependant, PROP utilise un système d'index pour trouver rapidement les codes du caractère désiré. Cet index constitue la première partie des codes et occupe 192 octets. Il contient les adresses des longueurs des codes (à partir du début de l'index, sur 2 octets). Voyons alors comment un caractère est codé : chaque groupe de codes à envoyer à l'imprimante est précédé d'un octet indiquant la longueur des codes, qui arrivent ensuite dans l'ordre de sortie. Tout cela , bien rangé, représente 993 octets pour le jeu de caractères standard, index compris.

Utilisation

Pour mettre en route PROP, il suffit de faire un BRUN ou un CALL 36608 après BLOAD. L'instruction & est initialisée. Pour utiliser PREPROP, il suffit de faire un autre BRUN ou CALL 768 après le chargement de PROP et de PREPROP. Un CALL 802 (ou CALL "adresse de chargement" + 34) déconnectera PROP. Pour PREPROP qui agit comme un PR#1, il suffit de faire un PRINT de ce que l'on veut imprimer. Pour &, il faut assigner une chaîne puis faire :
& chaîne$ ou & chaîne$;
selon que l'on veut ou non effectuer un saut de ligne.

Par ailleurs, un & seul fera imprimer tout ce que l'on avait fait entrer dans le buffer (avec & chaîne$;). S'il n'y a rien, seul un saut de ligne sera effectué. Notez que si chaîne$ contient un retour chariot (Ascii 13), l'impression, puis le saut de ligne seront effectués, même si un point-virgule est utilisé. Les caractères suivant le retour chariot seront traités normalement.

Note importante : ne faites surtout pas FP seul avec PROP en mémoire. Ce FP remet en place la HIMEM et les variables alphanumériques viendront écraser PROP après un nettoyage de la mémoire. Faites donc un NEW au lieu de FP, ou faites suivre le FP par un HIMEM:36608.

Définition des caractères

Si vous aimez le jeu de caractères standard fourni, il n'y a aucun besoin d'utiliser PROP.DEF, programme Basic de définition des caractères.

Dans le cas contraire, PROP.DEF vous demandera d'abord le nom du fichier de caractères à éditer (vous pouvez en effet sauvegarder différents jeux de caractères sous des noms différents). Un RETURN à vide indique au programme de travailler sur le fichier actuellement en mémoire, ce qui suppose bien sûr que celui-ci s'y trouve effectivement. Il faut obligatoirement travailler à partir d'un jeu existant, donc à partir du jeu standard pour la toute première édition.
Vous devrez ensuite préciser s'il s'agit d'un Module (routines d'impression plus codes) ou simplement d'un fichier Codes. Dans ce dernier cas, vous ne pourrez pas essayer directement les caractères créés mais les principes de définition restent les mêmes (voir plus loin la méthode de fusion entre les routines et les codes).

Lorsque les données d'identification du fichier sont fournies, le programme décode le jeu de caractères en mémoire, puis apparaissent une grille HGR et 4 lignes de commande. Pour éditer un caractère sur la grille, il suffit de taper ce caractére. Quelques touches de fonction sont prévues pour pallier l'absence de certains caractères sur le clavier de l'Apple II+ :

- CTRL-B permet de basculer entre les majuscules (MI=0 dans les lignes de commandes) et les minuscules (MI=1)
- CTRL-I correspond au caractère "trait soulignant" (Ascii 95) en majuscule
- CTRL-O correspond au caractère "[" en minuscule
- CTRL-P correspond au caractère "slash inverse" en minuscule

Une fois entrée, la lettre s'affiche en gros points sur l'écran. Pour positionner le "curseur", vous devez indiquer la position horizontale (1 chiffre de 0 à 9) puis la position verticale (un chiffre de 1 à 8). Ensuite, la frappe de la touche "0" change l'état du point à l'endroit du curseur : il l'allume s'il était éteint et inversement. A ce stade, toute autre touche que 0 laisse le point inchangé.
En ce qui concerne la position horizontale, il n'est pas possible d'atteindre tous les points d'une ligne au moyen des seuls chiffres 0 à 9. Si vous voulez modifier un point situé dans une position intermédiaire, placez tout d'abord le curseur sur le point situé à gauche de votre objectif et laissez-le inchangé. Tapez ensuite la "flèche à droite" en réponse à la demande de position horizontale et cela vous amènera à l'endroit voulu. Par exemple, le troisième point d'une ligne n'est pas accessible directement (le deuxième correspond à la position horizontale 1 et le quatrième à la position 2); placez vous alors sur la position 1 et la "flèche à droite" vous placera ensuite sur le troisième point.

Si vous possédez un joystick ou deux paddles, vous pouvez remplacer les lignes 4000 à 4050 du programme PROP.DEF par les lignes listées plus loin sous le nom "Version Paddles". Dans ce cas, le déplacement est donné par les poignées et l'un des boutons permet de modifier l'état d'un point à l'endroit du "curseur".

Les lettres sont affichées en matrices de points, qui reproduisent à l'écran très fidèlement ce qui sera imprimé sur papier. Pour les techniciens des Epson, je précise que l'impression se fait en double densité, mais pas en vitesse double, ce qui confère à ces caractères proportionnels une qualité d'impression supérieure. Bref, vous dessinez sur l'écran ce que vous voudriez voir sur papier et vous pouvez valider votre création par RETURN. Sinon, vous pouvez recommencer en appuyant sur ESC.

Si vous appuyez sur ESC lors de la demande du caractère à éditer, et si vous avez un module en mémoire (et non seulement un fichier codes) vous aurez droit à une impression de tous les caractères disponibles (Ascii 32 à 127). Par ailleurs, un RETURN au même niveau provoque le codage des données stockées dans les tableaux du Basic en une forme qui sera POKée directement dans la table qu'utilise PROP. Si vous ne voulez toutefois pas enregistrer sur disquette, entrez un RETURN à vide à la demande du nom de sauvegarde. Ce codage est nécessaire pour pouvoir essayer le nouveau jeu édité. Si vous voulez stocker ce nouveau jeu sur disquette, il suffit d'entrer un nom, de préférence différent de tous ceux qui sont présents sur la disquette. Après cela, vous pouvez terminer avec ESC, ou continuer avec n'importe quelle autre touche. Notez qu'à la sauvegarde, tout le module sera ou non enregistré en même temps que les codes selon que vous aurez indiqué M)odules ou C)odes au début de l'édition.
Si vous appuyez sur CTRL-S (sauvegarde) et non RETURN en lieu et place d'un caractère à éditer, le programme agira comme s'il avait dépassé le stade de codage et demandera directement le nom du fichier à sauvegarder. Les changements effectués après le dernier codage seront ignorés, bien que présents dans les tableaux du Basic.

Une dernière remarque : après l'appui du RETURN qui valide un caractère, une fourchette s'affiche sur l'écran : elle indique les deux "limites" du caractère, à partir desquelles on ne trouve rien. Ce calcul est automatique. Si vous voulez inclure quelques espaces de plus que ces limites, appuyez sur CTRL-L (pour "Limites") au lieu de RETURN, et le programme vous demandera la colonne de départ et celle d'arrivée (1 à 16 inclus). Si le caractère que vous éditez dégénère en "rien du tout", le programme vous demandera la largeur de l'espace pour ce caractère. Les deux cas standards, l'espace et le code Ascii 127 (Del), prennent trois espaces.

Les différents programmes

Le module PROP se nomme PROP.OBJ sur la disquette, et contient les codes standards.
Par contre, PROP.PROG ne les contient pas : c'est le programme objet directement assemblé à partir de PROP.SCE, programme source (Dos Tool Kit).
Les codes (index + codes) sont reproduits dans le fichier CODES, qui doit être chargé juste après la partie programme, soit à l'adresse $9023.
Cette disposition a été adoptée pour ne pas devoir garder de longs codes dans le programme source.
PROP.EXEC, fichier EXECutable, fait le nécessaire pour souder la partie programme et la partie codes afin de créer un fichier PROP.OBJ2, qui sera généré à partir des fichiers PROP.PROG (assemblage de PROP.SCE) et CODES (relogé).

PREPROP.SCE et PREPROP.OBJ sont les programmes source et objet de PREPROP.
PROP.DEF est le programme Applesoft qui édite les jeux de caractères pour PROP. Enfin, S.SHP contient la shape du point utilisé par PROP.DEF.

A propos du Dos Tool Kit, si vous avez une Carte Langage, éditez vos programmes avec le disque virtuel de Michel Haag (Pom's 12); cela accélère beaucoup les lectures/écritures et, surtout, l'assemblage se fait très rapidement.

■

```
]LIST:PROP.DEF

10  HIMEM: 36608: LOMEM: 16510
20  GOTO 9000
990  REM  CADRE
1000  HOME : HGR : HCOLOR= 3: POKE 34,20:
       VTAB 21
1010  FOR I = 0 TO 15: HPLOT H + I * H,0:
       HPLOT H + I * H,V * 8 + 2: NEXT
1020  FOR I = 0 TO 8: HPLOT H1,1 + I * V
       TO H * 16 + H1,I * V + 1: NEXT
1030  FOR I = H TO H * 16 STEP H * 5: HPL
       OT I,131 TO I,140: NEXT
1040  REM  ENTREE, CONVERTION CONTROLS
1050  HOME : VTAB 21: PRINT "RETURN POUR
       ENREGISTRER,": PRINT "ESCAPE POUR
       ESSAYER.": PRINT "CARACTERE (MI="M
       I"): ";: GET R$
1060  IF R$ =  CHR$ (15) THEN R$ =  CHR$
       (91): GOTO 1140
1070  IF R$ =  CHR$ (16) THEN R$ =  CHR$
       (92): GOTO 1140
1080  IF R$ =  CHR$ (9) THEN R$ =  CHR$ (
       95): GOTO 1140
```

```
1085  IF R$ =  CHR$ (12) THEN 5000
1090  IF R$ =  CHR$ (2) THEN MI = 1 - MI:
       PRINT : GOTO 1050
1100  IF R$ =  CHR$ (13) THEN 6000
1110  IF R$ =  CHR$ (27) AND T$ = "M" THE
      N 7000
1120  IF R$ =  CHR$ (19) THEN 6060
1130  IF R$ < " " THEN  PRINT : GOTO 1050
1140  INVERSE : IF MI = 1 AND  ASC (R$) >
       = 64 THEN RM$ = R$:R$ =  CHR$ (
      ASC (RM$) + 32): PRINT "MIN "RM$;:
       NORMAL : PRINT "     ASCII " ASC
      (R$): GOTO 1160
1150  IF R$ >  = "@" THEN  PRINT "MAJ ";
1155  PRINT R$;: NORMAL : PRINT "     ASC
      II " ASC (R$)
1160 R =  ASC (R$) - 32: IF R > 127 THEN
      R = R - 128
1990  REM  DECODAGE A PARTIR DE LA MEMOIR
      E
2000  FOR C = 1 TO D%(R)
2010 C% = P%(R,C)
2020  FOR I = 7 TO 0 STEP  - 1: IF C% >
      = 2 ^ I THEN C% = C% - 2 ^ I:G%(C,
      7 - I) = 1: GOTO 2040
2030 G%(C,7 - I) = 0
2040  NEXT I,C
2050  FOR I = C TO 17: FOR J = 0 TO 7:G%(
      I,J) = 0: NEXT J,I
2990  REM  DESSIN
3000  FOR I = 1 TO D%(R): FOR J = 0 TO 7
3010  IF G%(I,J) = 1 THEN  DRAW 1 AT I *
      H,J * V + V1
3020  NEXT J,I
3990  REM  EDITION
4000  VTAB 24: HTAB 1: PRINT "PH : 0 A 9
      OU ->  ";: IF  PEEK ( - 16384) < 1
      28 THEN 4000
4010 K =  PEEK ( - 16384): POKE  - 16368,
      0:KZ = K - 128: IF (KZ > 47 AND KZ
       < 58) OR KZ = 21 THEN 4032
4030  IF K > 139 THEN  ON K - 139 GOTO 50
      00,5020: ON K = 155 GOTO 1000
4031  GOTO 4000
4032  IF KZ = 21 AND KX < 9 THEN CX = ZX
      +  INT (H / 8): GOTO 4041
4034  IF KZ = 21 THEN 4000
4040 KZ = KZ - 48:KX = KZ:CX =  INT (KZ *
       255 / 153) + 1:ZX = CX
4041  VTAB 24: HTAB 1: PRINT "PV : 1 A 8
            ";: IF  PEEK ( - 16384) <
      128 THEN 4041
4042 KZ =  PEEK ( - 16384) - 128: POKE  -
      16368,0: IF KZ < 49 OR KZ > 56 TH
      EN 4041
4043 KZ = KZ - 48:CY =  INT (KZ * 255 / 2
      88)
4050  XDRAW 1 AT CX * H,CY * V + V1
4060  XDRAW 1 AT CX * H,CY * V + V1
4070  VTAB 24: HTAB 1: PRINT "0 CHANGE LE
      POINT          ";: IF  PEEK ( - 16
      384) < 128 THEN 4070
4071 K =  PEEK ( - 16384): POKE  - 16368,
      0:KZ = K - 128: IF KZ <  > 48 THEN
       4000
4080  FOR M = 0 TO 100: NEXT
4090  IF G%(CX,CY) = 0 THEN G%(CX,CY) = 1
      .: DRAW 1 AT CX * H,CY * V + V1: GO
      TO 4120
4100 G%(CX,CY) = 0: XDRAW 1 AT CX * H,CY
      * V + V1: IF CX > 1 THEN  IF G%(CX
       - 1,CY) = 1 THEN  DRAW 1 AT CX *
      H - H,CY * V + V1
4110  IF CX < 16 THEN  IF G%(CX + 1,CY) =
       1 THEN  DRAW 1 AT CX * H + H,CY *
       V + V1
4120  GOTO 4000
4990  REM  CTRL-L
5000  INPUT "COLONNE DEBUT: ";D: INPUT "C
      OLONNE FIN: ";L:L = L + 1 * (L < 1
      6): IF D < 1 OR L > 16 OR L < D TH
      EN 5000
5005  GOTO 5060
5010  REM  LECTURE ECRAN
5020  FOR C = 1 TO 16: FOR I = 0 TO 7: IF
       G%(C,I) = 0 THEN  NEXT I,C: INPUT
       "LONGUEUR DES ESPACES: ";L:D = 1:
       GOTO 5070
5030 I = 9: NEXT I:D = C: FOR C = 16 TO D
       STEP  - 1:L = 0: FOR I = 0 TO 7:
      IF G%(C,I) = 0 THEN  NEXT I,C: GOT
      O 5050
5040 I = 9: NEXT I
5050 L = C
5060  HPLOT D * H,150 TO D * H,159 TO L *
       H,159 TO L * H,150
5070  FOR I1 = D TO L:I = I1 - D + 1:P%(R
      ,I) = 0
5080  FOR J = 0 TO 7
5090  IF G%(I1,7 - J) = 1 THEN P%(R,I) =
       P%(R,I) + 2 ^ J
5100  NEXT J,I1:D%(R) = I
5110  GOTO 1000
5990  REM  CODAGE POUR MODULE PROP
6000 TA = IN + 192: FOR R = 0 TO 95:R1 =
       R * 2:TX = TA - IN
6010 T1 =  INT (TX / 256): POKE IN + R1,T
      X - T1 * 256: POKE I2 + R1,T1
6020  POKE TA,D%(R)
6030  FOR I = 1 TO D%(R)
6040  POKE TA + I,P%(R,I): NEXT
6050 TA = TA + I: NEXT
6060  PRINT : INPUT "NOM DU FICHIER A ENR
      EGISTRER: ";F$: IF F$ = "" THEN 61
      00
6070 R$ = "C": IF T$ = "M" THEN  PRINT "C
      )ODES  OU  M)ODULE? ";: GET R$: PR
      INT R$
6080  IF R$ = "C" THEN  PRINT D$"BSAVE"F$
      ",A"IN",L"TA - IN: GOTO 6100
6090  PRINT D$"BSAVE"F$",A$8F00,L"TA - IN
       + IN - 36608
6100  PRINT "PRESSEZ UNE TOUCHE.(ESC POUR
       FINIR)";: GET R$: IF R$ <  >  CHR
      $ (27) THEN 1000
6110  END
6990  REM  ESC: ESSAI SUR IMPRIMANTE
7000  HOME : PRINT "ESSAI: ALLUMEZ L'IMPR
      IMANTE.": GET R$
7010 R$ = "": FOR I = 32 TO 127:R$ = R$ +
       CHR$ (I): NEXT
7020  & R$: GOTO 1000
8990  REM  INITIALISATION
9000  TEXT : HOME : DIM P%(95,17),D%(95),
      G%(17,7):D$ =  CHR$ (4)
9005 H = 9:V = 16:V1 =  INT (V / 2) + 1:H
      1 =  INT (H / 2) + 1: REM  DONNEES
       POUR LES PROPORTIONS H/V A L'ECRA
      N
9010  POKE 232,0: POKE 233,64: ROT= 0: SC
      ALE= 1
9020  IF  PEEK (16384) = 1 AND  PEEK (163
      85) = 0 AND  PEEK (16386) = 4 AND
       PEEK (16387) = 0 AND  PEEK (16388
      ) = 44 THEN 9040: REM  TEST SI SHA
      PE PRESENT
```

```
9030  PRINT D$"BLOADS.SHP,A$4000"
9040  INPUT "NOM DE FICHIER A EDITER: ";F
      $: PRINT "M)ODULE  OU  C)ODES? ";:
       GET T$: PRINT T$
9050  IF F$ = "" THEN  PRINT "EN MEMOIRE.
      ": GOTO 9070
9060  PRINT D$"BLOAD"F$
9070  PRINT "DECODAGE:": IF T$ = "M" THEN
        CALL 36608
9080  REM  DECODAGE A PARTIR DES CODES
9090  IF T$ = "C" THEN IN =  PEEK (43634)
       +  PEEK (43635) * 256: GOTO 9110
9100 IN =  PEEK (36611) +  PEEK (36612) *
      256
9110 TA = IN + 192:I2 = IN + 1
9120  FOR R = 0 TO 95
9130 D%(R) =  PEEK (TA)
9140  FOR I = 1 TO D%(R):P%(R,I) =  PEEK
      (TA + I): NEXT
9150 TA = TA + I: PRINT  CHR$ (R + 32);:
      NEXT
9160 D$ =  CHR$ (4): GOTO 1000: REM  D$ E
      FFACE PAR CALL 36608
```

Version Joystick

```
JLIST 4000,4050

4000 CX =  INT ( PDL (0) / 17) + 1
4010  IF  PEEK ( - 16384) ( 128 THEN 4040
4020 K =  PEEK ( - 16384): POKE  - 16368,
      0
4030  IF K ) 139 THEN  ON K - 139 GOTO 50
      00,5020: ON K = 155 GOTO 1000
4040 CY =  INT ( PDL (1) / 32)
4050  XDRAW 1 AT CX * H,CY * V + V1
```

PROP.OBJ

```
*8F00.9404

8F00- 4C 05 8F 23 90 A9 4C 8D
8F08- F5 03 A9 26 8D F6 03 A9
8F10- 8F 8D F7 03 A9 00 85 73
8F18- 85 6F A9 8F 85 74 85 70
8F20- 20 B5 8F 4C F8 8F 20 B7
8F28- 00 F0 35 20 E3 DF 20 6C
8F30- DD 85 85 84 86 A0 00 B1
8F38- 85 85 1C C8 B1 85 85 1D
8F40- C8 B1 85 85 1E A0 00 B1
8F48- 1D 84 1B 20 66 8F A4 1B
8F50- C8 C4 1C D0 F2 20 B7 00
8F58- C9 3B D0 04 20 B1 00 60
8F60- A9 0D 20 66 8F 60 A6 06
8F68- 29 7F C9 0D F0 3D 38 E9
8F70- 20 90 F2 0A 9D 00 95 E8
8F78- 86 06 20 01 90 A0 00 B1
8F80- 08 18 69 02 85 FB 65 19
8F88- 85 19 90 02 E6 1A A5 FB
8F90- 18 65 FC 85 FC 90 02 E6
8F98- FD A5 FD C9 03 D0 C6 A5
8FA0- FC C9 B0 B0 06 A6 06 D0
8FA8- BC F0 11 E0 00 F0 64 20
8FB0- BC 8F 20 13 90 A2 00 86
8FB8- FC 86 FD 60 A9 1B 20 1A
8FC0- 90 A9 4C 20 1A 90 A5 19
8FC8- 20 1A 90 A5 1A 20 1A 90
8FD0- A2 00 BD 00 95 20 01 90
8FD8- A9 00 20 1A 90 20 1A 90
8FE0- A8 B1 08 A8 C8 84 07 A0
8FE8- 01 B1 08 20 1A 90 C8 C4
8FF0- 07 D0 F6 E8 E4 06 D0 DA
8FF8- A2 00 86 06 86 19 86 1A
9000- 60 A8 B9 23 90 18 69 23
9008- 85 08 C8 B9 23 90 69 90
9010- 85 09 60 A9 0D 20 1A 90
9018- A9 0A 2C C1 C1 30 FB 8D
9020- 90 C0 60 C0 00 C4 00 C9
9028- 00 CE 00 D8 00 E0 00 E8
9030- 00 F0 00 F3 00 F7 00 FB
9038- 00 03 01 0B 01 0E 01 16
9040- 01 19 01 22 01 2A 01 30
9048- 01 38 01 40 01 48 01 50
9050- 01 58 01 60 01 68 01 70
9058- 01 73 01 76 01 7B 01 83
9060- 01 88 01 90 01 9A 01 A8
9068- 01 B2 01 BC 01 C6 01 D0
9070- 01 DA 01 E4 01 F0 01 F6
9078- 01 FF 01 0B 02 15 02 23
9080- 02 2F 02 3B 02 45 02 51
9088- 02 5B 02 65 02 71 02 7B
9090- 02 89 02 97 02 A4 02 B0
9098- 02 BA 02 BF 02 C8 02 CD
90A0- 02 D3 02 DD 02 E0 02 EA
90A8- 02 F4 02 FC 02 06 03 0F
90B0- 03 16 03 1F 03 29 03 2F
90B8- 03 35 03 3D 03 43 03 51
90C0- 03 5A 03 62 03 6B 03 74
90C8- 03 7C 03 84 03 8C 03 96
90D0- 03 A0 03 AE 03 B8 03 C2
90D8- 03 CA 03 CF 03 D1 03 D6
90E0- 03 DD 03 03 00 00 00 04
90E8- 60 FA FA 60 04 E0 00 00
90F0- E0 09 2A 2C 38 68 AA 2C
90F8- 38 68 A8 07 20 52 52 FF
9100- 4A 4A 04 07 C2 C4 08 10
9108- 20 46 86 07 44 AA 92 AA
9110- 44 0A 10 02 20 C0 03 3C
9118- 42 81 03 81 42 3C 07 08
9120- 2A 1C 08 1C 2A 08 07 08
9128- 08 08 3E 08 08 08 02 05
9130- 06 07 08 08 08 08 08 08
9138- 08 02 02 02 08 01 02 04
9140- 08 10 20 40 80 07 38 44
9148- 82 82 82 44 38 05 22 42
9150- FE 02 02 07 46 8A 92 92
9158- 92 92 62 07 44 82 82 92
9160- 92 92 6C 07 08 18 28 48
9168- 88 FE 08 07 E4 A2 A2 A2
9170- A2 A2 9C 07 0C 12 32 52
9178- 92 12 0C 07 80 82 84 88
9180- 90 A0 C0 07 6C 92 92 92
9188- 92 92 6C 07 60 90 92 94
```

```
9190- 98 90 60 02 14 14 02 15
9198- 16 04 10 28 44 82 07 14
91A0- 14 14 14 14 14 14 04 82
91A8- 44 28 10 07 40 80 80 8A
91B0- 90 A0 40 09 38 44 82 92
91B8- AA AA BA 8A 72 0D 02 06
91C0- 0A 18 A8 C8 88 48 28 18
91C8- 0A 06 02 09 82 FE 92 92
91D0- 92 92 92 92 6C 09 38 44
91D8- 82 82 82 82 82 82 C4 09
91E0- 82 FE 82 82 82 82 82 42
91E8- 3C 09 82 FE 92 92 92 BA
91F0- 82 82 C6 09 82 FE 92 92
91F8- 90 B8 80 80 C0 09 38 44
9200- 82 82 82 82 92 92 DE 0B
9208- 82 FE 92 10 10 10 10 10
9210- 92 FE 82 05 82 82 FE 82
9218- 82 08 04 02 02 82 82 82
9220- FC 80 0B 82 FE 92 10 28
9228- 28 44 44 82 82 82 09 82
9230- FE 82 02 02 02 02 02 06
9238- 0D 82 FE 82 40 20 10 08
9240- 10 20 40 82 FE 82 0B 82
9248- FE 82 40 20 10 08 04 82
9250- FE 82 0B 38 44 82 82 82
9258- 82 82 82 82 44 38 09 82
9260- FE 92 90 90 90 90 90 60
9268- 0B 38 44 82 82 82 82 82
9270- 8A 8A 44 3A 09 82 FE 92
9278- 90 90 98 94 92 62 09 66
9280- 92 92 92 92 92 92 92 CC
9288- 0B C0 80 80 80 82 FE 82
9290- 80 80 80 C0 09 80 FC 82
9298- 02 02 02 82 FC 80 0D 80
92A0- C0 A0 10 08 04 02 04 08
92A8- 10 A0 C0 80 0D 80 C0 B0
92B0- 0C 02 0C 30 0C 02 0C B0
92B8- C0 80 0C 82 C6 C6 28 28
92C0- 10 10 28 28 C6 C6 82 0B
92C8- 80 C0 C0 20 22 1E 22 20
92D0- C0 C0 80 09 C2 86 86 8A
92D8- 92 A2 C2 C2 86 04 FE 82
92E0- 82 82 08 80 40 20 10 08
92E8- 04 02 01 04 82 82 82 FE
92F0- 05 40 80 80 80 40 09 01
92F8- 01 01 01 01 01 01 01 01
```

```
9300- 02 C0 20 09 04 0A 2A 2A
9308- 2A 2A 2A 1C 02 09 82 FC
9310- 22 22 22 22 22 22 1C 07
9318- 1C 22 22 22 22 22 12 09
9320- 1C 22 22 22 22 22 A2 FC
9328- 02 08 1C 2A 2A 2A 2A 2A
9330- 2A 18 06 12 7E 92 92 92
9338- 40 08 18 24 25 25 25 25
9340- 1E 20 09 82 FE 12 20 20
9348- 20 22 1E 02 05 22 22 BE
9350- 02 02 05 02 01 21 21 BE
9358- 07 82 FE 0A 14 14 22 22
9360- 05 82 82 FE 02 02 0D 22
9368- 1E 22 20 20 22 1E 22 20
9370- 20 22 1E 02 08 22 1E 22
9378- 20 20 22 1E 02 07 1C 22
9380- 22 22 22 22 1C 08 21 1F
9388- 25 24 24 24 24 18 08 18
9390- 24 24 24 24 25 1F 21 07
9398- 22 1E 22 20 20 20 10 07
93A0- 12 2A 2A 2A 2A 2A 24 07
93A8- 20 FC 22 22 22 22 04 09
93B0- 20 3C 22 02 02 02 22 3C
93B8- 22 09 20 30 28 04 02 04
93C0- 28 30 20 0D 20 30 28 04
93C8- 02 04 08 04 02 04 28 30
93D0- 20 09 22 22 14 14 08 14
93D8- 14 22 22 09 20 21 31 0A
93E0- 04 08 30 20 20 07 22 26
93E8- 26 2A 32 32 22 04 10 6C
93F0- 82 82 01 FF 04 82 82 6C
93F8- 10 06 40 80 80 40 40 80
9400- 03 00 00 00 00
```

PREPROP.OBJ

```
*300.33A

0300- AD 53 AA 85 FE AD 54 AA
0308- 85 FF 20 58 FF BA CA BD
0310- 00 01 18 69 21 8D 53 AA
0318- E8 BD 00 01 69 00 8D 54
0320- AA 60 A5 FE 8D 53 AA A5
0328- FF 8D 54 AA 60 20 4A FF
0330- A5 45 20 66 8F 20 3F FF
0338- 6C FE 00
```

PROP.EXEC

```
BLOADPROP.PROG
AD=PEEK(36611)+PEEK(36612)*256
PRINT"BLOADCODES,A"AD (Rem : il y a un C
      TRL-D devant BLOAD)
BSAVEPROP.OBJ2,A$8F00,L$600
```

S.SHP

```
*4000.4077

4000- 01 00 04 00 2C 36 3F 24
4008- 2C 2D 36 36 3F 3F 24 24
```

```
4010- 2C 2D 2D 36 36 36 3F 3F
4018- 3F 24 24 24 2C 2D 2D 2D
4020- 36 36 36 36 3F 3F 3F 3F
4028- 24 24 24 24 2C 2D 2D 2D
4030- 2D 36 36 36 36 36 3F 3F
4038- 3F 3F 3F 24 24 24 24 24
4040- 0C 2D 2D 2D 2D 2D 32 36
4048- 36 36 36 36 3B 3F 3F 3F
4050- 3F 3F 20 24 24 24 24 24
4058- 08 29 2D 2D 2D 2D 09 12
4060- 32 36 36 36 36 12 1B 3B
4068- 3F 3F 3F 3F 1B 58 58 20
4070- 24 24 24 24 08 21 00 18
```

PROP.SCE

```
SOURCE FILE: PROP.SCE
----- NEXT OBJECT FILE NAME IS PROP.SCE.O
      BJ0
8F00:                1          ORG  $8F00
8F00:                2 **********************
      **********
8F00:                3 *IMPRIMANTE PROPORTION
      NELLE
8F00:                4 *EPSON->PROPORTIONNEL

8F00:                5 *PAR:THIERRY HAN
8F00:                6 *DATE:3/5/1984
8F00:                7 *SYNTAXE:&CHAINE$(;)
8F00:                8 **********************
      **********
8F00:                9 ;DOS TOOL KIT
8F00:               10 ;
0006:               11 XREG     EQU  $6
0007:               12 YREG     EQU  $7
0008:               13 INP      EQU  $8
0019:               14 TOTAL    EQU  $19
      ;LONGUEUR JUSQU'A IMPRESSION
001B:               15 SAVE     EQU  $1B
001C:               16 LEN      EQU  $1C
001D:               17 ADR      EQU  $1D
00FB:               18 AREG     EQU  $FB
00FC:               19 TOT1     EQU  $FC
      ;LONGUEUR JUSQU'A CR
9500:               20 BUFFER   EQU  $9500

8F00:               21 ;
006F:               22 STREND   EQU  $6F

0073:               23 HIMEM    EQU  $73
00B1:               24 CHRGET   EQU  $B1
00B7:               25 CHRGOT   EQU  $B7

0085:               26 LAST     EQU  $85
03F5:               27 AMPVEC   EQU  $3F5
      ;VECTEUR &
C1C1:               28 BUSY     EQU  $C1C1
      ;POUR L'IMPRIMANTE
C090:               29 DATA     EQU  $C090
DD6C:               30 CHKSTR   EQU  $DD6C
DFE3:               31 PTRGET   EQU  $DFE3
8F00:4C 05 8F       32          JMP  DEBUT
8F03:               33 ;
8F03:23 90          34          DW   INDEX
      ;POUR PROP.DEF ET PROP.EXEC
8F05:               35 ;
8F05:               36 ;INITIALISATION
8F05:               37 ;
8F05:A9 4C          38 DEBUT    LDA  #$4C
      ;JMP
8F07:8D F5 03       39          STA  AMPVEC
8F0A:A9 26          40          LDA  #>START
8F0C:8D F6 03       41          STA  AMPVEC+1
8F0F:A9 8F          42          LDA  #<START
8F11:8D F7 03       43          STA  AMPVEC+2
8F14:A9 05          44          LDA  #>DEBUT
8F16:85 73          45          STA  HIMEM
8F18:85 6F          46          STA  STREND
8F1A:A9 8F          47          LDA  #<DEBUT

8F1C:85 74          48          STA  HIMEM+1
8F1E:85 70          49          STA  STREND+1

8F20:20 B5 8F       50          JSR  RAZTOT
8F23:4C F8 8F       51          JMP  INIT

8F26:               52 ;
8F26:               53 ;DEBUT DE &
8F26:               54 ;
8F26:20 B7 00       55 START    JSR  CHRGOT
8F29:F0 35          56          BEQ  PRT
      ;RIEN:IMPRESSION
8F2B:20 E3 DF       57          JSR  PTRGET
      ;TROUVE LE DESCRIPTEUR
8F2E:20 6C DD       58          JSR  CHKSTR
      ;SI NON-CHAINE, TYPE MISMATCH
8F31:85 85          59          STA  LAST
8F33:84 86          60          STY  LAST+1
8F35:A0 00          61          LDY  #0
8F37:B1 85          62          LDA  (LAST),Y
      ;LONGUEUR
8F39:85 1C          63          STA  LEN
8F3B:C8             64          INY
8F3C:B1 85          65          LDA  (LAST),Y
      ;ET ADRESSE DE LA CHAINE
8F3E:85 1D          66          STA  ADR
8F40:C8             67          INY
8F41:B1 85          68          LDA  (LAST),Y
8F43:85 1E          69          STA  ADR+1
8F45:A0 00          70          LDY  #0
8F47:               71 ;
8F47:B1 1D          72 LOOP     LDA  (ADR),Y
      ;PREND CHAQUE CARACTERE
8F49:84 1B          73          STY  SAVE
      ;DE LA CHAINE ET
8F4B:20 66 8F       74          JSR  PROP
      ;L'ENVOIE A PROP
8F4E:A4 1B          75          LDY  SAVE
8F50:C8             76          INY
8F51:C4 1C          77          CPY  LEN
8F53:D0 F2          78          BNE  LOOP
8F55:20 B7 00       79          JSR  CHRGOT

8F58:C9 3B          80          CMP  #$3B
      ;POINT-VIRGULE
8F5A:D0 04          81          BNE  PRT

8F5C:20 B1 00       82          JSR  CHRGET
      ;POUR INCREMENTER TXTPTR
8F5F:60             83          RTS
8F60:               84 ;
8F60:A9 0D          85 PRT      LDA  #$D
8F62:20 66 8F       86          JSR  PROP
      ;FORCE IMPRESSION PAR SAUT DE LIG
     NE
8F65:60             87 END      RTS
       ;POUR LES BRANCH
8F66:               88 ;
8F66:               89 ;ROUTINE DE STOCKAGE D
    ANS BUFFER
8F66:               90 ;
8F66:A6 06          91 PROP     LDX  XREG
      ;INDEX DANS BUFFER
8F68:29 7F          92          AND  #$7F
8F6A:C9 0D          93          CMP  #$D
      ;CODE 13 PROVOQUE L'IMPRESSION
8F6C:F0 3D          94          BEQ  CR
      ;ET SAUT DE LIGNE
8F6E:38             95          SEC
8F6F:E9 20          96          SBC  #$20

8F71:90 F2          97          BCC  END
8F73:0A             98          ASL  A
      ;AJUSTE POUR INDEXER
8F74:9D 00 95       99          STA  BUFFER,X
8F77:E8            100          INX
8F78:86 06         101          STX  XREG
8F7A:              102 ;
8F7A:20 01 90      103          JSR  GETADR
8F7D:A0 00         104          LDY  #0
8F7F:B1 08         105          LDA  (INP),Y
      ;LONGUEUR DES CODES POUR LA LETTR
     E
8F81:18            106          CLC
8F82:69 02         107          ADC  #2
      ;DEUX ESPACES ENTRE LETTRES
```

```
8F84:85 FB      108          STA  AREG
8F86:65 19      109          ADC  TOTAL
8F88:85 19      110          STA  TOTAL
8F8A:90 02      111          BCC  ADDTOT

8F8C:E6 1A      112          INC  TOTAL+1
8F8E:           113 ;
8F8E:A5 FB      114 ADDTOT   LDA  AREG
8F90:18         115          CLC
8F91:65 FC      116          ADC  TOT1
8F93:85 FC      117          STA  TOT1
8F95:90 02      118          BCC  TEST
8F97:E6 FD      119          INC  TOT1+1
8F99:           120 ;
8F99:           121 ;TESTE DEBORDEMENT
8F99:           122 ;
8F99:A5 FD      123 TEST     LDA  TOT1+1
8F9B:C9 03      124          CMP  #3
8F9D:D0 C6      125          BNE  END
8F9F:A5 FC      126          LDA  TOT1

8FA1:C9 B0      127          CMP  #176
        ;TOTAL 960 CODES/LIGNE
8FA3:B0 06      128          BCS  CR

8FA5:           129 ;
8FA5:           130 ;TESTE FIN DE BUFFER
8FA5:           131 ;
8FA5:A6 06      132          LDX  XREG
8FA7:D0 BC      133          BNE  END
8FA9:F0 11      134          BEQ  PRINT
8FAB:           135 ;
8FAB:           136 ;IMPRESSION + SAUT DE
      LIGNE + RAZ TOTAL
8FAB:           137 ;
8FAB:E0 00      138 CR       CPX  #0
8FAD:F0 64      139          BEQ  CROUT
        ;RIEN, JUSTE CR
8FAF:20 BC 8F   140          JSR  PRINT
8FB2:20 13 90   141          JSR  CROUT
8FB5:A2 00      142 RAZTOT   LDX  #0
8FB7:86 FC      143          STX  TOT1

8FB9:86 FD      144          STX  TOT1+1
8FBB:60         145          RTS
8FBC:           146 ;
8FBC:           147 ;IMPRIME BUFFER (SANS
      SAUT DE LIGNE)
8FBC:           148 ;
8FBC:A9 1B      149 PRINT    LDA  #$1B
        ;CODES DE L'IMPRIMANTE
8FBE:20 1A 90   150          JSR  COUT
        ;(MODE GRAPHIQUE DOUBLE DENSITE)
8FC1:A9 4C      151          LDA  #$4C

8FC3:20 1A 90   152          JSR  COUT
8FC6:A5 19      153          LDA  TOTAL
8FC8:20 1A 90   154          JSR  COUT
8FCB:A5 1A      155          LDA  TOTAL+1
8FCD:20 1A 90   156          JSR  COUT
8FD0:A2 00      157          LDX  #0
8FD2:BD 00 95   158 NEXTCHR  LDA  BUFFER,X
        ;LETTRE A SORTIR
8FD5:20 01 90   159          JSR  GETADR

8FD8:A9 00      160          LDA  #0
        ;DEUX ESPACES ENTRE LETTRES
8FDA:20 1A 90   161          JSR  COUT
8FDD:20 1A 90   162          JSR  COUT
8FE0:A8         163          TAY
8FE1:B1 08      164          LDA  (INP),Y
        ;LONGUEUR DES CODES
8FE3:A8         165          TAY
8FE4:C8         166          INY
8FE5:84 07      167          STY  YREG
8FE7:A0 01      168          LDY  #1
8FE9:B1 08      169 NEXTBYT  LDA  (INP),Y
        ;CODE A SORTIR
8FEB:20 1A 90   170          JSR  COUT
8FEE:C8         171          INY
8FEF:C4 07      172          CPY  YREG
8FF1:D0 F6      173          BNE  NEXTBYT
8FF3:E8         174          INX
8FF4:E4 06      175          CPX  XREG
8FF6:D0 DA      176          BNE  NEXTCHR
8FF8:           177 ;
8FF8:           178 ;RAZ DU BUFFER
8FF8:           179 ;
8FF8:A2 00      180 INIT     LDX  #0
8FFA:86 06      181          STX  XREG
8FFC:86 19      182          STX  TOTAL
8FFE:86 1A      183          STX  TOTAL+1
9000:60         184          RTS
9001:           185 ;
9001:           186 ;SOUS-ROUTINES
9001:           187 ;
9001:A8         188 GETADR   TAY
        ;CHERCHE ADRESSE DES CODES
9002:B9 23 90   189          LDA  INDEX,Y
        ;A PARTIR DE LA TABLE INDEX
9005:18         190          CLC
9006:69 23      191          ADC  #>INDEX
9008:85 08      192          STA  INP
        ;QUI POINTE EN "OFFSET"
900A:C8         193          INY
900B:B9 23 90   194          LDA  INDEX,Y
900E:69 90      195          ADC  #<INDEX
9010:85 09      196          STA  INP+1

9012:60         197          RTS
9013:           198 ;
9013:A9 0D      199 CROUT    LDA  #$D
        ;RETOUR CHARIOT
9015:20 1A 90   200          JSR  COUT
9018:A9 0A      201          LDA  #$A
        ;ET SAUT DE LIGNE
901A:           202 ;
901A:2C C1 C1   203 COUT     BIT  BUSY
        ;ENVOI DIRECT
901D:30 FB      204          BMI  COUT
        ;A L'IMPRIMANTE
901F:8D 90 C0   205          STA  DATA
9022:60         206          RTS
9023:           207 ;
9023:           208 ;TABLE DES INDEX
9023:           209 ;
9023:           210 INDEX    EQU  *

*** SUCCESSFUL ASSEMBLY: NO ERRORS
```

PREPROP.SCE

```
SOURCE FILE: PREPROP.SCE
----- NEXT OBJECT FILE NAME IS PREPROP.SC
      E.OBJ0
0300:             1          ORG  $300
0300:             2 **********************
      **********
0300:             3 *PREPROP:REVECTORISE C
      SWL
0300:             4 *POUR PROP
0300:             5 *CALL768 POUR BRANCHER
      PROP
0300:             6 *CALL789 POUR DEBRANCH
      ER
0300:             7 **********************
      **********
0300:             8 ;
AA53:             9 CSWL     EQU  $AA53
8F66:            10 PROP     EQU  $8F66
```

```
0045:              11 AREG     EQU  $45
       ;DE SAVE
00FE:              12 COUT1    EQU  $FE
```

```
FF4A:              13 SAVE     EQU  $FF4A
FF3F:              14 RESTORE  EQU  $FF3F
FF58:              15 RETURN   EQU  $FF58
0100:              16 STACK    EQU  $100
0300:              17 ;
0300:              18 ;INIT PROP
0300:              19 ;
0300:AD 53 AA      20          LDA  CSWL
0303:85 FE         21          STA  COUT1
       ;SAUVEGARDE L'ANCIEN
0305:AD 54 AA      22          LDA  CSWL+1
0308:85 FF         23          STA  COUT1+1
030A:              24 ;
030A:20 58 FF      25          JSR  RETURN
       ;CHERCHE ADRESSE RELOGEE
030D:BA            26 ICI      TSX
030E:CA            27          DEX
030F:BD 00 01      28          LDA  STACK,X
0312:18            29          CLC
0313:69 21         30          ADC  #>PROPZ-I
       CI+1
0315:8D 53 AA      31          STA  CSWL
0318:E8            32          INX
0319:BD 00 01      33          LDA  STACK,X
031C:69 00         34          ADC  #<PROPZ-I
       CI+1
031E:8D 54 AA      35          STA  CSWL+1
0321:60            36          RTS
0322:              37 ;
0322:              38 ;DES-INIT: REMET L'ANC
       IEN
0322:              39 ;
0322:A5 FE         40          LDA  COUT1
0324:8D 53 AA      41          STA  CSWL
0327:A5 FF         42          LDA  COUT1+1

0329:8D 54 AA      43          STA  CSWL+1
032C:60            44          RTS
032D:              45 ;
032D:              46 ;ROUTINE EXECUTEE AVAN
       T PROP:
032D:              47 ;SAUVE REGISTRES PUIS
       RESTORE A LA FIN
032D:              48 ;
032D:20 4A FF      49 PROPZ    JSR  SAVE

0330:A5 45         50          LDA  AREG
       ;SAVE MODIFIE A
0332:20 66 8F      51          JSR  PROP
0335:20 3F FF      52          JSR  RESTORE
0338:6C FE 00      53          JMP  (COUT1)
       ;JUMP AU VRAI COUT
```

Accès direct aux disquettes

Guy d'Herbemont

Dans le Pom's 10 nous était présenté un programme simple d'accès direct aux disquettes. Mais il y a plus simple encore ! La routine contenue dans la ligne 100 du programme proposé ici suffit en effet pour réaliser ce type d'opérations.

Le DOS assure lui même l'essentiel du travail en utilisant le buffer de 256 octets commençant en 46267 ($B4BB) et l'appel à RWTS se fait en 45111, après entrée des paramètres aux endroits indiqués.

Voici venue la fin du règne de la bande des quatre (IOB, table DCB, accumulateur et registre Y) sur RWTS, loyal serviteur scandalisé par les codes cachés...

Nous devons la découverte de ces adresses à Bill Parker (Call A.P.P.L.E. in Depth (N° 3), mais je ne le suivrai pas totalement dans sa démarche. En effet, après s'être inquiété des dialogues avec le buffer, il a aussitôt succombé aux charmes de l'ampersand, pour aboutir finalement à deux excellents programmes, certes, mais relativement longs et dont l'un est en assembleur.

Pour rester concis, et je l'espère utile, j'ai inclus la routine de la ligne 100 dans un petit programme Applesoft qui affiche le contenu d'un secteur choisi de deux manières :

- Affichage du code hexadécimal des 256 octets du buffer. Pour ce faire, l'appel au moniteur s'impose et le dialogue s'établit facilement avec la méthode de S.H.LAM (voir Recueil 1, page 94).
- Affichage des caractères concaténés (ligne 40). Les lignes 60 et 70 servent à modifier les caractères perturbateurs ou indésirables, tout en conservant le code 255 ($FF) qui s'affiche sous forme du damier (curseur de l'Apple //e) et permet notamment de retrouver dans la piste 17 ($11) la position des fichiers effacés que l'on souhaiterait récupérer.

La table des codes hexadécimaux comportant 33 lignes, il pourrait être utile de prévoir sa sortie sur imprimante.
L'exploration du contenu des secteurs de leurs disquettes promet bien des surprises à ceux qui n'ont pas encore eu l'occasion de s'y attarder. La routine de la ligne 100 peut évidemment être utilisée également pour l'écriture (C=2), mais il appartient alors à chacun de mesurer les conséquences de ses actes en fonction de sa connaissance de la structure des disquettes et du fonctionnement du DOS. ■

```
]LOAD ACCES.HERBEMONT
]LIST

1  REM  LECTURE DIRECTE
2  REM  SUR DISQUETTE
3  REM  ===============
5 C = 1: INPUT "PISTE : ";P: INPUT "SECTE
      UR : ";S:SLOT = 6:DRIVE = 1: ON P
      < 0 OR P > 34 OR S < 0 OR S > 15 G
      OTO 5: GOSUB 100
10  REM  LECTURE EN HEXADECIMAL
20 A$ = "B4BB.B5BA ND823G": FOR I = 1 TO
      LEN (A$): POKE 511 + I, ASC ( MID
      $ (A$,I,1)) + 128: NEXT : POKE 72,
      0: CALL  - 144: PRINT
30  REM  LECTURE EN CARACTERES
40 BUF = 46267: FOR I = 0 TO 255:A =  PEE
      K (BUF + I): GOSUB 60: PRINT A$;:
      NEXT : PRINT
50  END
60 A$ =  CHR$ (A): IF (A > 32 AND A < 127
      ) OR A > 161 THEN  RETURN
70 A$ = ".": RETURN
90  REM  ACCES DISQUE PAR LA ;ETHODE SIMP
      LIFIEE
100  POKE 45121,C: POKE 45975,P: POKE 459
      76,S: POKE 46583,SLOT * 16: POKE 4
      6584,DRIVE: CALL 45111: POKE 45121
      ,2: POKE 72,0: RETURN
```

L'Apple en multitâche ?

Jérôme Leclercq

Avoir un Apple pour travailler, c'est bien. Mais en avoir deux ? Le programme DOUBLE APPLE n'effectue pas la multiplication des pommes, mais permet de travailler SIMULTANEMENT sur deux programmes en Applesoft.

Les utilisations sont nombreuses : exploitation de deux versions d'un programme, consultation de l'un pendant que l'on édite l'autre, etc.

L'initialisation, après le chargement de la routine, est lancée par un &CLEAR. Votre programme précédent semble disparaître mais n'est pas effacé : la commande &INVERSE échange les deux programmes en mémoire. Ceux-ci peuvent être chargés, sauvés sur disque, édités et bien sûr exécutés sous la réserve que les variables soient globales pour l'ensemble des programmes : la valeur d'une variable commune est donc identique quelque soit le programme actif.

La mémoire disponible pour le second programme, après la commande &CLEAR, correspond à l'ensemble de la mémoire libre moins $3000 octets (paramètre MARGE de l'assemblage) qui sont réservés pour l'extension du premier programme.

Si ce dernier, en cours d'édition, approche très près de la frontière, une série de deux beeps se fait entendre à chaque touche enfoncée. Il est alors préférable de sauver les deux programmes, puis de les recharger en relançant la routine, plutôt que de risquer d'écraser le début du second programme.

Pour un aperçu des possibilités de la routine, lancer la démonstration avec EXEC DOUBLE APPLE.DEMO et examiner l'interférence des variables entre les 2 programmes.

■

```
 1 *----> DOUBLE APPLE <----*
 2
 3 * Cr(ation: 06/04/84 J. Leclercq
 4 * Modifi(: 20/06/84 A. Avrane
 5
 6           ORG  $300
 7
 8 SAV2TXT   =    $06
 9 SAV2END   =    $08
10 KSW       =    $38
11 TXTTAB    =    $67
12 PRGEND    =    $AF
13 CHRGET    =    $B1
14 SAVTXT    =    $EB
15 SAVEND    =    $ED
16 NUMERO    =    $EF
17 INTDOS    =    $3EA
18 AMPER     =    $3F5
19 NEW       =    $D64B
20 KEYIN     =    $FD1B
21 BELL      =    $FF3A
22 MARGE     =    $30
23
24 * ==========================
25 * Place & et la routine de saisie
26 * ==========================
27           LDA  #$4C        jmp
28           STA  AMPER
29           LDA  #<ENTRY
30           STA  AMPER+1
31           LDA  #>ENTRY
32           STA  AMPER+2
33
34           LDA  #<ENTRY2
35           STA  KSW
36           LDA  #>ENTRY2
37           STA  KSW+1
38           JMP  INTDOS
39
40 * ================
41 * Controle l'appel
42 * ================
43 ENTRY     =    *
44           CMP  #$BD        clear
45           BEQ  CLEAR
46           CMP  #$9E        inverse
47           BEQ  INVERSE
48           RTS
49
50 * ==============================
51 * Clear: initialise la scission
52 * ==============================
53 CLEAR     LDA  TXTTAB
54           STA  SAVTXT
55           LDA  TXTTAB+1
56           STA  SAVTXT+1
57           LDA  PRGEND
58           STA  SAVEND
59           LDA  PRGEND+1
60           STA  SAVEND+1
61
62           LDA  PRGEND
63           STA  TXTTAB
64           LDA  PRGEND+1
65           CLC
66           ADC  #MARGE
67           STA  TXTTAB+1
68
69           LDA  #2
70           STA  NUMERO
71           JSR  CHRGET
72           LDA  TXTTAB
73           SEC
74           SBC  #1
75           STA  PTRDEBUT
76           LDA  TXTTAB+1
77           SBC  #0
78           STA  PTRDEBUT+1
```

```
 79            LDA   #0
 80            STA   $3333       (cras(!
 81 PTRDEBUT =       *-2
 82            JMP   NEW
 83
 84 *================================
 85 *Inverse:(change les 2 programmes
 86 *================================
 87 INVERSE   LDA   #1
 88            CMP   NUMERO
 89            BNE   P2VERSP1
 90            LDA   #2
 91            STA   NUMERO
 92            BNE   ECHANGE     =jmp
 93 P2VERSP1  STA   NUMERO       A=1
 94 ECHANGE   LDA   TXTTAB
 95            STA   SAV2TXT
 96            LDA   TXTTAB+1
 97            STA   SAV2TXT+1
 98            LDA   PRGEND
 99            STA   SAV2END
100            LDA   PRGEND+1
101            STA   SAV2END+1
102            LDA   SAVTXT
103            STA   TXTTAB
104            LDA   SAVTXT+1
105            STA   TXTTAB+1
106            LDA   SAVEND
107            STA   PRGEND
108            LDA   SAVEND+1
109            STA   PRGEND+1
110            LDA   SAV2TXT
111            STA   SAVTXT
112            LDA   SAV2TXT+1
113            STA   SAVTXT+1
114            LDA   SAV2END
115            STA   SAVEND
116            LDA   SAV2END+1
117            STA   SAVEND+1
118            JMP   CHRGET
119
120 * ============================
121 * Saisie: controle d(bordement
122 * ============================
123 ENTRY2    =     *
124            JSR   KEYIN
125            PHA
126            LDA   NUMERO
127            CMP   #1
128            BNE   EXIT
129            LDA   PRGEND+1
130            CLC
131            ADC   #1          s(curit(
132            CMP   SAVTXT+1
133            BCC   EXIT
134            JSR   BELL
135            JSR   BELL
136 EXIT       PLA
137            RTS
138            END
```

```
*BLOAD DOUBLE APPLE
*300.3B7

0300- A9 4C 8D F5 03 A9 1A 8D
0308- F6 03 A9 03 8D F7 03 A9
0310- 9D 85 38 A9 03 85 39 4C
0318- EA 03 C9 BD F0 05 C9 9E
0320- F0 3A 60 A5 67 85 EB A5
0328- 68 85 EC A5 AF 85 ED A5
0330- B0 85 EE A5 AF 85 67 A5
0338- B0 18 69 30 85 68 A9 02
0340- 85 EF 20 B1 00 A5 67 38
0348- E9 01 8D 57 03 A5 68 E9
0350- 00 8D 58 03 A9 00 8D 33
0358- 33 4C 4B D6 A9 01 C5 EF
0360- D0 06 A9 02 85 EF D0 02
0368- 85 EF A5 67 85 06 A5 68
0370- 85 07 A5 AF 85 08 A5 B0
0378- 85 09 A5 EB 85 67 A5 EC
0380- 85 68 A5 ED 85 AF A5 EE
0388- 85 B0 A5 06 85 EB A5 07
0390- 85 FC A5 08 85 ED A5 09
0398- 85 EE 4C B1 00 20 1B FD
03A0- 48 A5 EF C9 01 D0 0F A5
03A8- B0 18 69 01 C5 EC 90 06
03B0- 20 3A FF 20 3A FF 68 60
```

```
]EXEC DOUBLE APPLE.DEMO

]MONICO

]BLOAD DOUBLE APPLE
]CALL 768
```

```
]LOAD DOUBLE APPLE.P1
]& CLEAR

]LOAD DOUBLE APPLE.P2
]& INVERSE
```

```
]LOAD DOUBLE APPLE.P1
]LIST

0  REM P1/A.AVRANE/200684
10  REM PROGRAMME 1
11  DEF  FN L(L) = L + 1 / K
20  FOR I = 1 TO 33
25  INVERSE : LIST 9,50: NORMAL
30 : &  INVERSE : GOTO 1
40 L =  FN L(L)
50  PRINT "L="L: NEXT
70  PRINT
80  PRINT "NOMBRE D'OR = "L
```

```
]LOAD DOUBLE APPLE.P2
]LIST

0  REM P2/A.AVRANE/200684
1  INVERSE : LIST 10,45: NORMAL
5  PRINT "I="I,
10  REM PROGRAMME 2
11 K = 1
12  DEF  FN K(K) = K * J
40  FOR J = 1 TO I
41 :K =  FN K(K): NEXT
50  PRINT "K="K
70  PRINT
80  &  INVERSE : GOTO 40
```

Initiation à l'assembleur (6)

Gérard Michel

Comme tous les langages, ou presque, l'assembleur peut servir à l'écriture de programmes complets. D'ailleurs, la plupart des logiciels "professionnels" vous sont livrés sous forme de codes machine, résultats de l'assemblage de volumineux programmes sources. Dans ce cas, toutes les opérations du logiciel ont été conçues en assembleur : entrée au clavier et affichage à l'écran, affectation des valeurs aux variables, calculs, accès disques, impression...
Même si l'on utilise alors intensément les routines offertes par la ROM de l'Apple ou par son DOS, il est évident que la mise au point d'un tel logiciel représente un travail de longue haleine et d'une grande minutie.

Autre utilisation possible de l'assembleur, plus souple et plus facile à mettre en oeuvre, l'écriture de modules de taille "abordable", utilisés en complément d'un langage évolué, fera l'objet du présent article.
Dans la mesure où l'Applesoft constitue le compagnon inséparable de votre Apple II ou //, nous examinerons donc les moyens par lesquels peut s'aménager une cohabitation entre ce Basic "évolué" et des routines en langage-machine issues de votre travail de création.
L'assembleur permettra alors de faire certaines choses plus vite ou mieux que le Basic, il en complètera le jeu d'instructions, mais l'essentiel de vos programmes sera toujours réalisé en Applesoft, garantissant ainsi une plus grande rapidité d'écriture, sinon un soulagement au niveau des maux de tête...

CALL : pour et contre

Pour appeler une routine à partir du Basic, il semble facile d'utiliser une instruction Basic comme CALL, que vous connaissez déjà bien. Pour appeler une routine débutant à l'adresse décimale A (point d'entrée), on insère *CALL A* à l'endroit voulu dans le programme et l'ordinateur procède à l'exécution de la routine avant de revenir à l'Applesoft.

C'est effectivement la solution la plus simple, car on laisse finalement au Basic le soin d'assurer lui-même la jonction avec les routines, sans avoir à se préoccuper de la façon dont il y parvient.
Le CALL est bien pratique pour exploiter certaines routines de la ROM : CALL −868 pour effacer la fin de la ligne courante, par exemple. De même pour utiliser des routines de votre cru, préalablement chargées à l'adresse adéquate.

Mais l'usage du CALL comporte des limitations qui réduisent en fait son champ d'application :

- Si la routine appelée réclame des paramètres, il n'est pas facile de les lui transmettre autrement que par des POKEs plus ou moins nombreux, à des adresses choisies à l'avance et qui doivent rester à la disposition de l'utilisateur.
- Si la routine comporte plusieurs points d'entrée, en fonction des traitements à effectuer, il faut faire des CALLs à des adresses différentes, ou POKEr quelque part un paramètre permettant à la routine de déterminer sur quel point d'entrée elle doit se brancher.
- Si vous modifiez les routines et que cela déplace les points d'entrée, il faut alors modifier également le programme Basic...

Nous présenterons donc ici deux autres solutions pour réaliser l'interface entre Basic et langage-machine. La première fait également appel à l'Applesoft "évolué", au travers de l'instruction &, dont la renommée n'est plus à faire. La seconde utilise en revanche une routine en langage-machine de l'Applesoft, dont nous parlerons en détail plus avant. Dans un cas comme dans l'autre, l'utilisation de ces techniques suppose un minimum de renseignements sur la façon dont l'Applesoft exécute et manipule les instructions d'un programme.

Analyse du RUN

Structure d'un programme

Un programme Applesoft est stocké en mémoire centrale à partir d'une adresse "pointée" par les adresses $67 et $68. En situation normale, c'est-à-dire en l'absence de manipulation des pointeurs par le programmeur, cette adresse de début est $801; on trouve alors #$01 dans $67 et #$08 dans $68. L'adresse $800 contient toujours pour sa part la valeur 0 si l'adresse de début des programmes est fixée à $801.
Si aucun programme ne se trouve en mémoire, après un NEW par exemple, les trois adresses successives $800, $801 et $802 contiennent la valeur 0.

Une ligne de programme est stockée de la façon suivante :

- Adresse de la ligne suivante, dans l'ordre octet bas / octet haut : si la deuxième ligne commence en $80A, on trouvera #$0A dans $801 et #$08 dans $802. Si la troisième ligne débute en $818, on aura alors #$18 dans $80A et #$08 dans $80B.
- Numéro de la ligne, dans l'ordre octet bas / octet haut.
- Codes des instructions de la ligne.
- La fin de la ligne est marquée par un 0.

On sait que l'on est à la fin du programme lorsque l'on trouve 0 et 0 dans les deux octets où l'on devrait autrement trouver l'adresse de la ligne suivante. Un programme Applesoft se termine donc par trois 0, un pour marquer la fin de la dernière ligne, et deux pour marquer la fin du programme. C'est la même configuration que celle des adresses $800 à $802 lorsqu'il n'y a pas de programme du tout.

Pour bien comprendre la suite, il serait bon de vous munir de votre Pom's 4 ou de votre recueil 1, et de l'ouvrir à l'article traitant des codes ASCII épluchés.
En effet, les instructions de l'Applesoft ne sont pas stockées "en clair", mais sous la forme de "tokens", c'est-à-dire de codes sur un octet, chacun de ces codes correspondant à une et une seule instruction. Ainsi, si vous avez un PRINT dans une ligne, vous ne trouverez pas en mémoire les codes ASCII des lettres P, R, I, N et T, mais tout simplement *BA*, qui est le token de PRINT en hexadécimal.
Les autres composantes des instructions (variables, constantes numériques, constantes alphanumériques...) sont en revanche conservées sous forme ASCII. Pour PRINT 39, par exemple, on aurait en mémoire *BA 33 39*. Il s'agit toujours d'ASCII positif, avec le bit 7 des codes à 0, car les tokens de l'Applesoft utilisent l'ASCII négatif (bit 7 à 1) et s'étendent des valeurs #$80 à #$EA.

L'Applesoft utilise un pointeur de programme, situé aux adresses $B8-$B9. Ce pointeur est géré par une routine commençant à l'adresse $B1 (baptisée souvent CHRGET), sur laquelle nous reviendrons plus loin en détail. Pour l'heure, il nous suffit de savoir que ce pointeur contient l'adresse du caractère du programme Basic (token, élément d'un nom de variable...) que l'interpréteur est en train d'analyser. Prenons l'exemple du petit programme suivant :

```
10 PRINT 39
```

Il est stocké en mémoire à partir de $801 sous la forme :

```
801- 09 08 0A 00 BA 33 39
808- 00 00 00
```

Lorsque l'interpréteur abordera l'analyse du token de PRINT (BA), le pointeur de programme $B8-$B9 contiendra l'adresse de BA, soit $805 (# $05 dans $B8 et #$08 dans $B9).

Lorsque vous lancez l'exécution de votre programme par RUN, l'Applesoft procède tout d'abord à quelques initialisations : remise à zéro des variables, positionnement du pointeur de programme au début de celui-ci (soit adresse contenue dans $67-$68, diminuée de 1), c'est-à-dire à $800 dans le cas standard...
On arrive ensuite sur une routine débutant en $D7D2 et qui constitue le coeur de l'interpréteur de l'Applesoft, car c'est elle qui "lit" les instructions du programme, en prépare l'analyse et assure le branchement sur les routines de traitement correspondant à chacune de ces instructions.

L'exécution du programme

Le listing ci-dessous reprend l'essentiel de la routine d'exécution des instructions. Nous en examinerons les points principaux, afin de déterminer dans quelles conditions on peut sans risques ajouter à l'Applesoft d'autres instructions, au moyen de routines "maison".

```
D7D2-   BA          TSX
D7D3-   86 F8       STX   $F8
D7D5-   20 58 D8    JSR   $D858
D7D8-   A5 B8       LDA   $B8
D7DA-   A4 B9       LDY   $B9
D7DC-   A6 76       LDX   $76
D7DE-   E8          INX
D7DF-   F0 04       BEQ   $D7E5
D7E1-   85 79       STA   $79
D7E3-   84 7A       STY   $7A
D7E5-   A0 00       LDY   #$00
D7E7-   B1 B8       LDA   ($B8),Y
D7E9-   D0 57       BNE   $D842
D7EB-   A0 02       LDY   #$02
D7ED-   B1 B8       LDA   ($B8),Y
D7EF-   18          CLC
D7F0-   F0 34       BEQ   $D826
D7F2-   C8          INY
D7F3-   B1 B8       LDA   ($B8),Y
D7F5-   85 75       STA   $75
D7F7-   C8          INY
D7F8-   B1 B8       LDA   ($B8),Y
D7FA-   85 76       STA   $76
D7FC-   98          TYA
D7FD-   65 B8       ADC   $B8
D7FF-   85 B8       STA   $B8
D801-   90 02       BCC   $D805
D803-   E6 B9       INC   $B9
D805-   24 F2       BIT   $F2
D807-   10 14       BPL   $D81D
D809-   A6 76       LDX   $76
D80B-   E8          INX
D80C-   F0 0F       BEQ   $D81D
D80E-   A9 23       LDA   #$23
D810-   20 5C D8    JSR   $D85C
D813-   A6 75       LDX   $75
D815-   A5 76       LDA   $76
D817-   20 24 ED    JSR   $ED24
D81A-   20 57 D8    JSR   $D857
D81D-   20 B1 00    JSR   $00B1
D820-   20 28 D8    JSR   $D828
D823-   4C D2 D7    JMP   $D7D2
D826-   F0 62       BEQ   $D88A
D828-   F0 2D       BEQ   $D857
D82A-   E9 80       SBC   #$80
D82C-   90 11       BCC   $D83F
D82E-   C9 40       CMP   #$40
D830-   B0 14       BCS   $D846
D832-   0A          ASL
D833-   A8          TAY
D834-   B9 01 D0    LDA   $D001,Y
D837-   48          PHA
D838-   B9 00 D0    LDA   $D000,Y
D83B-   48          PHA
D83C-   4C B1 00    JMP   $00B1
D83F-   4C 46 DA    JMP   $DA46
```

```
D842-   C9 3A       CMP   #$3A
D844-   F0 BF       BEQ   $D805
D846-   4C C9 DE    JMP   $DEC9
D849-   38          SEC
D84A-   A5 67       LDA   $67
D84C-   E9 01       SBC   #$01
D84E-   A4 68       LDY   $68
D850-   B0 01       BCS   $D853
D852-   88          DEY
D853-   85 7D       STA   $7D
D855-   84 7E       STY   $7E
D857-   60          RTS
D858-   AD 00 C0    LDA   $C000
D85B-   C9 83       CMP   #$83
D85D-   F0 01       BEQ   $D860
D85F-   60          RTS
D860-   20 53 D5    JSR   $D553
D863-   A2 FF       LDX   #$FF
D865-   24 D8       BIT   $D8
D867-   10 03       BPL   $D86C
D869-   4C E9 F2    JMP   $F2E9
D86C-   C9 03       CMP   #$03
D86E-   B0 01       BCS   $D871
D870-   18          CLC
D871-   D0 3C       BNE   $D8AF
D873-   A5 B8       LDA   $B8
D875-   A4 B9       LDY   $B9
D877-   A6 76       LDX   $76
D879-   E8          INX
D87A-   F0 0C       BEQ   $D888
D87C-   85 79       STA   $79
D87E-   84 7A       STY   $7A
D880-   A5 75       LDA   $75
D882-   A4 76       LDY   $76
D884-   85 77       STA   $77
D886-   84 78       STY   $78
D888-   68          PLA
D889-   68          PLA
D88A-   A9 5D       LDA   #$5D
D88C-   A0 D3       LDY   #$D3
D88E-   90 03       BCC   $D893
D890-   4C 31 D4    JMP   $D431
D893-   4C 3C D4    JMP   $D43C
D896-   D0 17       BNE   $D8AF
*
```

– $D7D2-$D7D3 : sauvegarde du pointeur de pile en $F8.

– $D7D5 : la routine en $D858 vérifie si l'on a tapé CTRL-C (code clavier #$83) en cours d'exécution du programme. Si tel n'est pas le cas, et c'est ce qui nous intéresse ici, on revient simplement par RTS.

– $D7D8-$D7DA : on charge le pointeur de programme dans les registres A et Y. Lorsque l'on arrive en $D7D2 pour la première fois après le RUN, le compteur de programme contient $800 dans le cas standard (adresse de début du programme moins 1). De manière générale, chaque fois que l'on aborde $D7D2 pour exécuter une nouvelle instruction, $B8-$B9 contient l'adresse de l'instruction à exécuter, diminuée de 1.

– $D7DC à $D7DF : en $75-$76 se trouve stocké le numéro de la ligne en cours d'exécution, toujours dans l'ordre poids faible / poids fort. Si l'on est en mode immédiat, l'octet haut du numéro de ligne est remplacé par #$FF dans $76; par conséquent, si X contient la valeur de $76 et si l'on augmente X de 1, on obtiendra 0 si $76 contenait #$FF, cas du mode immédiat. Le test BEQ de l'adresse $D7DF provoquera donc le branchement en $D7E5 si l'on est en mode immédiat.
Bien que ce soit un peu hors sujet, précisons que l'exécution des ordres Basic donnés en mode immédiat passe également par la routine $D7D2, comme le montre les instructions ci-dessus. On utilise donc également la routine CHRGET en $B1, mais le "pointeur de programme" pointe alors sur les adresses des instructions dans le buffer d'entrée au clavier, et non plus dans la zone de stockage des programmes.

– $D7E1-$D7E3 : en $79-$7A se trouve toujours stockée, en mode programme, l'adresse de l'instruction à exécuter diminuée de 1. L'instruction CONT utilise cette adresse pour relancer l'exécution. On copie donc en $79-$7A le contenu de $B8-$B9.

– $D7E5 à $D7E9 : sans modifier la valeur du pointeur de programme, on examine l'octet pointé par ce dernier. Si l'on trouve 0, tout va bien, et c'est effectivement le cas lorsqu'on arrive ici juste après RUN puisque $800 contient 0.
Sinon, on saute en $D842, où l'on vérifie cette fois s'il s'agit de ":" (#$3A). Dans l'affirmative, on retourne dans la routine, mais dans le cas contraire, on saute à $DEC9 qui envoie une SYNTAX ERROR.
Conclusion : quand on arrive en $D7D2 pour exécuter une nouvelle instruction, $B8-$B9 doivent pointer sur l'octet qui précède l'instruction en question et cet octet doit être "0" ou ":" afin d'éviter une erreur de syntaxe. C'est un point qu'il convient de garder en mémoire si l'on veut introduire de nouvelles instructions.

–$D7EB à $D7F0 : nous sommes dans le cas où $B8-$B9 pointe sur un 0, ce qui peut correspondre à deux situations : on est au tout début du programme ($800) ou on est à la fin d'une ligne (marquée par un 0, comme rappelé plus haut). En tout état de cause, on doit avoir juste après le 0 l'adresse de la ligne suivante, l'octet de poids fort se trouvant deux octets plus loin que la position actuelle pointée par $B8-$B9 (LDY #$02 en $D7EB).
Comme l'adresse de début d'un programme ne peut se trouver en dessous de $800 si l'on veut éviter de traumatiser profondément l'Apple, l'octet de poids fort de l'adresse d'une instruction ne peut être à 0 que si elle correspond à la marque de fin de programme (voir supra), ou à l'absence de programme ($800-$801-$802 à zéro). Si tel est le cas, on saute en $D826 qui nous amène en $D88A. Sans entrer dans les détails, précisons que le JMP $D43C nous ramène finalement en mode immédiat, comme toujours en fin d'exécution de programme Applesoft.

– $D7F2 à $D7FA : il s'agit cette fois de passer à la ligne suivante. Aux index 3 et 4 par rapport à la position actuelle pointée par $B8-$B9, on trouve le numéro de cette ligne, que l'on stocke alors en $75-$76

(numéro de la ligne en cours d'exécution).

– $D7FC à $D803 : en ajoutant 4 (valeur de Y transférée dans A) à la valeur de $B8-$B9, on déplace le pointeur de programme qui contient maintenant l'adresse de la première instruction de la ligne, diminuée de 1.

– $D805 à $D81A : c'est là que l'on revient si l'on était arrivé dans $D7D2 en pointant sur un ":" et non sur un 0. Les deux cas de figure se rejoignent.
Si le contenu de $F2 est négatif, on est en mode TRACE, sinon on passe directement à la suite (BPL $D81D), comme on le fait également si l'on est en mode immédiat ($76 contient alors #$FF).
En mode TRACE, on affiche un # (code ASCII $23) suivi du numéro de la ligne (lu en $75-$76) au moyen des routines $DB5C (affiche le contenu de l'accumulateur), $ED24 (affiche le contenu de X suivi de celui de A) et $DB57 (affiche un espace).

– $D81D : appel de la routine $B1, qui va nous retourner le premier caractère de l'instruction.
Cette routine CHRGET met à jour le pointeur de programme et va lire dans la zone de stockage du programme Applesoft les octets de celui-ci. Elle joue un rôle fondamental dans le fonctionnement de l'Applesoft "standard" et peut également rendre de grands services pour l'appel de routines en langage machine à partir du Basic. Aussi en analyserons nous en détail le contenu, au moyen du listing ci-dessous.

```
00B1-   E6 B8       INC   $B8
00B3-   D0 02       BNE   $00B7
00B5-   E6 B9       INC   $B9
00B7-   AD 05 02    LDA   $0205
00BA-   C9 3A       CMP   #$3A
00BC-   B0 0A       BCS   $00C8
00BE-   C9 20       CMP   #$20
00C0-   F0 EF       BEQ   $00B1
00C2-   38          SEC
00C3-   E9 30       SBC   #$30
00C5-   38          SEC
00C6-   E9 D0       SBC   #$D0
00C8-   60          RTS
00C9-   80          ???
00CA-   4F          ???
00CB-   C7          ???
00CC-   52          ???
00CD-   00          BRK
00CE-   00          BRK
00CF-   00          BRK
```

–$B1 à $B7 : incrémentation du pointeur de programme $B8-$B9, selon une méthode déjà décrite dans les articles précédents (l'octet de poids fort est incrémenté lorsque l'octet de poids faible passe à 0), et chargement de l'octet pointé par $B8-$B9 dans l'accumulateur.
Notez bien ici l'astuce des programmeurs qui ont placé le pointeur à l'intérieur même de la routine, juste après le code de l'instruction LDA.
Ainsi, la routine se modifie elle-même : on aura toujours en $B8-$B9 l'adresse de l'octet que l'on veut lire et que l'on retrouvera dans l'accumulateur après le LDA.
Dans notre listing, l'instruction en $B7 est LDA $0205 ; cela provient de ce que la liste a été effectuée en mode immédiat, $B8-$B9 pointant alors toujours sur le buffer d'entrée-clavier, situé aux adresses $200 à $2FF. Après un RUN, en revanche, $B8-$B9 est initialisé à l'adresse de début du programme moins 1 ; dans la situation normale, on aura donc en $B7 l'instruction LDA $0800. Ensuite l'adresse dont on charge le contenu dans A peut varier de deux façons : soit séquentiellement à chaque appel de $B1 (qui incrémente $B8-$B9), soit par manipulation directe (calcul) des valeurs de $B8-$B9, comme on l'a vu plus haut aux adresses $D7FC à $D803.
Il est important de noter que chaque modification de $B8-$B9 revient en fait à changer l'octet sur lequel portera l'analyse de l'interpréteur de l'Applesoft, soit à l'intérieur de l'examen d'une instruction, soit au moment de l'exécution d'une nouvelle instruction (voir les tests du début de la routine $D7D2). Si l'on veut par conséquent ajouter des instructions à l'Applesoft, sans le perturber outre mesure (le seuil de tolérance étant marqué par l'apparition d'une SYNTAX ERROR), il convient de vérifier que les valeurs de $B8-$B9 et de l'accumulateur sont convenables pour le Basic avant de lui rendre la main. C'est un point sur lequel nous reviendrons plus loin au travers de quelques exemples.

– $BA-$BC : le contenu de l'accumulateur, c'est-à-dire l'octet que l'on vient de lire dans le programme, est comparé à #$3A, code ASCII du ":", forme sous laquelle sont stockés les signes ":" que vous utilisez pour séparer plusieurs instructions sur une même ligne. Le test BCS provoque le branchement en $C8 (RTS) si A est supérieur ou égal à #$3A, ce qui sera notamment le cas pour les tokens de l'Applesoft et les lettres constituant le premier caractère d'un nom de variable.

– $BE-$CO : #$20 est le code ASCII de l'espace, forme sous laquelle il est stocké dans un programme. Si le caractère que l'on vient de lire est un espace, on l'ignore pour passer directement au caractère suivant en incrémentant le pointeur de programme.
Cela ne signifie pas toutefois, par exemple, que les espaces placés entre guillemets après un PRINT ne seront pas pris en compte, puisque vous pouvez constater aisément qu'ils sont respectés à l'affichage ou à l'impression. C'est en effet la routine qui effectue le traitement correspondant à un PRINT qui se charge de lire la donnée à imprimer et d'utiliser les espaces, et le branchement sur cette routine, comme nous le verrons plus loin, s'effectuera toujours avant que l'on ait rencontré et ignoré le premier espace dans une chaîne de caractères en le lisant par CHRGET ($B1). Ne sont donc ainsi effectivement délaissés que les espaces lus dans un programme dans le cadre de l'interprétation d'instructions pour lesquelles ils ne sont pas significatifs (espaces séparant l'instruction DATA du premier item de DATA, par exemple).

– $C2 à $C6 : nous sommes dans le cas où le contenu de l'accumulateur est inférieur à #$3A et différent de #$20. On procède alors à deux soustractions successives, mais à l'issue desquelles le contenu de l'accumulateur n'aura pas changé.
En effet, si l'on additionne #$30 (48 en décimal) et #$D0 (208 en décimal), on obtient #$100 (256). On ne manipule ici que des octets, raison pour laquelle on procède à la soustraction en deux étapes, mais puisque la retenue est toujours remise à 1 (SEC) avant l'opération, on ne retire finalement de l'accumulateur que le poids faible de # $30+#$D0, soit 00, et l'accumulateur retrouvera donc sa valeur initiale à l'issue du calcul.
En fait, cette manipulation apparemment neutre permet d'établir une différence entre les chiffres (codes ASCII de #$30 à #$39) et les autres caractères de code ASCII inférieur à #$30 :

- Si A est supérieur ou égal à #$30, A – #$30 sera inférieur à #$D0 en valeur absolue. Par suite, à l'issue de la soustraction (A – #$30) – #$D0, le bit de retenue C sera positionné à 0 (voir l'article consacré à SBC pour le résultat de la soustraction d'un octet à un autre plus petit que lui).
- Si A est inférieur à #$30, A – #$30 sera supérieur ou égal à #$D0 en valeur absolue, et le bit C sera positionné à 1 après (A – #$30) – #$D0.

En résumé, lorsque l'on sort de la routine CHRGET, la valeur du bit C dépend de la nature du caractère que l'on vient de lire dans le programme :

- C=1 si A est supérieur ou égal à #$3A (:), puisque l'on sort au test BCS de l'adresse $BC
- C=0 si A contient un chiffre (codes ASCII #$30 à #$39)
- C=1 si A contient un caractère de code ASCII inférieur à #$30

Ces différences seront ensuite exploitées par l'interpréteur et les routines correspondant à chaque instruction

de l'Applesoft pour en vérifier la bonne syntaxe.

Par ailleurs, le bit Z joue également un rôle, sur lequel nous reviendrons, en sortie de la routine $B1. Deux cas peuvent conduire à Z=1 :

- Le caractère lu est ":", ce qui amène Z=1 lors de l'instruction CMP #$3A.
- Le caractère lu est un 0 (fin de ligne Applesoft), ce qui provoque également Z=1 à l'issue de la dernière soustraction SBC #$D0, puisque le résultat obtenu alors est à nouveau 0.

Maintenant que nous savons ce qui se passe dans la routine CHRGET, nous pouvons ressortir de notre JSR $B1 de l'adresse $D81D et retourner dans l'interpréteur à l'adresse suivante, soit $D820. Celle-ci nous dirige aussitôt vers une autre sous-routine par un JSR $D828.

– $D828 : l'accumulateur contient toujours l'octet que l'on vient de lire dans le programme par le JSR $B1. Les indicateurs du registre d'état sont positionnés en fonction des résultats obtenus dans cette même routine $B1. En particulier, le bit Z est à 1 si l'octet lu est 0 ou ":" (voir ci-dessus). Le test BEQ branche donc sur $D857 (RTS) si l'on vient de lire l'octet de fin d'une ligne (0) ou le séparateur d'instructions (:). Dans l'un ou l'autre cas, on revient donc de suite en $D823 pour passer à l'analyse de l'instruction suivante, si elle existe, par un JMP $D7D2.

– $D82A-$D82C : les tokens de l'Applesoft vont de #$80 à #$EA. De plus, si c'est un token que l'on vient de lire par $B1, le bit C est à 1 et, par l'instruction SBC #$80, on ne retire donc que #$80 de la valeur du token (l'opposé de la retenue vaut 0). Si l'accumulateur contient un token, on aura donc toujours C=1 après la soustraction, d'où échec du test BCC.
On ira ainsi en $D83F si A n'est pas un token (nom de variable, constante, caractère de contrôle...) pour sauter en $DA46, routine de l'Applesoft traitant l'affectation des valeurs aux variables (c'est elle qui provoquera par la suite une erreur de syntaxe si cette dernière n'est pas correcte).

– $D82E-$D830 : c'est un token que l'on a lu par $B1 et, après le SBC #$80, l'accumulateur contient une valeur comprise entre 0 et #$6A. Après CMP #$40, on aura C=1 si A est supérieur ou égal à #$40; dans ce cas, BCS branche sur $D846 qui va nous envoyer SYNTAX ERROR.
Si l'on regarde la liste des tokens, on constate que le dernier qui passera les tests des adresses $D82A à $D830 sans provoquer d'erreur est NEW (code #$BF). Tous les tokens dont le code est supérieur à #$BF sortiront en erreur au BCS de l'adresse $D830. Or, les dits tokens (par exemple TO, FN, THEN...) sont en quelque sorte "non autonomes", en ce sens qu'ils doivent toujours être précédés d'un autre et ne viennent jamais au début d'une instruction. C'est donc la routine de traitement du token "maître" (par exemple FOR, DEF, IF...) qui se chargera ultérieurement de les "récupérer" dans le programme. Ceci explique pourquoi l'interpréteur envoie une SYNTAX ERROR si l'on arrive en $D828, alors que l'on commence seulement l'analyse d'une instruction, avec un token de code supérieur à #$BF.

– $D832 à $D83B : l'accumulateur contient le code d'un token valable en début d'instruction, mais diminué de #$80, soit une valeur comprise entre 0 et #$3F.
A partir de $D000 sont stockées sur deux octets, dans l'ordre poids faible / poids fort, les adresses de début, moins 1, des routines machine correspondant au traitement réalisé par chaque token de l'Applesoft. En $D000 - $D001, par exemple, on trouve l'adresse moins 1 de la routine du premier token dans l'ordre des codes, c'est-à-dire END; en $D002 - $D003, on trouve l'adresse moins 1 de la routine du second token FOR...
Prenons l'exemple concret de l'instruction HOME dont le token est #$97. On obtient son numéro dans la liste des tokens en lui retirant #$80, ce qui nous donne #$17. Comme chaque adresse de routine occupe deux octets, il faut multiplier #$17 par 2 pour savoir à quelle distance (index) du début de la table nous trouverons la routine de HOME. En effet, la routine du premier token (dont le numéro dans la liste est 0) est donnée en $D000+ 0 et $D000+1; la routine du second (numéro 1) en $D000+2 et $D000+3, celle du troisième (numéro 2) en $D000+4 et $D000+5 ... De façon générale, si T est le token d'une instruction Applesoft, N = T – #$80 est son numéro dans la liste des tokens et si A = 2 * N, on trouvera l'adresse de la routine machine correspondante (toujours diminuée de 1) aux adresses $D000+A et $D000+A+1. Dans le cas de HOME, A = 2 * #$17 = #$2E. Aux adresses $D02E et $D02F on trouve 57 FC, ce qui nous donne comme adresse finale $FC58 après ajout de 1. Or $FC58 (ou encore −936 en décimal) est l'adresse de la routine du moniteur qui efface tout l'écran et positionne le curseur en haut et à gauche. On voit ainsi que HOME, par l'intermédiaire de l'interpréteur, permet finalement d'appeler une routine en langage machine qui assure le traitement voulu, en l'occurrence le nettoyage de l'écran.
Revenons maintenant aux instructions de l'interpréteur qui réalisent cet appel de routine. Quand on arrive en $D832, on a en fait dans l'accumulateur le numéro du token dans la liste et, par ASL, on multiplie ce numéro par 2 avant de le charger dans le registre Y par TAY. Par LDA $D001, Y et PHA, on empile l'octet fort de l'adresse de la routine; par LDA $D000, Y et PHA, on empile son poids faible. Plus précisément, on a maintenant au sommet de la pile l'adresse de la routine moins 1.

– $D83C : par JMP $B1, on va lire le caractère qui suit le token que l'on vient d'interpréter. De plus, comme on utilise JMP et non JSR, le sommet de la pile est toujours constitué par l'adresse de la routine du token moins 1.
Par suite, lorsqu'on sort de CHRGET par RTS, le processeur dépile les deux octets du sommet, leur ajoute 1 (voir articles précédents) et saute à l'adresse ainsi obtenue. On se branche donc sur la routine de traitement du token, avec dans l'accumulateur le caractère qui suit le-dit token dans le programme. Le tour est joué et cela vous explique pourquoi la table débutant en $D000 contient les adresses des routines diminuées de 1 et non les adresses exactes.
N'oublions pas cependant que l'on est toujours sous le contrôle du JSR $D828 de l'adresse $D820; lorsqu'on arrive dans la routine du token, on a 22 D8 au sommet de la pile (adresse de retour $D823, diminuée de 1 et empilée avec le poids faible au sommet). D'une façon ou d'une autre, c'est-à-dire soit par un RTS, soit par un JMP $B1 pour utiliser le RTS de l'adresse $C8 par exemple, cette adresse sera utilisée au retour de la routine du token et l'on reviendra en $D823 pour exécuter un JMP $D7D2 et aborder l'analyse d'une autre instruction. Rappelons qu'il faut alors, pour éviter les erreurs, que $B8-$B9 pointe effectivement sur un 0 ou un ":". Dans le cadre de l'ajout d'instructions à l'Applesoft, le fait que leur analyse se fasse à l'issue d'un JSR $D828 et qu'il faille en revenir en pointant sur 0 ou ":" peut réclamer certaines précautions, qu'illustreront les exemples donnés plus loin.

– $D849 à $D857 : bien qu'elle ne fasse pas directement partie de notre sujet, précisons que cette routine correspond au traitement effectué par l'instruction RESTORE. Elle met le pointeur de DATA ($7D-$7E) à l'adresse de début du programme moins 1 (contenu de $67-$68 diminué de 1).

Utilisation de &

Rappelons tout d'abord que & est une instruction de l'Applesoft, dont le token est codé #$AF. Il s'agit donc d'un token valide en début d'instruction, et l'accès aux routines machine correspondant à & suivra le même processus que pour tout autre token. C'est par le biais de $D7D2 et de la sous-routine débutant en $D828 que se fera l'appel.
Dans la table $D000 des adresses de routines, les valeurs données pour & sont F4 03, et l'utilisation de & conduira finalement à un branchement sur $3F5. C'est à ce niveau que peut intervenir le programmeur, en mettant aux adresses $3F5 - $3F6 - $3F7 une instruction JMP $XXYY, soit 4C YY XX en langage machine, où YY désigne le poids faible de l'adresse où commence la routine d'exécution des instructions précédées de &, et XX le poids fort de cette même adresse.
Par exemple, si l'on met en $3F5 les codes 4C 00 03, l'interpréteur de l'Applesoft sautera à l'adresse $300 chaque fois qu'il rencontrera un & dans le programme. Si l'on met 4C 00 80, le saut se fera à l'adresse $8000... La routine concernée (en $300, ou en $8000...) doit alors se charger de lire la suite de l'instruction dans le programme et de réaliser le traitement demandé.

Exemple d'application

Supposons que le fonctionnement de certaines instructions de l'Applesoft ne soit pas conforme à nos voeux et que, par ailleurs, nous voulions lui en adjoindre quelques autres. Par hypothèse, nous savons comment écrire en assembleur les routines nécessaires et nous utiliserons & pour contraindre l'Applesoft à leur faire appel chaque fois que bon nous semblera...

La liste des adaptations à réaliser est la suivante :

- PRINT devra toujours effacer la ligne sur laquelle on veut afficher quelque chose avant de procéder à cet affichage. L'instruction sera &PRINT et utilisera exactement la même syntaxe que PRINT (&PRINT A, &PRINT ZZ$, &PRINT "TEXTE"; ...).
- Dans les instructions GET et INPUT, rebaptisées &GET et &INPUT, on refusera tout caractère de contrôle à l'exception de RETURN. Le reste de la syntaxe sera inchangé.
- L'instruction HOME, soit &HOME, effacera tout l'écran, quelles que soient les limites de la fenêtre.
- L'instruction NEW, soit &NEW, réalisera en revanche le traitement d'un HOME standard, c'est à dire effacement dans les limites de la fenêtre.
- L'instruction LET, soit &LET, fonctionnera comme un PRINT précédé d'un double beep. Ainsi, &LET "TEXTE", par exemple, produira un double beep et affichera TEXTE à l'écran.

Voyons maintenant comment programmer tout cela en analysant le listing source de la routine donné ci-après.

Version & : lisa 1.5

```
0300                 1            ORG $300
0300 A94C            2            LDA #$4C
0302 8DF503          3            STA $3F5
0305 A910            4            LDA #DEB
0307 8DF603          5            STA $3F6
030A A903            6            LDA /DEB
030C 8DF703          7            STA $3F7
030F 60              8  FIN       RTS
0310 F0FD            9  DEB       BEQ FIN
0312 C9BA           10            CMP #$BA
0314 D00E           11            BNE S0
0316 A424           12            LDY $24
0318 8406           13            STY $6
031A 209CFC         14            JSR $FC9C
031D A406           15            LDY $6
031F 8424           16            STY $24
0321 4C9E03         17            JMP M0
0324 C984           18  S0        CMP #$84
0326 F004           19            BEQ S1
0328 C9BE           20            CMP #$BE
032A D01F           21            BNE S3
032C A93A           22  S1        LDA #ENTREE
032E 8538           23            STA $38
0330 A903           24            LDA /ENTREE
0332 8539           25            STA $39
0334 20EA03         26            JSR $3EA
0337 4C9E03         27            JMP M0
033A AD00C0         28  ENTREE    LDA $C000
033D 10FB           29            BPL ENTREE
033F 2C10C0         30            BIT $C010
0342 C98D           31            CMP #$8D
0344 F004           32            BEQ S2
0346 C9A0           33            CMP #$A0
0348 90F0           34            BCC ENTREE
034A 60             35  S2        RTS
034B C997           36  S3        CMP #$97
034D D01D           37            BNE S4
034F A003           38            LDY #3
0351 B92000         39  B0        LDA $20,Y
0354 48             40            PHA
0355 88             41            DEY
0356 10F9           42            BPL B0
0358 2039FB         43            JSR $FB39
035B 2058FC         44            JSR $FC58
035E A000           45            LDY #0
0360 68             46  B1        PLA
0361 992000         47            STA $20,Y
0364 C8             48            INY
0365 C004           49            CPY #4
0367 D0F7           50            BNE B1
0369 4C8503         51            JMP S6
036C C9BF           52  S4        CMP #$BF
036E D006           53            BNE S5
0370 2058FC         54            JSR $FC58
0373 4C8503         55            JMP S6
0376 C9AA           56  S5        CMP #$AA
0378 D00B           57            BNE S6
037A 20DDFB         58            JSR $FBDD
037D 20DDFB         59            JSR $FBDD
0380 A9BA           60            LDA #$BA
0382 4CA103         61            JMP M1
0385 08             62  S6        PHP
0386 A476           63            LDY $76
0388 C0FF           64            CPY #$FF
038A D00B           65            BNE A1
038C A01B           66            LDY #$1B
038E 8438           67            STY $38
0390 A0FD           68            LDY #$FD
0392 8439           69            STY $39
0394 20EA03         70            JSR $3EA
0397 28             71  A1        PLP
0398 20B100         72            JSR $B1
```

```
039B 4C1003    73          JMP DEB
039E 20B700    74    M0    JSR $B7
03A1 BA        75    M1    TSX
03A2 E8        76          INX
03A3 E8        77          INX
03A4 9A        78          TXS
03A5 4C20D8    79          JMP $D820
```

```
*300.3A7

0300- A9 4C 8D F5 03 A9 10 8D
0308- F6 03 A9 03 8D F7 03 60
0310- F0 FD C9 BA D0 0E A4 24
0318- 84 06 20 9C FC A4 06 84
0320- 24 4C 9E 03 C9 84 F0 04
0328- C9 BE D0 1F A9 3A 85 38
0330- A9 03 85 39 20 EA 03 4C
0338- 9E 03 AD 00 C0 10 FB 2C
0340- 10 C0 C9 8D F0 04 C9 A0
0348- 90 F0 60 C9 97 D0 1D A0
0350- 03 B9 20 00 48 88 10 F9
0358- 20 39 FB 20 58 FC A0 00
0360- 68 99 20 00 C8 C0 04 D0
0368- F7 4C 85 03 C9 BF D0 06
0370- 20 58 FC 4C 85 03 C9 AA
0378- D0 0B 20 DD FB 20 DD FB
0380- A9 BA 4C A1 03 08 A4 76
0388- C0 FF D0 0B A0 1B 84 38
0390- A0 FD 84 39 20 EA 03 28
0398- 20 B1 00 4C 10 03 20 B7
03A0- 00 BA E8 E8 9A 4C 20 D8
```

```
 LOAD INI16.2.TEST
]LIST

5  PRINT  CHR$ (4)"BRUN INI16.2.OBJ"
10  TEXT : &  HOME : FOR I = 1 TO 10: &
      PRINT "**************************
      ************": NEXT : VTAB 2: HTAB
       1: &  PRINT "REECRITURE SUR CETTE
       LIGNE";
20  &  PRINT : &  PRINT "ENTREZ QUELQUE C
      HOSE";: &  INPUT " ? ";Z$: &  PRIN
      T "ENCORE EN 'GET'";: &  GET Y$: P
      RINT : &  PRINT Z$,Y$: &  PRINT ;:
       &  GET Z$
30  POKE 34,5: POKE 33,20: &  HOME : VTAB
       20: HTAB 1: &  LET "MESSAGE BRUYA
      NT.............................
      .";: &  GET Z$: PRINT : TEXT : FOR
       I = 1 TO 10: PRINT "************
      ************************": NEXT
40  POKE 34,18: POKE 33,10: POKE 35,22: V
      TAB 19: &  PRINT "AAAAAAAAAAAAAAA
      AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
      AAAAAAAAAA": GET Z$: &  NEW : GET
       Z$: &  HOME
```

– Lignes 2 à 8 : pour que les instructions & arrivent bien sur notre routine, il faut mettre en place le vecteur de l'&, c'est-à-dire donner à partir de l'adresse $3F5 un JMP à l'adresse voulue. Plutôt que d'utiliser des POKEs en Basic, nous confierons cette tâche aux toutes premières instructions de la routine assembleur.
On met tout d'abord #$4C (code de JMP) en $3F5 (lignes 2 et 3). L'assembleur LISA 1.5 utilisé ici offre en outre deux directives pour calculer les poids faible et fort d'adresses symboliques dont on ne connaît pas forcément l'implantation exacte au moment de l'écriture du source, ou dont l'implantation peut être modifiée en fonction de l'évolution du source lors de sa mise au point. DEB étant l'étiquette qui marque le début de la routine de traitement, #DEB donnera le poids faible de son adresse calculée au moment de l'assemblage et /DEB nous donnera son poids fort. Tous les assembleurs proposent de telles directives, avec des syntaxes légèrement différentes le cas échéant, mais que vous trouverez toujours dans la documentation de celui que vous utilisez (par exemple #< pour le poids faible et #> pour le poids fort en BIG MAC).
On met donc le poids faible de l'adresse en $3F6 (lignes 4 et 5) et son poids fort en $3F7 (lignes 6 et 7). En regardant la liste d'assemblage, on constate que DEB correspond finalement à l'adresse $310 et que ce sont bien #$10 et #$03 qui sont stockés en $3F6 et $3F7.
On sort ensuite par RTS pour revenir au Basic. Si le code objet de la routine est sauvé sur disquette, la mise en place du vecteur & pourra se faire simplement par un BRUN du fichier concerné en début de programme Applesoft. L'ensemble du code sera ainsi chargé en mémoire mais l'exécution s'arrêtera au RTS de la ligne 8, l'accès au reste de la routine, à partir de DEB, devant ultérieurement résulter de l'emploi de & (on aura en $3F5 : JMP $310 après le BRUN). Si vous rentrez ce code sans le sauver sur disquette, l'initialisation de & en début de programme pourra se faire au moyen de CALL 768.

– Ligne 9 : il est important de se rappeler ici que l'on arrive en DEB à l'issue de l'analyse d'un token. C'est donc le JMP $B1 de l'adresse $D83C, au sein de l'interpréteur, qui nous amène en $310. De ce fait, l'accumulateur contient déjà le caractère qui suit & dans le programme, caractère sur lequel pointe $B8-$B9, et les indicateurs du registre d'état sont positionnés en fonction de la nature du caractère lu par CHRGET. Le test BEQ sera donc positif si l'on a dans l'accumulateur un 0 ou un ":", ce qui marque la fin d'une instruction et la nécessité de ressortir de la routine. On reviendra alors en $D823, puis en $D7D2, et $B8-$B9 pointant sur 0 ou ":", l'exécution du programme Applesoft se poursuivra sans problème.

Lignes 10 à 17 : #$BA est le token de PRINT et si l'accumulateur contient #$BA nous sommes donc en train d'analyser une instruction &PRINT. Dans ce cas, on sauve la position horizontale du curseur (stockée en $24) en $6, on fait un saut à la routine du moniteur qui efface la fin de la ligne (JSR $FC9C, équivalent du CALL –868 que vous pouvez faire en Basic), on rétablit ensuite la position du curseur en $24 et on passe finalement en M0.
Nous reviendrons plus loin sur le traitement réalisé en M0, mais on peut noter dès maintenant que l'exécution de &PRINT n'est pas complète à l'issue des lignes 10 à 17, puisque l'on a bien effacé la fin de la ligne mais encore rien affiché. Dans la mesure où ce serait perdre notre temps que de ré-écrire les routines de PRINT de l'Applesoft, tout le problème consistera, lorsque nous serons en M0, à faire en sorte que l'interpréteur reprenne la main et traite le PRINT comme un PRINT standard du Basic, sans générer de message d'erreur.

–Lignes 18 à 27 : #$84 est le token de INPUT et #$BE celui de GET. Si l'on recontre une instruction &INPUT ou &GET, le traitement consiste à changer le vecteur de la routine d'entrée de caractères utilisée par le système (l'adresse de cette routine est donnée en $38-$39), comme nous l'avons déjà fait souvent dans les exemples illustrant les articles d'initiation précédents. Le JSR $3EA (ou encore CALL 1002) permet de faire prendre en compte cette modification par le DOS, qui supervise les entrées - sorties lorsqu'il est chargé.

Une fois le vecteur changé, on passe en M0. Là encore, le traitement réel de INPUT ou GET reste à faire, et nous en laisserons le soin à l'Applesoft...

– Lignes 28 à 35 : il s'agit de la routine d'entrée que nous voulons faire utiliser par le Basic dans ses INPUT et GET. Elle est particulièrement simple et vous n'aurez aucune peine à l'analyser.

– Lignes 36 à 51 : #$97 est le token de HOME. On empile les valeurs des limites de fenêtre, stockées aux adresses $20 à $23 (lignes 38 à 42). JSR $FB39 réalise le même traitement que l'instruction TEXT et JSR $FC58 réalise l'effacement de tout l'écran, puisque TEXT remet les limites de fenêtre à leurs valeurs maximales. On récupère ensuite les limites de fenêtre dans la pile (lignes 45 à 50) et on saute en S6.
Notez que, cette fois-ci, le traitement de l'instruction est terminé et on devrait donc pouvoir passer à l'instruction suivante du programme.

– Lignes 52 à 55 : #$BF est le token de NEW. On exécute un simple HOME standard (JSR $FC58) et, là encore, on doit passer à l'instruction suivante.

– Lignes 56 à 61 : #$AA est le token de LET. On fait deux appels consécutifs à $FBDD (émission d'un beep) puis on charge l'accumulateur avec le token de PRINT car il faudrait maintenant faire afficher le message par l'Applesoft, c'est-à-dire le tromper sur la nature de l'instruction et lui laisser finir le travail.

– Lignes 62 à 73 : c'est là que l'on arrive pour les instructions & dont le traitement est achevé par notre seule routine. Petite complication intermédiaire : remettre la routine d'entrée standard ($FD1B) en $38-$39 si l'on est en mode immédiat (signalé par un #$FF à l'adresse $76). Ainsi, lorsque le programme Applesoft est terminé, vous pourrez taper par exemple &HOME en mode immédiat pour rétablir la routine d'entrée standard et recouvrer l'usage normal des caractères de contrôle.
Si l'on est toujours en mode programme, par contre, il faut passer à l'instruction suivante. Pour ce faire, on lit le caractère suivant (JSR $B1) et on saute en DEB. Si la syntaxe est correcte, on doit avoir alors dans l'accumulateur un 0 ou un ":" et le test BEQ de DEB nous ramènera finalement en $D7D2 où l'exécution du programme se poursuivra correctement.

– Lignes 74 à 79 : on arrive en M0 dans le cas des instructions &PRINT, &INPUT ou &GET. Le JSR $B7 permet de remettre dans l'accumulateur, qui a pu être modifié plus haut, le token de PRINT, INPUT ou GET, puisque l'on relit l'adresse pointée par $B8-$B9, mais sans modifier ce pointeur.
On arrive en M1 dans le cas de &LET. Cette fois, on ne relit pas le token, car il faut au contraire le faire passer pour un PRINT, dont on a chargé le token dans l'accumulateur.
Il convient maintenant de rappeler que nous sommes arrivés en DEB à l'issue du JMP $B1 de l'adresse $D83C et que nous restons donc sous le contrôle du JSR $D828 dont l'adresse de retour occupe le sommet de la pile. Nous voulons faire reprendre l'exécution de PRINT, INPUT ou GET par l'Applesoft, comme s'il s'agissait d'instructions "normales" saisies par l'interpréteur dans le programme. L'idéal serait donc de venir se brancher en $D820, puisque c'est là que commence dans l'interpréteur l'analyse d'un token qu'il vient de lire et de charger dans l'accumulateur par le JSR $B1 de l'adresse $D81D. Pour que tout se passe bien par la suite, il faut donc effacer la trace du JSR $D828 dont nous ne pouvons revenir par RTS (sinon, le token n'est pas interprété, $B8-$B9 ne pointe pas sur 0 ou ":" et le résultat final est SYNTAX ERROR) et que nous ne pouvons pas davantage laisser au sommet de la pile avant de sauter en $D820 (sinon, la présence d'une adresse de retour excédentaire dans la pile finit par provoquer un mauvais branchement en retour de routine). C'est pourquoi on remonte le sommet de la pile de deux positions (TSX - INX - INX - TXS), afin de réaliser l'équivalent d'un POP et de faire "sauter" l'adresse de retour du JSR $D828 à l'extérieur de la pile. On peut alors retourner sans risque à l'interpréteur en $D820 et lui laisser terminer le travail.

Lorsque vous aurez saisi et sauvé cette routine de traitement d'instructions &, le programme de démonstration qui l'accompagne vous permettra d'en tester le fonctionnement.

Manipulation de CHRGET

L'appel de routines machine par & passe par l'interpréteur de l'Applesoft et comporte certaines contraintes : mise en place du vecteur en $3F5, respect du contrôle exercé par l'interpréteur sur l'analyse des instructions, nécessité, en cas de modification du traitement réalisé par certaines instructions du Basic, de modifier les programmes existants (remplacer ainsi les PRINT par des &PRINT...).
Il est possible de procéder autrement, et de prendre même le pas sur l'interpréteur, en exploitant le fait que la routine d'acquisition des caractères du programme Applesoft, CHRGET ($B1), se trouve en RAM et peut donc subir une modification de la part du programmeur. Après avoir chargé le caractère pointé par $B8-$B9 dans l'accumulateur en $B7, et plutôt que de retourner de suite à l'interpréteur, il faut alors placer un saut dans une routine "personnelle", une sorte de mini-interpréteur, qui traitera nos instructions en priorité et en exclusivité.

La liste ci-dessous donne un exemple de CHRGET modifié : en $BA est implanté un JMP $300, et en $300 devra commencer notre mini-interpréteur. Nous verrons qu'il faut veiller dans cette routine à respecter le traitement normal de CHRGET pour les caractères qui ne nous concernent pas (comparaison à #$3A, soustractions de #$30 et #$D0...) afin de ne pas perturber l'Applesoft.

```
0081-   E6 B8     INC   $B8
0083-   D0 02     BNE   $0087
0085-   E6 B9     INC   $B9
0087-   AD 05 02  LDA   $0205
008A-   4C 00 03  JMP   $0300
008D-   0A        ASL
008E-   C9 20     CMP   #$20
00C0-   F0 EF     BEQ   $0081
00C2-   38        SEC
00C3-   E9 30     SBC   #$30
00C5-   38        SEC
00C6-   E9 D0     SBC   #$D0
00C8-   60        RTS
00C9-   80        ???
00CA-   4F        ???
00CB-   C7        ???
00CC-   52        ???
00CD-   00        BRK
00CE-   00        BRK
00CF-   00        BRK
```

Pour mettre en place ce JMP, il est exclu d'utiliser des POKEs à partir du Basic : les POKEs sont en effet interprétés, bien sûr, et lus en mémoire par des appels à $B1. Si vous modifiez par POKEs la routine qui est en train de lire ces mêmes POKEs, il est évident que vous n'arriverez jamais au bout !
Vous disposez en fait de trois solutions :

- En mode immédiat, mettre le saut en place à partir du moniteur en tapant par exemple BA: 4C 00 03. Cette solution n'est à retenir que pour un premier test.
- Après avoir rentré le saut sous moniteur, sauver les codes correspondants dans un fichier par BSAVE Fichier, A$BA, L3. Vous pourrez alors le récupérer simplement au début de vos programmes Applesoft par BLOAD Fichier. Pour revenir facilement au Basic standard, il est alors pratique de disposer d'un second fichier, sauvé de la même façon, mais contenant les 3 codes du CHRGET "normal".
- Utiliser une petite routine machine, chargée et exécutée en début de programme, pour mettre les codes en place (le source d'une telle routine serait par

exemple : LDA #$4C - STA $BA - LDA #$00 - STA $BB - LDA #$03 - STA $BC - RTS). Cette méthode est moins pratique que la précédente, mais elle sera inévitable si vous travaillez sous ProDos, qui ne prend pas en compte le chargement d'un fichier directement dans CHRGET.

Exemple d'application

Nous reprendrons exactement le même problème que celui traité plus haut au moyen de &. Cette fois, l'appel de routines se fera par un JMP $300 implanté en $BA. Il n'est pas nécessaire de modifier la syntaxe des instructions Applesoft concernées (nous interprèterons les PRINT, GET et autres avant même que l'Applesoft ne le fasse).

Voyons maintenant quelles sont les différences entre cette nouvelle routine d'interprétation des instructions et la précédente.

– Ligne 9 (PRINT) : il n'y aura plus de JSR $B7 en fin de routine et on recharge directement l'accumulateur avec le token de PRINT.

– Lignes 15 et 21 (INPUT et GET) : même chose que ci-dessus. On empile le token de l'instruction avant de modifier A et on le dépile avant de passer à la fin de la routine.

– Ligne 46 (HOME) : le traitement de l'instruction est assuré totalement par notre routine et terminé lorsque l'on arrive à cette ligne 46. Il s'agit donc cette fois de passer à l'instruction suivante et d'acquérir le caractère qui suit le token de HOME dans le programme (normalement 0 ou ":"). Nous sommes toujours sous le contrôle du JSR $B1 de l'adresse $D81D, dont l'adresse de retour est au sommet de la pile. Par JMP $B1 (et non JSR $B1) on lit le caractère suivant en mettant à jour le pointeur de programme et, au prochain RTS, on revient en $D820. Comme A contient normalement 0 ou ":", on ressort de $D828 par le BEQ $D857 et on passe à l'instruction suivante (JMP $D7D2) en pointant sur un caractère valide.

– Ligne 50 (NEW) : même processus que ci-dessus.

– Lignes 56 à 72 (sortie après interprétation de nos instructions, ou sortie directe si le caractère ne nous concerne pas). Pour que tout se passe bien, il faut effectuer les mêmes traitements que le CHRGET "normal" dans sa partie $BA à $C8. A partir de $BE, CHRGET n'est plus affecté par la modification des codes et l'on pourra donc le réutiliser tel quel.
Il nous manque donc le CMP #$3A et le branchement alternatif BCS qui en résulte, que nous remettons aux lignes 56 et 57. Si le test BCS est positif, on doit quitter la routine sans autres manipulations de A ou du registre d'état mais il nous faut malgré tout rétablir la routine d'entrée standard en $38-$39 si l'on est en mode immédiat (registre d'état et accumulateur sont alors empilés avant modification et dépilés avant RTS). Dans le cas de BCS A0, le RTS de la ligne 72 nous renvoie en $D820.
Si le test BCS échoue, il faut en revanche poursuivre les manipulations, ce que nous ferons, après rétablissement éventuel de la routine d'entrée, par un JMP $BE. Dans ce cas, c'est le RTS de l'adresse $C8 qui nous ramènera en $D820.

Notez bien ce qui se passe dans ce système lorsqu'on utilise un token pour réaliser un traitement préparatoire (PRINT, GET, INPUT ou LET) et que l'exécution de la routine du token est laissée à l'Applesoft lui-même. Le traitement préparatoire est cette fois-ci réalisé avant que ne commence l'interprétation du token par le Basic ("entre" les adresses $D81D et $D820) et l'on rend ensuite la main à l'interpréteur comme s'il ne s'était rien passé (l'accumulateur contient le token, sur lequel pointe $B8-$B9 et les indicateurs du registre d'état sont positionnés en conséquence). Il n'est donc plus nécessaire de gérer la pile pour tromper l'interpréteur sur le point auquel il est parvenu dans son analyse.

Vous trouverez ci-après un petit programme de démonstration, comparable à celui utilisé avec &. Il est écrit avec des instructions Applesoft tout à fait banales, et vous pourrez donc l'exécuter sous contrôle du CHRGET normal ou sous contrôle du CHRGET modifié. Ne soyez pas surpris toutefois si, dans la première hypothèse, le résultat obtenu n'est pas tout à fait conforme aux espérances...

■

Version chrget : lisa 1.5

```
0300                 1          ORG $300
0300 C9BA            2          CMP #$BA
0302 D010            3          BNE S0
0304 A424            4          LDY $24
0306 8406            5          STY $6
0308 209CFC          6          JSR $FC9C
030B A406            7          LDY $6
030D 8424            8          STY $24
030F A9BA            9          LDA #$BA
0311 4C7403         10          JMP S6
0314 C984           11   S0     CMP #$84
0316 F004           12          BEQ S1
0318 C9BE           13          CMP #$BE
031A D021           14          BNE S3
031C 48             15   S1     PHA
031D A92C           16          LDA #ENTREE
031F 8538           17          STA $38
0321 A903           18          LDA /ENTREE
0323 8539           19          STA $39
0325 20EA03         20          JSR $3EA
0328 68             21          PLA
0329 4C7403         22          JMP S6
032C AD00C0         23   ENTREE LDA $C000
032F 10FB           24          BPL ENTREE
0331 2C10C0         25          BIT $C010
0334 C98D           26          CMP #$8D
0336 F004           27          BEQ S2
0338 C9A0           28          CMP #$A0
033A 90F0           29          BCC ENTREE
033C 60             30   S2     RTS
033D C997           31   S3     CMP #$97
033F D01D           32          BNE S4
0341 A003           33          LDY #3
0343 B92000         34   B0     LDA $20,Y
0346 48             35          PHA
0347 88             36          DEY
0348 10F9           37          BPL B0
034A 2039FB         38          JSR $FB39
034D 2058FC         39          JSR $FC58
0350 A000           40          LDY #0
0352 68             41   B1     PLA
0353 992000         42          STA $20,Y
0356 C8             43          INY
0357 C004           44          CPY #4
0359 D0F7           45          BNE B1
035B 4CB100         46          JMP $B1
035E C9BF           47   S4     CMP #$BF
0360 D006           48          BNE S5
0362 2058FC         49          JSR $FC58
0365 4CB100         50          JMP $B1
0368 C9AA           51   S5     CMP #$AA
036A D008           52          BNE S6
036C 20DDFB         53          JSR $FBDD
036F 20DDFB         54          JSR $FBDD
0372 A9BA           55          LDA #$BA
0374 C93A           56   S6     CMP #$3A
0376 B006           57          BCS A0
0378 207E03         58          JSR A0
037B 4CBE00         59          JMP $BE
037E 08             60   A0     PHP
037F A476           61          LDY $76
0381 C0FF           62          CPY #$FF
0383 D00D           63          BNE A1
0385 A01B           64          LDY #$1B
0387 8438           65          STY $38
0389 A0FD           66          LDY #$FD
038B 8439           67          STY $39
038D 48             68          PHA
038E 20EA03         69          JSR $3EA
0391 68             70          PLA
0392 28             71   A1     PLP
0393 60             72   A2     RTS
```

```
*300.393

0300- C9 BA D0 10 A4 24 84 06
0308- 20 9C FC A4 06 84 24 A9
0310- BA 4C 74 03 C9 84 F0 04
0318- C9 BE D0 21 48 A9 2C 85
0320- 38 A9 03 85 39 20 EA 03
0328- 68 4C 74 03 AD 00 C0 10
0330- FB 2C 10 C0 C9 8D F0 04
0338- C9 A0 90 F0 60 C9 97 D0
0340- 1D A0 03 B9 20 00 48 88
0348- 10 F9 20 39 FB 20 58 FC
0350- A0 00 68 99 20 00 C8 C0
0358- 04 D0 F7 4C B1 00 C9 BF
0360- D0 06 20 58 FC 4C B1 00
0368- C9 AA D0 08 20 DD FB 20
0370- DD FB A9 BA C9 3A B0 06
0378- 20 7E 03 4C BE 00 08 A4
0380- 76 C0 FF D0 0D A0 1B 84
0388- 38 A0 FD 84 39 48 20 EA
0390- 03 68 28 60
```

```
]LIST:INI16.1.TEST

10  TEXT : HOME : FOR I = 1 TO 10: PRINT
      "**********************************
      ******": NEXT : VTAB 2: HTAB 1: PR
      INT "REECRITURE SUR CETTE LIGNE";
20  PRINT : PRINT "ENTREZ QUELQUE CHOSE";
      : INPUT " ? ";Z$: PRINT "ENCORE EN
      'GET'";: GET Y$: PRINT : PRINT Z$
      ,Y$: PRINT ;: GET Z$
30  POKE 34,5: POKE 33,20: HOME : VTAB 20
      : HTAB 1: LET "MESSAGE BRUYANT....
      ..............................";: G
      ET Z$: PRINT : TEXT : FOR I = 1 TO
       10: PRINT "**********************
      ***************": NEXT
40  POKE 34,18: POKE 33,10: POKE 35,22: V
      TAB 19: PRINT "AAAAAAAAAAAAAAAAAAA
      AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
      AAAAAAA": GET Z$: NEW : GET Z$: H
      OME
```

Personnalisez vos disquettes Macintosh

Jean-Luc Bazanegue

Nous ne vous apprendrons rien en vous disant que, à chaque fois que l'on introduit une disquette d'amorçage pour initialiser le Macintosh, on obtient à l'écran l'affichage temporaire d'un dessin. La plupart du temps, ce dessin est limité au simple message "Bienvenue.", accompagné d'un icône représentant le Mac. En fait, tout un chacun peut remplacer cela par une image personnelle, à condition de disposer des logiciels "Macpaint" et "MS Basic". S'il n'est pas nécessaire de comprendre le fonctionnement du système pour utiliser le programme que nous vous proposons ici, il nous semble important de vous donner quelques explications, ne serait-ce que pour vous permettre d'adapter le principe à d'autres applications.

Structure d'un document "Macpaint"

Le plus grand document Macpaint que l'on puisse obtenir correspond, à peu de choses près, à une feuille de papier format A4. Un rapide calcul nous permet de savoir que ce document est constitué par 720 lignes de 576 points (soit 72 octets par ligne). Si l'image était sauvegardée sur la disquette sans modification, cela nous donnerait un fichier de 576 / 8 * 720 = 51840 octets, ce qui n'arrive jamais. Donc, l'image est codée. Après de nombreux essais et comprimés contre les maux de tête, nous avons réussi à déterminer la nature exacte d'un tel fichier.

Bloc de tête

Chaque fichier commence par un bloc de 512 octets qui n'a pas de rapport direct avec l'image :

- Les quatre premiers octets indiquent si les motifs standards placés au bas de l'écran Macpaint ont été redéfinis ou non. $00000000 = motifs standards; $00000002 = un ou plusieurs motifs redéfinis par l'utilisateur.
- Les trois cent quatre octets suivants constituent la définition des motifs (8 octets sont nécessaires à la représentation d'un motif - voir "Les routines en ROM du Macintosh", Pom's 15 - et Macpaint affiche 38 motifs).
- Les deux cent quatre octets restants ne semblent pas être signifiants puisqu'ils sont toujours à zéro, quel que soit le type ou la dimension du dessin. Cette zone est peut-être réservée pour une extension future du logiciel.

Codage des images

Nous arrivons dans la partie qui nous a donné le plus de travail puisque, pour trouver une constante susceptible de nous placer sur la bonne voie, nous avons dû réaliser un grand nombre de dessins "structurés", qu'il fallait ensuite "lire" octet par octet à partir du Basic. Nous vous livrons ici le résultat de cette longue recherche.

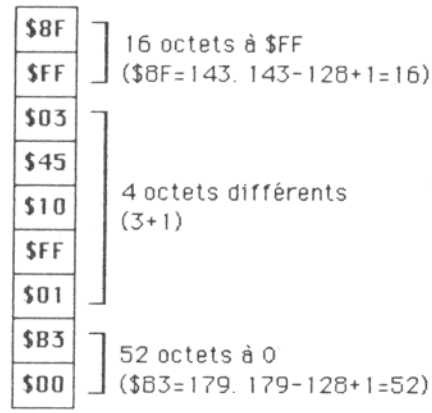
Dans tous les cas, même pour un très petit dessin, la totalité du document Macpaint est sauvegardée; ceci explique le fait qu'un document absolument vide occupe 2ko (soit quatre secteurs) sur la disquette lorsqu'il est sauvegardé.

Chaque ligne de points (constituée, comme nous l'avons vu précédemment, par 72 octets) est codée indépendamment des autres de la manière suivante :

Si des octets identiques sont juxtaposés, nous trouvons dans le fichier codé un octet contenant le nombre d'octets identiques moins un, avec son bit de poids fort (bit 7 ou "bit de signe") positionné à 1. Cet "octet compteur" est lui même suivi par la valeur de l'octet trouvé de façon répétitive. Ainsi, pour une ligne totalement vide (tous les octets à zéro), nous trouvons $C7 et $00 (199 et 0). Ceci correspond parfaitement à notre explication préalable, puisque 199 – 128 (valeur du bit 7) + 1 est égal à 72.

Dans le cas contraire, c'est à dire si des octets différents les uns des autres se suivent, l'octet compteur contient aussi le nombre d'octets moins un mais, cette fois, avec son bit de signe à 0. Cela permettra à la routine de décodage de déterminer si elle a affaire à des octets différents ou identiques. Puisque, dans le cas présent, nous avons des octets différents, l'octet utilisé comme compteur sera suivi par les *N* octets concernés. Si, avant codage, nous avons la suite d'octets : $FF, $FE, $A0, $10, la section correspondante dans le fichier codé sera : $03 (quatre octets moins un), $FF, $FE, $A0 et $10. Les deux possibilités peuvent, bien sûr, être combinées plusieurs fois pour une seule ligne de 572 points. On peut constater que cette méthode de codage est surtout rentable lorsque le nombre d'octets identiques est important. En effet, si l'on prend le cas extrême où l'on aurait, pour la totalité du document, uniquement des séquences de deux octets identiques, mais différents des deux octets précédents et suivants, l'image codée prendrait cinquante pour cent de plus de place que si elle n'était pas codée.

Pour conclure ce paragraphe, nous vous proposons un schéma représentant une ligne, une fois son codage effectué :



Constitution d'un fichier "Startup"

Pour qu'un fichier soit reconnu par le système comme étant l'image à afficher lors de l'amorçage, il doit :

- être de même nature qu'un document Macpaint avant codage;
- être baptisé du doux nom de "Startupscreen".

Le programme Basic accompagnant cet article décode le document Macpaint de votre choix, ce qui ne pose pas de problème puisque nous avons maintenant tous les éléments, et en fait un fichier que l'on peut considérer comme étant une "image mémoire" du document original. L'affichage du Macintosh étant constitué par 512 lignes de 342 points, seule la surface correspondante placée dans l'angle supérieur gauche du document à transformer est prise en compte.

```
10 ' Conversion de fichiers MacPaint en
   fichiers "Startupscreen".
20 '
30 DEFINT A-Y:OPTION BASE 1:DIM A(36):
   ON ERROR GOTO 290
40 CLS:CALL TEXTFONT(0)
50 PRINT"Ce programme transforme un document
   MacPaint"
60 PRINT"en un fichier Startupscreen."
70 PRINT"La partie supérieure gauche du document
   est utilisée"
80 PRINT"(342 lignes * 512 colonnes).":PRINT
90 PRINT"ATTENTION : si un fichiers Startupscreen
   se trouve sur la"
100 PRINT "disquette placée dans le lecteur
    interne, il sera détruit.":PRINT
110 INPUT"Fichier MacPaint à convertir ? ",Z$:
    OPEN"i",1,Z$
120 OPEN"o",2,"startupscreen"
130 L=342
140 M=32
150 PRINT"Conversion en cours..."
160 Z$=INPUT$(512,1)
170 FOR I=1 TO L:FOR J=1 TO M:A(J)=0:NEXT:X=0
180 NO=ASC(INPUT$(1,1)):IF NO<185 THEN O=NO+1:
    GOTO 230 ELSE O=256-NO+1:
    C=ASC(INPUT$(1,1))
190 FOR J=1 TO O:IF (J+X+1) MOD 2 THEN
    A((J+X+1)\2)=A((J+X+1)\2) OR C:GOTO 220
200 C!=C*256:IF C!>32767 THEN C!=C!-65536!
210 H=C!:A((J+X+1)\2)=A((J+X+1)\2) OR H
220 NEXT:X=X+O:GOTO 270
230 FOR J=1 TO O:X=X+1:C=ASC(INPUT$(1,1)):
    IF (X+1) MOD 2 THEN A((X+1)\2)=A((X+1)\2) OR
    C:GOTO 260
240 C!=C*256:IF C!>32767 THEN C!=C!-65536!
250 H=C!:A((X+1)\2)=A((X+1)\2) OR H
260 NEXT
270 IF X<72 THEN 180
280 FOR J=1 TO M:
    PRINT#2,CHR$(A(J)\256);CHR$(A(J) AND
    &HFF);:NEXT:NEXT:CLOSE:PRINT"Conversion
    effectuée.":GOTO 300
290 CLOSE:END:IF ERL=280 THEN PRINT"La
    disquette est saturée !":RESUME 300 ELSE IF
    ERL=110 THEN RESUME 110 ELSE RESUME 120
300 PRINT"Autre fichier à convertir ? (O/N)
310 Z$=INKEY$:IF Z$="" THEN 310 ELSE IF Z$="O" OR
    Z$="o" THEN 40 ELSE IF Z$="N" OR Z$="n" THEN
    END ELSE 310
```

La réalisation d'un fichier "Startupscreen" par le programme ici proposé demande environ huit minutes, mais son affichage au moment de l'amorçage est instantané. Désormais, lorsque vous enverrez une disquette à un correspondant, celui-ci saura d'où elle vient, même si vous avez omis de joindre vos coordonnées. De plus, ce petit programme prouve qu'il n'est pas nécessaire d'avoir le statut de "développeur de logiciels" pour personnaliser des disquettes. ■

Choisissez vos caractères

Lorsque l'on édite un programme Basic, la taille et le type des caractères qui apparaissent dans les fenêtres "List" et "Command" sont imposés (New York 12 points avec la première version du dossier système, et Geneve 12 points avec la version actuelle). Il est possible de changer cela en "POKant" le code de la police voulue à l'adresse $985, et la taille à l'adresse $987.

POKE &H985,4 : POKE &H987,9 nous fait passer en Monaco 9 points, ce qui permet de visualiser beaucoup plus de texte en une seule fois. La taille des caractères situés dans les menus est aussi modifiée, mais ce n'est pas un problème puisque, lorsque l'on programme, l'efficacité est plus importante que l'apparence de l'écran. ■

D'Applesoft à Microsoft

Lorsque l'on passe de l'Apple II au Macintosh, on peut être confronté à des "bugs" difficilement repérables, dûs à quelques différences entre les Basics utilisés.

Les opérateurs logiques ne fonctionnent pas de la même façon. Ainsi :

```
100 IF NOT A% THEN PRINT
"NON":
GOTO 120
110 PRINT "OUI"
```

provoque, avec le Basic Applesoft, l'affichage de "OUI" si la variable A% est égale à 0, "NON" si A% est égale à 1 (ou, tout simplement, différente de 0).
En Basic Microsoft, le texte affiché sera "NON", que la variable A% soit égale à 1 ou 0.
Cela est dû au fait que le Basic Applesoft réalise seulement un test logique, alors que le Basic Microsoft effectue une véritable opération logique. NOT 0 est égal à −1 (différent de 0 donc vrai), NOT 1 est égal à −2 (idem). Le Basic Microsoft affichera "NON" uniquement si A% est égale à −1 (NOT −1 est égal à 0). Le NOT étant souvent utilisé pour tester l'état d'un drapeau, il vaut mieux positionner celui-ci à 0 ou −1, plutôt qu'à 1 ou 0.

En Basic Applesoft, les boucles FOR - NEXT sont toujours exécutées au moins une fois, alors qu'avec le Basic Microsoft une boucle du type :

```
FOR I=0 TO 1 STEP −1:NEXT
```

ou encore :

```
FOR I=1 TO 0:NEXT
```

ne sera pas exécutée (passage immédiat à l'instruction suivante).

L'instruction STR$(X) du Basic Microsoft ajoute un espace au début de la chaîne résultante si le contenu de la variable X est positif : "FICHIER" + STR$(1) donne la chaîne "FICHIER 1". Ce n'est pas le cas avec le Basic Applesoft, où la même instruction retourne la chaîne "FICHIER1".

Si vous trouvez d'autres différences, n'oubliez pas de nous en faire part... ■

Call :un exemple d'application

Marianne Sutz

Le Basic Microsoft autorise l'appel de routines en langage machine, ce qui est la moindre des choses, avec l'instruction CALL. Cependant, la documentation fournie avec le logiciel n'est pas très claire sur ce point et un petit complément d'informations ne nous semble pas superflu.

Pour illustrer le fonctionnement de cette instruction, nous avons écrit une courte routine qui effectue le transfert du contenu d'un tableau de variables entières à deux dimensions, vers un tableau de variables à une dimension. Cette routine permet de combler en partie un défaut de l'instruction PUT, qui n'accepte pas les indices : PUT(10,10),A(B,C) provoque une "SYNTAX ERROR". Ce n'est pas vraiment gênant lorsque l'on a seulement deux ou trois PUT à exécuter mais, si l'on doit réaliser une animation qui nécessite 32 de ces instructions, la mémoire peut être très rapidement saturée. Il serait à la fois plus agréable et plus efficace d'écrire une boucle du type :

```
100 FOR I%=0 TO 31
110 PUT(10,10),A%(I%,0)
120 NEXT
```

plutôt que :

```
100 PUT(10,10),A0%
110 PUT(10,10),A1%
120 PUT(10,10),A2%
etc...
```

Pour comprendre parfaitement le fonctionnement de la routine, il est indispensable de connaître la structure des tableaux de variables entières; nous allons donc commencer par là.

Structure des tableaux

L'instruction VARPTR(X) retourne l'adresse en mémoire de la variable X (uniquement avec des variables numériques; le système diffère sensiblement pour les variables alphanumériques). De même, l'instruction VARPTR(V%(0)) retourne l'adresse du premier élément (variable entière : 2 octets) constituant le tableau V%(X). On peut noter au passage que, si un tableau V%(X) a été préalablement défini avec DIM, une variable V% est considérée par l'interpréteur comme totalement indépendante du tableau précité.

Au dessus de l'adresse du premier élément (soit en allant vers les adresses les plus basses), nous trouvons les arguments du tableau. Si l'on part de l'adresse contenant le premier octet de ce "descripteur", nous trouvons :

- Le nombre d'octets par éléments (8 = double précision, 4 = simple précision et 2 = entier).
- Les codes ASCII des deux premiers caractères constituant le nom du tableau. Si le nom est formé par un seul caractère, le second octet est nul.
- Le nombre de caractères constituant le nom du tableau moins deux. Pour un nom formé de moins de trois caractères, cet octet est nul.
- Les codes ASCII des caractères du nom (au delà du second caractère), avec le bit de signe positionné à 1 ($C1, soit 193, pour un "A" dont le code ASCII est normalement 65). Pour un nom dont le nombre de caractères est inférieur ou égal à deux, cette zone n'apparaît pas.
- Une valeur sur 24 bits (3 octets) représentant la position relative (par rapport à l'adresse courante) de la prochaine définition de tableau ou variable.
- Un octet contenant le nombre de dimensions du tableau.
- Deux octets indiquant le nombre d'éléments par dimension, répétés autant de fois que nécessaire. On trouve ces valeurs dans l'ordre inverse de la définition du tableau par DIM. Ainsi, pour A(3,10), nous trouvons $00 $0B $00 $04 pour OPTION BASE 0, et $00 $0A $00 $03 pour OPTION BASE 1. On constate donc que la seule différence entre OPTION BASE 1 et OPTION BASE 0 se trouve dans la définition du nombre d'éléments.

Si l'on prend des exemples concrets, pour ABCD%(10), nous trouvons dans la table des arguments (OPTION BASE 0):

$02 : nombre d'octets par éléments.
$41 $42 $02 $C3 $C4 : nom du tableau;
$00 $00 $5C : position relative de la prochaine définition de variable ou tableau;
$01 : nombre de dimensions;
$00 $0B : nombre d'éléments.
$XX $XX : premier élément du tableau, dont l'adresse est retournée par VARPTR(A%(0)).

Pour AB#(2,3,4), nous trouvons (OPTION BASE 1) :

$08 : nombre d'octets par éléments.
$41 $42 $00 : nom du tableau.
$00 $00 $C7 : adresse relative;
$03 : nombre de dimensions.
$00 $04 : nombre d'éléments pour la troisième dimension.
$00 $03 : nombre d'éléments pour la seconde dimension.
$00 $02 : nombre d'éléments pour la première dimension.
$XX $XX $XX $XX $XX $XX $XX $XX : premier élément de la première dimension.

Position des éléments dans un tableau

Pour un tableau à une dimension, il n'y a aucun problème puisque les éléments sont placés les uns après les autres en mémoire. Ainsi, pour un tableau de variables en simple précision, et si X représente l'adresse du premier élément, le second élément se trouve en X+4, le troisième en X+8, etc...

Pour un tableau à deux dimensions (A(3,9) par exemple), les choses se compliquent un peu puisque l'on y trouve (la première colonne de chiffres représente la première dimension) :

```
1 - 1
2 - 1
3 - 1
4 - 1
1 - 2
2 - 2
3 - 2
4 - 2
etc...
```

Ceci explique la position inversée des indicateurs d'octets par éléments.

Fonctionnement de CALL

L'instruction CALL peut être considérée comme une interface entre l'interpréteur Basic et le langage machine. Elle permet l'appel d'une routine et le retour au programme Basic (si tout se passe bien !). Il n'est pas possible, comme avec le Basic AppleSoft de l'Apple II, d'appeler une routine avec un "CALL adresse de la routine". Il faut obligatoirement que l'adresse se trouve dans une variable. Cette variable doit être de type simple ou double précision car les adresses peuvent être supérieure à 65535 (le microprocesseur MC 68000 L8 peut adresser 16777216 octets directement). De manière générale, et dans ce cas, il vaut mieux s'en tenir aux variables en simple précision puisque les variables en double précision occupent deux fois plus de place en mémoire, pour le même résultat.

TRES IMPORTANT : il ne faut jamais utiliser une variable numérique qui ne fait pas partie d'un tableau, si elle n'a pas été préalable-

ment définie, entre l'affectation de la variable destinée au CALL et le CALL lui même. En effet, l'initialisation d'une nouvelle variable provoque un déplacement des tableaux, ce qui fait que, si cette règle n'est pas respectée, le CALL peut appeler autre chose que l'adresse de branchement dans la routine, autrement dit : n'importe quoi... et bonjour la petite bombe ! Cette particularité peut donner des programmes à l'aspect étrange : la ligne 120 du programme joint à cet article paraît complètement stupide si l'on a pas connaissance de ce problème de déplacement.

Le CALL et la pile

La notion de pile est bien connue des utilisateurs du Mac venant de l'Apple II. Il faut néanmoins signaler quelques différences entre la gestion de cette zone de mémoire par le 6502 et le 68000 :

- la pile du 6502 se trouve en page 1 aux adresses $100 à $1FF alors que celle du 68000 peut être à n'importe quel emplacement;
- la pile du 6502 est limitée à 256 octets alors que celle du 68000 peut théoriquement occuper 16 Méga-octets (le pointeur de pile, qui est en fait le registre d'adresse 7, contient 32 bits). Avec le Basic Microsoft, et par défaut, la pile occupe 8 Ko. Ceci peut être modifié par l'instruction CLEAR : CLEAR,,1024 réduit la taille de la pile à 1 Ko, les 7 Ko ainsi libérés étant affectés au programme Basic;
- le pointeur de pile étant un registre d'adresse, on peut agir sur son contenu de la même manière qu'avec les autres registres; ce n'est pas le cas avec le 6502.

Nous conseillons à ceux d'entre-vous qui désirent en savoir plus sur le microprocesseur MC 68000 de se procurer l'ouvrage intitulé "Mise en oeuvre du 68000", aux éditions SIBEX. Ce livre est relativement cher (198 F), mais ce ne sera certainement pas un investissement inutile. Nous sommes en effet dans une situation paradoxale où l'on s'aperçoit que le seul ouvrage vraiment indispensable à ceux qui veulent connaître le fonctionnement du Macintosh n'est pas l'un des nombreux livres qui "traitent" directement de cet appareil !

Appel d'une routine sans passage de paramètres

Lors d'un appel de routine, l'adresse de retour à l'interpréteur Basic est placée au sommet de la pile, le contenu du pointeur de pile (Stack Pointer - SP) est décrémenté de 4 (adresse sur 32 bits, donc 4 octets) et pointe sur le premier octet de l'adresse en question. Pour revenir au programme Basic, il suffit d'utiliser l'instruction RTS (retour de sous-programme) qui place dans le compteur ordinal (PC) le contenu de l'adresse pointée par le SP, qui est ensuite incrémenté de 4 pour revenir à la valeur initiale (avant le CALL).

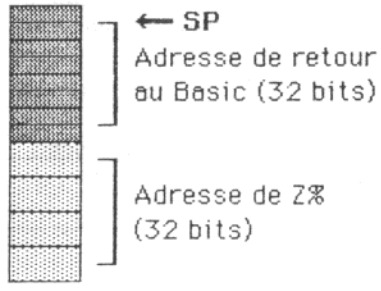
Appel d'une routine avec passage de paramètres

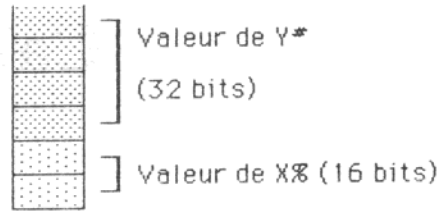
Il est possible de passer autant de paramètres que nécessaire entre le programme Basic et la routine en langage machine, ou dans le sens inverse. Tous les types d'arguments peuvent être communiqués :

- Les constantes ou expressions (B*10, par exemple) sont converties en une valeur entière sur 16 bits si possible, sur 32 bits sinon.
- Les variables entières restent sur 16 bits.
- Les variables en simple ou double précision sont converties en une valeur entière sur 32 bits.
- Lorsque l'on veut passer à la routine l'adresse d'une variable numérique plutôt que son contenu, il faut utiliser l'instruction VARPTR(X). Dans ce cas, la valeur est passée sur 32 bits.
- Pour une variable alphanumérique, VARPTR ne retourne pas l'adresse de la chaîne de caractères mais l'adresse où se trouvent, dans l'ordre, le nombre d'octets dans la chaîne (sur 16 bits) et l'adresse du premier caractère de la chaîne (sur 24 bits).

Dans tous les cas, si l'on doit passer des arguments par l'intermédiaire de variables, que ce soit pour une valeur ou une adresse, il est indispensable que la ou les variables aient été préalablement définies. De même, lorsque l'on doit passer l'adresse d'un tableau de variables, il faut respecter les mêmes règles que pour une variable contenant une adresse destinée à un CALL.

Les paramètres passés à la routine sont placés dans la pile dans l'ordre de leur apparition dans l'instruction CALL. L'adresse de retour au Basic est ensuite empilée. Ainsi, pour l'instruction CALL A(X%, Y#, VARPTR (Z%)), nous trouvons dans la pile :





Fonctionnement de la routine

Le but de cette routine est le transfert du contenu d'un tableau à deux dimensions dans un tableau à une dimension, afin de pouvoir réaliser une animation avec une seule instruction PUT, alors qu'il en aurait fallu un grand nombre (32 dans notre exemple) avec la méthode normale. L'appel de cette routine s'effectue avec le passage des paramètres suivants : adresse du premier élément de la première dimension du tableau de départ, adresse de l'élément (X,0) du même tableau, adresse du premier élément du tableau d'arrivée. Voyons ce que la routine fait de ces valeurs et, ligne par ligne, comment elle fonctionne.

1 - Le contenu du registre d'adresse 6 (32 bits) est placé dans la pile. Le pointeur de pile est décrémenté de 4 (il pointe donc 4 octets plus haut que l'adresse de retour), et sa nouvelle valeur est placée dans A6 (registre d'adresse 6). Cette manoeuvre est indispensable lorsqu'il y a passage de paramètres.
2 - L'adresse du premier élément du tableau de départ est la première valeur empilée au moment du CALL, et se trouve donc, compte tenu de la nature des 3 paramètres et de l'effet de l'instruction précédente, 16 octets plus bas que l'adresse pointée par A6. Cette instruction déplace donc une copie de l'adresse du premier élément du tableau de départ dans le registre d'adresse A0.
3 - Nous avons vu précédemment que les deux octets situés juste au dessus de l'adresse d'un tableau à deux dimensions indiquent le nombre d'éléments dans la première dimension. L'instruction MOVE −2(A0),D0 effectue une copie de ce nombre dans les 16 bits de poids faible du registre de donnée D0 (les 16 registres du 68000, qu'ils soient de donnée ou d'adresse, sont des registres 32 bits). Nous nous servirons de cette valeur ultérieurement.
4 - L'adresse du premier élément du tableau d'arrivée se trouve 8 octets plus bas que l'adresse pointée par A6. Nous faisons une copie de cette adresse dans A0.
5 - L'adresse du premier élément du tableau d'arrivée moins deux contient le nombre d'éléments. Nous

plaçons cette valeur dans les 16 bits de poids faible du registre de donnée D1. Ceci nous permet de savoir combien de mots de 16 bits devront être transférés et d'éviter ainsi des erreurs catastrophiques, puisque la routine ne pourra pas déplacer plus d'éléments que ne peut en contenir le tableau d'arrivée.

6 - D1, notre compteur d'éléments, est décrémenté de un. Nous verrons plus loin l'intérêt de cette opération.

7 - L'adresse du premier élément à déplacer depuis le tableau de départ est copiée dans le registre A1.

8 - La valeur du registre DO (qui contient le nombre d'éléments dans la première dimension du tableau de départ) est multipliée par deux, puisque le transfert porte sur des mots de 16 bits.

9 - Tranfert du premier élément. L'adresse contenue dans A0 étant auto-incrémentée, AO pointe sur l'élément suivant du tableau d'arrivée dès la fin d'exécution de cette instruction.

10 - Dans le paragraphe consacré à la position des éléments dans un tableau, nous avons vu que, dans le cas d'un tableau à deux dimensions, on trouve d'abord les premiers éléments, suivis des seconds éléments, et ainsi de suite... Donc, pour un tableau A(3,9), l'élément (1,0) se trouve 8 octets plus loin que l'élément (0,0). Ce déplacement se trouve dans DO, que nous ajoutons à A1 pour obtenir l'adresse du prochain élément à transférer depuis le tableau de départ.

11 - L'instruction DBRA D1,BOUCLE décrémente la valeur contenue dans les 16 bits de poids faible de D1. La valeur résultante est ensuite comparée à −1. Si le contenu de D1 est différent, le programme est dirigé vers l'étiquette "BOUCLE". Sinon, on passe à l'instruction suivante. On voit ici l'intérêt de la décrémentation de la ligne 6 puisque la sortie de boucle se fait lorsque D1 est égal à −1. En cas d'omission de la décrémentation, on transférerait un élément de trop, ce qui aurait certainement pour conséquence un "plantage" du Mac.

12 et 13 - La routine a fait son travail, on revient au programme Basic par RTS, sans avoir oublié de remettre les choses en état avec UNLK, qui est bien sûr l'inverse de LINK.

Source de la routine de transfert

```
 1  4E56 0000          LINK     A6,#0
 2  206E 0010          MOVEA.L  16(A6),A0
 3  3028 FFFE          MOVE     -2(A0),D0
 4  206E 0008          MOVEA.L  8(A6),A0
 5  3228 FFFE          MOVE     -2(A0),D1
 6  5341               SUBQ     #1,D1
 7  226E 000C          MOVEA.L  12(A6),A1
 8  D080               ADD.L    D0,D0
 9  30D1         BOUCLE MOVE     (A1),(A0)+
10  D3C0               ADDA.L   D0,A1
11  51C9 FFFA          DBRA     D1,BOUCLE
12  4E5E               UNLK     A6
13  4E75               RTS
```

```
10 ' Exemple d'animation utilisant une routine en langage machine pour le transfert de tableaux de variables entières.
20 '
30 DEFINT A-H:DEFSNG I-Z:DIM C(19),B(65),
   A(31,65),F(3),D(3)
40 DATA &h4E56,0,&h206E,16,&h3028,&hFFFE,
   &h206E,8,&h3228,&hFFFE,&h5341,&h226E,12,
   &hD080,&h30D1,&hD3C0,&h51C9, &hFFFA,
   &h4E5E,&h4E75
50 FOR B=0 TO 19:READ C(B):NEXT
60 FOR B=0 TO 3:F(B)=&HAA55:NEXT
70 D(0)=11:D(1)=11
80 FOR B=0 TO 31:D(2)=11+B:D(3)=D(2)
90 CALL FILLRECT(VARPTR(D(0)),VARPTR(F(0))):
   CALL FRAMERECT(VARPTR(D(0)))
100 GET(11,11)-(42,42),B
110 FOR E=0 TO 65:A(B,E)=B(E):NEXT:NEXT
120 Y=0:X=0:Z=VARPTR(C(0)):Y=VARPTR(A(0,0)):
    X=VARPTR(B(0))
130 FOR B=0 TO 31:CALL Z(Y,VARPTR(A(B,0)),X):
    PUT(100,100),B,PSET:NEXT:GOTO 130
```

Il semble inutile de s'attarder sur le petit programme Basic dont la compréhension est aisée lorsque l'on a tous les éléments. Nous n'avons pas joint l'équivalent de ce programme en utilisant la méthode normale, car cela aurait pris beaucoup de place et ne présentait pas d'intérêt. Ce dont vous pouvez être sûr, c'est qu'un tel programme prend vraiment beaucoup plus de place que celui que nous vous proposons. Le seul défaut de cette méthode est sa relative lenteur par rapport à une série de PUT, mais ce n'est pas vraiment un inconvénient, d'autant qu'avec la méthode classique il est souvent nécessaire d'insérer des boucles de délai. Faites l'expérience... ■

Restauration de disquette

Il arrive qu'il ne soit plus possible d'utiliser une disquette abîmée par une micro-coupure, un RESET intempestif... durant un accès au disque. On parvient parfois à restaurer la disquette avec la procédure suivante, à démarrer avec l'appareil éteint : – Mettre la disquette fautive dans le lecteur intégré.
– Allumer l'appareil en maintenant enfoncées les touches Option et Commande.

Disquette Macintosh

La disquette Macintosh comporte les programmes Macintosh publiés dans les Pom's 14 à 16 : le nouveau système, la police Cairo (Pom's 15) et le programme Disk Copy. La grande surprise que nous vous y avons ajouté pour ce numéro est le programme "Localizer".

Ce programme vous permet de modifier le clavier d'une disquette sans toucher au reste du système. C'est très utile car, quand vous remplacez le système d'une disquette US par un système français, vous écrasez tout ce qu'il pouvait y avoir par ailleurs dans le système américain, par exemple les polices. C'est ce qu'ont découvert avec grand déplaisir certains possesseurs de Multiplan US, après avoir copié un système français ne comportant pas la police Seattle.

MacPaint et le Basic

Pom's

Nous avons pensé qu'il pourrait être intéressant de récupérer des documents MacPaint depuis le Basic, pour en faire, par exemple, des décors de programmes de jeux ou, simplement, des présentations soignées de programmes plus sérieux. Le temps nécessaire à la réalisation d'un programme se trouve sérieusement raccourci puisque l'on peut "planter le décor " très rapidement avec Macpaint. En revanche, s'il est possible de faire la même chose avec uniquement le Basic, une présentation sophistiquée risque d'utiliser toute la mémoire disponible sur un 128 Ko et d'occuper la place nécessaire pour la partie logique du programme.

Mode d'emploi

Il faut, bien sûr, créer un document MacPaint dont on utilisera seulement les 300 premières lignes de 496 points, situées dans l'angle supérieur gauche de la page graphique. La deuxième étape consiste en la conversion du document en un fichier facile à charger depuis le Basic. Pour ce qui est de la méthode employée pour transformer le document, vous pouvez vous reporter à l'article "Personnalisez vos disquettes Macintosh", publié dans ce numéro. Vous pourrez d'ailleurs constater que les programmes sont presque identiques; seule la méthode employée pour la sauvegarde du fichier final est différente. Une fois le fichier converti, quelques lignes de Basic suffisent à son chargement.

Nous avons choisi d'utiliser 300 lignes de 496 points parce que cela correspond au remplissage de la fenêtre de sortie du Basic Microsoft, lorsqu'elle est ouverte au maximum. Cependant, vous pouvez choisir une autre dimension en modifiant les lignes 70 et 80 du programme de conversion. Pour un document de 200 lignes de 256 points, il faut que L (nombre de lignes) soit égal à 200, et que M (nombre de mots de 16 bits par ligne) soit égal à 16 (256 / 16 = 16). De même, le petit programme de chargement doit être modifié en conséquence.

Pour le chargement, bien que le programme soit très court, il est encore possible de libérer 66 octets en "deletant" le tableau A(32), inutile une fois l'affichage de l'image effectué.

■

```
10 ' Conversion de fichiers MacPaint en
     fichiers utilisables depuis un programme
     Basic.
20 '
30 DEFINT A-Y:OPTION BASE 1:DIM A(36):
     ON ERROR GOTO 230
40 CLS:CALL TEXTFONT(0)
42 PRINT"Ce programme transforme un document
     MacPaint"
43 PRINT"en un fichier séquentiel utilisable depuis
     un"
44 PRINT"programme en Basic Microsoft."
45 PRINT"La partie supérieure gauche du document
     est utilisée"
46 PRINT"(300 lignes * 496 colonnes).":PRINT
50 INPUT"Fichier MacPaint à convertir ? ",Z$:
     OPEN"i",1,Z$
60 INPUT"Fichier à créer ? ",Z$:OPEN"o",2,Z$
70 L=300
80 M=31
90 PRINT:PRINT"Conversion en cours..."
100 Z$=INPUT$(512,1)
110 FOR I=1 TO L:FOR J=1 TO M:A(J)=0:NEXT:X=0
120 NO=ASC(INPUT$(1,1)):IF NO<185 THEN O=NO+1:
      GOTO 170 ELSE O=256-NO+1:
      C=ASC(INPUT$(1,1))
130 FOR J=1 TO O:IF (J+X+1) MOD 2 THEN
      A((J+X+1)\2)=A((J+X+1)\2) OR C:GOTO 160
140 C!=C*256:IF C!>32767 THEN C!=C!-65536!
150 H=C!:A((J+X+1)\2)=A((J+X+1)\2) OR H
160 NEXT:X=X+O:GOTO 210
170 FOR J=1 TO O:X=X+1:C=ASC(INPUT$(1,1)):
      IF (X+1) MOD 2 THEN A((X+1)\2)=A((X+1)\2) OR
      C:GOTO 200
180 C!=C*256:IF C!>32767 THEN C!=C!-65536!
190 H=C!:A((X+1)\2)=A((X+1)\2) OR H
200 NEXT
210 IF X<72 THEN 120
220 FOR J=1 TO M:PRINT#2,A(J):NEXT:NEXT:
      CLOSE:PRINT"Conversion effectuée." :GOTO 240
230 CLOSE:IF ERL=220 THEN PRINT"La disquette est
      saturée !":RESUME 240 ELSE IF ERL=50 THEN
      RESUME 50 ELSE RESUME 60
240 PRINT"Autre fichier à convertir ? (O/N)
250 Z$=INKEY$:IF Z$="" THEN 250 ELSE IF Z$="O" OR
      Z$="o" THEN 40 ELSE IF Z$="N" OR Z$="n" THEN
      END ELSE 250
```

```
10 ' Chargement et affichage d'une image
     réalisée avec Macpaint et transformée
     avec le programme de conversion.
20 DEFINT A-Z:CLS
30 DIM A(32):A(0)=496:A(1)=1
40 OPEN"i",1,"nom du fichier"
50 FOR I=0 TO 299:FOR J=0 TO 30:
     INPUT#1,A(J+2):NEXT:
     PUT (0,I),A:NEXT:CLOSE
60 GOTO 60
```

Appel des routines en ROM

Gérard Michel et Jean-Luc Bazanegue

La tendance actuelle des constructeurs d'ordinateurs et, en particulier, d'Apple, est de dire qu'il n'est pas nécessaire de savoir programmer pour utiliser ce type de machine. Si cela est vrai, il est aussi indéniable que la plupart des particuliers qui possèdent un micro-ordinateur passent plus de temps à le programmer qu'à utiliser des programmes tout faits. Dans un même ordre d'idée, on nous répète sans cesse que l'on peut tout faire avec Multiplan; nous serons prêts à le croire le jour où l'on nous présentera un tel tableur écrit en ce "langage". Cela nous rappelle d'ailleurs la grande question métaphysique : "est-ce l'oeuf qui a pondu l'oeuf ou l'oeuf qui a pondu l'oeuf...".
Cette entrée en matière quelque peu rageuse traduit notre sentiment, lorsque l'on constate qu'aucune information technique n'est fournie à l'utilisateur, qui a quand même investi quelques milliers de francs dans un appareil, et serait en droit d'en savoir un peu plus. Nous sommes assez loin des "Tutorials" et autres manuels de référence de l'Apple II, et ce n'est pas la "littérature spécialisée" actuelle qui suffit à combler le vide. Ceci étant dit, lorsque l'on regarde ce qui se passe "à l'intérieur", le Macintosh est une machine passionnante qui utilise un processeur remarquable, et Pom's a bien l'intention "d'ouvrir la boîte".

Pour mieux connaître le fonctionnement de la "chose", nous n'avons pas trouvé d'autres solutions que le désassemblage de certaines zones de mémoire. Ce n'est pas une mince affaire, sans désassembleur, d'autant plus que toutes les instructions du 68000 sont codées sur au moins 16 bits, ce qui fait, étant donnés les 14 modes d'adressage autorisés, un grand nombre de codes machine possibles.

Dès le début de nos recherches, nous avons trouvé de nombreux codes ne correspondant pas à des instructions valides. Après avoir douté quelques instants de la fiabilité de notre méthode de désassemblage manuel, il s'est avéré que ces instructions illégales étaient utilisées pour l'appel des routines situées dans la ROM de 64 Ko, ainsi que pour quelques routines en RAM.

Instructions illégales

Lorsque le processeur rencontre un code ne correspondant pas à une instruction valide, il entame une procédure d'exception. Deux cas peuvent se présenter : code dont les quatre bits de poids fort correspondent à la combinaison binaire 1010 ou 1111 ($A ou $F); code dont les quatre bits de poids fort ne forment pas la combinaison 1010 ou 1111, mais dont la combinaison binaire ne permet pas de reconnaître une instruction valide. Nous ne traiterons ici que de la première solution, utilisée pour l'appel des routines.

Une procédure d'exception est effectuée de la manière suivante :

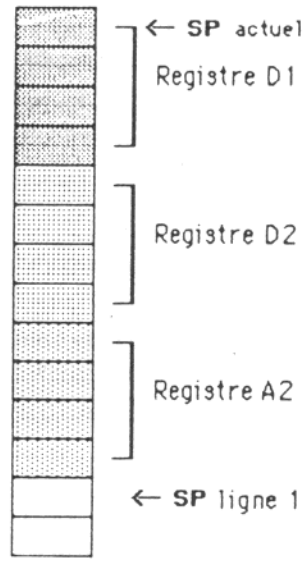
- Le processeur passe en mode superviseur, si toutefois il n'y était pas déjà. En effet, le 68000 peut fonctionner dans deux modes différents : le mode utilisateur, dans lequel la totalité des instructions ne peut pas être utilisée, et le mode superviseur qui autorise l'exploitation de toutes les instructions. Nous ne nous attarderons pas sur ce point car il semble que, sur le Macintosh, le 68000 soit toujours en mode superviseur.
- La logique du processeur génère un numéro de vecteur qui est : 10 pour un code commençant par 1010; 11 pour un code commençant par 1111.
- Le contenu actuel du compteur ordinal (Program Counter - PC) est mis en pile. Il est important de signaler que l'adresse placée dans la pile est celle de l'instruction illégale provoquant l'exception.
- Le contenu du registre d'état (Status Register - SR) est mis en pile. A ce stade, le contenu du registre pointeur de pile (Stack Pointer - SP) a été décrémenté de 6, par rapport à sa valeur avant le début de la procédure d'exception, puisque nous avons empilé le PC (32 bits) et le SR (16 bits).
- La valeur du numéro de vecteur est multipliée par quatre: pour une exception provenant d'un code dont les quatre bits de poids fort sont à 1010, le numéro de vecteur est 10, ce qui nous donne 40. Ceci correspond à l'adresse contenant l'adresse à placer dans le PC, donc l'adresse de la prochaine instruction à exécuter.

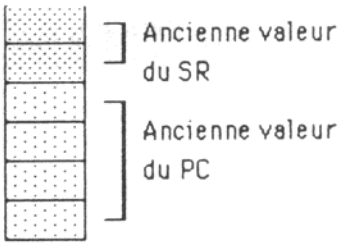
Tous les appels de routines étant effectués par des instructions illégales commençant par 1010, nous allons étudier de plus près le contenu de l'adresse 40, et surtout ce qui se passe lorsque l'on a obtenu cette valeur.

Routine de calcul des adresses

Pour savoir quelle est l'adresse contenue par l'adresse 40, il suffit, sous Basic, de faire des "PEEKs" aux adresses 40, 41, 42 et 43. Il est d'ailleurs important de signaler que toutes nos recherches sont effectuées avec le seul Basic Microsoft, soit avec des moyens accessibles à tout un chacun. Aux adresses sus-nommées, nous trouvons : 0, 64, 16 et 24, ce qui nous donne l'adresse 0 * 65536 + 64 * 4096 + 16 * 256 + 24, soit 266264 ou, en hexadécimal, $041018. Cette adresse (impressionnante quand on vient de l'Apple II) correspond au point d'entrée dans une routine (en ROM) qui calcule les adresses des autres routines, en fonction de la valeur du code qui a provoqué la procédure d'exception. Toujours à l'aide du Basic, nous avons lu les codes machine situés à partir de cette adresse, puis désassemblé (à la main !), ce qui nous donne la "pseudo-liste d'assemblage" que vous trouverez à la fin de cet article. La méthode employée pour le calcul des adresses n'étant pas forcément évidente, nous vous proposons quelques explications. Afin de les clarifier, nous allons prendre le cas du code illégal $A94C, qui provoque l'appel de la routine qui inverse l'état des pixels constituant la barre des menus.

1 - Le contenu du pointeur de pile est décrémenté de deux.
2 - Le registre d'adresse A2, ainsi que les registres de donnée D1 et D2, sont sauvegardés dans la pile qui, à ce stade, se présente ainsi :





3 - Le contenu de l'adresse pointée par le SP plus 16 (ancienne valeur du PC, soit l'adresse de l'instruction invalide) est transféré dans le registre d'adresse A2.
4 - Une copie de l'instruction invalide est placée dans les 16 bits de poids faible du registre de donnée D2. Ainsi, D2 (registre 32 bits) contient $XXXXA94C, les "X" représentant la valeur indéterminée contenue dans les 16 bits de poids fort. Après exécution de cette instruction, le registre A2, qui est auto-incrémenté, pointe sur l'instruction à traiter au retour de la routine.
5 - L'adresse de retour est placée dans la pile, écrasant ainsi l'adresse de l'instruction illégale.
6 - Une copie des 16 bits de poids fort de D2, qui contiennent l'instruction illégale, est placée dans D1.
7 - Le code de l'instruction illégale est comparé à la valeur immédiate $A800.
8 - Le bit "carry" du registre d'état est placé à 1 si le code de notre instruction illégale est inférieur à $A800. Comme ce n'est pas le cas, nous passons à l'instruction suivante.
9 - L'adresse de l'instruction suivante (ligne 10) est mis en pile. Le pointeur de pile est décrémenté de 4 et le compteur ordinal est chargé avec l'adresse de l'instruction située à l'étiquette "ETQ2". Pour que les choses soient bien claires : le pointeur de pile contient l'adresse où est sauvegardée le registre D1, moins 4.
16 - Un "ET" logique est réalisé entre les 16 bits de poids fort de D2 et la valeur immédiate $01FF. Comme nous avons $A94C dans le registre, la valeur résultante est $XXXX014C.
17 - Le bit 15 du registre D2 (bit de signe lorsque l'on utilise seulement les 16 bits de poids faible) est copié dans les 16 bits de poids fort. Le bit 15 étant à 0, nous obtenons la valeur $0000014C. Ceci permet de supprimer d'éventuelles valeurs parasites.
18 - On ajoute le registre D2 à lui-même, ce qui revient à multiplier son contenu par 2. Il contient maintenant $00000298.
19 - L'adresse $000400 est chargée dans le registre A2
20 - Le contenu (16 bits) de l'adresse calculée en ajoutant la valeur de A2 à celle de D2 est placé dans le registre D2. $000400 + $000298 = $000698. Cette adresse contient $8B9B.
21 - Le contenu des 16 bits de poids faible du registre D2 ($8B9B) est multiplié par deux. On a maintenant $1736 et le bit "carry" (retenue) est positionné à 1 puisque le résultat de l'opération ne "tient" pas dans 16 bits.
22 - L'instruction BCS (branchement si retenue à 1) nous dirige vers l'étiquette "ETQ4".
25 - Les 32 bits situés à l'adresse $02B2 sont ajoutés aux 32 bits du registre D2. Nous trouvons la valeur $00000B00, que nous ajoutons à $00001736 pour obtenir $00002236. Cette valeur est l'adresse de la routine d'inversion de la barre des menus. Cette routine n'est pas en ROM puisque l'adresse de base de cette dernière est $040000, adresse que nous trouvons d'ailleurs en $02AE. Nous avons constaté que les instructions illégales commençant par $A9 provoquent toujours un branchement sur une routine en RAM, alors que celles qui débutent par $A8 concernent des routines en ROM.
26 - Nous retournons à la ligne 10, après avoir incrémenté le pointeur de pile de 4. Ainsi, le SP pointe à nouveau sur l'adresse de sauvegarde de D1, comme avant l'exécution de la ligne 9.
10 - Le contenu de D2 (adresse de la routine) est placé dans la pile à l'endroit où se trouvaient les deux octets libres et l'ancienne valeur du registre d'état (d'où l'intérêt de la première instruction - SUBQ.L #2,SP - qui libérait deux octets dans la pile).
11 - Comparaison du code de l'instruction illégale (dont nous avions fait une copie à la ligne 6) avec la valeur $AC00. Le bit "carry" du registre d'état est placé à 1 ou à 0 en fonction du résultat de la comparaison, et sera utilisé un peu plus tard (quelques microsecondes !).
12 - Les valeurs sauvegardées à la ligne 2 sont replacées dans les registres D1, D2 et A2 afin d'obtenir la configuration de départ. De même, le SP contient l'adresse qu'il contenait lors de l'entrée dans la routine, adresse où se trouve maintenant l'adresse de la routine.
13 - Nous utilisons ici le bit "carry", positionné par la comparaison de la ligne 11. Si le code est inférieur à $AC00, ce qui est le cas, nous passons à l'étiquette "ETQ3".
15 - Puisque le SP pointe sur l'adresse de la routine, l'instruction RTS provoque le branchement. Le SP est ensuite incrémenté de 4, pointant ainsi sur l'adresse de retour au programme appelant.

Routine de calcul des adresses des routines en ROM

```
 1   041018  558F              SUBQ.L   #2,SP
 2   04101A  48E76020          MOVEM.L  A2/D2/D1,-(SP)
 3   04101E  246F0010          MOVEA.L  $0010(SP),A2
 4   041022  341A              MOVE     (A2)+,D2
 5   041024  2F4A0010          MOVE.L   A2,$0010(SP)
 6   041028  3202              MOVE     D2,D1
 7   04102A  0C42A800          CMPI     #$A800,D2
 8   04102E  6534              BCS      ETQ1
 9   041030  6112              BSR      ETQ2
10   041032  2F42000C          MOVE.L   D2,$000C(SP)
11   041036  0C41AC00          CMPI     #$AC00,D1
12   04103A  4C41AC00          MOVEM.L  (SP)+,D1/D2/A2
13   04103E  6502              BCS      ETQ3
14   041040  2E9F              MOVE.L   (SP)+,(SP)
15   041042  4E75        ETQ3  RTS
16   041044  024201FF    ETQ2  ANDI     #$01FF,D2
17   041048  48C2              EXT.L    D2
18   04104A  D442              ADD      D2,D2
19   04104C  45F80400          LEA      $000400,A2
20   041050  34322000          MOVE     0(A2,D2),D2
21   041054  D442              ADD      D2,D2
22   041056  6506              BCS      ETQ4
23   041058  D4B802AE          ADD.L    $02AE,D2
24   04105C  4E75              RTS
25   04105E  D4B802B2    ETQ4  ADD.L    $02B2,D2
26   041062  4E75              RTS
27   041064  08820008    ETQ1  BCLR     #8,D2
28   041068  6618              BNE      ETQ5
29   04106A  61D8              BSR      ETQ2
30   04106C  2442              MOVEA.L  D2,A2
```

```
31  04106E  48E700C0        MOVEM.L  A1/A0,-(SP)
32  041072  4E92            JSR      (A2)
33  041074  4CDF0300        MOVEM.L  (SP)+,A0/A1
34  041078  4CDF0406  ETQ6  MOVEM.L  (SP)+,D1/D2/A2
35  04107C  584F            ADDQ     #4,SP
36  04107E  4A40            TST      D0
38  041080  4E75            RTS
39  041082  61C0      ETQ5  BSR      ETQ2
40  041084  2442            MOVEA.L  D2,A2
41  041086  2F09            MOVE.L   A1,-(SP)
42  041088  4E92            JSR      (A2)
43  04108A  225F            MOVEA.L  (SP)+,A1
44  04108C  60EA            BRA      ETQ6
```

Nous avons étudié un seul cas car il n'est pas possible de les passer tous en revue. Néanmoins, le principe reste le même pour tous les appels de routines et l'important est de connaître la méthode utilisée. Pour la routine d'inversion de la barre des menus, on peut vérifier que notre calcul est juste en faisant, sous Basic :

A!=&h2236:CALL A!

■

Réservation de mémoire

La documentation qui accompagne le Basic Microsoft signale, à juste titre, que les routines en langage machine doivent être relogeables, c'est-à-dire que leur exécution doit pouvoir se faire quelle que soit leur position en mémoire. La raison de cette contrainte est que tout ce qui se trouve en mémoire vive se déplace en permanence. Par exemple, la création d'une variable déplace les tableaux de variables préalablement définis. Le système n'est pas criticable, puisque le 68000 permet la réalisation de codes relogeables très simplement, mais il peut être utile de se réserver un bloc de mémoire situé à une adresse connue, et qui n'aurait pas la bougeotte, pour faire des essais ou installer des tampons.

Nous vous proposons une petite routine qui créée une zone de mémoire fixe du nombre d'octets de votre choix, en fonction de la valeur contenue par la variable N! (256 dans notre exemple). L'adresse du premier octet de la zone est choisie par le système de gestion de la mémoire. Les 16 bits de poids fort de l'adresse sont placés dans A1%, les 16 bits de poids faible dans A2%. Le calcul de la position du bloc en mémoire est donc facile.

Il faut utiliser cette routine avec prudence car le système cherche toujours de la place, et fera tout pour en trouver, quitte à détruire un bloc de mémoire relogeable non protégé, mais utilisé par le Basic. Si l'on ne prend pas beaucoup de risque en se créant un bloc de 256 octets, ce n'est pas le cas si l'on essaie de réserver 10 Ko. ■

```
1 ' Réservation d'un bloc fixe de N
  octets
2 '
10 DIM R%(14)
20 DATA &h4E56,&h0000,&h202E,
   &h0010,&hA11E,&h226E,&h0008,
   &h3288,&h226E,&h000C,&h2008,
   &h4840,&h3280,&h4E5E,&h4E75
30 FOR I%=0 TO 14:READ R%(I%):NEXT
40 A1%=0:A2%=0:N!=256
50 M!=VARPTR(R%(0))
60 CALL M!(N!,VARPTR(A1%),
   VARPTR(A2%))
70 IF A1%<0 THEN A1!=A1%+65536!
   ELSE A1!=A1%
80 IF A2%<0 THEN A2!=A2%+65536!
   ELSE A2!=A2%
90 A!=A1!*65536!+A2!:
   PRINT "Le premier des" N! "octets se
   trouve à l'adresse" A!
```

Source de la routine

```
4E56 0000  LINK     A6,#0        ; voir l'article "CALL : exemple d'application"
202E 0010  MOVE.L   16(A6),D0    ; déplace une copie du contenu de la variable N! (situé
                                 ; dans la pile) dans le registre de donnée D0
A11E       _NewPtr               ; Trap (voir l'article "Appel des routines en ROM")
226E 0008  MOVEA.L  8(A6),A1     ; place l'adresse de la variable A2% dans le registre
                                 ; d'adresse A1
3288       MOVE     A0,(A1)      ; les 16 bits de poids faible de l'adresse du bloc de
                                 ; mémoire sont placés à l'adresse de la variable A2%
226E 000C  MOVEA.L  12(A6),A1    ; place l'adresse de la variable A1% dans le registre
                                 ; d'adresse A1
2008       MOVE.L   A0,D0        ; copie du registre d'adresse A0 dans le registre de
                                 ; donnée D0
4840       SWAP     D0           ; inverse les 16 bits de poids fort et les 16 bits de
                                 ; poids faible de D0
3280       MOVE     D0,(A1)      ; les 16 bits de poids fort de l'adresse du bloc de
                                 ; mémoire (qui sont maintenant dans les deux octets
                                 ; de poids faible de D0) sont placés à l'adresse où se
                                 ; trouve la variable A1%
4E5E       UNLK     A6
4E75       RTS
```

Entrée analogique améliorée

J. Gunther

Les entrées analogiques de l'Apple, également appelées "entrées des manettes de jeux", servent à numériser la valeur d'une résistance, variant de 0 à 150 kilo-ohms et contenue, par exemple, dans un joystick.

Il est facile de brancher autre chose qu'un potentiomètre : une thermistance (mesure de la température), une photorésistance (mesure de l'éclairage), ou un transistor (mesure de l'intensité) peuvent être utilisés par certaines applications.

En Basic, les entrées analogiques sont mesurables par les instructions PDL (0) à PDL (3) qui retournent une valeur comprise entre 0 et 255 (en pratique, l'interpréteur Applesoft se contente d'appeler la routine PREAD située en $FB1E).
Pour utiliser ces entrées comme instrument de mesure, il serait intéressant d'avoir une meilleure résolution.

Le principe général est de décompter le temps de décharge d'une capacité à travers la résistance étudiée. Le "top" de départ est donné par un appel à la mémoire $C070, et le "top" d'arrivée par la remise à zéro du bit 7 de la mémoire $C064 + X, X étant le numéro de l'entrée.

Le petit programme suivant :

```
        LDX #0
        LDA $C070
        LDA #$80
BOUCLE  INX
        BIT $C064
        BMI BOUCLE
        RTS
```

montre qu'à la sortie, le contenu du registre X varie de 0 à 255, puis revient à 0 et termine vers 130 lorsque l'on tourne le potentiomètre. La résolution peut donc être améliorée.

Le problème consiste à déterminer le nombre de tours effectués. Par le procédé ci-dessus, une valeur de 40 en sortie n'indique pas si la résistance se trouve à 40 ou à 255 + 40.

C'est donc l'objet du programme ANALOG, qui améliore considérablement la résolution (environ 3 fois). En contre-partie, le temps de recherche peut sembler long (jusqu'à 2 secondes) mais toutes les applications des entrées analogiques de l'Apple ne concernent pas des jeux vidéo à haute vitesse !

Pour rester simple, le programme ne contrôle que l'entrée numéro 0 (il faut modifier la variable d'assemblage ENTREE du source) et stocke le résultat aux adresses 00 et 01. Le programme ANALOG.DEMO fournit une démonstration.

■

```
 1 ******************************
 2 *          ANALOG            *
 3 ******************************
 4
 5 * J. Gunther
 6 *
 7 * assembl( AA/29sep84 Big Mac
 8 *
 9 * Programme permettant une lecture
    @ haute r(solution des
10 * entr(es analogiques.
11
12 ENTREE   =    0          num(ro pad
    dle ou autre
13 VAL      =    $00        page z(ro
    pour simpl;fier
14
15          LDA  #1
16          STA  VAL+1
17          STA  VAL
18
19 DEBUT    LDA  $C070      top de d(p
    art
20          LDX  VAL        double bou
    cle d'attente
21          LDY  VAL+1
22 BOUCLE   DEX
23          BNE  BOUCLE
24          DEY
25          BNE  BOUCLE
26
27          LDA  $C064+ENTREE
28          BPL  FINI       fini si (
    $80
29          LDA  #$80
30 BOUCLE2  BIT  $C064+ENTREE laisse l
    a d(charge se terminer
31          BMI  BOUCLE2
32
33          INC  VAL        incr(mente
    les compteurs
34          BNE  DEBUT
35          INC  VAL+1
36          INC  VAL
37          BNE  DEBUT      =jmp
38 FINI     RTS
```

```
*8000.8029

8000- A9 01 85 01 85 00 AD 70
8008- C0 A6 00 A4 01 CA D0 FD
8010- 88 D0 FA AD 64 C0 10 11
8018- A9 80 2C 64 C0 30 FB E6
8020- 00 D0 E3 E6 01 E6 00 D0
8028- DD 60
```

```
1  REM ****  ANALOG.DEMO  ****
2  REM AA/29sep84/pour J.Gunther
3  ONERR  GOTO 99
10  GR : COLOR= 2: FOR I = 0 TO 47: HLIN
     0,39 AT I: NEXT : TEXT : HTAB 1: V
     TAB 1
20  INVERSE : PRINT "  DEMONSTRATION DE L
     ECTURE ANALOGIQUE   ": NORMAL
30  PRINT  CHR$ (4)"BLOAD ANALOG,A$300":
     REM enti)rement relogeable
40  PRINT : PRINT  SPC( 4)"(APPUYEZ SUR C
     TRL-C POUR FINIR)" SPC( 5)
50  POKE 32,13: POKE 33,14: POKE 34,5: PO
     KE 35,23: HOME : PRINT
60  CALL 768: PRINT " PDL(0) = " RIGHT$ (
     "00" +  STR$ (257 * ( PEEK (1) - 1
     ) +  PEEK (0) - 1),3)" ";: GOTO 60
99  TEXT : HOME
```

Un désassembleur en Applesoft et WPL

Jean-François Rabasse

Pom's a déjà ouvert ses pages à des auteurs, parmi les plus "prestigieux", qui s'étaient attachés à démontrer que le fameux Visicalc pouvait servir à bien d'autres choses que celles auxquelles il semblait essentiellement destiné.

Dans le même ordre d'idées, je voudrais vous proposer ici une utilisation originale de WPL, le petit langage de programmation du traitement de textes Applewriter II et //e. Il s'agit en fait de le mettre à contribution, en complément d'un programme en Applesoft, pour reconstituer à partir d'un code en langage machine un fichier source en assembleur symbolique exploitable ensuite par LISA 2.5.

Evidemment, il ne faut pas attendre d'un tel système des performances extraordinaires en matière de désassemblage, qu'il s'agisse de la souplesse des manipulations ou de la rapidité des traitements. Mais il vous permettra néanmoins de réaliser un désassemblage précis, correct, et votre fichier source aura au moins le privilège de n'avoir pas été engendré par des moyens tout à fait ordinaires...

Introduction sous forme d'exemple

Pour illustrer les possibilités du logiciel, nous allons présenter un problème concret de désassemblage, appliqué à un petit morceau de la ROM Applesoft, situé entre les adresses $D31F et $D392. Ce fragment n'a pas grande signification en lui même, mais comporte une zone de données et une zone de programme.

Voici le listing que l'on obtient avec la commande L du moniteur :

Listing 1

```
D31F- 46 4F     LSR $4F
D321- 52        ???
D322- 4D 55 4C  EOR $4C55
D325- 41 20     EOR ($20,X)
D327- 54        ???
D328- 4F        ???
D329- 4F        ???
D32A- 20 43 4F  JSR $4F43
D32D- 4D 50 4C  EOR $4C50
D330- 45 D8     EOR $D8
D332- 43        ???
D333- 41 4E     EOR ($4E,X)
D335- 27        ???
D336- 54        ???
D337- 20 43 4F  JSR $4F43
D33A- 4E 54 49  LSR $4954
D33D- 4E 55 C5  LSR $C555
D340- 55 4E     EOR $4E,X
D342- 44        ???
D343- 45 46     EOR $46
D345- 27        ???
D346- 44        ???
D347- 20 46 55  JSR $5546
D34A- 4E 43 54  LSR $5443
D34D- 49 4F     EOR #$4F
D34F- CE 20 45  DEC $4520
D352- 52        ???
D353- 52        ???
D354- 4F        ???
D355- 52        ???
D356- 07        ???
D357- 00        BRK
D358- 20 49 4E  JSR $4E49
D35B- 20 00 0D  JSR $0D00
D35E- 42        ???
D35F- 52        ???
D360- 45 41     EOR $41
D362- 4B        ???
D363- 07        ???
D364- 00        BRK
D365- BA        TSX
D366- E8        INX
D367- E8        INX
D368- E8        INX
D369- E8        INX
D36A- BD 01 01  LDA $0101,X
D36D- C9 81     CMP #$81
D36F- D0 21     BNE $D392
D371- A5 86     LDA $86
D373- D0 0A     BNE $D37F
D375- BD 02 01  LDA $0102,X
D378- 85 85     STA $85
D37A- BD 03 01  LDA $0103,X
D37D- 85 86     STA $86
D37F- DD 03 01  CMP $0103,X
D382- D0 07     BNE $D38B
D384- A5 85     LDA $85
D386- DD 02 01  CMP $0102,X
D389- F0 07     BEQ $D392
D38B- 8A        TXA
D38C- 18        CLC
D38E- 69 12     ADC #$12
D38F- AA        TAX
D390- D0 D8     BNE $D36A
D392- 60        RTS
```

Ce listing met bien en évidence la zone de données ($D31F à $D364).

Le module Applesoft du désassembleur proposé (programme DSMTEXT) utilise la routine L du moniteur, plus des possibilités de listing en hexadécimal ou en ASCII.
Les listings réalisés, au moyen de commandes sur lesquelles nous reviendrons, sont créés à l'écran ou sur disquette.

A partir des deux commandes :
D31F.D364A
D365.D392L
il donne le résultat suivant :

Listing 2

```
D31F- ASC 'FORMULA TOO COMPLE'
D331- ASC "X"
D332- ASC 'CAN'T CONTINU'
D33F- ASC "E"
D340- ASC 'UNDEF'D FUNCTIO'
D34F- ASC "N"
D350- ASC ' ERROR'
D356- HEX 0700
D358- ASC ' IN '
D35C- HEX 000D
D35E- ASC 'BREAK'
D363- HEX 0700
D365- BA        TSX
D366- E8        INX
D367- E8        INX
D368- E8        INX
D369- E8        INX
D36A- BD 01 01  LDA $0101,X
D36D- C9 81     CMP #$81
D36F- D0 21     BNE $D392
D371- A5 86     LDA $86
D373- D0 0A     BNE $D37F
D375- BD 02 01  LDA $0102,X
D378- 85 85     STA $85
D37A- BD 03 01  LDA $0103,X
D37D- 85 86     STA $86
D37F- DD 03 01  CMP $0103,X
D382- D0 07     BNE $D38B
D384- A5 85     LDA $85
D386- DD 02 01  CMP $0102,X
D389- F0 07     BEQ $D392
D38B- 8A        TXA
D38C- 18        CLC
D38D- 69 12     ADC #$12
D38F- AA        TAX
D390- D0 D8     BNE $D36A
D392- 60        RTS
```

La syntaxe choisie pour le désassemblage en ASCII est la syntaxe LISA, avec les options ASCII positif (bit de poids fort à 0 - ASC') ou négatif (bit de poids fort à 1 - ASC"), et passage en HEX pour les codes de contrôle.

L'étape suivante consiste à remettre en forme la présentation des lignes pour obtenir une syntaxe assembleur "source", puis à réaliser l'analyse du texte :

- rassembler les arguments,
- créer les tables des variables (page 0 et adresses 4 chiffres),
- introduire des noms symboliques et éliminer toutes les adresses en tête.

C'est un travail typique de traitement de texte, c'est pourquoi on fait maintenant appel à un module écrit en WPL (DSMSYMB) qui, à partir du fichier disque généré par DSMTEXT puis relu en Applewriter, donne le listing suivant :

```
INS
;PROGRAMME  TEST
;
;TABLE PAGE 0
Z86 EPZ $86
Z85 EPZ $85
;
;TABLE DES ADRESSES
W0101 EQU $0101
W0102 EQU $0102
W0103 EQU $0103
;
;
 ORG $D31F
 OBJ $800
;
LBD31F ASC 'FORMULA TOO C
```

```
OMPLE'
     ASC "X"
     ASC 'CAN'T  CONTINU'
     ASC "E"
     ASC 'UNDEF 'D FUNCTIO'
     ASC "N"
     ASC ' ER ROR'
     HEX 0700
     ASC ' IN  '
     HEX 000D
     ASC 'BREAK '
     HEX 0700
     TSX
     INX
     INX
     INX
     INX
LBD36A LDA   W0101,X
     CMP    #$81
     BNE    LBD392
     LDA    Z86
     BNE    LBD37F
     LDA    W0102,X
     STA    Z85
     LDA    W0103,X
     STA    Z86
LBD37F CMP   W0103,X
     BNE    LBD38B
     LDA    Z85
     CMP    W0102,X
     BEQ    LBD392
LBD38B TXA
     CLC
     ADC    #$12
     TAX
     BNE    LBD36A
LBD392 RTS
 END
```

Les noms symboliques sont créés artificiellement en utilisant les chiffres de l'adresse correspondante, précédés d'une lettre (indispensable pour l'entrée sous l'assembleur). La lettre est Z pour une variable page 0, W pour une variable à 4 chiffres hexa et LB pour une variable "Label" (point d'entrée dans la zone désassemblée).
Le programme DSMSYMB rajoute de plus tout ce qui est nécessaire à l'entrée sous LISA : accès à l'éditeur (INS), directives ORG et OBJ, fin de source END, et, invisible sur le listing mais néanmoins présent, le CTRL-E permettant la sortie de l'éditeur.
Le fichier obtenu est un fichier TEXT, disponible sous Applewriter. On le transforme en source assembleur en tapant, à partir de LISA bien sûr, la commande CTRL-D EXEC "Nom du fichier".

Programme de désassemblage DSMTEXT

Description

Ce programme en Applesoft est assez court et découpé en petits sous-programmes. Son analyse ne devrait pas poser de problèmes, mais réclame quelques précisions :

– Le programme, prévu pour coexister avec un code machine, peut être chargé n'importe où en RAM, par ajustement du pointeur "Début de Basic". Il détermine son adresse de chargement (ligne 2030) et positionne HIMEM de manière à occuper exactement 8000 octets tout compris (ligne 2045). Tout le reste de la RAM est protégé.

– Il désassemble en mnémoniques au moyen de la routine LIST du moniteur. La routine complète désassemblant par blocs de 20 instructions, on utilise la sous-routine $F8D0 qui désassemble une instruction à l'adresse PTR = $3A, et retourne la longueur (1, 2 ou 3 octets) à l'adresse LG = $2F (ligne 2040).

– Pour afficher la valeur d'un octet en hexadécimal, on utilise la routine moniteur $FDDA (écriture en hexa du contenu de l'accumulateur), par l'intermédiaire d'une mini routine machine appelée PA. On POKE la valeur de l'octet à afficher dans une adresse "boîte à lettres" $06, et on appelle PA qui se résume à "LDA $06 - JMP $FDDA". Ce code de 5 octets est POKé lors de l'initialisation dans la ligne Basic 1 (ligne A NE PAS MODIFIER OU SUPPRIMER).

– Pour convertir en décimal les adresses hexa saisies au clavier, on utilise la technique dite "du fainéant" consistant à ne pas refaire ce que le moniteur fait si bien. Les adresses hexa, sous forme de variables chaînes sont introduites aux adresses RAM $06 à $09 par la technique classique de S.H.LAM. (lignes 34 à 38). Les adresses décimales sont récupérées par des doubles PEEK. L'inconvénient de cette méthode, par ailleurs très rapide à programmer, est que l'on ne teste pas la validité des symboles frappés (0 à 9 et A à F). Si vous entrez une chaîne qui n'est pas la représentation valide d'une adresse hexa, le moniteur proteste, fait BIP et vous abandonne "planté" sous moniteur. Pas de panique, faites CTRl-C, RETURN, GOTO 1000 (surtout pas RUN), RETURN, et c'est reparti sans perte d'informations, sauf votre dernière commande (la mauvaise).

– Enfin dernier point et peut-être le plus important, Apple Computer Inc, société sympathique au demeurant, n'a pas prévu que l'on ait envie d'utiliser simultanément la routine moniteur LIST et la routine RWTS du DOS. Ces deux routines ont en commun 4 adresses page 0, de $2C à $2F. Cela ne gêne pas RWTS qui utilise ces adresses pour des stockages temporaires mais chaque appel au disque modifie les variables de LIST qui fait alors n'importe quoi. Comme LIST est en ROM on la laisse tranquille et on "déménage" les variables de RWTS des adresses $2C à $2F aux adresses $FC à $FF (ligne 2065). Ceci impose d'ailleurs d'utiliser DSMTEXT avec un DOS tout à fait standard et non relogé. Comme chaque fois que l'on se trouve avec un DOS exotique en machine, il est préférable de supprimer la commande INIT (ligne 2070).

Utilisation

Pour une utilisation simple et rapide, le programme fonctionne avec un système de touches clés, donnant accès à des fonctions rappelées par un menu en haut d'écran. Lorsqu'on lance le programme, on se trouve en mode attente LIST donnant accès à 4 commandes, C, E, X et W.

Listing d'une zone mémoire

Frapper la touche C (sans RETURN), puis les adresses de début et de fin de la zone à lister, suivant la syntaxe classique du moniteur ADR1.ADR2 (4 chiffres hexa obligatoires par adresse, séparés par un point). Frapper ensuite une lettre : L pour désassembler par la routine moniteur, H pour lister en hexadécimal, A pour lister en ASCII. Terminer par RETURN.
Le listing est exécuté à l'écran. On peut l'interrompre, puis le relancer à l'aide de la barre d'espace. On peut le terminer avant la fin par la touche Escape.
Le programme demande ensuite si la commande doit être mémorisée. A vous de voir si le résultat vous satisfait, le but étant de créer un petit fichier de commandes de désassemblage successives, permettant de constituer un fichier source combinant l'analyse des zones de données (ASC ou HEX) et celles des instructions de l'assembleur (voir l'exemple du Listing 2 présenté plus haut, obtenu par un fichier à 2 commandes).
Le programme autorise jusqu'à 10 commandes successives, ce qui s'avère suffisant pour traiter la majorité des cas.
Le travail ainsi effectué, on retourne en mode attente LIST.

Edition des commandes

La frappe de E fait passer dans un mini éditeur qui récapitule les commandes de désassemblage (si vous en avez déja mémorisées, sinon un -E-est affiché), et permet d'en supprimer (S), de modifier l'ordre des commandes (D), ou d'en éditer pour correction éventuelle (E).
On quitte l'éditeur (Q), pour revenir en mode LIST.

Exécution des commandes

La frappe de X lance l'exécution à l'écran du désassemblage, les commandes en mémoire étant exécutées en séquence. La barre d'es-

pace et Escape jouent le même rôle que dans l'exécution au coup par coup. A la fin, retour au mode LIST.

Création du fichier disque

Lorsque l'exécution de votre séquence de commandes fournit un résultat conforme à vos voeux, la frappe de W démarre la création du fichier sur disque. Après avoir demandé le nom du fichier et les adresses de début du code machine, le programme relance un listing mais sur la disquette, puis revient en mode LIST.
Attention, le programme ne demande pas de numéro de lecteur; si vous en possédez plusieurs, assurez-vous que le lecteur implicite contient la bonne disquette destination du fichier texte (celui-ci devant nécessairement se trouver sur un disque qui contient les deux programmes WPL DSMSYMB et DSMSYMB.2).

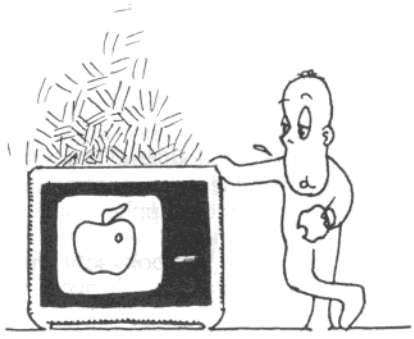
Place en mémoire

Comment disposer en RAM le code-machine à désassembler et le programme Applesoft DSMTEXT ?
En général, il est plus pratique de charger le code-machine à son adresse réelle de fonctionnement. Ceci facilite la lecture des listings de désassemblage qui comportent beaucoup de sauts ou d'appels de sous-routines. Si le code est à une adresse supérieure à $2800, vous lancerez alors DSMTEXT par un simple RUN. Sinon, il vous faudra d'abord remonter le pointeur début de BASIC derrière le code binaire.
Il existe un certain nombre de cas où le code ne peut pas être lu à son adresse réelle de fonctionnement : programme machine qui "tourne" en carte langage, ou bien dans l'extension RAM de la carte Text du //e, ou encore qui occupe la place habituelle du DOS.
On devra alors charger le code ailleurs. Le programme prévoit la possibilité de faire un décalage d'adresses, mais par remplacement du premier caractère hexa de l'adresse et non par calcul (WPL n'est pas réellement adapté pour effectuer des calculs en hexadécimal). Il est donc absolument indispensable de déplacer le code en conservant les trois chiffres bas de l'adresse. Le remplacement est effectué pour deux chiffres successifs : un programme fonctionnant à l'adresse $D04A, pourra être chargé en RAM à $404A par exemple, toutes les adresses en $4xxx seront remplacées par $Dxxx et les adresses $5xxx par $Exxx. Ceci limite à 4k la taille des codes désassemblables. Cette restriction n'est pas gênante dans la mesure où la limite pratique des codes, imposée par Applewriter, lui est inférieure.

Les adresses réelle et d'implantation du code-machine sont demandées par DSMTEXT au moment de créer le fichier texte (commande W).

Programme de traitement DSMSYMB

Une fois créé le fichier texte, "rebootez" le système avec Applewriter //, puis démarrez le programme WPL par la commande habituelle : CTRL-P puis DO DSMSYMB. Le programme vous demande le nom du fichier texte et vous pouvez enfin aller prendre un repos bien mérité, le reste du traitement étant automatique.



Lorsque le programme est terminé, le fichier texte définitif se trouve en mémoire Applewriter pour consultation et sur le disque sous le nom du fichier original produit par DSMTEXT (ce fichier original est bien sûr perdu). Ce fichier est prêt à une entrée sous LISA en mode EXEC.
Une remarque : le programme DSMSYMB utilise deux fichiers temporaires nommés REF et PG0, qui sont supprimés du disque en fin de travail. Assurez-vous de ne pas utiliser une disquette qui contient des programmes personnels portant un de ces deux noms, il leur arriverait malheur.

Le programme WPL a dû être scindé en deux modules (DSMSYMB et DSMSYMB.2), car Applewriter limite à 2K la zone réservée à WPL. Il n'appelle pas de commentaires particuliers, toutes les opérations successives étant signalées par un message à l'écran (et, accessoirement, les messages servent à documenter le listing).
L'opération la plus longue est la création des tables d'adresses (début du module 2). Pour un code important (2 à 3K) l'opération peut prendre d'une demi-heure à plus d'une heure (c'est l'occasion où jamais de profiter du beau temps...).

Bilan de l'opération

Le produit final est, comme on l'a montré dans le petit exemple de départ, un pseudo programme source. Sur le plan de la syntaxe, il est prêt à être exploité sous assembleur. On peut, si on le désire, profiter de ce que le fichier est encore en mémoire sous Applewriter, pour remplacer les noms symboliques générés par DSMSYMB par des noms plus explicites. On consultera les tables d'adresses créées et une documentation sur les routines système et leur appellation standard. On utilisera alors la fonction "recherche et remplacement" d'Applewriter. Par exemple : <CTRL-F> /WFD1B/KEYIN/A
ou bien
<CTRL-F>/WFDED/COUT/
A etc....
Ne pas oublier de sauvegarder alors le fichier modifié avant de "booter" LISA.

La taille maximale des codes désassemblables est fonction de la capacité mémoire de votre système sous Applewriter (environ 27K sur un Apple //e, ou Apple II+ avec carte langage, environ 47K sur un //e avec carte 80 colonnes étendue).
La routine LIST du moniteur génère en moyenne 30 caractères par ligne d'instruction. Une instruction correspondant à 1, 2 ou 3 octets, comptez un rapport 15 entre la taille du code machine et celle du fichier texte, un peu moins si le code comporte beaucoup de tables et zones de données.

Il est important de signaler un gros point faible de ce logiciel de désassemblage : il ne sait pas traiter complètement les adressages immédiats.
Soit par exemple le code machine "A9 41". Ce code sera désassemblé par LDA #$41 mais quelle est l'instruction source originale ? On peut proposer :
- LDA #$41
- LDA #'A'
- LDA #TOTO si TOTO=$6041
- LDA /LULU si LULU=$41FF

Les "matheux" diront qu'il n'y a pas bijection du programme source sur le code objet.

En pratique on devra donc être attentif à la signification des adressages immédiats. Dans l'ensemble le désassemblage sera correct, mais dans des cas tels qu'un code utilisant une table de données, avec des instructions assembleur du type "LDA #TABLE - LDY /TABLE", le désassemblage en arguments immédiats perd sa signification et le nouveau code objet ne fonctionnera plus si l'on décale les adresses du programme source. Il sera alors nécessaire de retoucher les instructions source pour réintroduire un nom symbolique.

En conclusion, si ce logiciel peut décharger le programmeur (patient) de la partie "bestiale" du travail de dé-

sassemblage, il ne dispense pas, heureusement, de réfléchir et de comprendre comment fonctionne le code que l'on est en train de traiter, avant de le modifier.

Note aux lecteurs qui souhaitent utiliser le programme et ne possèdent pas la disquette d'accompagnement

Pour entrer le programme Applesoft d'après le listing, pas de problèmes, mais quelques indications peuvent être utiles pour le WPL (à entrer sous Applewriter).

1) ATTENTION à la frappe ! WPL ne supporte pas les "typos" approximatives et, en général, une erreur de frappe se traduit, non par un message sympathique tel que SYNTAX ERROR à tel endroit, mais par ... "rien du tout". Le programme semble fonctionner et, à la fin, le résultat est totalement "farfelu". Ce type de BUG est pénible à traquer.
Attention aussi à ne pas oublier le blanc initial des lignes instructions. Seules les lignes portant une étiquette commencent en colonne 1.

2) Un certain nombre d'instructions effectuent des recherches et remplacements sur des chaînes comportant un nombre précis de Blancs (Espace). Il n'est pas toujours évident de compter des blancs sur un listing et, afin de permettre une frappe correcte, ce dernier comporte un certain nombre de lignes de commentaire composées de points : P ... .
Ces lignes signalent le nombre et la position des blancs dans l'instruction immédiatement suivante.

3) Quelques instructions comportent des caractères de contrôle, à frapper avec la procédure CTRL-V,CTRL-x,CTRL-V. Comme ces caractères n'apparaissent pas au listing, voici les lignes qui en comportent, le caractère étant représenté par (CTRL-x). Ces 8 caractères seront à remplacer par 1 caractère contrôle.
X1 PPR(CTRL-L)
X3 F##END%(CTRL-E)%#
X4 PPR(CTRL-L)
X15 S$D#(CTRL-E)%#
X18 PPR(CTRL-G)(CTRL-G)

Note aux lecteurs qui utilisent un autre assembleur que LISA 2.5

On récapitule ici tous les symboles et directives utilisés et destinés à LISA, avec les lignes de programme où ils apparaissent. Ceux qui ont un autre assembleur vérifieront la syntaxe et feront les remplacements éventuels (le remplacement doit être fait avec beaucoup de rigueur, sans introduire de caractères blancs supplémentaires et sans en supprimer).
La directive ou le symbole sont présentés entre crochets [.....], avec les blancs nécessaires.

- [HEX] Assemblage d'une zone en hexadécimal

Dans DSMTEXT ligne 2060
Dans DSMSYMB ligne X2

- [ASC] Assemblage d'une zone en ASCII

Dans DSMTEXT ligne 2060

- ["] Encadre une définition ASC "xxxxxxxx" en ASCII négatif (Bit 7 à 1)

Ce caractère est défini par CHR$(34), dans DSMTEXT lignes 2055 et 2060.

- ['] Encadre une définition ASC 'xxxxxxxx' en ASCII positif (Bit 7 à 0)

CHR$(39) dans DSMTEXT lignes 2055 et 2060.

- [INS] Entrée dans l'éditeur de LISA (en mode EXEC)

Dans DSMSYMB.2 ligne X16

- [ORG $] Origine du programme à assembler

Dans DSMSYMB.2 ligne X17

- [OBJ $800] Origine du code objet en machine

Dans DSMSYMB.2 ligne X17

- [END] Fin du programme source

Dans DSMSYMB ligne X3

- [(CTRL E)] Sortie de l'éditeur de LISA

Dans DSMSYMB ligne X3
Dans DSMSYMB.2 ligne X15

- [EQU $] Affectation d'une adresse hexa à une variable

Dans DSMSYMB.2 ligne X6

- [EPZ $] Affectation d'une adresse page 0 à une variable

Dans DSMSYMB.2 ligne X11

- [;] Début d'une ligne commentaire. Ce symbole est destiné à LISA mais est aussi utilisé abondamment comme marqueur par DSMSYMB. Si votre assembleur utilise un autre symbole commentaire (étoile ou petit Mickey), faites bien les remplacements partout :

Dans DSMSYMB.2 lignes X5, X7, X8, X9, X10, X12, X13, X14, X16 (2 fois), X17 (2 fois).

■

```
]LIST:DSM.TEXT

1  REM *****
2  GOSUB 2005: GOTO 1015
10  REM
12  REM  AFFICHAGE D'UNE ADRESSE HEXA
14  PRINT : POKE 6, FN HI(AD): CALL PA: P
      OKE 6, FN LO(AD): CALL PA: PRINT "
      -";: RETURN
20  REM
22  REM  DESASSEMBLE DES ADRESSES A1 A A2
24  FOR AD = A1(IC) TO A2(IC): POKE PTR,
      FN LO(AD): POKE PTR + 1, FN HI(AD)
      : CALL LI:AD = AD +  PEEK (LG): GO
      SUB 305: NEXT : RETURN
30  REM
32  REM  CONVERSION DECIMALE DES ADRESSES
       DE LA COMMANDE N$
34 B$ = "06:" +  MID$ (N$(IC),3,2) + " "
      +  MID$ (N$(IC),1,2) + " " +  MID$
       (N$(IC),8,2) + " " +  MID$ (N$(IC
      ),6,2) + " N D9C6G"
36  FOR I = 1 TO  LEN (B$): POKE 511 + I,
       ASC ( MID$ (B$,I,1)) + 128: NEXT
      : POKE 71,0: CALL  - 144
38 A1(IC) =  FN DP(6):A2(IC) =  FN DP(8):
       RETURN
40  REM
42  REM  NOUVELLE LIGNE DE DESASS.
44 DEB = 0:CP = 0: GOSUB 14: PRINT M$(FS)
      ;: RETURN
50  REM
52  REM  DES. EN HEXA
54 DEB = 1:FS = 1: FOR AD = A1(IC) TO A2(
      IC)
56  IF DEB THEN  GOSUB 44
58  POKE 6, PEEK (AD): CALL PA:CP = CP +
      1: IF CP = 12 THEN DEB = 1: PRINT
      CL$(FS);
60  GOSUB 305: NEXT : PRINT CL$(FS);: RET
      URN
100  REM
105  REM  TEST CARACTERE
110 K =  PEEK (AD): ON K / 32 + 1 GOTO 12
      0,125,125,125,120
115 FO = 2: RETURN
120 FO = 1: RETURN
125 FO = 3: RETURN
200  REM
205  REM  DESASS. EN ASCII
210 AD = A1(IC): GOSUB 105:FS = FO: GOSUB
       44
215  FOR AD = A1(IC) TO A2(IC): GOSUB 105
220  IF FO <  > FS THEN DEB = 1: PRINT CL
      $(FS);:FS = FO: GOSUB 44
225  IF DEB THEN  PRINT CL$(FS);: GOSUB 4
```

```
      4
230   ON FS GOTO 235,240,240
235   POKE 6, PEEK (AD): CALL PA: GOTO 245
240   PRINT  CHR$ (K + (FS - 3) * 128);
245 CP = CP + 1: IF CP = 12 + 12 *  INT (
      FS / 2) THEN DEB = 1
250   GOSUB 305: NEXT : PRINT CL$(FS);: RE
      TURN
300   REM
305   REM  PAUSE
310   IF WR THEN  RETURN
315   IF  PEEK (49152) = 155 THEN  POKE 49
      168,0:AD = A2(IC):IC = IM: RETURN
320   IF  PEEK (49152) <  > 160 THEN  RETU
      RN
325   POKE 49168,0
330   IF  PEEK (49152) = 160 THEN  POKE 49
      168,0: RETURN
335   IF  PEEK (49152) = 155 THEN 315
340   GOTO 330
400   REM
405   REM  ANALYSE DE LA COMMANDE
410 C$(IC) =  RIGHT$ (N$(IC),1): IF N$(IC
      ) = "W" AND IM =  > 0 THEN  POP :
      GOTO 605
415   IF N$(IC) = "E" THEN  POP : GOTO 300
      0
420   IF N$(IC) = "X" AND IM =  > 0 THEN
      POP : GOTO 705
425   IF  LEN (N$(IC)) <  > 10 OR  MID$ (N
      $(IC),5,1) <  > "." THEN 450
430   GOSUB 32
435   IF C$(IC) = "L" THEN 22
440   IF C$(IC) = "H" THEN 52
445   IF C$(IC) = "A" THEN 205
450   PRINT
455   PRINT  CHR$ (4);"CLOSE"
460   PRINT  CHR$ (7);"COMMANDE ";N$(IC);"
      INCORRECTE !": POP : GOTO 1015
600   REM
605   REM  CREATION DU FICHIER TEXTE
610   HOME : PRINT "CREATION DU FICHIER TE
      XTE": PRINT
615   INPUT "NOM DU FICHIER ? ";NA$: IF NA
      $ = "" THEN 1015
620   PRINT : PRINT "ADRESSE ORIGINE VRAIE
      DU CODE MACHINE ?": PRINT "(4 CHI
      FFRES HEXA)          $";: INPUT "";O
      G$
625   PRINT : PRINT "IMPLANTATION ? (OU RE
      TURN SI LA MEME)": PRINT "(4 CHIFF
      RES HEXA)          $";: INPUT "";OB$
630   IF OB$ = "" THEN OB$ = OG$
635   IF  RIGHT$ (OG$,3) <  >  RIGHT$ (OB$
      ,3) THEN  PRINT  CHR$ (7); CHR$ (7
      ): PRINT "SEUL LE PREMIER CHIFFRE
      PEUT CHANGER.": PRINT " DESOLE !":
      GOTO 1015
640 B$ =  LEFT$ (OB$,1):G$ =  LEFT$ (OG$,
      1)
645 T$ = "(A1)" + B$ + "(R1)" + G$
650   IF B$ = "9" THEN B$ = "@"
655   IF G$ = "9" THEN G$ = "@"
660 T$ = T$ + "(A2)" +  CHR$ (1 +  ASC (B
      $)) + "(R2)" +  CHR$ (1 +  ASC (G$
      )) + "( ORG $" + OG$
665   PRINT  CHR$ (4);"OPEN ";NA$;".ADRS"
670   PRINT  CHR$ (4);"WRITE ";NA$;".ADRS"
675   PRINT T$
680 WR = 1
685   PRINT  CHR$ (4);"OPEN";NA$
690   PRINT  CHR$ (4);"DELETE";NA$
695   PRINT  CHR$ (4);"OPEN";NA$
700   PRINT  CHR$ (4);"WRITE";NA$
705   FOR IC = 0 TO IM: GOSUB 405: NEXT
710   PRINT
715   IF  NOT WR THEN 1015
720   PRINT  CHR$ (4);"CLOSE"
725 WR = 0
1000  REM
1005  REM  PROGRAMME PRINCIPAL
1010  REM
1015  GET N$(JC): PRINT N$(JC);: IF N$(JC
      ) <  > "C" THEN  PRINT :IC = JC: G
      OSUB 405: GOTO 1035
1020  IF JC = 10 THEN  PRINT : PRINT  CHR
      $ (7); CHR$ (7);" PLUS DE PLACE !
      ": GOTO 1015
1025  INPUT "";N$(JC):IC = JC
1030  GOSUB 405: PRINT : PRINT
1035  PRINT "COMMANDE ";N$(JC);"  A CONSE
      RVER ? O/N ";: GET T$: PRINT T$
1040  IF T$ = "O" THEN IM = JC:JC = JC +
      1
1045  IF JC = 9 THEN  PRINT  CHR$ (7);"AT
      TENTION,DERNIERE COMMANDE ": PRINT
       "    AVANT ECRITURE DU FICHIER !"
      : PRINT
1050  GOTO 1015
2000  REM    INITIALISATION
2005 IM =  - 1
2010  DIM N$(11)
2015  DEF  FN HI(X) =  INT (X / 256): REM
      PARTIE HAUTE ADRESSE
2020  DEF  FN LO(X) = X - 256 *  FN HI(X)
      : REM  PARTIE BASSE
2025  DEF  FN DP(X) =  PEEK (X) + 256 *
      PEEK (X + 1): REM  DOUBLE PEEK
2030 PA =  FN DP(103) + 5: REM  CODE MACH
      INE POUR SORTIE CONTENU DE $06 EN
      HEXA
2035  POKE PA,165: POKE PA + 1,6: POKE PA
      + 2,76: POKE PA + 3,218: POKE PA
      + 4,253
2040 PTR = 58:LG = 47:LI = 63696: REM  DE
      SASSEMBLE 1 INSTRUCTION
2045  HIMEM: PA + 7995
2050  REM  FORMATS,MNEMONIQUES
2055 CL$(1) =  "   ":CL$(2) =  CHR$ (34) +
      CL$(1):CL$(3) =  CHR$ (39) + CL$(
      1)
2060 M$(1) =  "   HEX ":M$(2) =  "   ASC "
      +  CHR$ (34):M$(3) =  "   ASC " +
      CHR$ (39)
2065  POKE 47489,252: POKE 48687,253: POK
      E 48881,253: POKE 48975,253: POKE
      49015,253: POKE 49033,253: POKE 48
      622,254: POKE 48662,255: POKE 4867
      1,255: REM  MODIF RWTS
2070  POKE 43140,64: REM  SUPRESSION INIT
2075  TEXT : HOME :T$ = "ADR1.ADR2"
2080  INVERSE : PRINT "          ";: HTAB
      12: PRINT "C";: NORMAL : PRINT T$;
      : INVERSE : PRINT "L";: NORMAL : P
      RINT " ...DESASSEMBLAGE"
2085  INVERSE : PRINT " DSMTEXT ";: HTAB
      12: PRINT "C";: NORMAL : PRINT T$;
      : INVERSE : PRINT "H";: NORMAL : P
      RINT " ...LISTE EN HEXA"
2090  INVERSE : PRINT "          ";: HTAB
      12: PRINT "C";: NORMAL : PRINT T$;
      : INVERSE : PRINT "A";: NORMAL : P
      RINT " ..LISTE EN ASCII"
2095  INVERSE : PRINT "  LIST   ";: HTAB
      12: PRINT "E";: NORMAL : PRINT " .
      .....EDITEUR DE COMMANDES"
```

```
2100  INVERSE : PRINT "          ";: HTAB
       12: PRINT "X";: NORMAL : PRINT " .
       .....EXECUTION A L'ECRAN"
2105  INVERSE : PRINT "          ";: HTAB
       12: PRINT "W";: NORMAL : PRINT " .
       ..ECRITURE FICHIER DISQUE"
2110  PRINT "          ----------------
       ----------"
2115  POKE 34,7: RETURN
3000  REM
3005  REM   EDITEUR DES COMMANDES
3010  TEXT : HOME
3015  INVERSE : PRINT "          ";: HTAB
       12: PRINT "S";: NORMAL : PRINT "UP
       PRIMER UNE COMMANDE"
3020  INVERSE : PRINT " DSMTEXT ";: HTAB
       12: PRINT "D";: NORMAL : PRINT "EP
       LACER UNE COMMANDE"
3025  INVERSE : PRINT " EDITEUR ";: HTAB
       12: PRINT "E";: NORMAL : PRINT "DI
       TER UNE COMMANDE"
3030  INVERSE : PRINT "          ";: HTAB
       12: PRINT "Q";: NORMAL : PRINT "UI
       TTER L'EDITEUR"
3035  HTAB 12: PRINT "----------------
       ---"
3040  POKE 34,6: HOME
3045  FOR IC = 0 TO IM: PRINT IC;"  -";N$
       (IC);"-": NEXT
3050  POKE 34,20: HOME
3055  GET C$
3060  GOSUB 3155
3065  IF C$ = "S" THEN 3095
3070  IF C$ = "D" THEN 3110
3075  IF C$ < > "E" THEN  PRINT  CHR$ (7
       ): GOTO 3050
3080  PRINT "EDITER NO ? ";: GET C$: GOSU
       B 3155: PRINT C$: GOSUB 3165
3085  ON ER GOTO 3080,3090
3090  HTAB 1: PRINT "?";N$( VAL (C$));: H
       TAB 2: INPUT "";N$( VAL (C$)): GOT
       O 3040
3095  PRINT "SUPPRIMER NO ? ";: GET C$: G
       OSUB 3155: PRINT C$: GOSUB 3165
3100  ON ER GOTO 3095,3105
3105  FOR IC =  VAL (C$) TO IM - 1:N$(IC)
       = N$(IC + 1): NEXT :IM = IM - 1:J
       C = JC - 1: GOTO 3040
3110  PRINT "DEPLACER NO ? ";: GET C$: GO
       SUB 3155: PRINT C$:IO =  VAL (C$):
       GOSUB 3165
3115  ON ER GOTO 3110,3120
3120  PRINT "DESTINATION NO ? ";: GET C$:
       GOSUB 3155: PRINT C$:ID =  VAL (C
       $): GOSUB 3165
3125  ON ER GOTO 3120,3130
3130 KZ$ = N$(IO):N$(IO) = N$(ID):N$(ID)
       = KZ$: GOTO 3040
3155  IF C$ = "Q" THEN  GOSUB 2075: POP :
       GOTO 1015
3160  RETURN
3165 ER = 2: IF C$ >  STR$ (IM) OR C$ < "
       0" THEN ER = 1: PRINT  CHR$ (7)
3170  RETURN
```

```
      PND
X1 PPR
 PPR       PROGRAMME   DSMSYMB
 PPR        ================
 PPR     JF-R                        AOUT 84
 PPR
 PPR
 PPR
 PPR
 PPR      MODULE  DSMSYMB NO 1 : PREPARATI
       ON
 PPR
 PPR
 PPR
 PIN  NOM DU FICHIER ? =$D
 PPR CHARGEMENT
 NO
 L$D
 B
 PPR
 PPR-OK
 PPR RECHERCHE DES ???
 B
 P       ...  ..........   ...
AOP2 F#-   &&          ???   %#
 ?
 PGOX2
 PGOFOP2
 P     ..  ...
X2 F/-  /-   HEX/
 O?
 P ........   ...
 F/        ???   //
 O?
 PGOAOP2
FOP2 PPR-OK
 PPR ELIM. DES CODES HEXA
AOP3 B
 P  ...  .  ...
 F#-   && #-   #A
 PGOAOP3
 E
 D
X3 F## END%%#
 O?
 PPR-OK
 PPR RECALAGE DES ADRESSES
 P LECTURE
 PAS$D.ADRS=$C
 PLS$C/A1>/</N=$A
 PLS$C/R1>/</N=$B
 PCS/$A/$B/
 PGOFPREP
 PSRRC
 PAS$D.ADRS=$C
 PLS$C/A2>/</N=$A
 PLS$C/R2>/</N=$B
 PSRRC
 PGOFPREP
RC B
 P RECALAGE DES INSTRUCTIONS
 F#%$A#%$B#A
 P
 P RECALAGE DES BRANCHEMENTS
 PASBCC=$C
 PSRRECMN
 PASBCS=$C
 PSRRECMN
 PASBVS=$C
 PSRRECMN
 PASBVC=$C
 PSRRECMN
 PASBNE=$C.
 PSRRECMN
 PASBEQ=$C
 PSRRECMN
 PASBMI=$C
 PSRRECMN
 PASBPL=$C
 PSRRECMN
 PRT
RECMN B
 P   ...       ...
 F/$C   $$A/$C   $$B/A
 P
 PRT
FPREP PPR
 PPR-OK
```

```
 S$D
 PPR
 PPR RECHERCHE DES ARGUMENTS
NO
 P     ...    ...
 L$D#-    &&&    #%#AN
 P
 B
 F##%#
 0?
 PPR
 PPR    - MARQUAGE
 F/$/W/A
 PPR    - ADRS IMMEDIAT
 B
 F<#=><<A
 P
 PPR    - ELIM. DES SYMBOLES D'ADRESSAGE
 B
 F/(//A
 P
 B
 F/)//A
 P
 B
 F/,//A
 P
 B
 F/X//A
 P
 B
 F/Y//A
 P
AOP5 B
 F#%%#%#A
 PGOAOP5
SREF
PPR
PPR-OK
PPR SAUVEGARDE ARGUMENTS PAGE 0
B
F#W&&&&%##A
P
B
F/W/Z/A
P
SPG0
PPR
PPR-OK
PPR SAUVEGARDE REFERENCES
NO
LREF
B
F#W&&%##A
P
PPR
PPR-OK
PPR
PPR         PREPARATION TERMINEE
PPR
PPR    CHARGEMENT DU MODULE DSMSYMB.2
PDODSMSYMB.2
X4 PPR
 PPR
 PPR          DSMSYMB.2
 PPR
 PPR CREATION TABLE DES ADRESSES
 PPR
 PPR
 E
 D
 F<<WFIN><
 0?
 E
 L$D
 E
 D
X5 F<<>;TABLE DES ADRESSES><
 0?
 P
 PAS$D.ADRS=$C
 PLS$C#ORG $#%#N=$A
 PAS$A=$B
 PGOAD1
STR PPR
 B
 PLS#<>W<>(N=$A
 F(W$A)<<A
 PCS/$A/FIN/
 PGOFINI
AD1 PPR$$A
 B
 P       ..
 F<>$A-   <>LB$A<A
 PGOREFINT
 B
 F/$$A/W$A/A
 P
 E
 D
X6 F<<W$A EQU $$A><
 0?
 PGOSTR
 PPR
 P LABELS INTERNES
REFINT B
 F/$$A/LB$A/A
 P
 PGOSTR
FINI B
 P          ...   .......
 F#%&&&&-    #%         #A
 P
 PPR
 PPR-OK
 PPR SAUVEGARDE TABLE ADRESSES
 E
 D
X7 F##;%#
 0?
 B
X8 F/;TAB/
 ?
X9 SREF#;%#
 PPR
 PPR-OK
 PPR CREATION TABLE PAGE 0
 B
 LPG0
 F<<ZFIN><
 0?
 E
 D
X10 F<<>;TABLE PAGE 0><
 0?
 P
 B
 P ...    ...
 F/    #$/    @@/A
 P
STR0 PPR
 B
 PLS#<>Z<>(N=$A
 F(Z$A)<<A
 PCS/$A/FIN/
 PGOFINI0
 PPR$$A
 B
 F/$$A/Z$A/A
 P
 E
 D
X11 F<<Z$A EPZ $$A><
 0?
 PGOSTR0
FINI0 B
 P ...    ...
 F/    @@/    #$/A
 P
 PPR
 PPR-OK
 PPR SAUVEGARDE
 E
 D
X12 F##;%#
 0?
 B
X13 F/;TABLE P/
```

```
 ?
X14 SPG0#;%#
 B
 F#%%%##
 0?
X15 S$D#%#
 PPR
 PPR-OK
 PPR MISE EN FORME DU FICHIER FINAL
 NO
 B
X16 F##INS%;PROGRAMME  $D%;%#
 0?
 LPG0
 LREF
```

```
 E
 D
X17 F##;% ORG $$B% OBJ $800%;%#
 0?
 L$D
 PPR
 PPR ET SAUVEGARDE
 S$D
 OFREF
 FPG0
 F$C
 ?
 B
X18 PPR
 PQT
```

Les codes ASCII du //e

Guy d'Herbemont

J'ai été intéressé par le programme "GESMASK modifié" publié dans Pom's 14, et brusquement interloqué en constatant qu'il m'était impossible de sauver le moindre masque par la commande CTRL - SHIFT - P, pourtant prévue à cet effet.

En examinant le programme source en assembleur, j'ai constaté, à la ligne 82, que l'accès à la sauvegarde du masque résultait de la frappe d'une touche, ou séquence de touches, ayant pour code clavier $80. J'en ai déduit que les auteurs du programme concerné utilisaient un Apple II+, sur lequel la frappe de CTRL - SHIFT - P, ou encore "CTRL - arrobas", correspond effectivement à la frappe du caractère "nul" dont le code clavier est $80.
Malheureusement, sur un Apple //e, il n'en va pas de même. La frappe de CTRL - SHIFT - P est équivalente à celle de CTRL - P et retourne $90 pour code clavier. On comprend alors que la comparaison avec $80 se soit avérée désespérément infructueuse...

Considérant en outre que le //e possède un double clavier AZERTY / QWERTY, j'ai voulu vous proposer un petit programme très simple permettant de lister les codes de toutes les touches du //e, dans l'un et l'autre cas, afin de rendre service aux utilisateurs qui pourraient se trouver confrontés comme moi à ce problème de différences de codes.
Pour en revenir au cas de GESMASK, il apparaît ainsi que, sur le //e, c'est "CTRL - à" (ou encore "CTRL - 0") qu'il faut taper pour obtenir $80 si l'on est en AZERTY, et "CTRL - 2" (ou "CTRL - SHIFT - 2") pour le même résultat si l'on est en QWERTY.

L'utilisation du programme est simple : il suffit de taper sur toutes les touches de chaque rangée dans les différents cas possibles (normal, majuscules, CTRL et CTRL + SHIFT), une fois avec le clavier en mode AZERTY et une seconde fois en mode QWERTY.

La version listée ci-dessous vous donnera un tableau des codes en ASCII positif décimal (tous inférieurs à 128), correspondant aux valeurs que vous retournerait la fonction ASC si elle fonctionnait correctement pour toutes les touches.

Pour ceux qui s'intéressent plutôt aux codes écrans négatifs exprimés en hexadécimal (valeurs que l'on teste en assembleur en sortie d'une routine de saisie de caractères, comme dans le cas de GESMASK), les petites adaptations suivantes permettront d'obtenir le tableau souhaité de la même façon :

- Saisir et sauver sur disquette la petite routine machine ASCII.O
- Rajouter une ligne "5 PRINT CHR$ (4) " BLOAD ASCII.O
- Remplacer la ligne 210 par "210 CALL 768" ■

```
]LOAD ASCII.HERBEMONT
]LIST

1   REM  TABLEAU DES CODES CLAVIER
2   REM  ------------------------
10   TEXT : HOME : PRINT  CHR$ (4)"PR#1"
15   PRINT : PRINT  SPC( 10)"CODES CLAVIER
       AZERTY": GOSUB 20
16   PRINT : PRINT  SPC( 10)"CODES CLAVIER
       QWERTY": GOSUB 20
17   PRINT  CHR$ (4)"PR#0": END
20   PRINT "NORMAL": GOSUB 30
21   PRINT "SHIFT": GOSUB 30
22   PRINT "CTRL": GOSUB 30
23   PRINT "SHIFT+CTRL": GOSUB 30
24   RETURN
30   PRINT "RANGEE ESC: ";:N = 14: GOSUB 2
       00
40   PRINT "RANGEE TAB: ";:N = 14: GOSUB 2
       00
50   PRINT "RANGEE CTRL ";:N = 13: GOSUB 2
       00
60   PRINT "RANGEE X   : ";:N = 11: GOSUB 2
       00
70   PRINT "RANGEE ESP: ";:N = 5: GOSUB 20
       0
80   RETURN
200   FOR I = 1 TO N
210   GET A$: PRINT  PEEK (49152)" ";
220   NEXT : PRINT : RETURN
```

```
*300.30B

0300- 20 0C FD 20 DA FD A9 A0
0308- 20 ED FD 60
```

STARTUP et saisie de la date en Pascal

Eric Pascual

La gestion de directory sous PASCAL UCSD permet, entre autres avantages, de dater les versions de fichiers automatiquement. Mais cette date ne sera correcte que si l'utilisateur prend systématiquement la peine de mettre à jour la date système en "bootant". Or cette opération ne peut se faire normalement que par l'option D(ate du FILER. Malheureusement, 9 fois sur 10, on néglige de le faire, et au bout de quelques temps, on ne se souvient plus de la chronologie des versions de programmes.

Il est pourtant assez simple d'écrire un petit programme possédant les fonctions suivantes :

- Saisie rapide de la date : rentrer uniquement le jour si le mois est inchangé.
- Vérification de celle-ci.
- Enregistrement de la nouvelle date sur la disquette (pour le prochain "boot") et en mémoire pour qu'elle soit utilisée dans la directory.

De plus si on appelle ce programme SYSTEM.STARTUP, il sera exécuté à chaque "boot", ce qui permet d'avoir une date système toujours ... à jour.

Les bonnes adresses

Deux endroits nous intéressent:
- la position de la date système sur la disquette;
- la position de cette même date en mémoire.

Pour la première, on peut se reporter, entre autres, à l'article de M. Crimont "Un catalogue généralisé en Pascal" paru dans le numéro 2 de Pom's. Pour ceux d'entre vous qui n'auraient pas ce numéro, rappelons que la directory d'un disque PASCAL occupe les blocs 2 à 5 inclus et est structurée comme indiqué dans le listing 1 donné en fin d'article.

Etant donné que seul le premier bloc nous intéresse, nous utiliserons une variable d'un type légèrement différent de REPERTOIRE dans le programme. Le type REPERT n'a de description précise que pour l'entête, le complément à la taille du bloc étant occupé par un ARRAY de 243 octets. En effet, le champ DISQUE occupe 26 octets, il en reste donc 486 dans le bloc, ce qui correspond à la place occupée par notre tableau. Pour lire (respectivement écrire) ce bloc, il suffit d'utiliser la fonction UNITREAD (respectivement UNITWRITE).

Pour ce qui est de la position en mémoire, elle varie selon les versions du système :
- version 1.0 : $A912, $A913 (−22254, −22253).
- version 1.1 : $AA18, $AA19 (−21992, −21991).

Le programme

Il va lire le répertoire sur le disque, affiche la dernière date enregistrée, demande la nouvelle date (en refusant les entrées incorrectes), puis l'enregistre dans le répertoire du disque et en mémoire aux adresses ci-dessus.

Le listing du programme étant abondamment commenté, il est inutile de reprendre ici sa description détaillée.
Pour le lancer automatiquement lors du "boot", il suffit de le stocker sur la disquette APPLE1: et de l'appeler SYSTEM.STARTUP.

Pour finir, voici un complément de bibliographie sur PASCAL UCSD pour Apple //:
- Gestion de fichiers et de périphériques pour Apple // en Pascal, de Hervé Haut (Editions de PSI).
- Le système Pascal UCSD (tomes 1 et 2), de Thierry Chamoret (Edi-TEST).

Ces deux ouvrages sont particulièrement intéressants et fort bien faits. On y apprend énormément de choses. Merci à leurs auteurs.

■

```
REPERTOIRE = record
               DISQUE: ENTETE;
               FICHIER: array !1..77! of DESCRFIC
             end;
avec

ENTETE = record
           unused1          :integer;       (* inutilisé                *)
           PREBLOC          :integer;       (* num 1er bloc utilisable *)
           unused2          :integer;       (* inutilisé                *)
           NOM              :string!7!;     (* nom du disque            *)
           NBBLOCS          :integer;       (* nombre de blocs du disq.*)
           NBFICHIERS       :integer;       (* nombre de fichiers       *)
           unused3          :integer;       (* inutilisé                *)
           DATESYS          :date;          (* date système             *)
           unused4          :integer;       (* inutilisé                *)
           unused5          :integer;       (* inutilisé                *)
         end;

DATE  = packed record
          MOIS   :1..12;
          JOUR   :1..31;
          ANNEE  :1..99
        end;
```

```
{-------------------------------------------------------------------------}
{                                                                         }
{       Programme STARTUP                                Version du 05/09/84    }
{                                                                         }
{       Assure les fonctions suivantes:                                   }
{               - lecture dans le repertoire de la deniere date de boot   }
{               - affichage du message de bienvenue.                      }
{               - saisie date du jour et enregistrement sur le disque et  }
{                 en memoire                                              }
{                                                                         }
{-------------------------------------------------------------------------}

program startup;

const   volume  =4;                     { Numero du boot disk             }
        adrdate =-21992;                { Adresse date systeme version 1.1 }
                                        {   (-22254 pour version 1.0)     }

type    date    =packed record
                  mois  :1..12;
                  jour  :1..31;
                  annee :0..99
                 end;

        memdate =record case boolean of         { pour pouvoir acceder a la date }
                  true  :(datesys  :^date);     { en memoire (voir article       }
                  false :(adresse  :integer)    { "PEEKs et POKEs en PASCAL"     }
                 end;                           { POMS numero 12 page 5          }

        entete  =record
                  extfut1       :integer;       { inutilise                      }
                  prebloc       :integer;       { numero du 1er bloc libre       }
                  catdis        :integer;       { inutilise                      }
                  nom           :string[7];     { nom du disque                  }
                  blocs         :integer;       { nbre de blocs sur le disque    }
                  nbfichiers    :integer;       { nbre de fichiers sur disque    }
                  extfut2       :integer;       { inutilise                      }
                  datesys       :date;          { date systeme enregistree       }
                  extfut3       :integer;       { inutilise                      }
                  extfut4       :integer        { inutilise                      }
                 end;

        repert  =record
                  header        :entete;
                  filler        :array[1..243] of integer
                  { pour completer le record a 512 octets (taille d'un bloc)   }
                 end;

var     mmjjaa  :memdate;
        wdate   :date;
        rep     :repert;
        change  :boolean;

{-------------------------------------- Ecriture de la nouvelle date systeme  }
{                                      en memoire.                          }
procedure writedate(vardate:date);

begin
   mmjjaa.adresse         :=adrdate;
   mmjjaa.datesys^.jour   :=vardate.jour;
   mmjjaa.datesys^.mois   :=vardate.mois;
   mmjjaa.datesys^.annee  :=vardate.annee
end;


{-------------------------------------- Saisie nouvelle date systeme          }
```

```
procedure saisdate;

const   x0      =62;                    { position du debut de la zone de saisie }
        y0      =18;                    {    (x0 = colonne    y0 = ligne)        }


var     strdate :string;
        jj      :integer;
        mm      :integer;
        aa      :integer;
        ok      :boolean;

{ :::::::::::::::::::::::::::::::::::: Saisie des elements de la date          }
   procedure saisjjmmaa;

   var     j1,j2   :integer;
           m1,m2   :integer;
           a1,a2   :integer;
           c       :char;
           ok      :boolean;

   begin
    change:=true;

    { ============================== Lecture 1er chiffre du jour               }
    repeat
      gotoxy(x0,y0);
      read(c);
      if ord(c)=32 then begin                   { si RETURN, arret saisie     }
         change:=false;
         exit(saisjjmmaa)
      end;

      j1:=ord(c)-ord('0');                                  { test de validite  }
      ok:=(j1 in [0..3]);
      if not ok then writeln(chr(7))
    until ok;

    { ============================== Lecture 2eme chiffre du jour              }
    repeat
      gotoxy(x0+1,y0);
      read(c);
      j2:=ord(c)-ord('0');                                  { test de validite  }
      ok:= (       ((j1=0       ) and (j2 in [1..9]))
                or ((j1 in [1,2]) and (j2 in [0..9]))
                or ((j1=3       ) and (j2 in [0,1] ))       );
      if not ok then writeln(chr(7))
    until ok;

    jj:=10*j1 + j2;

    { ============================== Lecture 1er chiffre du mois               }
    repeat
      gotoxy(x0+3,y0);
      read(c);
      if ord(c)=32 then exit(saisjjmmaa);       { si RETURN, arret saisie     }

      m1:=ord(c)-ord('0');                                  { test de validite  }
      ok:=(m1 in [0,1]);
      if not ok then writeln(chr(7))
    until ok;

    { ============================== Lecture 2eme chiffre du mois              }
    repeat
      gotoxy(x0+4,y0);
```

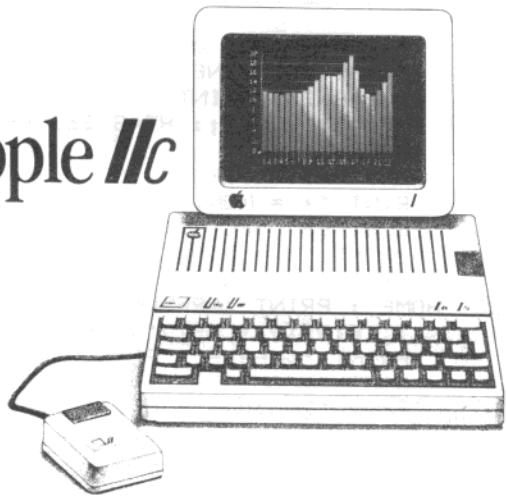
```
      read(c);
      m2:=ord(c)-ord('0');                                    { test de validite    }
      ok:= (       ((m1=0) and (m2 in [1..9]))
                or ((m1=1) and (m2 in [0..2]))       );
      if not ok then writeln(chr(7))
    until ok;

    mm:=10*m1 + m2;

    { ============================== Lecture 1er chiffre de l'annee              }
    repeat
      gotoxy(x0+6,y0);
      read(c);
      if ord(c)=32 then exit(saisjjmmaa);       { si RETURN arret saisie        }

      a1:=ord(c)-ord('0');                                    { test de validite    }
      ok:= (a1 in [0..9]);
      if not ok then writeln(chr(7))
    until ok;

    { ============================== Lecture 2eme chiffre de l'annee             }
    repeat
      gotoxy(x0+7,y0);
      read(c);
      a2:=ord(c)-ord('0');                                    { test de validite    }
      ok:=(a2 in [0..9]);
      if not ok then writeln(chr(7))
    until ok;

    aa:=10*a1 + a2;

  end;

{ ::::::::::::::::::::::::::::::::::::::: Verification coherence date saisie    }
  function verifdate :boolean;

  begin
   if mm in [4,6,9,11]
   then verifdate:=(jj<31)
   else if mm<>2
        then verifdate:=true
        else if aa mod 4 <> 0
             then verifdate:=(jj<29)
             else verifdate:=(jj<30)
  end;


{ ::::::::::::::::::::::::::::::::::::::: Debut SAISDATE                        }
begin
 jj:=rep.header.datesys.jour;
 mm:=rep.header.datesys.mois;
 aa:=rep.header.datesys.annee;

 {------------------------------------ Affichage date actuelle                 }
 gotoxy(0,17);
 write('Current date is ',jj,'-');
 if mm in [1..12]
 then case mm of
       1: write('Jan');
       2: write('Feb');
       3: write('Mar');
       4: write('Apr');
       5: write('May');
       6: write('Jun');
       7: write('Jul');
```

```
       8: write('Aug');
       9: write('Sep');
      10: write('Oct');
      11: write('Nov');
      12: write('Dec')
 end
 else write('???');
 writeln('-',aa);

 {------------------------------------ Saisie nouvelle date si besoin          }
 writeln('Enter new date if needed (form <01..31>-<01..12>-<00..99>)  :');
 gotoxy(x0,y0);
 writeln('..-..-..');

  repeat
   saisjjmmaa;                                { Lecture reponse                 }
   gotoxy(x0,y0);                             { Reecriture date complete        }
   if jj<=9 then write('0');write(jj,'-');
   if mm<=9 then write('0');write(mm,'-');
   if aa<=9 then write('0');write(aa);

   if not change
      then ok:=true                           { Si saise par RETURN => rien a faire }
      else ok:=verifdate;                     { Sinon on verifie la nouvelle date   }
   gotoxy(25,20);
   if not ok
      then write('>>>>    Date invalide    <<<<',chr(7))
      else write('                          ')
  until ok;

  {------------------------------------ Si modification alors on memorise la   }
  {                                     nouvelle date pour les mises a jour.   }
  if change
  then begin
   with wdate   do
        begin
         jour  :=jj;
         mois  :=mm;
         annee :=aa
        end;
   rep.header.datesys:=wdate
  end

 end;                                         { de la procedure SAISDATE        }

 {-------------------------------------- Lecture de 1er bloc du repertoire     }
 procedure readrep(var varrep:repert);

 begin
  unitread(volume,varrep,sizeof(varrep),2)
 end;

 {-------------------------------------- Mise a jour du 1er bloc du repertoire }
 procedure writerep(var varrep:repert);

 begin
   unitwrite(volume,varrep,sizeof(varrep),2)
 end;

 {-------------------------------------- Affichage du message de bienvenue     }
 procedure welcome;

 begin
  page(output);
  gotoxy(0,10);
```

```
 writeln('Welcome ',rep.header.nom,', to Apple II Pascal 1.1');
 writeln;
 writeln('Based on UCSD Pascal System Version II.1');
end;



{-------------------------------------------------------------------------------}
{                  PROGRAMME PRINCIPAL                                          }
{-------------------------------------------------------------------------------}


begin
  readrep(rep);                          { Lecture du 1er bloc de la directory  }
  welcome;                               { Affichage du message de bienvenue    }
  saisdate;                              { Saisie de la date                    }
  if change
  then begin                             { Si differente de l'ancienne,         }
       writedate(wdate);                 {  1 - enregistrement en memoire       }
       writerep(rep)                     {  2 - mise a jour de la directory     }
       end
end.
```

Tout ce que vous avez toujours voulu savoir...

Apple *IIc*



Macintosh™

Apple et GMS c'est une rencontre. Apple c'est toute une gamme d'ordinateurs personnels pour professionnels.
L'Apple *IIe* et son jeune frère compact l'Apple *IIc*. Ils sont très sérieux pour la gestion, la tenue des stocks ou le traitement de texte...
Et puis il y a Macintosh et sa souris. On clique sur la souris, on appelle le programme. Tout un menu est affiché par symboles, les éléments sont simples et les combinaisons infinies.
Enfin il y a GMS, une équipe de professionnels qui vous accueilleront et vous conseilleront personnellement.
Alors tout ce que vous avez toujours voulu savoir... 345.28.52.

INFORMATIQUE GMS

Informatique GMS 212-214 avenue Daumesnil 75012 Paris.

Apple

"Le nom Apple et le logo Apple sont des marques déposées d'Apple Computer, Inc." ™ Apple Computer, Inc est le licencié de la marque Macintosh.

ROUX, SEGUELA, CAYZAC & GOUDARD

PLE+CRAE+APA

Yvan Koenig

Si vous avez PLE, CRAE, TOOLKIT et une extension 16K, je vous offre la possibilité d'avoir ces trois utilitaires simultanément en mémoire (MEM) avec, en plus, un tableau des lignes référencées et une routine regroupant des lignes consécutives.

CRAE se place sous les buffers du DOS de $7400 à $8FFF. LOADAPA, utilisé lorsque PLE est en place, charge APA de $81E1 à $8FFF. GR+LINREF utilise $7FD0 à $81D9. Il est possible grâce à CRAPA.OBJ de reporter CRAE dans l'extension Bank1 de $D000 à $EC00. De son côté, APA+DIVERS peut se loger en Bank2 de $D000 à $DFFF et de $F000 à $FC00.

La commande &' permet d'appeler en MEM l'utilitaire qui n'y est pas. Lorsque APA+DIVERS est actif, on accède à LINREF par &&, à GROUPE par &GR L1,L2.

Pour avoir ces possibilités, entrez les programmes CRAPA et GR+LINREF soit en assembleur, soit directement en hexadécimal. Chargez PLE, puis faites RUN LOADAPA afin de placer APA sous les buffers du DOS. Placez alors GR+LINREF en $7FD0 et sauvez le tout par BSAVE APA+DIVERS, A$7FD0,L$1030. Chargez à nouveau PLE, effacez 63999, tapez le listing PLECRAPA et sauvez sous ce nom.

Vous pouvez alors initialiser une disquette avec PLECRAPA puis la compléter avec PLE.EDIT.PROD , APA+DIVERS et enfin CRAPA.OBJ.

Vous avez alors un outil très complet pour l'édition de vos programmes Basic. Je cite pour mémoire : avec CRAE (voir Pom's 1), vous avez la recherche et la modification de chaînes, un dump mémoire, une numérotation automatique, et la conversion HEX – DEC.

Avec APA+DIVERS, vous avez les variables référencées, la renumérotation, la réunion de programmes, la suppression de REM, les lignes référencées, etc...

Je précise que &GR L1,L2 se borne à afficher les lignes L1 à L2 comme si elles étaient regroupées, à vous de valider si vous le souhaitez en relisant avec le curseur.

&GR L1 affiche la ligne L1 compactée au maximum, les espaces parasites étant omis et les instructions PRINT remplacées par ?.

Je n'ai pas créé de pense-bête pour indiquer quel utilitaire est actif à un moment donné; taper & envoie le message READY si CRAE est actif, et envoie le prompt Applesoft dans le cas contraire.
La mémoire disponible pour les programmes Basic est exactement celle que l'on a avec le couple PLE + CRAE. ■

```
]LIST:PLECRAPA

10  TEXT : HOME : CALL  PEEK (175) +  PEE
    K (176) * 256 - 1822: CALL  - 2257
    2: PRINT "VOUS AVEZ PLE"
30  PRINT  CHR$ (4)"BLOADCRAPA.OBJ,A$300"
    : POKE 827,8: POKE 829,6
50  GOSUB 620: PRINT  CHR$ (4)"BLOAD APA+
    DIVERS": CALL 786
60  GOSUB 600: PRINT  CHR$ (4)"BLOADPLE.E
    DIT.PROD": CALL 768
70  POKE 111,240: POKE 112,115: POKE 115,
    240: POKE 116,115: POKE 827,6: POK
    E 829,8
80  PRINT : INVERSE : PRINT "  CRAE ACTIF
    ";: HTAB 21: PRINT "&'";: NORMAL :
     PRINT " POUR BASCULER": VTAB 22:
    NEW
600  PRINT : PRINT "1. APPEND";: HTAB 26:
     PRINT "6. QUOTE": PRINT "2. CHANG
    E";: HTAB 26: PRINT "7. VERIFY OFF
    ": PRINT "3. DUMP";: HTAB 26: PRIN
    T "8. AUTOLINE": PRINT "4. FIND";:
     HTAB 26: PRINT "9. RENUMBER": PRI
    NT "5. LIST";: HTAB 26: PRINT "10.
    MODIFY"
610  PRINT "11.QUICK & DIRTY": HTAB 26: P
    RINT "* = MONITOR": PRINT "! = FRE
    E BYTES" SPC( 11)"!F= HEX > DEC":
    PRINT "% = BIN. LOAD ADR" SPC( 8)"
    %1= DEC > HEX": RETURN
620  HOME : PRINT "&RENUMBER <START>,<INC
    >,<FIRST>,<LAST>": PRINT "&HOLD":
    PRINT "&MERGE": PRINT "&LENGTH": P
    RINT "&COMPRESS": PRINT "&SHOW": P
    RINT "&NOSHOW": PRINT "&AUTO <STAR
    T>,<INC>": PRINT "&MANUAL": PRINT
    "&XREF": PRINT "&KEYS"
630  INVERSE : VTAB 10: HTAB 30: PRINT "&
    GR L1,L2": HTAB 30: PRINT "&&=LINR
    EF": NORMAL : RETURN
```

GR + LINREF.SCE

```
 1 ;
 2 ;*** GROUPE DES LIGNES ***
 3 ;*** SUR L'ECRAN  POUR ***
 4 ;*** EDITION EVENTUELLE **
 5 ;
 6         ORG $7FD0
 7         OBJ $800
 8 ;
 9 CH      EPZ $24
10 LINNUM  EPZ $50
11 FORPNT  EPZ $85
12 LOWTR   EPZ $9B
13 DSCTMP  EPZ $9D
14 CHRGET  EPZ $B1
15 CHRGOT  EPZ $B7
16 TMINUS  EPZ $C9
17 ;
18 AMPERV  EQU $3F5
19 TKNTBL  EQU $D0D0
20 FNDLIN  EQU $D61A
21 NEWSTT  EQU $D7D2
22 ISCNTC  EQU $D858
23 LINGET  EQU $DA0C
24 CRDO    EQU $DAFB
25 OUTQST  EQU $DB5A
26 OUTDO   EQU $DB5C
27 SYNTAX  EQU $DEC9
28 LINPRT  EQU $ED24
29 ;
30 ;
```

```
 31 E.APA   EQU $8310
 32 ;
 33 DIVERS  CMP #$AF              ; &
 34         BNE AUTRE
 35         JMP LINREF
 36 AUTRE   CMP #$88              ; GR.OUPE
 37         BEQ COMPAC
 38         JMP E.APA
 39 COMPAC  JSR CHRGET
 40 STRTLN  JSR LINGET
 41         JSR FNDLIN
 42         JSR CHRGOT
 43         BEQ ENDMAX
 44         CMP #TMINUS
 45         BEQ ENDLN
 46         CMP #',
 47         BNE ERRP
 48 ENDLN   JSR CHRGET
 49         JSR LINGET
 50         BNE ERRP
 51 ENDMAX  PLA
 52         PLA
 53         LDA LINNUM
 54         ORA LINNUM+1
 55         BNE FIRSTL
 56 ERRP    JMP SYNTAX
 57 FIRSTL  LDY #1
 58         JSR CRDO
 59         INY
 60         LDA (LOWTR),Y
 61         TAX
 62         INY
 63         LDA (LOWTR),Y
 64         STY FORPNT
 65         JSR LINPRT
 66         LDY FORPNT
 67         JMP NOCR
 68 ;
 69 NXLN     LDY #1
 70         LDA (LOWTR),Y
 71         BEQ FIN
 72         JSR ISCNTC
 73         INY
 74         LDA (LOWTR),Y
 75         TAX
 76         INY
 77         LDA (LOWTR),Y
 78         CMP LINNUM+1
 79         BNE ENDRNG
 80         CPX LINNUM
 81         BEQ PRNMB
 82 ENDRNG  BCS FIN
 83 PRNMB   STY FORPNT
 84         LDA #':
 85         BNE PRCHR
 86 LSTLN   LDY FORPNT
 87         JMP NOCR
 88 PRCHR   JSR OUTDO
 89 NOCR    INY
 90         LDA (LOWTR),Y
 91         BNE PROCHR
 92         TAY
 93         LDA (LOWTR),Y
 94         TAX
 95         INY
 96         LDA (LOWTR),Y
 97         STX LOWTR
 98         STA LOWTR+1
 99         BNE NXLN
100 FIN     JSR CRDO
101         JMP NEWSTT
102 KEYCHR  INY
103         BNE K1
104         INC DSCTMP+1
105 K1      LDA (DSCTMP),Y
106 RTN0    RTS
107 PROCHR  BPL PRCHR
108         STY FORPNT
109         CMP #$BA             ;'PRINT'
110         BNE PRO1
111         JSR OUTQST
112         JMP LSTLN
113 PRO1    SEC
114         SBC #$7F
115         TAX
116         LDY #TKNTBL
117         STY DSCTMP
118         LDY /TKNTBL-$100
119         STY DSCTMP+1
120         LDY #$FF
121 NXKEY   DEX
122         BEQ PRKEY
123 N1      JSR KEYCHR
124         BPL N1
125         BMI NXKEY
126 PRKEY   JSR KEYCHR
127         BMI P2
128         JSR OUTDO
129         BNE PRKEY
130 P2      JSR OUTDO
131         JMP LSTLN
132 ;
133 ;* TABLEAU DES LIGNES APPELEES *
134 ;
135 ;
136 TXTTAB EPZ $67
137 CURLIN EPZ $75
138 FL1    EPZ $77
139 FL2    EPZ $78
140 OLDTXP EPZ $79
141 DATPTR EPZ $7D
142 ;
143 KBD    EQU $C000
144 KBDSTB EQU $C010
145 ;
146 COUT   EQU $FDED
147 ;
148 ;
149 ;
150 LINREF JSR CHRGET
151         LDA TXTTAB
152         STA OLDTXP
153         LDA TXTTAB+1
154         STA OLDTXP+1
155         LDA #0
156         STA FL2
157 D1      LDY #0
158         LDA (OLDTXP),Y
159         PHA
160         INY
161         LDA (OLDTXP),Y
162         BEQ D2
163         PHA
164         PLA
165         STA OLDTXP+1
166         PLA
167         STA OLDTXP
168         JMP D1
169 D2      PLA
170 ;
171         JSR CRDO
172         LDA TXTTAB
173         STA OLDTXP
174         LDA TXTTAB+1
175         STA OLDTXP+1
176 ENCORE  LDA #0
```

```
177          STA FL1
178          LDY #2
179          LDA (OLDTXP),Y
180          STA CURLIN
181          INY
182          LDA (OLDTXP),Y
183          STA CURLIN+1
184          JSR UNELIG
185          LDY #0
186          LDA (OLDTXP),Y
187          PHA
188          INY
189          LDA (OLDTXP),Y
190          BEQ TERMIN
191          STA OLDTXP+1
192          PLA
193          STA OLDTXP
194          JMP ENCORE
195 TERMIN   PLA
196          RTS
197 ;
198 UNELIG   LDA TXTTAB
199          STA DATPTR
200          LDA TXTTAB+1
201          STA DATPTR+1
202 U1       LDY #4
203 U2       LDA (DATPTR),Y
204          BEQ U4
205          CMP #$C5              ;AT
206          BCS U3
207          CMP #$85              ;DEL
208          BCC U3
209          BEQ YA1NUM
210          CMP #$AB              ;GOTO
211          BEQ YA1NUM
212          CMP #$AC              ;RUN
213          BEQ YA1NUM
214          CMP #$C4              ;THEN
215          BEQ YA1NUM
216          CMP #$B0              ;GOSUB
217          BEQ YA1NUM
218          CMP #$BC              ;LIST
219          BEQ YA1NUM
220 U3       INY
221          BNE U2
222 U4       LDY #0
223          LDA (DATPTR),Y
224          PHA
225          INY
226          LDA (DATPTR),Y
227          STA DATPTR+1
228          PLA
229          STA DATPTR
230          LDA (DATPTR),Y
231          BNE U1
232          RTS
233 ;
234 YA1NUM   JSR NUMGET
235          LDA FL2
236          BEQ Y4
237          LDA LINNUM
238          CMP CURLIN
239          BNE Y4
240          LDA LINNUM+1
241          CMP CURLIN+1
242          BNE Y4
243          TYA
244          PHA
245          LDA FL1
246          BNE Y1
247          LDA #1
248          STA FL1
249          JSR TOUCHE
250          JSR CRDO
251          LDA #$AD              ;TIRET
252          JSR COUT
253          LDY #2
254          LDA (OLDTXP),Y
255          TAX
256          INY
257          LDA (OLDTXP),Y
258          JSR LINPRT
259          LDA #7
260          STA CH
261          BNE Y3
262 Y1       LDA #$AC              ;VIRGULE
263 Y2       JSR COUT
264 Y3       LDY #2
265          LDA (DATPTR),Y
266          TAX
267          INY
268          LDA (DATPTR),Y
269          JSR LINPRT
270          PLA
271          TAY
272 Y4       LDA (DATPTR),Y
273          CMP #$2C              ;VIRGULE
274          BEQ YA1NUM
275          JMP U2
276 ;
277 NUMGET   LDA #0                ;PRESQUE
       LE LINGET APPLESOFT
278          STA LINNUM
279          STA LINNUM+1
280          STA FL2
281 C1       INY
282          LDA (DATPTR),Y
283          CMP #$20              ;ESPACE
284          BEQ C1
285 C10      LDA (DATPTR),Y
286          SEC
287          SBC #$30
288          BCC C3
289          CMP #$A
290          BCS C3
291          INC FL2
292          PHA
293          ASL LINNUM
294          ROL LINNUM+1
295          LDX LINNUM+1
296          LDA LINNUM
297          ASL LINNUM
298          ROL LINNUM+1
299          ASL LINNUM
300          ROL LINNUM+1
301          ADC LINNUM
302          STA LINNUM
303          TXA
304          ADC LINNUM+1
305          STA LINNUM+1
306          PLA
307          ADC LINNUM
308          STA LINNUM
309          BCC C2
310          INC LINNUM+1
311 C2       INY
312          BNE C10
313 C3       RTS
314 ;
315 TOUCHE   LDA KBD
316          BPL T2
317          STA KBDSTB
318 T1       LDA KBD
319          BPL T1
320          STA KBDSTB
321 T2       RTS
322 ;
323 LONG     EQU *-DIVERS
324 FINI     END
```

GR + LINREF

```
*7FD0.81D8

7FD0- C9 AF D0 03 4C 9B 80 C9
7FD8- 88 F0 03 4C 10 83 20 B1
7FE0- 00 20 0C DA 20 1A D6 20
7FE8- B7 00 F0 10 C9 C9 F0 04
7FF0- C9 2C D0 10 20 B1 00 20
7FF8- 0C DA D0 08 68 68 A5 50
8000- 05 51 D0 03 4C C9 DE A0
8008- 01 20 FB DA C8 B1 9B AA
8010- C8 B1 9B 84 85 20 24 ED
8018- A4 85 4C 45 80 A0 01 B1
8020- 9B F0 34 20 58 D8 C8 B1
8028- 9B AA C8 B1 9B C5 51 D0
8030- 04 E4 50 F0 02 B0 20 84
8038- 85 A9 3A D0 05 A4 85 4C
8040- 45 80 20 5C DB C8 B1 9B
8048- D0 1B A8 B1 9B AA C8 B1
8050- 9B 86 9B 85 9C D0 C6 20
8058- FB DA 4C D2 D7 C8 D0 02
8060- E6 9E B1 9D 60 10 DB 84
8068- 85 C9 BA D0 06 20 5A DB
8070- 4C 3D 80 38 E9 7F AA A0
8078- D0 84 9D A0 CF 84 9E A0
8080- FF CA F0 07 20 5D 80 10
8088- FB 30 F6 20 5D 80 30 05
8090- 20 5C DB D0 F6 20 5C DB
8098- 4C 3D 80 20 B1 00 A5 67
80A0- 85 79 A5 68 85 7A A9 00
80A8- 85 78 A0 00 B1 79 48 C8
80B0- B1 79 F0 0A 48 68 85 7A
80B8- 68 85 79 4C AA 80 68 20
80C0- FB DA A5 67 85 79 A5 68
80C8- 85 7A A9 00 85 77 A0 02
80D0- B1 79 85 75 C8 B1 79 85
80D8- 76 20 F0 80 A0 00 B1 79
80E0- 48 C8 B1 79 F0 08 85 7A
80E8- 68 85 79 4C CA 80 68 60
80F0- A5 67 85 7D A5 68 85 7E
80F8- A0 04 B1 7D F0 21 C9 C5
8100- B0 1A C9 85 90 16 F0 29
8108- C9 AB F0 25 C9 AC F0 21
8110- C9 C4 F0 1D C9 B0 F0 19
8118- C9 BC F0 15 C8 D0 DB A0
8120- 00 B1 7D 48 C8 B1 7D 85
8128- 7E 68 85 7D B1 7D D0 C8
8130- 60 20 85 81 A5 78 F0 44
8138- A5 50 C5 75 D0 3E A5 51
8140- C5 76 D0 38 98 48 A5 77
8148- D0 20 A9 01 85 77 20 C8
8150- 81 20 FB DA A9 AD 20 ED
8158- FD A0 02 B1 79 AA C8 B1
8160- 79 20 24 ED A9 07 85 24
8168- D0 05 A9 AC 20 ED FD A0
8170- 02 B1 7D AA C8 B1 7D 20
8178- 24 ED 68 A8 B1 7D C9 2C
8180- F0 AF 4C FA 80 A9 00 85
8188- 50 85 51 85 78 C8 B1 7D
8190- C9 20 F0 F9 B1 7D 38 E9
8198- 30 90 2C C9 0A B0 28 E6
81A0- 78 48 06 50 26 51 A6 51
81A8- A5 50 06 50 26 51 06 50
81B0- 26 51 65 50 85 50 8A 65
81B8- 51 85 51 68 65 50 85 50
81C0- 90 02 E6 51 C8 D0 CD 60
81C8- AD 00 C0 10 0B 8D 10 C0
81D0- AD 00 C0 10 FB 8D 10 C0
81D8- 60
```

CRAPA.OBJ

```
*300.37A

0300- A0 6E 8C F6 03 A0 03 8C
0308- F7 03 A0 01 84 2C A0 08
0310- D0 10 A0 61 8C F6 03 A0
0318- 03 8C F7 03 A0 02 84 2C
0320- A0 00 A9 00 85 08 85 06
0328- A9 D0 85 07 A9 74 85 09
0330- B9 83 C0 B9 83 C0 A2 10
0338- A0 00 B1 06 91 08 C8 D0
0340- F9 E6 07 E6 09 CA D0 F2
0348- A5 2C C9 02 D0 06 C6 2C
0350- A9 F0 85 07 A2 0C C6 2C
0358- F0 DE AD 8A C0 AD 82 C0
0360- 60 C9 27 F0 03 4C D0 7F
0368- 20 B1 00 4C 00 03 C9 27
0370- F0 03 4C 00 87 20 B1 00
0378- 4C 12 03
```

Conversion minuscules/Majuscules

Alexandre Avrane

Ce programme de conversion est destiné aux possesseurs d'Apple II ou d'Apple II+ sans ROM d'affichage des minuscules. Il convertit tous les caractères minuscules situés dans un programme Applesoft en caractères majuscules correspondants.

Il vient en complément du module SHIFT paru dans Pom's 14 et permet de modifier définitivement un programme sans devoir lancer SHIFT à chaque exécution.
De plus, il est destiné plus particulièrement aux programmes Applesoft qui tournent sous ProDOS (tel que C.Q.F.D.) car SHIFT ne fonctionne que sous Dos 3.3. Le programme une fois converti en majuscules sous DOS 3.3 peut effectivement l'être ensuite en ProDOS, sans problème.

L'utilisation est très simple et ne nécessite aucun commentaire particulier. En revanche, je vous laisse découvrir son principe de fonctionnement qui repose sur la génération en cascade de plusieurs fichiers EXEC. ■

SHIFT

```
*300.34A

0300- A9 0F 85 36 A9 03 85 37
0308- A9 00 85 48 4C EA 03 8D
0310- 4A 03 08 8A 48 AD 4A 03
0318- A2 07 DD 3A 03 F0 0C CA
0320- 10 F8 C9 E0 90 08 38 E9
0328- 20 D0 03 BD 42 03 8D 4A
0330- 03 68 AA 28 AD 4A 03 4C
0338- F0 FD C0 DC E0 FB FC FD
0340- FE FF C1 C3 A1 C5 D5 C5
0348- AD A1 00
```

```
1   REM   SHIFT.CONVERT
2   REM (C) 1984 ALEXANDRE AVRANE
3   REM 27/10/84
55   IF  PEEK (55) <  > 158 THEN  FLASH :
       PRINT "SHIFT.CONVERT DOIT S'EXECUT
       ER SOUS DOS!": NORMAL : END
100 D$ =  CHR$ (4)
110 Q$ =  CHR$ (34)
120  TEXT : HOME
130  PRINT "CONVERSION EN MAJUSCULES D'UN
       PROGRAMME  APPLESOFT COMPORTANT
       DES MINUSCULES"
140  PRINT : PRINT
150  INPUT "PROGRAMME INITIAL: ";F1$
160  INPUT "PROGRAMME CONVERTI:";F2$
170  PRINT D$"NOMONICO"
180  PRINT D$"BRUN SHIFT"
190  PRINT D$"OPEN Z1"
200  PRINT D$"DELETE Z1"
210  PRINT D$"OPEN Z1"
220  PRINT D$"WRITE Z1"
230  PRINT "LOAD"F1$
240  PRINT "63999";
250  PRINT ":?"Q$D$"CLOSE Z1"Q$;
260  PRINT ":?"Q$D$"OPEN Z2"Q$;
270  PRINT ":?"Q$D$"WRITE Z2"Q$;
280  PRINT ":POKE33,20";
290  PRINT ":LIST0-63998";
300  PRINT ":POKE33,40";
310  PRINT ":?"Q$"63999"Q$;
320  PRINT ":?"Q$"SAVE"F2$Q$;
330  PRINT ":?"Q$"DELETE Z1"Q$;
340  PRINT ":?"Q$"DELETE Z2"Q$;
350  PRINT ":?"Q$D$"CLOSE Z2"Q$;
360  PRINT ":?"Q$D$"MONI"Q$;
370  PRINT ":?"Q$D$"EXEC Z2"Q$
380  PRINT "RUN 63999"
390  PRINT D$"CLOSE Z1"
400  PRINT D$"EXEC Z1"
```

CALLs à gogo

Roland JOST

Troisième article destiné à vous familiariser avec la manipulation en Applesoft des ressources ultimes de votre Apple : après "Pokes à Gogo" (Pom's 11) et "Peeks à Gogo" (Pom's 13), voici les "Calls à Gogo". Peut-être aurions-nous dû appeler la série "PEEKs et POKEs et CALLe-gram" ?

Comparer deux zones mémoire, en déplacer une, afficher un nombre en hexadécimal, lire ou écrire un secteur sur la disquette, afficher de droite à gauche, et bien d'autres choses encore, sont possibles en utilisant les programmes du Moniteur, de l'interpréteur Applesoft ou du DOS. En effet il suffit souvent de POKEr quelques adresses en page zéro, de faire un CALL, et le problème est résolu.

Un certain nombre de sous-programmes peuvent être appelés directement à partir du Basic. Pour d'autres, il est nécessaire de charger l'accumulateur et/ou les registres d'index avant l'appel du sous-programme. Le programme en assembleur AMPERCALL vous facilitera la tâche. Dans la liste ci-dessous, ces CALLs sont précédés d'un ampersand (&).

Enfin, il est possible d'appeler des sous-programmes de l'interpréteur, à condition de POKEr certains paramètres en mémoire. Ces CALLS seront signalés par un #.

CALL −144 (65392) : lecture du tampon d'entrée et exécution des commandes : ce CALL est l'élément central de la célèbre routine de S.H.LAM :

```
C$=C$+" N D9C6G":FOR Z=1
TO LEN(C$):
POKE 511+Z,
ASC(MID$(C$,Z,1))+128:
NEXT:POKE 72,0:CALL −144
```

permet l'exécution de la chaîne C$ représentant une commande donnée sous Moniteur. Par exemple :

- Désassembler 20 instructions : C$ = "300L"
- Copier la page graphique 1 en page graphique 2 : C$ ="4000<2000.3FF8M"
- Afficher une zone mémoire en hexadécimal : C$ = "300.3FF"

Passons aux CALLs proprement dits :

CALL −151 (65385) : passage au mode Moniteur à partir du Basic.

CALL −155 (65381) : comme précédemment, avec émission d'un 'bip'.

CALL −167 (65369) : idem plus retour en mode TEXT.

CALL −182 (65354) : sauvegarde des registres A, X, Y, P et S respectivement en $45, $46, $47, $48 et $49 (69 à 73).

CALL −193 (65343) : restaure les registres à partir du contenu des adresses $45 à $49.

CALL −198 (65338) : émission d'un bip.

CALL −211 (65325) : affiche ERR et émet un 'bip'.

CALL −310 (65226) : lance l'exécution du programme machine pointé en $3F9 - $3FA (appelé par Ctrl-Y en mode moniteur).

CALL −321 (65215) : affiche les registres A, X, Y, P, S.

CALL −327 (65209) : restaure les registres et appelle le programme pointé par $3A - $3B (58 - 59).
Exemple d'utilisation : appel de la sous-routine RWTS :

```
IOB=256*(PEEK(PEEK(996)+256*
EEK(997))
+PEEK(PEEK(999)+256*
EEK(1000)) :
REM adresse de la table IOB
CALL −182 : POKE 69,IOB/256 :
POKE 71,IOB−PEEK(69)*256 :
POKE 59,3:
POKE 58,217: POKE 49,0: CALL
−327
```

Attention : les paramètres nécessaires doivent au préalable être POKés dans la table IOB (voir programme AMPERCALL).

CALL −336 (65200) : branchement au Basic. Le programme est effacé.

CALL −380 (65156) : passage en mode NORMAL.

CALL −384 (65152) : passage en mode INVERSE.

CALL −418 (65118) : passage en mode Moniteur et désassemblage de 20 instructions. L'adresse de début doit être POKée en $3A et $3B.

```
POKE 59,AD/256 :
POKE 58,AD−256*PEEK(59) :
CALL −418
```

désassemblera 20 lignes à partir de l'adresse AD. Un autre CALL −418 désassemblera les 20 lignes suivantes, etc...

# **CALL −458** (65078) : vérification de zones mémoire. Exemple :
D1 = début de la 1ère zone
F1 = fin de la zone 1
D2 = début de la zone 2
D1, F1 et D2 sont POKés respectivement en $3C - $3D, $3E - $3F et $42 - $43.

```
CALL −610: POKE 61,D1/256:
POKE 60,D1−PEEK(61)*256:
POKE 63,F1/256:
POKE 62,F1−PEEK(63)*256:
POKE 67,D2/256:
POKE 66,D2−256*PEEK(67): CALL
−458
```

Les différences seront affichées (le CALL −610 remet le registre Y à zéro, ce qui est absolument nécessaire car il est utilisé comme compteur par les routines VERIFY et MOVE).

CALL −468 (65068) : sous-programme MOVE permettant le déplacement de zones mémoire. Utiliser l'instruction précédente en remplaçant simplement CALL −458 par un CALL −468.
Exemple : recopier la page graphique 2 dans la page 1 :

```
CALL −610: POKE 61,64: POKE
60,0:
POKE 63,95: POKE 62,255: POKE
67,32:
POKE 66,0 :CALL −468
```

& **CALL −550** (64986) : affiche le contenu de l'accumulateur (en hexadécimal).

& **CALL −570** (64966) : addition ou soustraction de nombres hexadécimaux (inférieurs à $FF). Les deux nombres sont POKés en 60 et 62 et A doit contenir le code ASCII de + ou −.
Exemple :

```
INPUT "A,B":A,B : POKE
60,A:
POKE 62,B: &CALL −570,43,0,0 :
REM addition.
```

Pour la soustraction faire &CALL −570, 45, 0, 0.

CALL −589 (64947) : dump hexadécimal. L'adresse de début doit être POKée en 60 - 61 ($3C - $3D), l'adresse de fin en 62 - 63 ($3E - $3F).

CALL −629 (64907) : envoie un RETURN avec effacement de la ligne depuis le curseur jusqu'au bord droit de la fenêtre de texte.

CALL −651 (64885) : attente de la frappe d'une touche.

CALL −657 (64879) : attente d'une chaîne de caractères qui est stockée dans le tampon d'entrée. Tous les caractères sont acceptés : virgule, double point, etc.
Exemple :

```
CALL−657: POKE 72,0: CALL
−144
```

attente d'une instruction Moniteur, exécution de cette instruction et retour au Basic. L'instruction doit être suivie de "N D9C6G". Non utilisable en mode immédiat.

CALL −662 (64874) : comme ci-dessus mais affichage préalable du curseur.

CALL −665 (64871) : comme ci-dessus mais effectue d'abord un retour chariot.

CALL −715 (64821) : affichage du curseur et attente d'un caractère.

CALL −741 (64795) : affichage du curseur et attente d'un caractère. A utiliser à la place de GET si le programme appelle un fichier DOS.

& **CALL −856** (64600) : WAIT. La durée de la pause dépend de la valeur de l'accumulateur :
A délai en millisecondes
10 0,4
50 7,1
100 26,9
200 105,1
255 169,8

CALL −868 (64668) : efface la ligne de texte entre la position du curseur et la marge droite de l'écran (identique à ESC F).

CALL −912 (64624) : déplace tout le contenu de l'écran d'une ligne vers le haut. Exemple : pour un SCROLLING de l'écran de N lignes : FOR L=1 TO N : CALL −912 : NEXT

CALL − 922 (64614) : effectue un saut de ligne du curseur sans changement de colonne (identique à CTRL J).

CALL −926 (64610) : effectue un retour chariot (CR).

CALL −936 (64600) : vide la fenêtre d'écran texte et positionne le curseur dans le coin supérieur gauche de la fenêtre d'écran (correspond à l'instruction HOME de l'Applesoft).

CALL −958 (64578) : efface l'écran entre la position du curseur et le bas de la fenêtre (identique à ESC F).

CALL −998 (64538) : fait remonter le curseur d'une ligne.
FOR L=1 TO N : CALL −998 : NEXT : remonte le curseur de N−2 lignes.

CALL −1008 (64528) : déplace le curseur vers la gauche.
Testez les instructions suivantes :
A$="DEMONSTRATION":FOR J=1 TO LEN(A$):
HTAB 27 + J: FOR I=1 TO 20: CALL−1008:
CALL −1008:CALL −1008:
PRINT MID$(A$,J,1)" ":: NEXT I,J

CALL −1036 (64500) : déplace le curseur vers la droite.

CALL −1169 (64367) : modifie le RESET et le dirige vers le sous-programme dont l'adresse aura été POKée préalablement en 1010 et 1011 (décimal).
Exemple :

POKE 1010,102 :POKE 1011,213:CALL−1169
l'appui sur la touche RESET (précédé par CTRL sur le //e) fera exécuter le programme Applesoft. En effet le RESET pointe sur $D566 (54630 - voir plus loin).

CALL −1216 (64320) : passage en mode graphique basse résolution (GR). Plus généralement retour à la page graphique définie précédemment.

CALL −1223 (64313) : passage en mode TEXT.

CALL −1318 (64218) : affiche les registres A, X, Y, P, S.

CALL −1370 (64166) : démarrage à froid.

CALL −1438 (64098) : effectue un RESET.

& **CALL −1718** (63818) : affiche X blancs. Si X = 0 affiche 256 blancs.

#& **CALL −1988** (63548) : efface la portion haute et gauche de l'écran pointée par $2D (45) et Y. Exemple : POKE 45,Vert. : & CALL −1988, 0, 0, Horiz.

& **CALL −1992** (63544) : efface l'écran GR jusqu'à la ligne spécifiée dans Y (compris entre 0 et 47). Exemple : & CALL −1992, 0, 0, Y.

CALL −3082 (62454) : en mode haute résolution, peint tout l'écran avec la dernière couleur utilisée.
Exemple :
HCOLOR=C: HPLOT 0,0: CALL 62454
peint l'écran dans la couleur C (C compris entre 0 et 7).
Ou encore :
POKE 228,X: HPLOT 0,0: CALL 62454
avec X compris entre 0 et 255 : fonds divers.

CALL −3086 (62450) : vide la page HGR en cours d'utilisation. Peut se faire tout en restant en mode TEXT.

CALL −3102 (62434) : passage en mode HGR avec effacement (CALL 62430 pour ITT 2020).

CALL −3112 (62424) : passage en mode HGR2 avec effacement (CALL 62420 pour ITT).

CALL −3456 (62080) : passage en mode FLASH.

CALL −3465 (62071) : passage en mode INVERSE.

CALL −3469 (62067) : passage en mode NORMAL.

CALL −4818 (60718) : affiche la valeur actuelle de FAC (accumulateur flottant).

& **CALL −4828** (60708) : affiche en décimal les 2 octets dans X et A.

& **CALL −9414** (56122) : affiche une chaîne pointée par Y et A et se terminant par $00 ou $22.
Exemple :
A$="BONJOUR": FOR L=1 TO LEN(A$):
POKE 8191+L, ASC(MID$(A$,I,1)):NEXT L:
POKE L+8191,0: TEXT: HOME:
&CALL 56122,0,0,32

CALL −10601 (54935) : remet le pointeur en début de programme Basic et lance l'exécution. Les variables sont conservées.

CALL −10644 (54892) : équivalent de la commande CLEAR.

CALL −10677 (54859) : équivalent de la commande NEW.

CALL −10906 (54630) : exécution du programme Basic. Les variables sont perdues (commande RUN de l'Applesoft).

CALL −10964 (54572) : attente d'une chaîne de caractères qui sera stockée dans le tampon d'entrée. Tous les caractères tels que ";" ":" "," sont acceptés (non utilisable en mode immédiat).
Exemple :
CALL −10964: POKE 113,0: POKE 114,32:
&CALL 58850,255,0,2
saisie d'une chaîne de caractères dans le tampon d'entrée et déplacement en $2000. Cette chaîne pourra être affichée par un &CALL 56122,0,0,32.

CALL −20926 (44610) : affiche un nombre entre 0 et 255, cadré à droite sur 3 colonnes. Le nombre à afficher doit préalablement être POKé en 68 ($44).

CALL −22572 (42964) : déplace les buffers (tampons) du DOS. La nouvelle adresse doit être POKée en 40193 et 40192.
Exemple :
POKE 40193,PEEK(40193)−N: CALL 42964
libère N*256 octets entre $9C00 et les tampons en déplaçant ceux-ci de N pages vers le bas. HIMEM est diminué en conséquence et ni FP ni RESET ne modifient HIMEM. Cela est intéressant pour stocker dans une zone protégée un sous-programme qui ne risque pas d'être écrasé, sauf après un BOOT. Pour revenir à la normale, on pourra faire un POKE 40193, 156 : POKE 40192, 211 : CALL 42964.

CALL −23186 (42350) : effectue un CATALOG du dernier drive utilisé. Pour avoir le CATALOG des deux drives d'un slot, faire :
CALL 42350:
POKE 43624,3−PEEK(43624):
CALL 42350

CALL 1016 : lance l'exécution du programme machine pointé par CTRL-Y.

CALL 1013 : lance l'exécution du programme machine pointé par & ($3F5 - $3F7).

CALL 1002 : reconnecte les pointeurs du DOS.
Exemple : modifier le sous-programme de sortie :

POKE 54,0 : POKE 55,3 : CALL 1002
les sorties seront dirigées vers une routine en $300.

CALL 979 : ré-initialisation du DOS avec effacement du programme Basic.

CALL 976 : ré-initialisation du DOS sans effacement du programme Basic.

Conclusion

Dans certains cas, il est nécessaire de modifier le contenu des registres avant l'appel d'un sous-programme. Or l'Applesoft ne permet pas la modification directe d'un registre.
Pour y parvenir avant un CALL, il existe deux possibilités :

1) Utiliser les adresses de sauvegarde de ces registres ($46 et suivantes). La séquence d'instructions est alors la suivante :
– un CALL −182 sauvegarde les registres.
– on POKE la ou les nouvelles valeurs dans les adresses correspondant respectivement à A, X et Y.
– l'adresse du sous-programme à effectuer doit être POKée en $3A et $3B (58 et 59) qui sont les adresses de sauvegarde du PC (PCL et PCH).
– on termine par un CALL −327 qui restaure les registres et lance l'exécution du sous-programme pointé par PCL et PCH.

2) Utiliser l'utilitaire AMPERCALL qui fournit également la possibilité de POKEr dans deux octets mémoire consécutifs.

Appels :
- &CALL adresse,A,X,Y
  (fournir obligatoirement une valeur à chacun des registres)
- &POKE adresse,expression
  (toute valeur inférieure à 65535)

Ce programme est relogeable. Avant la première utilisation, il faut revectoriser l'ampersand.

■

AMPERCALL

```
0300-  C9 8C      CMP   #$8C     : instruction CALL ?
0302-  D0 26      BNE   $032A    :
                                   CALL
0304-  20 B1 00   JSR   $00B1    :
0307-  20 4A FF   JSR   $FF4A    : sauvegarde des registres
030A-  20 67 DD   JSR   $DD67    : on evalue la formule
030D-  20 52 E7   JSR   $E752    :
0310-  A5 50      LDA   $50      :
0312-  85 3A      STA   $3A      : l'adresse du sous-
0314-  A5 51      LDA   $51      : programme est mise
0316-  85 3B      STA   $3B      : en $3A et $3B
0318-  20 4C E7   JSR   $E74C    : evalue A
031B-  86 45      STX   $45      :
031D-  20 4C E7   JSR   $E74C    : evalue X
0320-  86 46      STX   $46      :
0322-  20 4C E7   JSR   $E74C    : evalue Y
0325-  86 47      STX   $47      :
0327-  4C B9 FE   JMP   $FEB9    : restauration des registres et appel
032A-  C9 B9      CMP   #$B9     : instruction POKE ?
032C-  F0 01      BEQ   $032F    :
032E-  60         RTS            : POKE
032F-  20 B1 00   JSR   $00B1    : saisie de l'adresse
0332-  20 67 DD   JSR   $DD67    :
0335-  20 6A DD   JSR   $DD6A    :
0338-  20 52 E7   JSR   $E752    :
033B-  A5 50      LDA   $50      :
033D-  85 06      STA   $06      :
033F-  A5 51      LDA   $51      :
0341-  85 07      STA   $07      :
0343-  20 BE DE   JSR   $DEBE    : teste la virgule
0346-  20 67 DD   JSR   $DD67    : evalue la variable
0349-  20 6A DD   JSR   $DD6A    :
034C-  20 52 E7   JSR   $E752    :
034F-  A0 00      LDY   #$00     :
0351-  A5 50      LDA   $50      :
0353-  91 06      STA   ($06),Y  :
0355-  C8         INY            :
0356-  A5 51      LDA   $51      :
0358-  91 06      STA   ($06),Y  :
035A-  60         RTS            : retour au BASIC.
```

Précisions et errata

Suite à l'article concernant le disque virtuel 64K, publié dans Pom's 14, un lecteur attentif apporte quelques précisions. Celles-ci vous permettront d'utiliser, sans perturbation éventuelle, ce programme en affichage 80 colonnes. Le nombre d'octets à protéger n'est plus alors 516 mais 1024.
Les modifications à apporter aux programmes RWAUXINIT et RWAUX sont les suivantes :
Dans RWAUXINIT remplacer :

– La ligne 51 par :
LDA #$0C

Dans RWAUX remplacer :

– Les lignes 12 et 13 respectivement par :
ORG $DF65
OBJ $5065

– La ligne 71 par :
LDY #$05

– A la ligne 123, L$100 devient L$102

– Et enfin, compléter la ligne 30 par 0607. Cette ligne devient alors :
TB2 HEX 000104050607.

Micro-Informations

Jean-michel Gourévitch

Applemaniaques, mes frères, voici un début d'année en forme de feu d'artifice : la ligne Apple II se diversifie et se multiplie, le Macintosh s'installe. Des nouveautés, en voici pour tout le monde.

Commençons par sonder les murs de Cupertino; là-bas, au siège d'Apple, on en a déjà beaucoup entendu, mais jamais autant qu'en ce début d'année 1985.

La première nouveauté va causer des joies, mais provoquer aussi certaines déceptions, l'Apple //c devient le nouveau standard de la famille Apple II. Une petite ligne qui déclenche de nombreuses conséquences :

−1 Les anciens Apple II et //e pourront être mis au niveau grâce à un kit que commercialisera Apple. Un nouveau processeur "65C02" remplacera le "vieux" 6502, donnant ainsi accès aux 27 nouvelles instructions. De nouvelles ROMs contenant les graphiques de la souris (Mousetext) se substitueront aux anciennes. Il est vraisemblable que, dans le courant de l'année, tous les Apple //e intègreront ce kit (l'apparition du //c n'a pas tué le //e). Au contraire, le //e ne s'est jamais aussi bien vendu; à tel point que les commerciaux d'Apple ont dû annuler d'urgence toutes les promotions à prix "cassés" du //e.

−2 La seconde conséquence est nettement moins agréable. La modification de mise à niveau s'effectue par ECHANGE du processeur, et non par addition d'une carte. Dans ces conditions, tous les programmes des Apples II qui tournaient mal ou bizarrement sur le //c seront affectés des mêmes défauts sur les ordinateurs "kités". Bonsoir la compatibilité ! Il y a des Multiplans, des PFS et des Applewriters qui ne s'en relèveront pas.

−3 La dernière conséquence entraînera une dépense supplémentaire pour tous ceux qui avaient rechigné devant l'achat d'une extension de mémoire. Le standard du //c étant d'une capacité de mémoire de 128 Ko, il faudra se procurer les 64 Ko supplémentaires (à moins de disposer d'une carte 80 colonnes étendues, d'une carte RVB Chat Mauve, etc...)

La deuxième nouveauté, et elle est de taille pour la famille des Apple II, est la naissance du "grand" frère du //c, connu actuellement sous le nom de //x. Dans une interview au mensuel britannique Apple Users, John Sculley, le directeur général d'Apple, a laissé entendre que le //x contiendrait, à la place du lecteur de disquettes de 5 pouces 1/4 traditionnel (et tragiquement limité à 143 Ko), un lecteur de 3 pouces 1/2 (le format des disquettes du Macintosh, d'une capacité de 360 Ko), voire même, un disque dur de ce format. Enfin, le nouveau //x serait équipé d'origine de 256 Ko de mémoire vive. La "bête" serait disponible courant 1985.

La troisième nouveauté est un réseau local destiné à l'utilisation de plusieurs Macintosh (jusqu'à 32). Le réseau utiliserait "l'Applebus", un serveur contenant un stockage de masse sur disque dur.
Enfin, comme si tout cela ne suffisait pas, Apple présenterait dans le courant de l'année une imprimante à laser dont le prix constituerait, à lui seul, une sorte de révolution (on parle de moins de 12000 F...)

En attendant, 1984 a vu le succès d'un ordinateur bien différent de Big Brother : le Macintosh s'impose, surtout avec 512 Ko. Déjà, à Cupertino, on envisage des perfectionnements : la couleur ! A en croire la revue américaine A +, le Mac en couleurs porterait comme nom de code Rainbow (Arc en ciel). Un circuit de Texas Instrument et une sortie sur l'arrière de Macintosh permettraient de le brancher sur un moniteur couleur. On nous avait bien prévenu chez Apple : ça ne sera pas donné (comptez 800 dollars sans le moniteur...)

Pour les voyageurs, le Macintosh de poche avec écran plat et alimentation autonome arrive. Un créneau porteur aux Etats Unis. Voici pour l'avenir, mais il existe déjà bien des moyens d'engloutir des fortunes dans les Apple d'aujourd'hui.

Améliorez votre Apple II

En commençant par la mémoire de masse. Une seule solution : le disque dur. Avec un nouveau Profile commercialisé par Apple. Allelouiah ! il a vu sa capacité doubler : 10 Megaoctets, contre 5 pour l'ancien, de quoi stocker le contenu de 70 disquettes de 5 pouces 1/4. Dommage que le prix soit conséquent : 23460 F TTC.
Rêvons un peu en consultant la publicité parue pour le "Sider", un disque dur, produit par First Class Peripherals, d'une capacité de 10 Megaoctets. Il supporte les systèmes Dos 3.3, ProDos, CP/M 2,23, CP/M 1.0, 1.5, 2.0, Apple Pascal 1.1 et 1.2 et permet de "booter" sur le disque dur. L'esthétique est superbe. Prix : 695 dollars. A vous de faire la conversion en francs... Il est vrai qu'Apple vient de mettre en place, pour les membres du Club Apple, une formule de crédit spécial baptisé Apple Check.

Désormais les accents circonflexes existent sur la version française d'Apple Works qu'Apple sort enfin (apparemment, la francisation n'était pas si facile). Apple Works, dont la version américaine a été testée dans Pom's, est un logiciel intégré et performant contenant un tableur, un gestionnaire de fichiers et un traitement de texte, pour Apple //e ou //c (accompagné d'un manuel de 330 pages et d'un livret de travaux pratiques pour 2490 F TTC).
L'une des lacunes d'Apple Works, était de ne pas disposer de fonctions de mailing et de correction. Voici l'oubli réparé (en anglais seulement hélas) grâce à Megaworks. Un logiciel de mailing et de corrections (30000 mots inclus et la possibilité d'en inclure 10000 autres). L'autre point faible d'Apple Works résidait dans les graphiques. Facile, avec Graphworks qui permet d'éditer sous forme de camemberts, d'histobarres ou de courbes, les données d'Apple Works (80 dollars). A noter, pour les "afficionados" d'Apple Works que celui-ci sera bientôt disponible pour Macintosh.

Imitez l'Apple //e

Pendant qu'Apple s'échine à transformer le //e en //c, les fabricants de périphériques se cassent la tête pour faire l'inverse. Qui a dit que le //c était une machine fermée ? Sûrement pas les britanniques de Cirtech. Ils ont réalisé un module CP/M qui s'installe dans le coffret du // c et joue le rôle d'une carte Z80. Elle est même, paraît-il un peu plus rapide. Prix : 95 Livres.
L'utilisation du //c avec une ancienne imprimante posait un problème : le standard des imprimantes était jadis parallèle et est devenu série. Voici donc des convertisseurs de série en parallèle. Ils permettent de brancher le //c sur un périphérique nécessitant une interface parallèle (c'était notamment le cas de l'ancienne imprimante DMP d'Apple). Le boîtier Hamlet de Belkin Components permet ce miracle (un mini miracle seulement, ma mère) pour 99 dollars. La serial box de PBI Software le réalise aussi pour 90 dollars.

Pour l'Apple II

Une carte multi fonctions, réalisée par AST Research, qui dispose d'un branchement pour modem ou pour imprimante, et en prime d'une horloge. Le tout sur la même carte pour consommer moins de courant.
Un complément de poignet pour l'ordinateur. C'est la montre Seiko PC Datagrap distribuée par Markline. Elle vous permet de stocker 2 Ko à votre poignet dans 12 dossiers différents. On peut y entrer des numéros de téléphone, des itinéraires, des dates, etc... La montre se relie à l'interface RS 232 C (interface série) de l'Apple. Un logiciel, livré avec, permet de transférer et de stocker les informations sur la montre. Prix de l'ensemble : 120 dollars. En plus, elle donne l'heure...

Enfin, un accessoire permettant de stocker des documents provenant de trois ordinateurs, et destinés à une seule imprimante parallèle. Le multi buffer de Scooter stocke 64 Ko, permet de les réimprimer à la demande, joue à la fois le rôle d'un buffer, et celui d'une boîte de sélection lorsqu'on ne dispose que d'une seule imprimante. Prix : 389 dollars.

A remarquer aussi une version à paraître du célèbre simulateur de vol Flight Simulator II de Sublogic. Cette nouvelle version permet de voler à plusieurs. Les Apples sont reliés par câble ou par modem, et on peut voler en formation, apercevoir l'appareil d'un ami, voire lutter contre lui en combat aérien...

Macintosh, c'est gagné.

Il existe, en matière de micro-informatique, deux moyens de s'assurer qu'un ordinateur s'impose sur le marché. La première, c'est de constater l'apparition des logiciels de copie de programmes. La seconde consiste à compulser les catalogues des fabricants d'accessoires. Pour Macintosh, pas de doute, c'est gagné. Voici une première version de BitCopy II Plus pour Mac (les amateurs apprécieront). Une revue américaine, Mac World, s'est déjà exclusivement voué au culte de la nouvelle idole.

Les accessoires

On en trouve pour tous les goûts et pour toutes les bourses. Kensington Microware propose un socle pivotant pour Mac, un modem portable, une unité de contrôle permettant d'alimenter l'ordinateur et ses accessoires, et de choisir entre la sortie modem, une imprimante et un accessoire. Des petits élévateurs métalliques, pour incliner l'imprimante Imagewriter vers l'avant, permettent simultanément de mieux voir ce qu'on imprime et de loger le papier sous l'imprimante.
MicroRain a construit un meuble, baptisé MacStation, qui loge le Macintosh, un drive, la documentation et la souris et permet d'installer l'imprimante au sommet.
Assimilation Process a remplacé la souris par un Mac Turbo Touch, une boule qu'on fait rouler avec la main.
Cette firme propose aussi des logiciels et des câbles pour relier le Macintosh à des imprimantes à marguerite et à des Epson.
Et surtout un logiciel, Mac Memory Disk, offrant l'utilisation d'une partie de la mémoire d'un Mac de 512 Ko comme pseudo disque. Les avantages sont : la suppression des accès programmes et la rapidité. Apple travaillerait aussi sur un tel disque.

D'autres logiciels permettent de relier le Mac à des imprimantes Nec (Macprint), ou à des imprimantes à marguerite (Proprint de Creighton).

A remarquer encore le Thunderscan, de Thunderware, une caméra qui permet de numériser des photos et de les reproduire avec Mac. A noter que cette caméra s'installe à la place de la cartouche ruban de l'Imagewriter, et qu'il suffit de faire défiler la photo à reproduire sur l'imprimante. Prix : 229 dollars.

Les logiciels

Des programmes pour Mac comme s'il en pleuvait. Des plus simples : des jeux de caractères comme ceux de Mac The Knife 2, ou Fluent Fonts de Casady. Des logiciels astucieux, comme Mac Publisher, de Boston Telecomputer, donne la possibilité d'éditer une lettre d'informations (100 dollars). Des programmes sophistiqués comme Fact Finder, de MacWare, permettant d'entrer des informations sous n'importe quelle forme, puis de les extraire facilement par mots clés, dates, etc...

Le Pascal UCSD pour le Macintosh

Le P-system est, depuis peu, implanté sur le Macintosh. En effet, BUS Informatique, unique représentant de SOFTECH en France, distribue la version IV.1 du Pascal UCSD.
Le système comporte, outre le compilateur Pascal, les compilateurs Fortran 77 et Basic, un éditeur pleine page, un assembleur, un générateur de code natif...
Sur le Macintosh, le Pascal UCSD s'emploie exactement comme sur toutes les autres machines. Comme dans les précédentes versions, on retrouve : les mêmes commandes (toujours sur deux niveaux; il est regrettable que la souris ne joue pas, ici, son rôle habituel !), la segmentation des programmes, la compilation séparée, les mémoires dynamiques...
Le Pascal UCSD permet d'accéder à toutes les fonctions du Macintosh. Il est muni d'une bibliothèque, lui donnant la possibilité d'utiliser MACDRAW. Une UNIT est destinée à la gestion de la souris...
La portabilité du système est assurée par BUS.

Catalogues du CXP

L'édition 1984 / 85 des catalogues du CXP est sortie : 10 tomes de 200 pages chacun en moyenne. Progiciels système, comptabilité, gestion financière et aide à la décision, de la production à la vente, paie et gestion du personnel, bureautique - gestion documentaire - CAO, professions libérales, progiciels sectoriels (2 tomes), progiciels scientifiques et techniques. Malgré toutes ces informations, on n'y trouve pas les logiciels de Pom's... Ces catalogues sont des répertoires et non des bancs d'essai.

Adresses :

Apple Seedrin - ZA de Courtaboeuf - av. de l'Océanie - BP 131 - 91944 Les Ulis Cedex.
First Class Peripherals - PO Box 6187 - Lehig Valley - PA 18001 USA.
Megahaus - 5M703 Oberlin d 2 - San Dlego - CA 92121 USA.
PBI Software - 1155 B H Chess Drive - Foster City - CA 94404 USA.
Cirtech - Currie Road - Ind. Estate - Galashiels - SelkirkshireTD1 2BP UK.
Belkin Components - 4718 W Rosecrans - Hawthorne CA 90250 USA.
AST Research - 2121 Alton Av. Irvine - California 92714.
Markline - PO Box C5 Belmont - MA O2178 USA.
Scooter, Ohm Electronics - 746 Vermont - Palatine IL 60067 USA.
Sublogic - 713 Edgebrook Drive Champaign - IL 61820.
Kensington Microware - 251 Park Av.South NY - NY 10010.
Creighton Dev. Inc. - 4931 Birch St. Newport Beach - CA 92660.
Thunderware - 19 G Orinda Way - Orinda - CA 94563.
Casady - PO Box 223 779 Carmel - CA 93922.
Boston Telecomputer - 19 Ledge Hill Road Boston - MA 02132. **Bus Informatique** - 3, rue de la Boétie - 75008 Paris - Tél 265 06 04.
CXP - 5, rue de Monceau - 75008 Paris - Tél 225 19 60.

Signature

Patrice Neveu

Certains systèmes utilisent le principe de la signature en fin de listing, pour identifier les programmes. Il peut être effectivement pratique de savoir rapidement si deux programmes sont différents ou si tel listing correspond bien à tel fichier sur disque.

C'est l'objectif de la routine en assembleur présentée ici. Elle a été écrite avec Big Mac et se place en $300. Un CALL 768 initialise le vecteur de l'ampersand à $30A. Ainsi, à tout moment, on obtient la signature du programme Applesoft en mémoire en tapant simplement &.

Quelques explications

Cette routine utilise les pointeurs :

– TXTTAB ($67 – $68) pour connaître l'adresse de début du programme Basic (généralement $801).

– PGREND ($AF – $B0) pour connaître l'adresse de fin du programme Basic.

Elle se sert également de routines courantes telles que COUT, CROUT, et LINPRT ($ED24) pour afficher en décimal les nombres hexa contenus dans les registres A et X.

Son principe

Considérer tous les octets du programme, ceux de rang pair et ceux de rang impair. On effectue un EOR des octets impairs (1er, 3ème...) avec l'octet de stockage IDENTIFIE et on réalise simultanément la même opération avec les octets pairs, sur l'octet IDENTIFIE + 1.

NB : il faut toutefois remarquer que PGREND situe plus exactement le début de la zone libre, non utilisée par le programme Basic. Il en résulte que SIGNATURE soustrait 3 à cette adresse, afin de pointer sur la fin du programme Applesoft et non sur le début de la zone utilisable.

■

```
 1
 2 *    CREE UN CODE IDENTIFICATEUR
 3 *     POUR PROGRAMMES APPLESOFT
 4
 5
 6 ***************
 7 * LABELS       *
 8 *              *
 9 ***************
10
11 TXTTAB     = $67
12 ALIRE      = $06
13 IDENTIFIE  = $08
14 PRGEND     = $AF
15 PGRFIN     = $18
16
17 AMPERV     = $03F5
18 COUT       = $FDED
19 LINPTR     = $ED24
20 CROUT      = $FD8E
21
22
23
24           ORG $300
25
26
27
28 *********************************
29 * INITIALISE L'AMPERSAND        *
30 *                               *
31 *********************************
32
33
34           LDA #$0A
35           STA AMPERV+1
36           LDA #$03
37           STA AMPERV+2
38
39
40 *********************************
41 * PREND EN TXTTAB ($67-68)      *
42 * LE DEBUT DU PROGRAMME, PUIS   *
43 * PREND EN PRGEND ($AF-B0)      *
44 * LA FIN DU PROGRAMME.          *
45 * FAIT UN 'EOR SUR 2 OCT. AVEC  *
46 * LES OCTETS DU PROGRAMME BASIC *
47 *                               *
48 *********************************
49
50
51           CLC
52
53           LDA TXTTAB
54           STA ALIRE
55           LDA TXTTAB+1
56           STA ALIRE+1
57
58           LDA PRGEND+1
59           STA PGRFIN+1
60
61           LDA PRGEND
62           CLC
63           SBC #$03
64           STA PGRFIN
65           BCC CONT4
66           JMP CONT5
67
68 CONT4     LDA PRGEND+1
69           STA PGRFIN+1
70           DEC PGRFIN+1
71
72 CONT5     LDA #$00
73           STA IDENTIFIE
74           STA IDENTIFIE+1
75
76           TAY
77
78 CONT      LDA ALIRE+1
79           CMP PGRFIN+1
80           BCC CONT1
81           BEQ CONT3
82           JMP CONT2
83
84 CONT3     CPY PGRFIN
85           BCS CONT2
86
87 CONT1     LDA (ALIRE),Y
88           CLC
89           INY
90           BNE CONT0
91           INC ALIRE+1
92
93 CONT0     EOR IDENTIFIE
94           STA IDENTIFIE
95           LDA (ALIRE),Y
96           EOR IDENTIFIE+1
97           STA IDENTIFIE+1
98
99           CLC
100          INY
```

```
101             BNE CONT
102             INC ALIRE+1
103             JMP CONT
104
105 CONT2       JSR CROUT
106             LDY #$00
107
108 PRNTCODE    LDA CODEMSG,Y
109             BEQ AFFCODE
110             JSR COUT
111             INY
112             BNE PRNTCODE
113
114 AFFCODE     LDA IDENTIFIE+1
115             LDX IDENTIFIE
116             JSR LINPTR
117
118             RTS
119
120
121 *********************************
122 * MESSAGES                      *
123 *                               *
124 *********************************
125
126
127 CODEMSG     EQU *
128             ASC "SIGNATURE: "
129             HEX 00
130
```

```
*300.37D

0300- A9 0A 8D F6 03 A9 03 8D
0308- F7 03 18 A5 67 85 06 A5
0310- 68 85 07 A5 B0 85 19 A5
0318- AF 18 E9 03 85 18 90 03
0320- 4C 29 03 A5 B0 85 19 C6
0328- 19 A9 00 85 08 85 09 A8
0330- A5 07 C5 19 90 09 F0 03
0338- 4C 5A 03 C4 18 B0 1B B1
0340- 06 18 C8 D0 02 E6 07 45
0348- 08 85 08 B1 06 45 09 85
0350- 09 18 C8 D0 DB E6 07 4C
0358- 30 03 20 8E FD A0 00 B9
0360- 72 03 F0 06 20 ED FD C8
0368- D0 F5 A5 09 A6 08 20 24
0370- ED 60 D3 C9 C7 CE C1 D4
0378- D5 D2 C5 BA A0 00
```

```
]LIST:DEMOSIGNATURE

1  TEXT : POKE 33,40: HOME : INVERSE : HT
       AB 14: PRINT "DEMOSIGNATURE": NORM
       AL
5  PRINT  CHR$ (4)"BLOAD SIGNATURE"
10  VTAB 5: PRINT "'SIGNATURE' EST UNE SO
       US-ROUTINE          PERMETTANT AUX P
       ROGRAMMES APPLESOFTS DE SIGNER, C'
       EST A DIRE DE DONNER LEUR      IDEN
       TITE."
20  PRINT : PRINT "PAR EXEMPLE, LA SIGNAT
       URE DE CE           PROGRAMME EST :"
       : CALL 768
25  PRINT
```

Trucs et Astuces

Eurêka ! Il existe un PEEK permettant de savoir si un point de l'écran haute résolution est allumé ou éteint.

A = PEEK(Y/64) :
D = Y - 64 * A :
B = INT(D/8) :
C = D - 8 * B :
E = INT(X/7) :
P = PEEK(8192 * PG + 1024 * C + 128 * B +40 * A + E) :
P = P/2^(X - E * 7)/2 :
P = (P-INT(P)>.5)

avec X, Y et PG les coordonnées horizontale, verticale et le numéro de la page haute résolution en cours, on obtient P = 1 si le point est allumé, P = 0 sinon.
Si vous n'aimez pas les lignes Applesoft aussi complexes, voici encore plus court mais en assembleur.
Après avoir BRUNé le programme HSCREEN (relogeable) suivant, il suffit de taper :
POKE 226,Y :
C = USR (X)
avec X et Y les coordonnées et P défini comme précédemment.

L'Ampersand est une merveilleuse instruction que nous a offert l'Applesoft (très discrètement d'ailleurs). Malheureusement, la quasi totalité des routines qui ont été développées se montrent d'un égoïsme prodigieux à l'égard de leurs consoeurs.
Nombreux sont donc les lecteurs de Pom's qui, ignorant l'assembleur, doivent perpétuellement charger les mêmes fichiers pour pouvoir utiliser simultanément plusieurs routines sous Ampersand.
Une méthode relativement simple, mais cependant peu connue consiste à modifier tous les appels de la forme :
& <paramètres éventuels>
par l'instruction :
CALL <adresse> , <paramètres éventuels>(selon les cas, il faudra ou non inclure une virgule entre l'adresse et les paramètres).
Pour calculer l'adresse, il suffit d'initialiser la routine de manière habituelle, puis de calculer :
ADR = PEEK(1014) + 256 * PEEK(1015)
Après avoir remplacé tous les appels à cette routine via l'Ampersand par un CALL à la valeur ADR, on peut alors charger une seconde routine qui sera directement appelée par l'&.
Vite un exemple : tapez FP pour fixer HIMEM au plus haut, puis lancez l'utilitaire RENUMBER de la disquette système du Dos; vous obtiendrez :
ADR = 36352 et il sera possible de lancer RENUMBER par :
CALL 36352 pour renuméroter un programme et
CALL 36352H (au lieu de &H) pour le placer en attente de fusion.

Certaines cartes graphiques Epson (Rom APL.B) posent parfois, et de manière inexplicable, de sérieux problèmes; par exemple, les impressions d'assemblage par Big Mac se révèlent impossibles.

L'origine provient de la ROM de la carte : à son entrée les trois premières adresses de la page zéro ($0000 à $0002), qui seront utilisées, sont sauvegardées (en $C808) sur la pile dans l'ordre 0 - 1 - 2. En sortie, malheureusement, on les restaure (en $C99F) dans l'ordre 0 - 2 - 1. Erreur fatale qui, en inversant deux adresses, fait chuter à coup sûr tout programme les utilisant.

La solution serait simple s'il ne s'agissait pas d'une ROM, donc relativement coûteuse à faire refaire par un particulier. La parole est donc donnée à M3C et Technology Ressources, respectivement ancien et actuel importateurs d'Epson.

Courrier des lecteurs

Alexandre Avrane et Alexandre Duback

Je voudrais soulever le problème de la compatibilité du logiciel MOUSEPAINT avec d'autres imprimantes que l'IMAGEWRITER. Ayant, en effet, cédé à l'attrait de la souris AppleMouse sur le //e (qui permet de rêver à un Macintosh plein de slots...), me voici bien déçu de ne pouvoir sortir les mises en page de MOUSEPAINT sur mon Epson RX-80FT.
Je me tourne vers Pom's et ses lecteurs (avant d'aller réclamer à Cupertino), pour trouver la solution logicielle ou matérielle à cette limitation des périphériques utilisables avec MOUSEPAINT.

Yvan Pellecuer - 69300 Caluire

Trois questions sur ProDOS :
1-Quels sont les manuels Apple disponibles ?
2-Comment le fichier Startup charge-t-il le Basic.System ?
3-Comment formater une disquette ProDOS sans passer par la disquette des utilitaires système ?

Christian Coquelet - 77420 Champs-sur-Marne

Pour ces trois points, reportez-vous à l'article sur ProDOS de ce numéro.

Après avoir fait la conversion sous ProDOS d'un programme de gestion de stock, celui-ci refuse de fonctionner. Les instructions du genre "Poke 43646, x" pour changer de lecteur ne sont pas appréciées. La lecture des fichiers directs pose aussi quelques problèmes. Enfin, le manuel livré détaille le fonctionnement des utilitaires mais quid des instructions nouvelles et de la syntaxe ?.

Michel Jacquemard - 74420 Boège

Stop ! Eloignez-vous tous de votre Apple et notez bien : les programmes, conseils, astuces diverses et autres que vous avez pu apprendre pour le Dos 3.3 ne fonctionnent pas sous ProDOS. Au mieux, un POKE ou un CALL intempestif se révèleront sans conséquence; au pire, vous aurez détruit votre programme en mémoire, voire l'intégrité de votre disquette. Malgré des apparences semblables pour l'utilisateur, le DOS 3.3 et ProDOS diffèrent radicalement dans leur architecture interne. Même les CALLs effectués vers les vecteurs de la page 3 ne donnent plus les mêmes résultats. Donc abstinence de tout patch dévastateur; promis ?

L'acquisition récente d'un Apple //c m'a fait passer de la préhistoire de l'informatique (je possédais un Sharp PC-1211) à l'ère moderne. Votre revue, fort bien faite, contribue largement à mon initiation. En tant que nouvel utilisateur, de nombreuses questions se posent à moi :
1-Dans un article sur la compatibilité Apple //c (Pom's 13), Guy Lapautre parle de programmes utilitaires tels que renumérotation et fusion de programmes qui figureraient sur la disquette utilitaires système de l'Apple //c; or je n'y ai pas trouvé ces programmes.
D'autre part, il semble exister des utilitaires ProDOS tels que FILER, CONVERT, etc... que je ne possède pas. Où puis-je me les procurer ?
2-Ma première utilisation, outre les jeux (on ne résiste pas à Lode Runner), a été la gestion familiale (il faut se donner bonne conscience). Or, je n'ai pu réussir à convertir sous ProDOS le programme Gescompte de Pom's 10.
3-Comment modifier Gescompte pour qu'il travaille en 80 colonnes ?
4-Désirant m'initier au Pascal, j'ai acheté le livre "Découvrez Pascal sur Apple II". Il y est dit que, lorsque l'on ne possède qu'un lecteur, il faut d'abord booter sur la disquette Apple3, puis insérer la disquette Apple0 et presser Reset. Je suis incapable d'activer Pascal avec cette méthode; y a-t-il un Pascal spécial pour le //c ?

P. Pariysiti - 94370 Noiseau

1-L'utilitaire de renumérotation est inclus dans une disquette ProDOS Tool-Kit commercialisée séparément. En revanche, FILER, CONVERT et le ProDOS Formatter sont normalement livrés avec ProDOS. Demandez à votre revendeur de vous les fournir.
2-Convertissez à nouveau Gescompte en ProDOS, en partant de votre fichier correct sous DOS 3.3; avant de l'exécuter, modifiez toutes les lignes contenant l'instruction VERIFY (cette commande DOS 3.3 n'existe pas sous ProDOS), en la remplaçant par une instruction OPEN suivie d'un CLOSE (lignes 2040, 21060,...). D'autre part, les noms de fichiers ProDOS ne peuvent contenir plus de 15 caractères et ne doivent pas comporter, en particulier, d'espace : modifiez les lignes 1220 à 1230 en conséquence.
3-Il n'est pas très difficile de modifier Gescompte pour le mode 80 colonnes. Si un lecteur s'est passionné pour le problème, qu'il nous en fasse part et nous transmettrons.
4-Bizarre, bizarre, ça devrait fonctionner... Toutefois, les anciennes versions de Pascal UCSD ne parviennent pas à booter sur le //c car elles sont incapables de reconnaître la "carte 80 colonnes". Peut-être votre problème vient-il de là.

Comment obtenir sur imprimante les accents circonflexes et les trémas, sans passer par un logiciel de traitement de textes ?

J.-R. Vailong - 73000 Chambéry

La notice de votre imprimante vous fournit le code ASCII de l'accent circonflexe et du tréma (généralement 94 et 126 respectivement); il faut alors insérer un caractère "espace arrière" (code ASCII 8) entre la lettre et son accent. La ligne Basic suivante écrit la phrase "Pour être ivre à Noël... ":

```
PRINT "Pour e"; CHR$(8); CHR$(94); "tre ivre à Noe"; CHR$(8); CHR$(126); "l..."
```

Je possède un Apple II avec une carte langage équipée d'une Eprom 2716 me faisant accéder à des fonctions très utiles (interruption du programme, sortie par Reset, commande de reboot). Malheureusement ProDOS refuse de se charger, et j'ai donc du remettre la ROM normale. Comment pourrais-je réutiliser mon Eprom sans devoir à chaque fois déssouder et ressouder ?

Hong Hai Vuong - 75015 Paris

Votre Eprom, un peu particulière (Lockbuster ?), ampute votre carte langage de 2 K octets de RAM, et ProDOS se trouve trop à l'étroit dans les 14 Ko restants pour fonctionner. La solution la plus économique est probablement de racheter une seconde carte langage et d'y placer ProDOS. Pour déplacer ProDOS vers un slot autre que le slot 0, observez la page mémoire $BF, et plus particulièrement les adresses $BFA0 −$BFF5; elles contiennent les vecteurs utilisés par ProDOS pour activer alternativement ROM et RAM.

Comment peut-on transférer un fichier sur IBM 36 pour le traiter sur Lisa ?

S.Berthier - 37000 Grenoble

A priori, les logiciels LisaTerminal et MacTerminal permettent une liaison synchrone aux normes IBM 3270/BSC et SNA/SDLC, ou asynchrone (TTY et VT52/VT100 de Digital Equipment). D'autre part, on peut toujours brancher deux modems, mais le débit est bien sûr beaucoup plus faible (1200 bauds contre 1 Méga-bauds maximum). Si un lecteur a tenté l'expérience...

Comment peut-on bloquer le clignotement du curseur et modifier la sonnette (CTRL-G) de l'Apple comme le font certains logiciels ?

C.Babin - 33200 Mérignac

Ces deux modifications ne peuvent s'effectuer qu'en assembleur, en modifiant en $36 – $39 les adresses respectives des routines de sortie et d'entrée des caractères. Voici, à titre d'exemple, une routine (tournant sous DOS 3.3) donnant un "beep" semblable à celui des utilitaires fournis avec ProDOS :

```
 1 *       *******
 2 *       * ZIO *
 3 *       *******
 4
 5 * (C) 1984 Alexandre Avrane
 6
 7 * Routine modifiant le "Bell"
 8 * (Ctrl-G) du Moniteur
 9
10            LDA   #<CSW2
11            STA   $36
12            LDA   #>CSW2
13            STA   $37
14            JMP   $3EA
15
16 CSW2       CMP   #$87        ctrl-G ?
17            BNE   CSW2X
18            LDA   #$20
19            STA   CSW2ZZ
20 CSW21      LDA   #2
21            JSR   $FCA8
22            STA   $C030
23            LDA   #$24
24            JSR   $FCA8
25            STA   $C030
26            DEC   CSW2ZZ
27            BNE   CSW21
28            RTS
29 CSW2X      JMP   $FDF0
30 CSW2ZZ     DFB   0
```

J'essaie de comprendre les documentations en anglais, fournies par l'importateur de l'Enhancer II, le Sup'R Terminal et la carte Legend 64k mais j'y perd mon "français".

J.P. Merle - 13001 Marseille

A notre connaissance, les traductions de ces manuels n'existent pas.

1-Comment obtenir sur l'imprimante et à l'écran les nombreux caractères annoncés par le menu "Pomme" du Macintosh ? Pour un texte scientifique, beaucoup d'entre eux seraient très utiles : racine, lettres grecques, symboles mathématiques...

2-Comment obtenir plus de 6 lignes d'impression par pouce ?

3-Comment récupérer les caractères accentués sur l'imprimante à partir du Basic ?

4-Comment lister le catalogue d'une disquette sur l'imprimante sans passer par la copie d'écran ?

Jacques Boutique - 83 rue Louis Casimir Ranson - 87000 Limoges

1-L'option "clavier" dans le menu "Pomme" n'est accessible qu'avec la police de caractères Chicago. Pour les caractères spéciaux que vous souhaitez obtenir, il serait bon de changer de police. La plus adaptée aux textes scientifiques serait, semble-t-il, la police Geneva. Pour une impression sur papier, si votre texte est un fichier Mac Write, vous n'aurez aucun problème.

2-Par défaut, l'impression est de 6 lignes par pouce (ESC A). Il est toutefois possible de la modifier. En effet, ESC T24 correspond à un interligne classique et ESC T16 supprime les interlignes. Ainsi, pour un espacement de lignes particulier il suffit de taper en mode immédiat :
LPRINT CHR$(27) "TXX"; XX étant la valeur choisie. Cette commande peut être introduite dans un programme.
Pour obtenir 8 lignes par pouce tapez simplement :
LPRINT CHR$(27) "B".

3-Lorsque vous listez un programme Basic, l'imprimante reçoit des codes ASCII. S'ils sont inférieurs à 32 ou supérieurs à 127, l'imprimante ne comprend plus ! Deux solutions sont alors à envisager :
– Transférer votre programme dans un fichier Mac Write et lister (attention vous ne pouvez transférer que 8 K à la fois).
– Utiliser le programme ci-dessous; il remplace tous les caractères accentués par des "@" et envoie l'équivalent d'un List.

```
10 'Le programme à lister doit
   être préalablement sauvegardé
   sous la forme d'un fichier
   "texte".
20 CLS:INPUT "fichier ? ",F$:
   OPEN"i",1,F$
30 WHILE NOT EOF(1)
40 LINE INPUT#1,L$
50 FOR B%=1 TO LEN(L$)
60 C$=MID$(L$,B%,1):C%=ASC(C$):
   IF C%<32 OR C%>127 THEN LPRINT
   "@"; ELSE LPRINT C$;
70 NEXT:LPRINT
80 WEND
90 CLOSE
```

4-Nous ne nous sommes pas encore penchés sur ce problème, mais tenterons de le faire dès que possible.

J'ai lu dans Pom's 13 la lettre de M. Delacourte, selon laquelle PURPLESOFT ne tolèrerait pas de programme Basic de plus de 2,5 K. Fort heureusement, il n'en est rien. Il suffit de charger PURPLESOFT avant le programme Basic. Pour éviter une limitation à 6,5 K il est nécessaire de charger le programme Basic au-dessus de la page graphique 1.

L'utilisation sous PURPLESOFT d'un programme Basic (12000 octets – chargé en 16384) exploitant la routine PROG48K (publiée dans Pom's 5), relogée au-dessous de la page graphique 1, est possible grâce à la dérivation SYNTAX ERROR décrite dans le manuel de la carte Chat Mauve, qui permet d'employer conjointement les ampersands de PROG48K et de PURPLESOFT.

Monsieur Bodin - La Prunerie - 24430 Razac

Le programme TRANS que j'ai développé, écrit en assembleur Z80, permet la communication entre un Apple sous CP/M et une autre machine sous le même système. L'utilisateur est guidé pas à pas, en particulier pour les opérations de commutation du modem.
La transmission binaire, 8 bits avec parité, se fait par blocs de 256 octets avec polling et cheksum. Tous les fichiers sont transmissibles en toute sécurité. La vitesse de transmission est fixée par défaut à 1200 bauds; il est toutefois possible de la modifier (elle peut aller jusqu'à 9600 bauds).

Plusieurs versions de TRANS sont en cours de réalisation.
En première installation, TRANS est livré pour deux sites en relation avec une assistance technique au prix de 4000 F HT; en extension, pour un nouveau site relié à un ensemble déjà équipé, il en coûtera 1000 F HT.
Pour plus de renseignements n'hésitez pas à me contacter.

Léon Kuntz - 65 rue Nollet - 75017 Paris

■

Bibliographie

Alexandre Duback

Pour Apple //e et //c

Apple programming for learning and teaching de Frederick H. Bell, Prentice Hall - 305 pages - $22.05.

Un cinquantaine de programmes détaillés, et environ 80 petits programmes exemples, pour apprendre à programmer. Bien fait et pédagogique.

Handbook of Applesoft Basic for the Apple II and //e de Roy Earl Myers et David I. Schneider, Prentice Hall - 320 pages - $22.05.

Consacre une ou plusieurs pages à chaque mot réservé du Basic Applesoft. Utilisable aussi par les possesseurs de //c. Bien fait, bon ouvrage de référence pour les anglophones.

Voici l'Apple //c de D. Goodman, Hachette - 165 pages - 95 FF. Traduit de l'anglais.

Voici le premier livre "en français" consacré exclusivement à l'Apple //c. Cet ouvrage s'adresse aux débutants et détaille l'appareil, vu de l'extérieur comme de l'intérieur. On a droit à un chapitre de comparaisons avec d'autres matériels : à quoi cela peut-il bien servir dans un ouvrage destiné aux possesseurs du //c ? Ensuite, un chapitre par application présente les possibilités du matériel : traitement de texte, calculs, gestion de fichers, jeux, ... Clair et propre, mais pas essentiel.

La programmation du 6502 de A.P. Stephenson, MicroDunod - 184 pages - 95 FF. Traduit de l'anglais.

Bon ouvrage d'initiation au langage machine du 6502.

L'Apple animé 3D de Phil Cohen, Eyrolles - 184 pages - 90 FF. Traduit de l'anglais.

Apprend à programmer le graphique et fournit des listings de programmes de création de tables de formes, d'édition... Il n'y a que des programmes Basic, agrémentés d'un texte explicatif bien détaillé.

The power of Appleworks de Robert E. Williams, Prentice Hall - 230 pages - $25,95.

Compilation assez complète, quoiqu'un peu primaire, des potentialités du logiciel. Hélas, seule une partie des possibilités de communication entre les trois applications (traitement de texte, gestion de fichiers, tableur) est expliquée, et ce de façon accessoire. On devrait pouvoir faire mieux sur le sujet, mais pour le moment il n'y a rien d'autre.

Pour le Macintosh

Le guide Marabout du Macintosh de Robert Van Loo, Marabout - 200 pages - 185 FF.

Pour les débutants du Macintosh, présentation générale du Mac et introduction à l'utilisation de MacPaint, MacWrite et Multiplan (version US). Ce livre a été rédigé avec le Macintosh. Ceci dit, travail honnête, mais qui apporte peu au-delà des documentations originales : il y a déjà deux ou trois livres de ce genre en français.

Guide de l'utilisateur Macintosh de Joseph Caggiano, Sybex - 180 pages - 98 FF. Traduction.

Mêmes objectifs que l'ouvrage précédent, mais avec deux fois moins de matière (présentation plus aérée).

Macintosh le magnifique de Merl K. Miller et Mary A. Myers, Editions du PSI - 157 pages - 95 FF. Traduit de l'anglais.

Mêmes thèmes que les ouvrages précédents, mais il s'agit beaucoup plus d'une présentation des possibilités que de l'apprentissage du Macintosh et de ses programmes standards.

Multiplan pour Macintosh de Gérard Santraille et Hervé Thiriez, Editions du PSI.

Ouvrage complet sur le Multiplan du Macintosh, comportant de nombreux tableaux préparés relatifs à divers domaines d'application. C'est le premier ouvrage écrit sur le programme Multiplan en français, version 1.02 (juillet 84).
Une disquette d'accompagnement facultative reprend tous les tableaux de l'ouvrage et propose en outre un programme écrit en Basic Microsoft pour obtenir la liste détaillée du contenu d'un tableau : formules, formats, noms de zones, découpages en fenêtres, ...

D'intérêt général

Guide de l'Informatique Individuelle de H. Varley et I. Graham, Hachette - 225 pages - 99 FF. Traduit de l'anglais.

Cette encyclopédie couvre, à raison d'une à deux pages par thème, une grande variété de sujets liés de près ou de loin à la micro-informatique. Clair et bien présenté. Educatif.

Encyclopédie de la micro-informatique de Peter Rodwell, Hachette - 206 pages - 149 FF. Traduit de l'anglais.

Même principe que l'ouvrage précédent, en grand format (21 * 29,7). Nombreuses illustrations en couleurs. Parfait pour les étudiants.

dBase II sans embûches de G. Grigorieff, Eyrolles - 176 pages - 115 FF.

La pratique de dBase II de Jean-Claude Guillemot, Eyrolles - 208 pages - 140 FF.

La pratique de dBase II de Carl Townsend, Turgeon - Diffusion PSI - 210 pages - 200 FF. Traduit de l'anglais..

Les avis sont partagés sur ces trois ouvrages qui, en première analyse, paraissent tous sérieux. Le premier a l'avantage de s'articuler autour de la programmation d'une application complète (librairie). Le second possède un sommaire plus détaillé que le troisième.

■

Le clavier du //c

Au fil des jours d'utilisation du //c, certaines touches nécessitent une "force de frappe" supérieure à la normale. Pour y remédier, voici quelques conseils.
– Otez tous les cabochons du clavier.
– Retirez les deux caches en plastique (attention l'un d'entre eux est collé en quelques points au clavier)
– Vous découvrez alors un superbe clavier et, à la base de chaque tige blanche (support des cabochons), un petit ressort.

La première solution consiste à retirer tous les ressorts, la seconde à tordre légèrement une des branches de chacun d'entre eux en l'éloignant de la tige blanche. Remontez le tout et votre clavier sera comme neuf !

Multiplan et le //c

Si vous utilisez Multiplan en 80 colonnes sur le //c, vous risquez quelques surprises. En effet, le progamme ne reconnaît plus l'IMAGEWRITER et votre tableau à l'écran ne sera plus qu'une ligne de caractères "w" sur papier. Pour un tableau de 8 colonnes standards (80 caractères par ligne) il vous suffit de recharger Multiplan, configuré en 40 colonnes, et d'imprimer. En revanche, pour un tableau de 13 colonnes (130 caractères par ligne) il faut utiliser les caractères ultra comprimés de l'IMAGEWRITER. Tout est expliqué dans Pom's 9. Cependant, il est nécessaire d'y apporter une modification. En commande SORTIE, sous-commande OPTIONS, il faut remplacer l'initialisation ^!c par ^°c (c étant un caractère de contrôle lié à l'imprimante) car ! n'est autre qu'une commande de recalcul manuel du tableau en mémoire. Il ne reste plus qu'à modifier, dans la commande PAGE, le nombre de caractères par ligne en fonction du caractère de contrôle choisi.

pom's

DISQUETTES (sauf précision, toutes les disquettes fonctionnent sous DOS 3.3 sur Apple II + , //e ou //c)

HAIFA	(cf. Pom's n° 5)	à	55,00 F	
H-BASIC	(cf. Pom's n° 8)	à	150,00 F	
MUSIC	(cf. Pom's n° 10)	à	80,00 F	
DISK-MANAGER	(cf. Pom's n° 11)	à	450,00 F	
DBSTAG (CP/M)	(cf. Pom's n° 11)	à	450,00 F	
JEUX A	(cf. Pom's n° 12)	à	80,00 F	
JEUX B	(cf. Pom's n° 12)	à	80,00 F	
BASICIUM	(cf. Pom's n° 13)	à	150,00 F	
DEMO CX SYSTEME 64K	(cf. Pom's n° 13)	à	55,00 F	
DEMO JANE 64K	(cf. Pom's n° 13)	à	55,00 F	
E.P.E.	(cf. Pom's n° 15)	à	150,00 F	
MACINTOSH	(cf. Pom's n° 15)	à	150,00 F	
PASCAL	(cf. Pom's n° 15)	à	80,00 F	

RECUEILS

N° 1, recueil des revues 1 à 4	à	130,00 F	
Disquettes d'accompagnement des numéros 1 à 4	à	150,00 F	
N° 2, recueil des revues 5 à 8	à	130,00 F	
Disquettes d'accompagnement des numéros 5 à 8	à	190,00 F	

ANCIENS NUMEROS

REVUES ☐ 4 ☐ 7 ☐ 8	à	35,00 F	
REVUES ☐ 9 ☐ 10 ☐ 11 ☐ 12 ☐ 13 ☐ 14 ☐ 15 ☐ 16	à	40,00 F	
DISQUETTES ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐ 11 ☐ 12 ☐ 13 ☐ 14 ☐ 15 ☐ 16	à	55,00 F	

ABONNEMENTS

POUR 6 NUMEROS à partir du n° .....

ABONNEMENT SANS DISQUETTES	à	200,00 F	
ABONNEMENT AVEC DISQUETTES (Apple II + , //e, //c)	à	480,00 F	

TOTAL TTC .....

Supplément expédition par avion à l'étranger .....

MONTANT DU REGLEMENT .....

Ces tarifs comprennent l'envoi postal en France métropolitaine. CEE et Suisse
Supplément avion hors CEE : 10 F par numéro et/ou disquette

Envoyez ce bon et votre règlement à : ***(Abonnés : n'oubliez pas de joindre l'étiquette-adresse).***

Editions MEV - 64-70, rue des Chantiers - 78000 VERSAILLES

Nom : .....

Adresse : .....

SPÉCIAL PROGRAMMES MICRO-INFORMATIQUE

ISSN 0183-570 X

ALICE
AMSTRAD CPC 464
APPLE
ATARI
BBC
COMMODORE
DAI
DRAGON 32
ÉLECTRON
EXL 100
HECTOR
LASER
LYNX
MO 5/TO 7
MSX
ORIC
SHARP
TANDY
TI 99/4A
YENO SC 3000
ZX 81 ET SPECTRUM

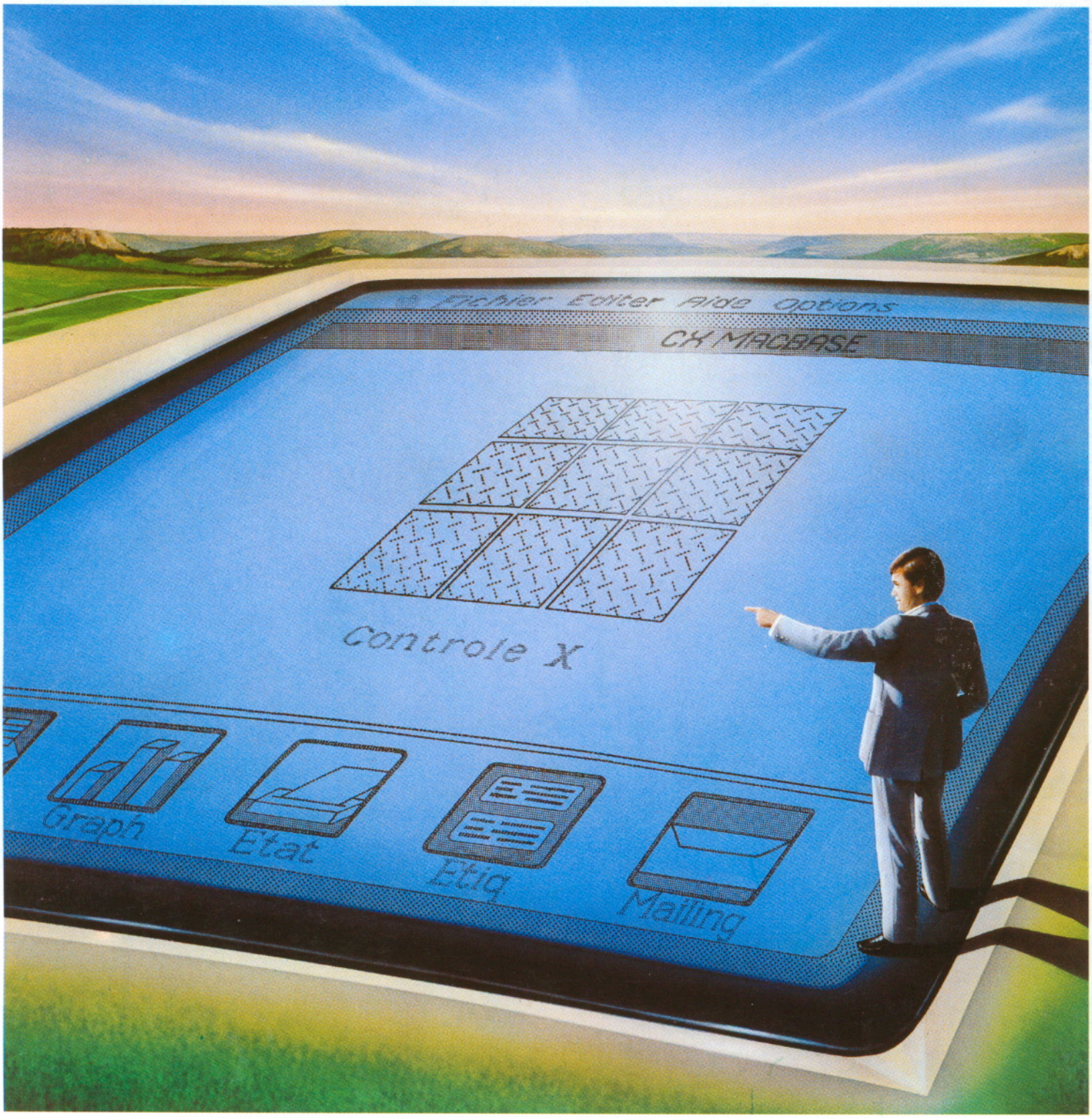
EN VENTE DANS TOUS LES KIOSQUES

NUMÉRO SPÉCIAL 65 HORS SÉRIE DE L'ORDINATEUR INDIVIDUEL

40 PROGRAMMES BASIC INÉDITS

ADAPTABLES SUR TOUS ORDINATEURS. JEUX, UTILITAIRES, ENSEIGNEMENT, **PLUS LE DICTIONNAIRE DES BASIC.**

M 2946 – 65 HS – 30 F **PRIX : 30 FF** – BELGIQUE : 231 FB – CANADA : 3.95 $C – SUISSE : 9,50 FS



CX MacBase

PARTENAIRE D'UN NOUVEAU MONDE

Macintosh

– "Où suis-je ?" demanda Macintosh.
– "Dans un nouveau monde" répondit CX MacBase, "Nos esprits viennent de fusionner, rien ne sera plus comme avant".

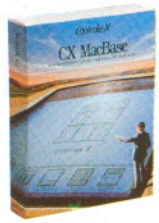
Aujourd'hui, la réalité dépasse la fiction. Cela ne pouvait arriver que sur Macintosh avec CX MacBase, le logiciel qui stimule, accélère et prolonge la pensée.

Avec CX MacBase vous saurez tout de suite fichier, texter, tabler, graphiquer, mixer à volonté. Vous avez champ libre : tout apparaît sur l'écran d'un seul coup de souris "magique".

Quel que soit votre domaine d'activité, sortez du rang ! Vous n'êtes plus fait pour marcher au pas, en ligne et en colonne forcées. Voyagez librement, au gré de votre pensée, de fichier en fichier, de texte en calcul, de tableau en graphe. Vous étonnerez votre entourage par des lettres très personnalisées, des rapports enrichis de tableaux, d'images ou de graphiques en trois dimensions...

CX MacBase, classe, calcule, gère, range, imprime, "coupe et colle" un nouveau monde.

CX MacBase, c'est presque de la sorcellerie ! Pourtant ce n'est pas sorcier !...



Controle X

Les logiciels à tout imaginer

Tour Maine-Montparnasse. 75755 Paris Cedex 15. Téléphone : (1) 538.98.87

Apple Computer Inc est le licencié de la Marque Macintosh. CX MacBase est une marque déposée de Controle X.