PREMIERE PARTIE

UN APERCU DE BIG MAC

A. UTILITES DU LANGAGE D'ASSEMBLAGE

Certains peuvent se demander quelle est l'utilité d'un langage d'assemblage ou pourquoi utiliser un langage d'assemblage. Le BASIC me convient parfaitement. Etant donné que nous n'avons pas suffisamment de place ici pour épuiser le sujet, nous allons essayer de répondre brièvement à ces questions.

Les langages pour ordinateurs sont souvent qualifiés de "haut niveau" ou "bas niveau". Les BASIC, FORTRAN, PASCAL, ... sont des langages de haut niveau. Un langage de haut niveau est un langage qui utilise habituellement des mots anglais comme commandes et passent par une série d'interprétations ou de compilations avant d'être placés en mémoire. Le temps que prend ce processus est la raison pour laquelle les programmes en langages de haut niveau tournent beaucoup mois vite qu'un programme équivalent en langage d'assemblage. En plus, il consomme normalement une plus grande quantité de mémoire.

En pratique, votre ordinateur ne comprend que deux choses : ouvert ou fermé. Toutes les opérations sont manipulées comme des additions ou des soustractions mais à très grande vitesse. Les seuls nombres que le système comprend sont des nombres de base 2 (système binaire) où 1 est représenté par 00000001 et 2 est représenté par 00000010.

Le microprocesseur 6502 a des registres de 8 bits et un registre de 16 bits dans l'unité arithmétique et logique. Toutes les données sont finalement manipulées dans ces registres. Même ces codes de très bas niveau requièrent un programme pour fonctionner correctement. Ce programme est logé dans le microprocesseur lui-même. Le programme du microprocesseur fonctionne en trois cycles : il va chercher une instruction en mémoire RAM, la décode et l'exécute.

Ces instructions existent en RAM sur des groupes de un, deux ou trois "bytes", un "byte" contenant 8 bits.

L'utilisateur peut entrer ses données sous la forme d'un mnémonique de trois caractères, une sorte de code dont les caractères forment une association avec opération du microprocesseur, par exemple LDA équivaut à charger l'accumulateur (LoaD Accumulator). L'APPLE II standard possède un mini assembleur qui permet la programmation simple en langage d'assemblage. Mais même cela n'est pas suffisant pour créer un programme à la fois long et compréhensible. En plus des trois caractères du mnémonique, un langage d'assemblage puissant permet au programmeur d'utiliser des "étiquettes" qui représentent au moment de leur entrée une case mémoire encore indéfinie où un segment particulier du programme devra etre logé. En plus, un langage d'assemblage puissant a des possibilités de numéroter des lignes, semblables à celles du BASIC qui permettront au programmeur d'insérer des lignes dans le programme et d'effectuer d'autres opérations d'édition. Et c'est cela que BIG MAC peut faire.

Finalement, un langage de haut niveau comme le BASIC a aussi un programme d'assemblage qui prend une commande comme par exemple un "PRINT" la réduit en un simple byte hexadécimal avant de le ranger en mémoire.

B. CARACTERISTIQUES

BIG MAC est un "TED-BASED" éditeur assembleur, ce qui signifie que, malgré qu'il soit original, il adhère à presque toutes les conventions du précurseur TED II+ au niveau des commandes, mnémoniques, ...

Le TED ASM original a été écrit par Randy Wiggington et Garry Shannon. Il a été largement distribué "sous la table" par des groupes d'utilisateurs et des particuliers sous un grand nombre de noms et de variétés de versions. Apparemment, chacun y a ajouté ses petites modifications. BIG MAC ne fait pas exception. Il représente un grand pas en avant du fait de ses MACRO-INSTRUCTIONS et apparait sur le marché comme un des éditeurs-assembleurs les plus sophistiqués pour l'APPLE II, gardant les commandes faciles à employer de TED de telle façon qu'il soit abordable pour un débutant en langage d'assemblage.

Les changements que BIG MAC contient en plus des MACROS, sont l'emploi des opérateurs logiques AND, OR, EOR, et l'opération mathématique de division, la possibilité de lister avec ou sans les numéros de lignes et une édition sensiblement plus rapide. De plus, le module édition inclut maintenant beaucoup de commandes supplémentaires pour faciliter l'édition et le programme annexe READER permet à n'importe quel fichier texte APPLE d'être lu dans le buffer de l'éditeur, autorisant de ce fait de se servir de fichiers source d'autres assembleurs comme le "DOS tool kit".

BIG MAC est un éditeur-assembleur entièrement compatible avec TED II+ sauf pour quelques mnémoniques rarement utilisés. Il peut faire des macros instructions, de l'assemblage conditionnel et d'autres commandes qui ne se trouvent pas dans TED II+. Le nom (mais pas le code) de certaines commandes a été emprunté à l'assembleur APPLE et à d'autres assembleurs pour augmenter la compatibilité. Cet assembleur est basé à l'origine sur le TED II+ mais a été pratiquement totalement réécrit. Les différences les plus évidentes qu'un utilisateur de TED II+ remarquera, sont le formattage de texte en mode édition, la rapidité de l'éditeur (pratiquement 20 fois supérieure) et l'absence de "bugs" numérotés.

BIG MAC est un assembleur corésident, ce qui signifie qu'il doit contenir le code source en mémoire pendant l'assemblage. Cela présente plusieurs avantages sur des assembleurs se trouvant sur disquette et beaucoup de gens, dont l'auteur, préfèrent de loin ce système. Malgré tout, cela présente le désavantage de ne pouvoir assembler en une fois un fichier source d'une taille maximum malgré tout assez importante. Tous les efforts ont été faits pour optimiser la place mémoire disponible pour BIG MAC et il est conseillé d'en faire de même en n'encombrant pas la mémoire avec des utilitaires dans le genre de "Program Line Editor" quand on utilise BIG MAC. Ces utilitaires sont très bien à leur place mais marchent sur les plates-bandes d'un bon assembleur ou éditeur.

BIG MAC nécessite un système 48K et travaille aussi bien en DOS 3.2 qu'en DOS 3.3

DEUXIEME PARTIE

BIG MAC

A. MODE EXEC

C:CATALOG

Après vous avoir montré le catalogue, cette commande accepte n'importe quelle commande concernant la disquette utilisant la syntaxe du DOS standard. Contrairement aux commandes APPEND, LOAD et SAVE, vous devez taper le suffixe .S pour un fichier source. Cette facilité est accordée pour "locker" ou "unlocker" les fichiers source. Ne l'utilisez pas pour charger ou pour sauver des fichiers. Si vous ne voulez pas faire une commande disquette, tapez simplement "RETURN". Pour annuler une commande partiellement tapée, utilisez CTRL-X ou assurez-vous qu'elle n'a pas la bonne syntaxe (tapez par exemple quelques virgules) ou ramenez le curseur à sa position de départ.

L:LOAD

Cette commande est utilisée pour charger un fichier source de la disquette. Le nom du fichier vous sera demandé. Vous ne devez pas taper le suffixe .S car BIG MAC le fait automatiquement. Si vous avez tapé L par erreur, tapez simplement "RETURN" deux fois et la commande sera annulée sans affecter les fichiers se trouvant en mémoire. Après un LOAD (ou APPEND), vous vous trouvez placés automatiquement dans le mode éditeur exactement comme si vous aviez tapé un E. Le fichier source sera automatiquement placé à l'adresse correcte. Les commandes LOAD et SAVE affichent le dernier nom utilisé du fichier, suivi d'un "?" clignotant. Si vous pressez la touche "Y", le nom de fichier courant sera utilisé pour la commande. Si vous pressez une autre touche (par exemple "RETURN"), le curseur se positionnera sur le premier caractère du nom du fichier et vous pourrez taper le nom désiré. "RETURN" seul à ce moment annulle la commande.

S:SAVE

Utilisez cette commande pour sauver un fichier source sur la disquette. Comme dans la commande LOAD, ne tapez pas le suffixe .S; vous pouvez annuler la commande par un "RETURN" seul. Notez que l'adresse et la longueur du fichier source sont affichés dans le menu. Il sont là uniquement pour information et vous ne devez pas les utiliser pour sauver votre fichier; l'assembleur s'en souvient mieux que vous et envoie ces informations au DOS automatiquement. Comme dans la commande LOAD, le nom du fichier sera affiché et vous pouvez taper "Y" pour sauver le fichier sous le même nom ou n'importe quelle autre touche pour modifier ce nom.

A:APPEND

Cette commande charge un programme source specifié et le place à la fin du fichier courant dans la mémoire. Elle travaille de la même façon que la commande LOAD et n'affecte pas le nom de fichier par défaut. Elle ne sauve pas le fichier ajouté mais vous pouvez le faire si vous le désirez.

D:DRIVE

Quand vous tapez D, le drive utilisé pour sauver et charger change de 1 en 2 ou de 2 en 1. Le drive courant est spécifié dans le menu. Quand BIG MAC est lancé, le drive courant sera celui utilisé pour le BRUN BIG MAC. Il n'y a pas de commande pour spécifier le numéro de slot mais cela peut être fait en tapant C (pour CATALOG) et en faisant exécuter la commande CATALOG,Sn où n représente le numéro du slot.

E:EDITOR

Cette commande vous place dans l'éditeur-assembleur. Il place automatiquement les tabulations par défaut pour l'éditeur ainsi que celles pour les fichiers source.

Z:ZERO TABS

Cette commande est la même que E mais met toutes les tabulations à zéro. Elle est appropriée aux fichiers texte qui ne sont pas des fichiers source. C'est juste une facilité et la même chose peut être accomplie en faisant TABS "RETURN" dans le mode éditeur.

O:SAVE OBJECT CODE

Vous n'êtes autorisés à employer cette commande qu'après un assemblage réussi d'un fichier source. Dans ce cas, vous verrez l'adresse et la longueur du code objet dans le menu. Comme pour l'adresse du fichier source, elle est donnée à titre d'information. Notez que l'adresse du code objet affichée est celle spécifiée par la commande ORG dans le programme (ou à défaut $8000) et non la localisation courante actuelle du code objet assemblé (qui est $8000 ou une autre adresse spécifiée par OBJ dans le programme). Quand vous utilisez cette commande, un nom vous est demandé pour le code objet. Contrairement au programme source, aucun suffixe n'est ajouté à ce nom. De ce fait vous pouvez utiliser sans danger le même nom que pour le programme source (sans .S évidemment). Quand le code objet est sauvé sur le disque, son adresse sera correcte, c'est-à-dire celle spécifiée dans le menu. Quand vous chargerez ce programme, il se placera à cette adresse qui peut être n'importe quelle adresse qui a été spécifiée par ORG. Donc, il n'y a généralement pas besoin d'utiliser un OBJ dans le programme source tant que le code objet n'est pas trop long pour tenir dans l'espace disponible en $8000 et au-dessus.

Q:QUIT

Cette commande existe en BASIC. Elle met également une adresse de branchement de telle façon que l'on puisse revenir à BIG MAC en tapant CALL 6. Ce retour constitue un "départ à chaud", c'est-à-dire qu'il ne détruit pas le fichier source qui se trouve en mémoire. Cette commande peut être utilisée pour commander les unités de disquettes si cela paraît plus confortable que la commande donnée par C.

B. L'EDITEUR

1. MODE DE COMMANDE

Il existe trois modes de base dans l'éditeur : le mode commande, le mode insertion ou ajout et le mode édition. Le mode principal est le mode commande qui se signale par ":".

Pour beaucoup de commandes dans ce mode, seule la première lettre de la commande est nécessaire, le reste étant optionnel. Nous montrons la commande nécessaire en majuscules et les commandes optionnelles en minuscules. Dans certaines commandes, vous devez spécifier un numéro de ligne, un champ ou une liste de champs. Un numéro de ligne est juste un nombre. Un champ est une paire de numéros de lignes séparés par une virgule. Une liste de champs est un ensemble de champs séparés par des slashes (/).

Plusieurs commandes admettent la spécification d'une chaîne de caractères. La chaîne de caractères doit être délimitée par un caractère non numérique autre que le slash (/). On délimite généralement à l'aide de simples ou doubles guillemets (" ').

Dans l'éditeur, les numéros de ligne sont fournis automatiquement. Vous ne devez jamais les taper quand vous entrez du texte mais seulement quand vous êtes en mode commande. Si un numéro de ligne dans un champ est supérieur au numéro de la dernière ligne, il est automatiquement ajusté au dernier numéro de ligne.

. Les commandes sont:

HImem (nombre décimal entre 9472=$2500 et 39584=$9AA0)

Cette commande est rarement nécessaire. Elle place la limite supérieure du fichier source et de la table des symboles générée par l'assembleur. Son but principal est de protéger la place mémoire du fichier objet d'un usage par la table des symboles au cours de l'assemblage. (En fait c'est la table des symboles qui serait détruite par le fichier objet). Le HIMEM par défaut est $8000 et donc n'a pas besoin d'être spécifié sauf si vous utilisez une adresse objet autre que la valeur par défaut. Vous ne pouvez pas spécifier un HIMEM en-dessous de $2500 (à l'intérieur de BIG MAC) ou au-dessus de $9AA0.

NEW

Efface le fichier source présent, remet HIMEM à $8000 et démarre à neuf.

PR£(07)

Mêmes fonctions qu'en BASIC. Utilisés principalement pour envoyer un listing d'édition ou d'assemblage vers une imprimante.

USER

Cette fonction fait un JSR $3F5

TABS nombre,nombre,...
TABS nombre,nombre,...,"caractère de tabulation"

Cette fonction met des tabulations dans l'éditeur et n'a aucun effet sur le listing d'assemblage. Le nombre maximum de tabulations est fixé à 9 mais la valeur ne peut excéder 39. Le caractère de tabulation par défaut est d'un espace mais n'importe quel caractère peut être spécifié. L'assembleur ne reconnait que l'espace comme seul caractère valide de séparation des étiquettes, des mnémoniques et des opérandes. Si vous ne spécifiez les caractères de tabulation, les derniers employés seront utilisés. Un TABS "RETURN", met toutes les tabulations à zéro.

LENgth

Cette commande donne la longueur en bytes du fichier source et le nombre de bytes restants avant HIMEM (habituellement $8000).

Where (numéro de ligne)

Cette commande affiche en hexadécimal la localisation en mémoire du départ d'une ligne spécifiée. "Where 0" donnera la localisation de la fin de la source.

MONitor

Cette commande sort du moniteur. Elle met les adresses de retour à 0,6 et au vecteur de CTRLY ($3F8). Donc, on peut retourner au système en faisant 6G, 0G ou CTRLY. Cela rétablit les pointeurs de la page zéro qui ont été sauvés dans une zone protégée de BIG MAC luimême. Donc, CTRL-Y vous donne une entrée correcte même si vous avez altéré les pointeurs de la page zéro pendant que vous êtes dans le moniteur. DOS n'est pas connecté quand on utilise cette entrée dans le moniteur. Cette facilité est intéressante pour les programmeurs expérimentés d'APPLE et n'est pas recommandée aux débutants. Si vous retournez accidentellement au BASIC avec un CTRL-C, ne paniquez pas, BIG MAC pardonne beaucoup. Pressez simplement "RESET". Avec l'ancien moniteur ROM, vous pouvez taper 6G pour retourner au BIG MAC. Avec la ROM Autostart, "RESET" reconnecte le DOS et vous pouvez taper INT ou FP suivant le BASIC dans lequel vous vous trouvez et frapper ensuite CALL 6 pour retourner au BIG MAC.

TRuncON

Cette commande place un flag qui, durant un LIST ou un PRINT, termine l'impression d'une ligne quand elle trouve un espace suivi de deux points. Cela facilite la lecture d'un fichier source sur un écran de 40 colonnes.

TRuncOFF

Cette commande retourne à la condition par défaut du flag de tronquation. (Cela arrive aussi automatiquement en entrant dans l'éditeur à partir du mode EXEC ou de l'assembleur).

Quit

Cette commande permet de sortir dans le mode EXEC.

ASM

Cette commande passe le contrôle à l'assembleur. Celui-ci va essayer d'assembler le fichier source. D'abord, il vous demande si vous voulez mettre à jour la source (update the source Y/N). Cela sert à vous rappeler la date ou le numéro d'identification dans votre fichier source. Si vous répondez "N", alors l'assemblage commence. Si vous répondez "Y", alors la première ligne du fichier source sera affichée avec un "/" et vous serez placés en mode éditeur. Quand vous avez terminé d'écrire dans cette ligne, pressez "RETURN" et l'assemblage commmencera. Si vous utilisez la commande d'édition CTRL-C pour supprimer une commande, vous retournerez en mode commande éditeur et toutes les commandes d'entrées-sorties que vous avez établi par PR£... seront déconnectées. Cela arrivera également s'il n'y a pas de ligne avec un "/".

Notez qu'en établissant une ligne commentaire avec un ";*/" en début, vous avez une méthode presqu'automatique de garder trace de multiples versions d'un programme.

Delete (numéro de ligne)
Delete (champ)
Delete (liste de champs)

Cette commande efface les lignes spécifiées. Comme, à la différence du BASIC, les numéros de lignes sont fictifs, ils changent avec une insertion ou un effacement de ligne. Donc, vous devez spécifier le champ supérieur d'abord!

Replace (numéro de ligne)
Replace (champ)

Cette commande efface la ou les lignes spécifiées et vous place en mode insertion à cet endroit.

List
List (numéro de ligne)
List (champ)
List (liste de champs)

Cette commande liste le fichier source avec les numéros de lignes ajoutés. Les caractères de contrôle dans le fichier source sont indiqués en inverse sauf si le listing est envoyé sur une imprimante ou un autre périphérique non-standard. Le listing peut être interrompu par un CTRL-C ou en pressant la touche "/". Vous pouvez arrêter le listing en pressant la barre d'espacement et avancer ligne par ligne en pressant à nouveau la barre d'espacement. N'importe quelle autre touche relance le listing.

Print
Print (numéro de ligne)
Print (champ)
Print (liste de champs)

Cette commande est identique à LIST excepté les numéros de lignes qui n'apparaissent pas.

/
/.(numéro de ligne)

Cette commande continue de lister à partir du dernier numéro de ligne listé ou, quand un numéro de ligne est spécifié, à partir de cette ligne. Ce listing continue jusque la fin du fichier ou jusqu'à ce qu'il soit stoppé comme dans la commande LIST.

Find (d-chaîne de caractères)
Find (champ) (d-chaîne de caractères)
Find (liste de champs) (d-chaîne de caractères)

Cette commande liste les lignes contenant la chaîne de caractères spécifiée. Elle peut être interrompue par CTRL-C ou "/". Comme la touche CTRL-L fonctionne en mode commande, vous pouvez l'utiliser pour trouver ou changer des chaînes de caractères avec des minuscules.

Change (d-chaîne de caractères:d-chaîne de caractères)
Change (champ)(d-chaîne de caractères:d-chaîne de caractères)
Change (liste de champs)(d-chaîne de caractères:d-chaîne de caractères)

Cette commande permet de changer la première chaîne de caractères spécifiée et de la remplacer par la seconde. Les chaînes de caractères doivent avoir les mêmes délimiteurs. Par exemple, pour changer "ordinateur" en "ordinateurs" dans le champ 10,100, vous devez taper C10,100"ordinateur" "ordinateurs". Si aucun champ n'est précisé, la modification s'effectue dans tout le fichier source. Avant que l'opération de modification ne s'effectue, il vous est demandé si vous voulez changer tout (ALL) ou certains (SOME). Si vous choisissez certains en tapant "S", l'éditeur s'arrête chaque fois qu'il rencontre la première chaîne caractères et affiche la ligne comme elle apparaitrait avec le changement. Si vous pressez "ESCAPE", ou n'importe quel caractère de contrôle, le changement affiché ne s'effectuera pas. N'importe quelle autre touche, comme par exemple la barre d'espacement, fera en sorte d'accepter la modification. CTRL-C ou "/" annule le processus Change.

COPY (champ) TO (numéro de ligne)
COPY (numéro de ligne) TO (numéro de ligne)

Cette commande copie les champs juste au-dessus du numéro de ligne spécifié. Elle n'efface rien du tout.

MOVE (champ) TO (numéro de ligne)
MOVE (numéro de ligne) TO (numéro de ligne)

Cette commande est identique à COPY mais, après avoir copié, les champs originaux sont automatiquement effacés. Vous terminez toujours avec les mêmes lignes qu'avant, mais dans un ordre différent.

Edit
Edit (numéro de ligne)
Edit (champ)
Edit (liste de champs)
Edit (d-chaîne de caractères)
Edit (numéro de ligne)(d-chaîne de caractères)
Edit (champ)(d-chaîne de caractères)
Edit (liste de champs)(d-chaîne de caractères)

Cette commande affiche le champ,... ligne par ligne pour être édité et vous place dans le mode éditeur. Si un d-chaîne de caractères est trouvé, alors seules les lignes contenant cette chaîne sont présentées.

Pour les commandes contenant un d-chaîne de caractères, le caractère "^" permet de remplacer n'importe quel caractère. Donc, "Jon^s" trouvera aussi bien "Jones" que "Jonas".

2. LE MODE AJOUT/INSERTION

Add

Cette commande vous place dans le mode "Ajout". Cela ressemble fort à la commande BASIC "AUTO" qui effectue une numérotation automatique des lignes. Vous pouvez entrer des minuscules (utiles pour les commentaires si vous disposez d'un clavier minuscules) en tapant CTRL-L. Cela agit comme un verrouilleur de touches, donc un autre CTRL-L vous ramène en majuscules. (Noter que les fonctions de CTRL-L sont différentes dans le mode éditeur). Pour sortir du mode ajout, tapez simplement "RETURN" comme premier caractère d'une ligne. Vous pouvez entrer un ligne vide en tapant un blanc et puis "RETURN" (Cela n'entrera pas un espace dans le texte, cela évite simplement de sortir de ce mode). Vous pouvez également sortir de ce mode en faisant CTRL-X, mais cette commande annulle la ligne courante.

Insert (numéro de ligne)

Cette commande vous permet d'entrer un texte juste au-dessus de la ligne dont le numéro est spécifié. A part cela sa fonction est fmême que celle de "Add".

CTRL-L

Cette commande verrouille le clavier en majuscules ou minuscules suivant le cas. Par défaut, le clavier est verrouillé en majuscules.

CTRL-X

Annule la dernière ligne courante entrée et retourne au mode éditeur.

3. MODE EDITEUR

Après avoir tapé E dans l'éditeur, vous êtes placés en mode éditeur. La première ligne du champ que vous avez spécifié est affichée à l'écran avec le curseur sur le premier caractère. La ligne est tabulée comme dans le listing et le curseur sautera aux tabulations quand vous vous déplacerez avec les flèches. Quand vous avez terminé, pressez "RETURN". La ligne sera enregistrée telle qu'elle apparaît à l'écran, peu importe où se trouve le curseur. Les commandes et les fonctions d'édition sont très semblables mais non identiques à celles du "NEIL KONZEN'S PROGRAM LINE EDITOR".

Les commandes d'édition sont :

CONTROL I

Commence l'insertion de caractères. Elle se termine par n'importe quel caractère de contrôle, comme les flèches ou "RETURN".

CONTROL D

Efface le caractère où se trouve le curseur.

CONTROL F

Trouv la localisation suivante du caractère tapé après le CONTROL F.

CONTROL L

Change la case du caractère où se trouve le curseur.

CONTROL O

Même fonction que CONTROL I, sauf qu'elle insère n'importe quel caractère de contrôle (y compris les caractères de commande comme CONTOL O).

CONTROL R

Annule la correction d'une ligne et enregistre la ligne avant la modification.

CONTROL Q

Efface tous les carctères à partir du curseur et termine l'édition de la ligne.

CONTROL C

Permet de sortir du mode éditeur et retoune dans l'éditeur par un démarrage à chaud. La ligne qui était éditée est enregistrée sous sa forme originale.

CONTROL B

Place le curseur en début de ligne.

CONTROL N

Place le curseur un espace après la fin de la ligne.

"RETURN"

Accepte la ligne telle qu'elle apparait à l'écran et va chercher la ligne suivante à éditer ou fait un démarrage à chaud si le champ à éditer est terminé.

L'éditeur replace automatiquement des espaces dans les commentaires et les chaînes ASCII avec des espaces en inverse. Quand on liste, il les reconvertit de telle sorte que vous ne vous en rendez pas compte. Il procède de la sorte pour éviter des tabulations ou des chaînes ASCII inappropriées. Dans le cas des chaînes ASCII, il ne le fait que dans le cas où le délimiteur est constitué de simples ou doubles guillemets. Vous pouvez de toute façon faire la même chose en éditant la ligne, en rempaçant le premier délimiteur par des guillemets, en pressant "RETURN" et en éditant à nouveau la ligne et en changeant le deuxième délimiteur pour celui désiré.

C. L'ASSEMBLEUR

Cette documentation ne vise pas à vous apprendre l'assembleur. Elle explique simplement la syntaxe que vous utiliserez dans vos fichiers source et vous renseigne sur les commandes qui sont autorisées dans cet assembleur.

1. FORMAT DES NOMBRES

Cet assembleur accepte les données numériques en mode décimal, hexadécimal et binaire. Les nombres hexadécimaux doivent être précédés par un "$" et les nombres binaires par un "%". Donc les trois instructions suivantes sont équivalentes.

```
LDA £100     LDA £$64     LDA £$1100100     LDA£$01100100
```

Il est à noter que les zéros sont ignorés. Le signe £ indique qu'il s'agit de nombres et l'effet de cette instruction est de charger le nombre décimal 100 dans l'accumulateur.

Un nombre qui n'est pas précédé du signe £ est considéré comme étant une adresse. Donc,

```
LDA 1000     LDA $3E8     LDA %1111101000
```

sont trois moyens pour charger l'accumulateur avec le byte se trouvant dans la mémoire d'adresse $3E8.

Utilisez un format pour les nombres pour clarifier le programme. Par exemple la base de données

```
DA  $1
DA  $A
DA  $64
DA  $3E8
DA  $2710
```

est moins mystérieuse que

```
DA  1
DA  10
DA  100
DA  1000
DA  10000
```

2. FORMAT DU CODE SOURCE

Voici une ligne type du code source :

ETIQUETTE MNEMONIQUE OPERANDE ;COMMENTAIRE

Une ligne contenant uniquement un commentaire doit commencer par une "*". L'assembleur accepte une ligne vide dans le code source et la traite comme l'instruction SKP 1 (voir la section des pseudo mnémoniques).

Le nombre d'espaces séparant les champs n'est pas important sauf pour le listing d'édition qui n'accepte qu'un espace.

La taille maximale d'une étiquette est de 13 caractères mais s'il y a plus de plus de 8 caractères pour une étiquette, les listings d'assemblage seront en désordre. Une étiquette doit commencer par un caractère d'au moins une colonne de large, dans le code ASCII et ne peut contenir aucun caractère plus petit que zéro dans le code ASCII.

L'assembleur n'examine que les 3 premiers caractères du mnémonique (avec comme seule exception le mnémonique supplémentaire POPD), donc vous pouvez, par exemple utiliser "PAGE" au lieu de "PAG" à la condition que la quatrième lettre ne soit pas un "D". L'assembleur listera en tronquant les mnémoniques à 7 lettres et n'imprimera pas bien un mnémonique de plus de 4 lettres sauf s'il n'y a pas d'opérande.

La longueur maximale de la combinaison opérande + commentaire est de 64 caractères. L'assembleur vous indiquera une erreur si vous en utilisez plus. Une ligne de commentaires est également limitée à 64 caractères.

3. EXPRESSIONS

Pour clarifier la syntaxe acceptée et/ou requise par l'assembleur, nous devons définir ce que l'on entend par "expression". Les expressions sont construites à partir "d'expressions primitives" en utilisant des opérations arithmétiques et logiques. Les expressions primitives sont :

1. Une étiquette
2. Un nombre décimal
3. Un nombre hexadécimal (précédé de "$")
4. Un nombre binaire (précédé de "%")
5. N'importe quel caractère ASCII précédé ou entouré de guillemets
6. Le caractère * (mis à la place de l'adresse actuelle)

Tous les formats de nombres acceptent des données de 16 bits et les zéros devant ne sont jamais nécessaires. Dans le cinquième cas des exemples ci-dessous, la valeur de l'expression est juste la valeur ASCII du caractère. Le bit supérieur sera à 1 si un " est utilisé et à 0 si un ' est utilisé.

L'assembleur accepte les 4 opérations arithmétiques : +, -, / et *. Il accepte également les 3 opérations logiques :

! = OU exclusif . = OU & = ET

Voici quelques exemples d'expressions correctes :

```
ETIQUETTE1-ETIQUETTE2
2*ETIQUETTE+$231
1234+%10111
"K"-ETIQUETTE
"0"!ETIQUETTE
ETIQUETTE&$7F
*-2
ETIQUETTE.%10000000
```

Les parenthèses ont une autre signification et ne sont pas permises dans les expressions. Toutes les opérations arithmétiques et logiques sont effectuées de gauche à droite .(Donc, 2+3*5 sera assemblé en 25 et non en 17).

4. DONNEES IMMEDIATES.

Pour les mnémoniques du genre LDA, CMP, ..., qui accepte des données immédiates (c'est-à-dire des nombres et non des adresses), le mode immédiat est signalé par £. Par exemple, LDX £3.

En outre :

£<expression	donne le byte de poids faible de l'expression
£>expression	donne le byte de poids fort de l'expression
£expression	donne également le byte de poids faible (le 6502 n'accepte pas des données sur 2 bytes)
£/expression	est une syntaxe optionnelle pour avoir le byte de poids fort de l'expression

La capacité de l'assembleur d'évaluer des expressions comme :

LAB1-LAB2-1

est très utile pour les types de codes suivants :

```
COMPARE   LDX  £TROUVE-DATA-1
BOUCLE    CMP  DATA.X
          BEQ  TROUVE
          DEX
          BPL  BOUCLE
          JMP  REJECT    ;pas trouvé
DATA      HEX  E3BC3498
TROUVE    RTS
```

Avec ce type de code, si vous effacez certaines des "DATA", alors le X.INDEX approprié pour la boucle de comparaison est automatiquement ajusté.

5. MODES D'ADRESSAGE

a. MNEMONIQUES DU 6502

L'assembleur accepte évidemment tous les codes du 6502 avec les mnémoniques standard. Il accepte aussi BLT = branchement si plus petit (Branch if Less Then) comme équivalent à BCC, et BGE = branchement si plus grand ou égal (Branch if Greater or Equal) comme équivalent à BCS.

Il existe 12 modes d'adressage pour le 6502. La syntaxe appropriée pour ces modes est :

MODE D'ADRESSAGE	SYNTAXE	EXEMPLES
Implicite	MNEMONIQUE	CLC
Accumulateur	MNEMONIQUE	ROR
Immédiat (donnée)	MNEMONIQUE£expr	ADC£$F8
		CMP£"M"
		LDX£>ETIQUETTE1-ETIQUETTE2-1
Page zéro (adresse)	MNEMONIQUE expr	ROL 6
indexé X	MNEMONIQUE expr,X	LDA SE0,X
indexé Y	MNEMONIQUE expr,Y	STX LAB,Y
Absolu (adresse)	MNEMONIQUE expr	BIT $300
indexé X	MNEMONIQUE expr,X	STA $4000,x
indexé Y	MNEMONIQUE expr,Y	SBC ETIQUETTE-1,Y
Indirect	JMP (expr)	JMP ($3F2)
Préindexé X	MNEMONIQUE(expr,X)	LDA (6,X)
Postindexé Y	MNEMONIQUE(expr,Y)	STA ($FE),Y

Notez qu'il n'y a pas de différence de syntaxe pour la page zéro et les modes d'adressage absolu. L'assembleur utilise automatiquement le mode d'adressage en page zéro à bon escient. Dans les modes indirects indexés, seule une expression page zéro est autorisée et l'assembleur fournira un message d'erreur si l'"expr" n'évalue pas une adresse en page zéro.

Notez aussi que le mode accumulateur n'accepte pas une opérande. Certains assembleurs perverses requièrent de mettre un "A" dans le champ opérande de ce mode.

L'assembleur décidera de la légalité du mode d'adressage de n'importe quel mnémonique donné.

b. 16 MNEMONIQUES SUPPLEMENTAIRES

L'assembleur accepte également les 16 mnémoniques supplémentaires en plus des mnémoniques standard. Les 16 registres usuels supplémentaires R0 à R15 n'ont pas besoin d'être mis en "équation" et le R est optionnel. Les utilisateurs de TED II+ seront contents d'apprendre que le mnémonique SET travaille comme il doit le faire, c'est-à-dire avec des nombres et des étiquettes. Pour le mnémonique SET, on peut utiliser soit un espace soit une virgule pour séparer la partie registre et la partie données de l'opérande; par exemple, SET R3,ETIQUETTE est équivalent à SET R3 ETIQUETTE. Il est à noter que le mnémonique NUL est assemblé en un byte (le même que HEX 0D) et non comme un 2 bytes skip comme il serait intepréte par la "ROM SWEET 16". C'est intentionnel et c'est fait pour des raisons internes.

c. PSEUDO MNEMONIQUES

I. directives

EQU expression (EQUals)
expression (syntaxe optionnelle)

Utilisé pour définir la valeur d'une étiquette, habituellement une adresse extérieure ou une constante souvent utilisée à laquelle on désire donner un nom significatif. Il est recommandé que ces instructions soient localisées au début du programme. L'assembleur n'autorise pas une équation à un nombre en page zéro après que l'étiquette mise en équation ait été utilisée car il pourrait en résulter un mauvais code (voyez aussi "variables").

ORG expression (ORiGin)

Etablit l'adresse à laquelle le programme est mis pour tourner. Par défaut, cette valeur est $8000. Habituellement, il n'y aura qu'un seul ORG et il se trouvera au début du programme. A part pour la commande du mode EXEC "object code save", cette commande n'a aucune fonction.

OBJ expression (OBJet)

Etablit l'adresse à laquelle le code objet sera implanté pendant l'assemblage. Par défaut, cette valeur est $8000. Il est rarement nécessaire d'utiliser ce mnémonique et il est fortement conseillé aux programmeurs inexpérimentés de ne pas l'utiliser. Vous devez faire attention de ne pas spécifier une adresse en conflit avec BIG MAC (ce qui inclut toute la mémoire au-dessus de la fin de la table des symboles éventuelle).

END

Ce mnémonique est rarement utilisé. Il indique à l'assembleur d'ignorer le reste du code source. Les étiquettes qui apparaissent aprés un END ne seront pas reconnues.

II. formats

LST ON ou OFF (LiST)

Ce mnémonique est utilisé quand on veut envoyer le listing à l'écran et/ou sur un autre périphérique. Vous pouvez, par exemple, l'utiliser pour envoyer seulement une partie du listing vers l'imprimante. Il peut y avoir autant d'instuctions LST qu'on le désire dans le programme source. Si la position de LST est sur OFF à la fin de l'assemblage, la table des symboles ne sera pas imprimée. L'assembleur, en fait, ne contrôle que le troisième caractère de l'opérande pour voir si c'est un espace ou non. Donc, LST SALUT aura le même effet que LST OFF. La directive LST n'a aucun effet sur la génération actuelle du code objet. Si la condition LST est sur OFF, le code objet sera généré beaucoup plus vite, mais il est recommandé de ne le faire qu'avec des programmes déjà mis au point.

EXP ON ou OFF (EXPand)

EXP ON affichera une MACRO en entier durant l'assemblage. Si la directive est sur OFF, on aura une impression du "PMC pseudo-op" uniquement. Par défaut, la condition est sur ON.

PAU (PAUse)

A la seconde passe, cette directive provoquera une pause dans l'assemblage jusqu'à ce qu'une touche soit pressée. (Cela peut également être réalisé à partir du clavier en tapant la barre d'espacement).

PAG (PAGe)

Cette directive envoie un saut de page ($8C) à l'imprimante. Ca n'a pas d'effet sur le listing écran.

AST expression (ASTerisks)

Cette directive envoie des astérisques au listing en nombre égal à la valeur de l'opérande. Le format du nombre est le format normal, de telle façon que AST 10 envoie (en décimal) 10 astérisques. Le nombre est traité modulo 256 avec 0 correspondant à 256 astérisques. Cela diffère du TED II+ qui reconnait l'opérande comme expression hexadécimale et a donc besoin d'être convertie.

SKP expression (SKiP)

Cette directive envoie un nombre de retours charriot au listing correspondant au nombre de l'opérande. Le format du nombre est le même que dans AST.

III. chaînes de caractères

ASC d-chaîne de caractères (ASCii)

Cette directive place un chaîne de caractères ASCII délimitée dans le code objet. La seule restriction sur le délimiteur est qu'il ne se trouve pas dans la chaîne elle-même. Des délimiteurs différents ont des effets différents. N'importe quel caractère inférieur (dans le code ASCII) au simple guillemet produira une chaîne avec le bit haut à 1, dans les autres cas, il sera à 0. Donc, par exemple, les délimiteurs !"£$%& produiront une chaîne "négative" ASCII et les délimiteurs '()*+/ produiront une chaîne "positive" ASCII. Habituellement, on choisit les délimiteurs simples ou doubles guillemets, mais des autres délimiteurs donnent la possibilité de mettre dans la chaîne des simples ou doubles guillemets.

DCI d-chaîne de caractères (Dextral Character Inverted)

Cette directive est identique à ASC à la différence que la chaîne est placée en mémoire avec le dernier caractère étant de bit supérieur opposé aux autres.

INV d-chaîne de caractères (INVerse)

Cette directive place une chaîne en mémoire dans le format inverse. Le choix de tous les délimiteurs a le même effet.

FLS d-chaîne de caractères (FLaSh)

Cette directive place une chaîne en mémoire dans le format clignotant. Le choix de n'importe quel délimiteur a le même effet.

IV. données et allocation

DA expression (Define Adress)

Cette directive range la valeur de l'opérande (2 bytes), habituellement une adresse, dans le code objet, le byte de poids faible pour commencer. Par exemple, DA $FDF0 générera F0 FD.

DDB expression (DeFine Double Bytes)

Idem à la précédente mais place le byte de poids fort pour commencer.

DFB expression (DeFine Byte)

Cette directive place les bytes spécifiés par l'opérande dans le code objet. Elle accepte plusieurs bytes de données qui doivent être séparés par des virgules et ne pas comprendre d'espace. C'est le format standard des nombres qui est utilisé comme d'habitude. Le symbole "£" est acceptable mais ignoré tout comme"<". Le symbole "'" peut être utilisé pour spécifier le byte de poids fort d'une étiquette; dans le cas contraire, c'est le byte de poids faible qui est chaque fois pris. Le symbole ">" ne doit apparaître que comme premier caractère d'une expression ou immédiatement après "£". Donc, l'instruction DFB >ETIQUETTE1-ETIQUETTE2 produira le byte de poids fort de la valeur de ETIQUETTE1-ETIQUETTE2. Par exemple,

DFB $34,100,ETIQUETTE1-ETIQUETTE2,%1011,>ETIQUETTE1-ETIQUETTE2

est une relation correctement formatée et générera le code objet (hex) 34 64 DE 0B 09 pour ETIQUETTE1 = $81A2 et ETIQUETTE2 = $77C4.

HEX donnée hexadécimale

Cette directive est une alternative à DFB qui permet une insertion facile de données hexadécimales. Contrairement à tous les autres cas, le "$" n'est ni requis ni accepté. L'opérande doit être constituée de nombres hexadécimaux ayant chacun 2 chiffres (par exemple 0F et non F). Ils peuvent être séparés par des virgules ou peuvent être adjacents. Il se produira un message d'erreur si l'opérande contient un nombre impair de chiffres ou se termine par une virgule ou, dans tous les cas, contient plus de 64 caractères.

DS expression

Cette directive ne génère aucun code mais ajuste simplement le pointeur du code objet de telle façon qu'un espace équivalent à la valeur de l'opérande soit réservé.

V. conditionnels

DO expression

Cette directive, avec ELSE et FIN sont des pseudo-opérateurs d'assemblage conditionnel. Si l'opérande évalue zéro, alors l'assembleur arrêtera de générerer le code objet (jusqu'à ce qu'il rencontre un autre conditionnel). Sauf pour les noms de MACRO, il ne reconnaîtra aucune étiquette dans une telle zone de code. Si l'opérande évalue un nombre différent de zéro, alors l'assemblage se poursuivra normalement. C'est très utile pour les macros. C'est également utile pour les sources définies pour générer des codes légèrement différents pour des situations différentes. Par exemple, si vous écrivez un programme pour aller sur un "chip" ROM, alors vous voudrez une version pour la ROM et une autre, avec des petites différences, pour créer une version RAM dans le but de mise au point. De même, dans un programme avec du texte, vous pouvez vouloir une version pour les Apple avec minuscules et une version pour les Apple sans minuscules. En utilisant l'assemblage conditionnel, la modification de tels programmes devient plus facile du fait que vous n'avez pas besoin de faire les modifications dans deux versions séparées du code source. Chaque DO doit être terminé par un FIN et chaque FIN doit être précédé d'un DO. Un ELSE ne peut apparaître que dans une structure DO,FIN. Les structures DO,FIN peuvent avoir 8 lignes maximum d'écart (avec éventuellement des ELSE à l'intérieur). Si une condition DO est à 0, l'assemblage ne se poursuivra que lorsqu'un FIN sera rencontré où si un ELSE se trouve à ce niveau. Les structures DO,FIN sont valides pour mettre des conditionnels dans les MACROS.

ELSE

Cette directive inverse la condition d'assemblage (ON --> OFF ou OFF --> ON).

FIN

Cette directive annule le dernier DO.

VI. macros

MAC (MACro)

Cette directive signale le commencement de la définition d'une MACRO. Elle doit être libellée avec le nom de la MACRO. Le nom que vous utilisez est alors réservé et ne peut être référencé par une autre instruction que PMC. (Donc, quelque chose comme DA NOM ne sera pas accepté si NOM est l'étiquette dans une MAC. De toute façon, la même chose peut être simulée en faisant précéder la MACRO par une ETIQUETTE EQU*, ou une ETIQUETTE DS 0, etc.... Il est rarement nécessaire de faire cela). Voyez la section consacrée aux MACROS pour plus de détails.

EOM (End Of Macro)
<<< (autre syntaxe)

Cette directive signale la fin de la définition d'une MACRO. Elle peut être utilisée pour des branchements à la fin de la MACRO ou une de ses copies.

PMC nom macro (Put MaCro)
>>> nom macro (autre syntaxe)

Cette directive dit à l'assembleur d'assembler une copie de la MACRO spécifiée à l'endroit où elle se trouve. Voyez la section des MACROS pour de plus amples informations.

6. VARIABLES

Les étiquettes commençant par un " " sont considérées comme des variables. Elles ne peuvent être définies que par un EQU et ne peuvent être utilisées pour étiqueter autre chose. Elles peuvent être redéfinies aussi souvent que vous le voulez. Le but désigné de l'emploi de variables sont les MACROS mais elles ne sont pas réservées à cet usage.

Des références à une variable précédant la définition de cette variable sont impossibles (du mois avec des résultats corrects) mais l'assembleur lui assignera une valeur. Donc, une variable doit être définie avant d'être utilisée.

D. MACROS

1. DEFINIR UNE MACRO

La définition d'une macro commence par :

NOM MAC (pas d'opérande)

avec NOM dans le champ "étiquette". Sa définition se termine par le pseudo mnémonique EOM ou <<<. L'étiquette NOM ne peut être référencée par rien d'autre que PCM NOM (ou >>>NOM).

Vous pouvez simplement définir une macro la première fois que vous en avez besoin dans le programme. Toutefois, il est préférable de définir d'abord toutes les macros au début du programme avec la condition d'assemblage sur "OFF" et vous y référer quand vous en avez besoin.

Des macros ne peuvent pas être emboîtées. Les restrictions au niveau de la mémoire sont trop importantes que pour permettre cette possibilité rarement utilisée. Un essai d'emboîter des macros produirait un message d'erreur dans les macros emboîtées.

Une référence à une macro avant qu'elle ne soit définie n'est pas possible. Cela produirait un message d'erreur "NOT MACRO". Donc, une macro doit être définie avant d'être appelée par un PMC.

Les conditionnels DO, ELSE, et FIN peuvent être utilisés dans une macro.

Les étiquettes se trouvant dans une macro sont mises à jour chaque fois qu'un PMC est rencontré. Donc, elles peuvent être uilisées pour faire un branchement à l'intérieur d'une macro aussi bien que pour faire un branchement en arrière.

Les messages d'erreurs générés par des erreurs dans des macros interrompent généralement l'assemblage à cause de leurs effets pernicieux. De tels messages indiqueront généralement le numéro de ligne d'un PMC plutôt que le numéro de la ligne dans la macro où elle a été rencontrée.

Comme il y a de l'espace supplémentaire dans la version BIG MAC carte langage, l'emboîtement de macros est autorisé.

2. VARIABLES SPECIALES

Huit variables appelées 1 à 8 sont prédéfinies et sont toute désignées pour la facilité d'emploi dans les macros. Elles sont utilisées dans une commande PMC comme suit :

```
>>>NOM expr1,expr2,expr3,...
```

qui assignera la valeur de l'expr1 à la variable 1, celle de l'expr2 à la variable 2, etc.... Voici un exemple de cette utilisation :

```
TEMP EQU      $10
     DO       0
ECH  MAC
     LDA       1
     STA       3
     LDA       2
     STA       1
     LDA       3
     STA       2
     <<<
     FIN
     >>>      ECH $6,$7,TEMP
     >>>      ECH $1000,$6,TEMP
```

(Ce segment de programme échange le contenu des mémoires $6 et $7, en utilisant TEMP comme intermédiaire, puis échange le contenu des mémoires $6 et $1000.)

Si certaines des variables spéciales sont utilisées dans la définition d'une macro, alors leur valeur doit être spécifiée dans l'instruction PMC (ou >>>). Dans le listing d'assemblage les variables spéciales seront remplacées par les expressions correspondantes.

L'assembleur accepte certains autres caracatères à la place de l'espace entre le nom de la macro et les expressions dans une instruction PMC. Par exemple, vous pouvez utiliser "/", ",", "-", ou "(". Les virgules sont de toute façon requises et aucun espace supplémentaire n'est autorisé.

3. PROGRAMME SIMPLE

Le programme simple de la page 29 de l'édition anglaise illustre l'usage des macros avec une variable non standard. Il serait de toutes façons plus facile d'utiliser 1 à la place de MSG. (Dans ce cas l'équation de la variable pourrait être éliminée et les valeurs pour 1 devraient être spécifiées dans les lignes >>>.

E. TABLE DES SYMBOLES

La table des symboles est imprimée après l'assemblage sauf si LST OFF a été invoqué. Elle apparaît d'abord dans l'ordre alphabétique puis dans l'ordre numérique. La tables des symboles est signalée comme suit :

MD = Définition de la macro
M = Etiquette définie dans la macro
V = Variable
? = Un symbole qui n'a pas été référencé

Dans le programme ils sont signalés en positionnant les bits 7 à 4 du byte de la longueur des symboles :

? = bit 7 MD = bit 5 M = bit 4

De plus, le bit 6 est positionné pendant l'impression par ordre alphabétique pour signaler les symboles imprimés, et supprimé pendant l'impression par ordre numérique. L'impression des symboles est formattée pour une imprimante 80 colonnes ou une qui envoie un retour chariot après 40 colonnes.

F. INFORMATIONS TECHNIQUES

1. ADRESSES IMPORTANTES

Les adresses et la longueur de BIG MAC sont A$803,L$1CFB. Vous ne devez jamais faire BSAVE BIG MAC avec un fichier source en mémoire; utilisez NEW dans le mode éditeur d'abord. La source est placée en $2500 quand elle est chargée, peu importe son adresse originale.

Les pointeurs importants, comme dans le TED II+, sont :

DEPART DE LA SOURCE	en	$A,$B	(toujours mis en $2500)
HIMEM	en	$C,$D	(par défaut en $8000)
FIN DE LA SOURCE	en	$E,$F	

Quand vous sortez vers le BASIC ou vers le moniteur, ces pointeurs sont sauvés en $809-$80E. Ils sont rendus quand on rentre dans BIG MAC.

Une entrée a chaud dans la mode EXEC se fait en $806 = 2054. C'est l'entrée donnée par un CALL 6, si vous avez utilisé "Q" pour passer du mode EXEC au BASIC. Une entrée via $803 = 2051 effacera le nom du fichier par défaut mais à part cela c'est la même entrée à chaud. Une entrée à chaud directement dans l'éditeur est possible en faisant C186, mais elle n'est pas recommandée car elle risque de perdre certains pointeurs.

L'entrée dans l'éditeur déconnecte le DOS (de ce fait vous pouvez utiliser des commandes comme INIT sans conséquences désastreuses). Rentrer dans la mode EXEC déconnecte les entrées/sorties que vous avez pu établir via la commande PR£ de l'éditeur et reconnecte le DOS. Sortir de l'assembleur déconnecte aussi les extensions d'entrées/sorties.

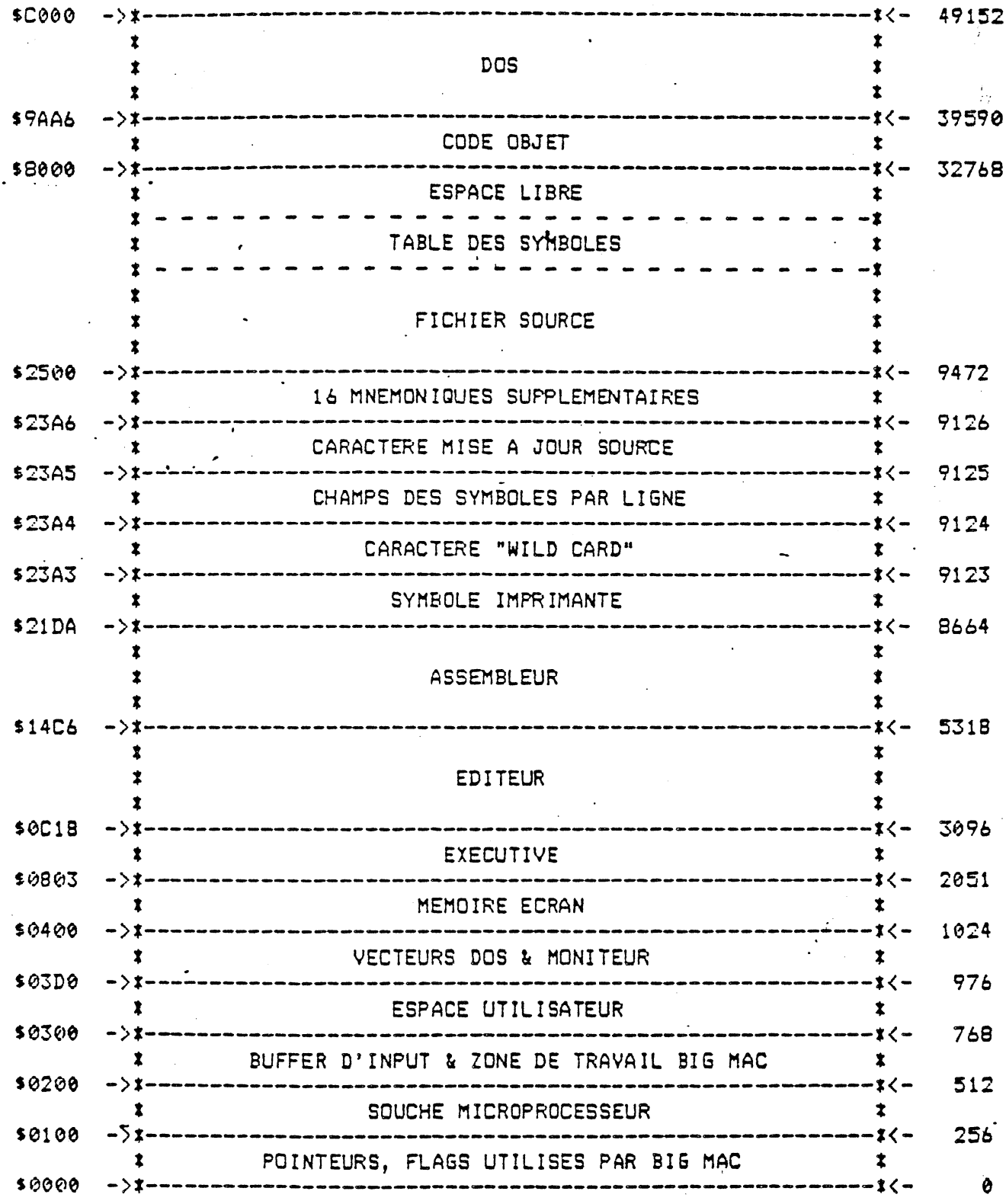
Il y a 3 bytes de données que vous pouvez vouloir changer pour satisfaire vos propres besoins. Pour cette raison, ils ont été placés dans des mémoires accessibles juste en-dessous des 16 mnémoniques supplémentaires comme montré dans le tableau.

$23A5 (16 MNE - 1) contient le caratère "/" ($AF), qui est cheché par l'opération "UPDATE SOURCE ?" du module assembleur.

$23A4 (16 MNE - 2) contient le nombre (couramment 4) de colonnes qui sont imprimées dans la table des symboles avant qu'un retour charriot soit renvoyé. Ceci peut être changé pour accommoder les imprimantes avec des largeurs de lignes plus grandes ou moins grandes. Chaque colonne de symboles represente 20 colonnes d'imprimante.

$23A5 (16 MNE - 3) contient le caractère WILD CARD "^" ($DE) utilisé par l'éditeur dans la commande F(ind) et autres routines de recherche.

2. CARTE DE LA MEMOIRE



3. EN CAS DE CRASH

Parfois, à cause de l'électricité statique ou d'autes causes, BIG MAC ou n'importe quel programme peut crasher. Du fait de cette possibilité, vous devez sauvez régulièrement les programmes que vous écrivez. De toute façon, en cas de crash, vous pouvez relire vos programmes à l'aide de la procédure suivante :

- Sortir de BIG MAC (par RESET si nécessaire).
- Appeler le moniteur par CALL-151.
- Noter les 2 bytes d'adresse $80D et $80E. (Si vous opérez depuis l'Integer, ce qui est généralement préférable,, utilisez alors les bytes d'adresse $E
- et $F à la place).
- Faire BLOAD BIG MAC (PAS BRUN !!).
- Replacez les 2 bytes que vous avez notés en $80D et $80E (pas $E et $F).
- Tapez 303G "RETURN" pour faire un démarrage à chaud de BIG MAC.
- Tapez E pour entrer dans l'éditeur.
- Tapez W0, ce qui imprimera la fin de la source en HEX.
- Si ce nombre paraît raisonnable, lister la source et vérifiez-la.
- Sinon, tapez MON pour aller dans le moniteur. Essayer de trouver la fin de la source en mémoire. Si l'assemblage a commencé, il doit y avoir un $FF à la fin de la source. Mettez $80D et $80E à l'adresse de la fin de la source (le byte de poids faible en $80D). Tapez CONTROL Y pour retourner au BIG MAC. Si tout n'est pas OK à ce stade, vous avez un problème.
- Si le DOS a été perdu, vous avez un gros problème. Dans ce cas, à la place de suivre la marche décrite ci-dessus, notez les bytes d'adresse $80D et $80E (ou $E et $F), rebooter (à partir d'une copie, pas de l'original), faites BRUN BIG MAC, allez dans le moniteur et rétablissez les adresses $80D et $80E, comme ci-dessus et d'après vos données, puis tapez CONTROL Y pour retourner dans le BIG MAC et vérifiez. C'est pour cette raison que MAC est fourni comme une disquette copie 48K.

4. UTILISATION DES FICHIERS SOURCE TED II+ AVEC BIG MAC

Vous pouvez simplement charger les fichiers créés avec TED II+ et, dans la plupart des cas, les assembler avec BIG MAC. Quelques changements mineurs au niveau des mnémoniques peuvent être nécessaires. En particulier, vous devrez généralement retirer les mnémoniques OBJ, étant donné qu'ils sont rarement utilisés dans BIG MAC.

De plus, l'éditeur de BIG MAC possède quelques raffinements qui ne seront pas utilisés avec cette procédure. Le fichier peut être converti dans la forme qui aurait été établie en utilisant BIG MAC en faisant simplement tourner le fichier en mode édition et en faisant "RETURN" à chaque ligne. Pour un fichier important, un programme spécial FIX a été ajouté; celui-ci fait cette opération automatiquement. Charger simplement votre programme; dans le mode EXEC, tapez C pour CATALOG et faites BRUN FIX. Le FIX sera fait presqu'instantanément et retournera automatiquement au BIG MAC.

Parmi d'autres actions, FIX enlève les espaces supplémentaires à la fin des lignes. Il n'enlève pas les espaces à l'intérieur des lignes de telle façon qu'il puisse être utilisé avec des fichiers texte qui ne sont pas des fichiers source. De toutes façons, cela peut être fait en tapant C" " " plusieurs fois depuis l'éditeur, jusqu'à ce qu'il n'y ait plus de changements. Vous devez quand même faire attention avec cette commande si vous ne voulez pas faire de tels changements dans les commentaires et les chaînes ASCII.

G. MESSAGES D'ERREURS

BAD OPCODE (mauvais mnémonique)

Apparaît quand le mnémonique n'est pas valide (peut-être mal orthographié) ou que le mnémonique se trouve dans le champ "ETIQUETTE".

BAD ADRESS MODE (mauvais mode d'adressage)

Le mode d'adressage n'est pas une instruction 6502 valide; par exemple, JSR (LABEL) ou LDX (LABEL),Y.

BAD BRANCH (mauvais branchement)

Un branchement à une adresse qui est hors du champ, c'est-à-dire à plus de 127 bytes.

BAD OPERAND (mauvaise opérande)

Opérande illégale. Cela arrive aussi si vous utilisez un EQU pour mettre en équation une étiquette en page zéro après que l'étiquette ait été utilisée. Cela peut aussi vouloir dire que votre opérande a plus de 64 caractères ou qu'une ligne de commentaires dépasse 64 caractères. Ce type d'erreur arrête l'assemblage.

DUPLICATE SYMBOL (symboles identiques)

A la première passe, l'assembleur trouve 2 étiquettes semblables.

MEMORY FULL (mémoire pleine)

A la première passe, la table des symboles excède HIMEM ($8000 par défaut). L'assemblage est arrêté par ce type d'erreur.

UNKNOW LABEL (étiquette inconnue)

Votre programme se réfère à une étiquette inexistante. Cela arrive également si vous essayez de vous référencer à la définition d'une macro avec autre chose que PMC. Cela peut aussi arriver si l'étiquette référencée est dans une zone avec l'assemblage conditionnel sur OFF.

NOT MACRO (pas une macro)

Référence à une macro avant sa définition ou référence par PMC à une étiquette qui n'est pas une macro.

NESTED MACROS (macros emboîtées)

Définition d'une macro à l'intérieur d'une autre ou PMC dans la définition d'une macro.

BAD LABEL (mauvaise étiquette)

Ce message est provoqué par un EQU ou un MAC sans étiquette, une étiquette trop longue ou contenant un caractère illégal.

H. VERSION CARTE LANGAGE

Si vous avez une version carte langage de BIG MAC, toutes les différences entre cette version et la version standard sont décrites dans le Supplément Carte Langage.

TROISIEME PARTIE

AUTRES PROGRAMMES

A. SOURCEROR

1. INTRODUCTION

SOURCEROR est un désassembleur sophistiqué et facile à utiliser tout désigné pour compléter l'assembleur BIG MAC. Il transforme les programmes binaires en programmes source BIG MAC, en général avec une grande célérité. SOURCEROR désassemble aussi bien en mnémoniques standards qu'en mnémoniques supplémentaires du BIG MAC.

La partie principale de SOURCEROR est appelée SRCRR.OBJ mais il ne peut pas tourner directement car il peut écrire sur les buffers du DOS et planter le système. Pour cette raison, un petit programme appelé SOURCEROR est fourni. Celui-ci tourne dans le buffer d'input et n'interfére donc avec aucun programme en mémoire. Ce petit programme vérifie simplement la taille de la mémoire, se débarasse de tous les programmes comme PLE qui pourraient entrer en conflit avec le programme source principal, installe MAXFILES 1 et lance SRCRR.OBJ (à $8A00.$9AA5). Pour minimiser la possibilité d'accident, SRCRR.OBJ a une location par défaut à $4000 et si vous le lancer par un BRUN, il ne fera rien du tout. Si vous essayer de le lancer par un BRUN à son adresse désignée ($8A00), vous allez au devant de gros ennuis. SOURCEROR exige l'écran standard APPLE et ne fonctionne pas sur un écran 80 colonnes.

2. UTILISATION DE SOURCEROR

1. - Chargez le programme à désassembler. Bien que SOURCEROR puisse traiter un programme à n'importe quel endroit de la mémoire, l'adresse originale du programme est préférable car il n'entrera pas en conflit avec le programme SOURCEROR et avec l'élaboration du code source. En cas de doute, chargez le en $800 ou $803. Des petits programmes en $4000 ou des programmes moyens au-dessus de $6000 ne causeront probablement aucun problème à leur adresse originale.

2. - Faites BRUN SOURCEROR

3. - Vous recevrez un message disant que l'adresse par défaut du fichier source se trouvera en $2500. Cette adresse a été sélectionnée parce que c'est l'adresse utilisée par BIG MAC et parce qu'elle n'entrera pas en conflit avec les adresses de la plupart des programmes binaires que vous voudrez désassembler. Tapez simplement "RETURN" pour accepter cette adresse par défaut ou spécifiez l'adresse (HEX) que vous désirez. Vous pouvez également accéder à une partie "secrète" à ce niveau. Cela s'obtient en tapant CONTROL S après (ou au lieu de) l'adresse de la source. Il vous sera alors demandé de spécifier une adresse non-standard pour l'interpréteur "SWEET 16". Cette possibilité vous est procurée pour faciliter le désassemblage de programmes qui utilisent une version RAM de "SWEET 16".

4. - Ensuite, le programme vous demandera de frapper "RETURN" si le programme à désassembler se trouve à son adresse originale. Si ce n'est pas le cas, vous devez préciser (en hex) l'adresse actuelle du code à désassembler. Il vous sera alors demandé l'adressse originale de ce programme.
Pendant le désassemblage, vous devez utiliser l'adresse originale du programme et non pas l'adresse où le programme réside actuellement. Il vous semblera que vous désassemblez le programme à son adresse originale, mais en fait, SOURCEROR désassemble le programme à son adresse actuelle et transfère les adresses.

5. - Ensuite, vous verrez enfin la page-titre qui contient un éventail des commandes à utiliser dans le désassemblage. Vous pouvez alors commencer le désassemblage ou utiliser une des autres commandes. Votre première commande doit contenir une adresse hexadécimale. Ensuite, ce sera optionnel comme nous l'expliquerons plus tard.
A partir de ce moment, et jusqu'à la fin de l'exécution, vous pouvez taper RESET pour retourner au début du programme SOURCEROR. Si vous tapez RESET une deuxiéme fois, vous sortirez de SOURCEROR et retournerez au BASIC (au cas où vous utilisez le moniteur autostart).

3. COMMANDES UTILISEES DANS LE DESASSEMBLAGE

Les commandes de désassemblage sont très similaires à celles utilisées par le désassembleur du moniteur APPLE. Toutes les commandes acceptent une adressse hexadécimale sur 4 digits avant la lettre de commande. Si ce nombre est omis, alors le désassemblage continue à partie de l'adresse actuelle. Il n'y a que lors de l'entrée initiale qu'un nombre doit être spécifié.

Si vous spécifiez un nombre plus grand que l'adresse actuelle, un nouveau ORG sera créé.

Plus communément, vous voudrez spécifier une adresse inférieure à la valeur par défaut actuelle. Dans ce cas, le désassembleur vérifiera si l'adresse spécifiée est la même que l'adresse d'une des lignes précédentes. Dans ce cas, il reculera simplement à cet endroit. Dans le cas contraire, il retournera à l'adresse suivante utilisée et créera un nouveau ORG. Les lignes suivantes de la source seront effacées. C'est généralement préférable d'éviter des nouveaux ORG quand c'est possible. Si vous créez un nouveau ORG et n'en voulez pas, essayer de reculer d'un bit supplémentaire jusqu'à ce que vous n'obteniez plus de ORG dans le désassemblage.

L (List)

C'est la commande principale de désassemblage. Elle désassemble 20 lignes de code. Elle peut être répétée (par exemple, 2000LLL désassemblera 60 lignes de code à partir de $2000). Si un JSR de l'interpréteur SWEET 16 est trouvé, le désassemblage est automatiquement commuté en mode SWEET 16.

La commande L continue tout le temps le même mode (SWEET 16 ou NORMAL) de désassemblage.

Si un mnémonique illégal est rencontré, alors, le haut-parleur retentira et le mnémonique sera affiché par 3 points d'interrogation clignotants. C'est uniquement pour attirer votre attention sur la situation. Dans le code source lui-même, les mnémoniques non reconnus sont convertis en données hexadécimales mais ne sont pas affichées à l'écran.

S (Sweet)

Cette commande est identique à L mais force le désassemblage à commencer en mode SWEET 16. Le mode SWEET 16 retourne au mode normal 6502 quand le code SWEET 16 RTN est trouvé.

N (Normal)

Cette commande est identique à L mais force le désassemblage à commencer en mode normal 6502.

H (Hex)

Cette commande crée le mnémonique de données HEX. Sa valeur par défaut est de 1 byte de données. Si vous insérez un byte (d'1 ou 2 digits) d'un nombre hexadécimal après un H, alors ce nombre sera généré.

T (Text)

Cette commande essaye de désassembler la donnée de l'adresse courante comme si c'était une chaîne ASCII. Suivant la forme de la donnée, elle sera (automatiquement) désassemblée sous le pseudo mnémonique ASC, DCI, INV ou FLS. Le délimiteur approprié " ou ' est choisi automatiquement. Le désassemblage sera interrompu quand la donnée rencontrée est inappropriée, quand 62 caractères ont été traités ou quand le bit haut de la donnée change. Dans le dernier cas, le mnémonique ASC est changé automatiquement en DCI.

Parfois, le changement en un DCI est inapproprié. Ce changement peut être inhibé en utilisant la commande TT au lieu de la commande T.

Occasionnellement, la chaîne désassemblée peut ne pas s'arrêter à l'endroit approprié du fait que le code suivant ressemble à une donnée ASCII pour le SOURCEROR. Dans ce cas, vous pouvez limiter le nombre de caractères mis dans la chaîne en insérant un nombre hexadécimal d'1 ou 2 digits après la commande T. La commande T ou TT peuvent aussi être utilisées pour établir la limite correcte entre une chaîne ASCII normale et une clignotante. Le moment de faire cela est généralement évident.

Les lettres minuscules qui apparaissent dans une chaîne sont affichées (dans SOURCEROR, et pas dans BIG MAC) en lettres clignotantes.

W (Word)

Cette commande désassemle les 2 prochains bytes à l'adresse courante comme un mnémonique DA. En option, si la commande WW est utilisée, alors ces 2 bytes sont désassemblés en un mnémonique DDB. Enfin, si la commande W- est utilisée, les 2 bytes sont désassemblés sous la forme DA ETIQUETTE-1. (La dernière commande est souvent la forme appropriée quand le programme utilise l'adresse en la mettant sur la pile. Vous pouvez détecter cela pendant le désassemblage ou seulement après que le programme ait été désassemblé. Dans ce dernier cas, il peut être avantageux de recommencer votre assemblage avec des notes en mains.)

4. COMMANDES DE MISE EN PAGE

/ (Cancel) (annuler)

Cette commande a pour fonction principale d'annuler la dernière commande. Plus exactement, elle rétablit la dernière adresse par défaut (l'adresse utilisée pour une commande, pas nécessairement attachée à une adresse). C'est un facilité qui vous permet de ne pas taper une adresse quand un retour est désiré. Par exemple, supposez que vous tapiez T pour désassembler un texte. Vous pouvez ne pas savoir à quoi vous attendre, donc vous pouvez simplement taper L pour voir ce qui va en résulter. Alors, par exemple, le texte doit être suivi d'une donnée hexadécimale (comme un $8D pour un retour charriot), tapez simplement la commande H appropriée.

R (Read) (lire)

Cette commande vous permet de scruter la mémoire d'une façon qui fait ressortir le texte inséré. Pour scruter les données de $1234 à $1333, tapez 1234R. Après cela, le fait de taper R vous permettra de voir la page suivante de la mémoire. Les nombres que vous utilisez pour cette commande sont totalement indépendants des adresses de désassemblage. Donc, vous pouvez désassembler puis utiliser (adresse)R puis L (seul, et le désassemblage se poursuivra comme si vous n'aviez jamais tapé R). Si vous n'avez pas l'intention d'utiliser l'adresse par défaut quand vous retournez au désassemblage, il est prudent de noter l'endroit où vous voulez poursuivre ou d'utiliser / avant R.

I (Instructions) (instructions)

Cette commande imprime la page-titre pour vous rappeler les commandes autorisées.

Q (Quit) (quitter)

Cette commande termine le désassemblage et se branche sur la procédure de fin, ce qui est automatique. Si vous tapez une adresse avant Q, alors le pointeur d'adresse est ramené à ce point (non inclus) avant de poursuivre le processus. Donc, à la fin du désassemblage, les lignes désassemblées comprennent :

```
2341- 4C 03 E0      JMP     $E003
2344- A9 BE 94      LDA     $94BE,Y
```

et la dernière ligne n'est pas significative; tapez alors 2344Q pour annuler la dernière ligne et retenir la première.

E. PROCESSUS DE FIN

Après la commande Q, le programme travaille quelques minutes pour générer le code désassemblé. Si vous tapez RESET à ce moment, vous retournerez au BASIC et perdrez le code désassemblé.

Ce processus prend une ou deux secondes pour des programmes courts ou une ou deux minutes pour des programmes plus longs.

Quand le processus est terminé, il vous est demandé si vous désirez sauver la source (si vous ne le faites pas, elle sera perdue). Si vous voulez la sauver, il vous sera demandé un nom de fichier. SOURCEROR ajoutera un suffixe .S à ce nom et le sauvera sur la disquette.

Le drive utilisé sera le drive utilisé quand vous avez fait BRUN SOURCEROR. Donc, changez de disque avant d'utiliser cette commande si vous voulez.

Pour regarder ce programme source désassemblé, faites BRUN BIG MAC et chargez-le.

F. RELATIONS AVEC LE PROGRAMME SOURCE TERMINE

Dans la plupart des cas, quand vous aurez un peu d'expérience et pour autant que vous preniez un minimum de précautions, le programme source n'aura pas ou peu de défauts.

Vous pouvez noter que certains DA auraient été plus appropriés dans des formats DA ETIQUETTE-1 ou dans des DDB ETIQUETTE. Dans ce cas, et dans d'autres similaires, le mieux est peut-être de désassembler à nouveau avec des notes sous les yeux. Le désassemblage est si rapide et sans douleur que c'est souvent plus rapide que d'essayer de modifier la source de façon adéquate.

Le programme source contiendra toutes les étiquettes extérieures ou non reconnues à la fin, dans une table d'équation. Vous devez lire cette table attentivement. Elle ne peut contenir aucune équation en page zéro sauf celles résultant d'un DA, JMP ou JSR. C'est presque sûrement un signe d'erreur (de votre part, pas de SOURCEROR) dans le désassemblage. Elle peut provenir d'une tentative de désassemblage d'un champ donnée comme si c'était un mnémonique. Notez que si vous essayer d'assembler la source dans ces conditions, vous obtiendrez une erreur dès que les équations apparaitront. Si vous devez éventuellement le faire, déplacez les équations au début du programme et vous n'obtiendez pas d'erreur, mais l'assemblage NE SERA PAS CORRECT. Donc, il est important d'analyser la situation d'abord. Le problème apparait pour les raisons suivantes : si, par exemple, le désassembleur trouve la donnée AD 00 8D, il la désassemblera , de façon correcte, en LDA $008D. L'assembleur, de toute façon, assemble toujours ce code comme une instruction en page zéro, donnant les 2 bytes AD 8D. Occasionnellement, vous trouverez des programmes qui utilisent cette forme pour une instruction en page zéro. Dans ce cas, vous devrez la changer en une donnée HEX pour qu'elle s'assemble dans sa forme originale. Plus souvent, c'est la donnée en première position, à la place du code et cela devra être réorganisé pour obtenir un assemblage correct.

7. LE MESSAGE "MEMORY FULL" (MEMOIRE PLEINE)

Quand le fichier source approche de $600 l'adresse de départ de SOURCEROR, (c'est-à-dire quand il va au-delà de $8400), vous verrez le message "MEMORY FULL" (= mémoire pleine) et "HIT A KEY" (= frappez une touche). SOURCEROR ira directement au processus final. La raison de la marge de $600 est que SOURCEROR a besoin de place pour ce processus. Il est possible, mais peu probable, qu'une partie de SOURCEROR soit écrasé pendant ce processus final, mais cela ne pose pas de problème étant donné que cette partie de programme n'est pas utilisée à ce moment. Il y a une provision de chevauchement "secret" quand la mémoire est pleine. Si la touche que vous tapez est CONTROL O (pour Override = chevauchement), alors SOURCEROR vous demandera une nouvelle commande. Vous pouvez utiliser cette possibilité pour spécifier l'endoit de fin désiré. Vous pouvez aussi l'utiliser pour aller un peu plus loin que SOURCEROR le veut et désassembler quelques lignes de plus. Evidemment, vous ne devez pas pousser cela à l'extrême.

8. HELLO ET GREETINGS

Initialement, la disquette a deux programmes appelés "HELLO". En chargeant ou "runnant" HELLO la première fois, un programme titre animé sera présenté. Quand le programme aura été exécuté, il s'effacera de lui-même, et quand vous ferez tourner le programme par la suite, c'est le BIG MAC qui se chargera directement. Le programme d'animation se trouve également sur la disquette sous le nom "GREETINGS".

C. LES PROGRAMMES ANNEXES DU FICHIER TEXTE

1. INTRODUCTION

Deux programmes appelés READER et WRITER vous offrent la possibilité d'utiliser BIG MAC comme un éditeur de textes limité et en plus, vous permet de lire dans les fichiers source créés dans un format fichier texte par un autre assembleur comme le DOS TOOL KIT d'Apple, puis de le convertir dans la syntaxe de BIG MAC.

2. READER (lecteur)

Ce programme peut lire n'importe quel fichier texte séquentiel dans le buffer d'édition de BIG MAC qui peut être utilisé pour éditer comme un fichier source normal. Quand le texte a été édité, il peut être sauvé comme un fichier binaire en utilisant la commande normale de BIG MAC S(ave) ou peut être sauvé comme un fichier texte en utilisant le programme WRITER.

N'importe quel fichier texte séquentiel peut être lu et édité par exemple données EXEC, source assembleur, ... READER ajoutera le préfixe .T au nom du fichier; donc, si votre fichier texte ne possède pas ce préfixe, vous devez d'abord changer son nom avant d'appeler READER. Plus tard, vous pourrez à nouveau lui donner son nom original si vous le désirez, Des numéros de drives ou de slots peuvent être spécifiés en ajoutant "D?" ou un paramètre approprié au nom du fichier donné à READER. Des lignes plus longues que 256 caractères ($100 bytes) seront scindées en 2 lignes et devront être prises en considération.

INSTRUCTIONS :

1. BRUN BIG MAC
2. Tapez "C" pour le catalogue
3. Après COMMAND:, tapez BRUN READER
4. READER vous demandera le nom du fichier que vous voulez charger; tapez-le
5. Quand le programme est chargé, vous serez renvoyé au mode EXEC de BIG MAC. Tapez "E" pour entrer dans l'éditeur.

3. WRITER (celui qui écrit)

Ce programme écrit un fichier BIG MAC dans un fichier texte. Il ne fait pas de différence, quelle que soit la forme originale du fichier; une fois qu'il est dans le buffer d'édition, WRITER le sortira comme un fichier texte.

Avant d'utiliser WRITER, assurez-vous que le fichier est intact en utilisant les commandes d'édition L(ist) ou P(rint) car WRITER effacera le fichier original si vous ne changez pas le nom. Comme READER, WRITER ajoute un préfixe .T au nom du fichier et, de la même façon, vous pouvez spécifier des drives ou des slots en utilisant les paramètres D ou S.

INSTRUCTIONS :

1. BRUN BIG MAC (si nécessaire)
2. L(oad) ou create a file
3. Tapez C pour Catalog
4. Après COMMAND:, tapez BRUN WITER
5. WRITER vous demandera le nom du fichier à sauver; tapez-le.

D. LA LIBRAIRIE MACRO

Une librairie MACRO avec trois programmes macros à titre d'exemple est incluse sur la disquette. Le but de cette librairie est de fournir un guide pour le nouveau venu aux macros et la façon dont elles peuvent être utilisées dans un programme d'assemblage. Notez que toutes les macros sont définies au début du fichier source, puis chaque programme place les macros où elles sont nécessaires. Des conditionnels sont utilisés pour déterminer quel programme d'exemple doit être assemblé.

E. SUPPLEMENT ROM AUTOSTART

Le supplément est un groupe de routines utilitaires qui résident dans les ROM F4 et F8 d'un Apple II standard. Elles ne se trouvent pas dans l'Apple II plus ni dans l'Apple II qui a le vieux moniteur ROM (qui a été remplacé par la ROM autostart). Les instructions pour utiliser le pas à pas, la commande TRACE et le mini-assembleur peuvent être trouvées dans le livre de référence. Les programme de Wozniak ont été modifiés par Guil Banks qui, en plus, a écrit le programme "CONVERT" inclus. Une documentation complète pour le supplément est aussi contenue dans le programme SUPPLEMENT.DOC.

Le "SUPPLEMENT" peut être entré depuis le moniteur par la fonction utilisateur CONTROL Y et exige que votre programme HELLO BASIC contiennent les 2 lignes suivantes :

```
100  HIMEM:29440:REM $8D00
110  POKE 1016,76:POKE 1017,0:POKE 1018,141: REM FAIRE SAUTER CTRL Y DE
     3F8 A $8D00
```

Le curseur de "SUPPLEMENT" est le caractère ")", et toutes les commandes du moniteur peuvent être entrées depuis "SUPPLEMENT" sauf la commande CTRL A qui branche le moniteur. Un slash "/" vous fait sortir du mini assembleur et vous renvoie à "SUPPLEMENT". La fonction TRACE se termine également par un slash. Dans le mode TRACE, un CONTROL S provoquera une pause dans le listing jusqu'à ce qu'une touche soit pressée.

La routine CONVERT convertit un nombre hexadécimal entre 0 et FFFF en un nombre décimal et l'affiche à la fois sous sa forme normale et sous sa forme en complément à 2. On entre dans CONVERT avec un CONTROL T et on en sort par un slash. Le curseur dans le mode CONVERT est un "^".

Pour passer du mode décimal en mode hexadécimal, tapez "^T" suivi de "RETURN". Pour passer du mode hexadécimal en mode décimal, tapez "^H" suivi d'un "RETURN".

CONVERT restera dans le mode de conversion précisé par le paramètres T ou H jusqu'à ce que le paramètre opposé soit tapé ou jusqu'à ce qu'un slash soit tapé. Notez que le contrôle retourne au moniteur si une touche, autre que celles précisées est tapée. CONVERT n'accepte pas les nombres négatifs en input.

EXEMPLES DE CONVERSION

```
^T c/rtn  ^936  c/rtn  affiche : 03A8FC58
^h c/rtn  ^FC58   c/rtn  affiche : 64400 936
```

F. SWEET 16

1. INTRODUCTION

SWEET 16 est probablement le programme le moins utilisé et le plus mal compris dans l'Apple II.

Exactement comme les BASIC Applesoft et Integer, SWEET 16 est un langage. Toutefois, comparé au BASIC, il doit être classé dans les langages de bas niveau, présentant une forte ressemblance avec le langage conventionnel d'assemblage du 6502.

Pour utiliser SWEET 16, vous devez étudier le langage et, pour citer WOZ, "la liste des mnémoniques est courte et simple". Woz est évidemment Mr Apple et le créateur de SWEET 16.

SWEET 16 se trouve en ROM sur tous les Apple II de l'adresse $F689 à l'adresse $F7FC. Il a ses propres commandes de mnémoniques et d'instructions et utilise les routines SAVE et RESTORE depuis le moniteur Apple pour préserver les registres du 6502 quand ils sont utilisés, permettant à SWEET 16 d'être utilisé comme une sous-routine.

Il utilise les 32 premières adresses de la page zéro pour mettre les registres de ses 16 doubles bytes et n'est par conséquent pas compatible avec le BASIC APPLESOFT sans aménagement.

L'article original "SWEET 16 : LA MACHINE DE REVE DU 6502" a d'abord été publié dans "BYTE MAGAZINE" en Novembre 1977 et plus tard dans le "WOZ PAK" original. Cet article est inclus ici pour aider à la compréhension et à l'implantation de SWEET 16.

Des exemples de l'utilisation de SWEET 16 peuvent être trouvés dans le "PROGRAMMER'S AID No 1" et dans les programmes RENUMBER, RELOCATE et APPEND. Le "PROGRAMMER'S AID OPERATING MANUAL" contient les listings complets d'assemblage de source indexés en page 65.

Le programme de démonstration, conçu pour être une introduction ssimple, est constitué de 3 parties :

1. Programme Basic Integer
2. Sous-routine en langage machine
3. Sous-routine SWEET 16

Le but du programme est de déplacer des données. Les paramètres du déplacement seront entrés dans le programme en Basic Integer.

Le "CALL 768" ($300) à la ligne 120, branche le programme sur une sous-routine en langage machine; cette sous-routine à pour simple but de d'entrer une sous-routine SWEET 16 et par la suite de retourner au BASIC (adresses $300, $301, $302 et $312 respectivement). La sous-routine SWEET 16 effectue évidemment le déplacement et est entrée aux adresses hexadécimales $303 à $311 (voyez le listing No 3).

Après le déplacement, l'écran affichera 3 lignes de données, chacune de 8 bytes de long, et attendra l'entrée de nouveaux paramètres. Les 3 lignes de données affichées à l'écran se présenteront comme suit :

Ligne 1 : les 8 premiers bytes de données démarrant à l'adresse $800, qui est la source de données fixée qui doit être déplacée (dans ce cas, la chaîne A$).

Ligne 2 : les 8 bytes de données commençant à l'adresse entrée comme destination du déplacement (byte de poids fort uniquement).

Ligne 3 : les 8 premiers bytes de données commençant à l'adresse $0000 (les 4 premiers registres de SWEET 16).

L'affichage de 8 bytes de données a été choisi pour simpifier l'illustration du foncionnement.

Le Basic Integer a son moyen propre d'enregistrer la chaîne A$. Du fait que le nom choisi pour la chaîne est rangé sur 2 bytes, un total de 5 bytes "de ménage" précède les données entrées pour A$, laissant seulement 3 bytes pour l'affichage. Le Basic Integer ajoute aussi un byte "de ménage" à la fin de la chaîne, connu sous le nom de "fin de chaîne". De ce fait, pour la facilité de l'affichage, et pour voir le byte de fin de chaîne au huitième byte, la chaîne entrée au clavier doit se limiter à 2 caractères et apparaitra sur les sixième et septième byte. En plus, les paramètres entrés doivent contenir le numéro des bytes à déplacer. Un champ habituel pour cette démonstration est le champ de 1 à 8 inclus, mais il est évident que ça marche de 1 à 255.

Finalement, l'adresse de départ de la destination du déplacement doit être entrée. De nouveau, pour la simplicité, le byte haut seulement est entré et le programme vous permet de choisir entre un chiffre décimal entre 9 et le H.O.B. du pointeur de programme 1 pour ne pas avoir de problème superflu (dans cette démonstration, choisissez un nombre entre 9 et 144 pour un Apple 48K).

Les 8 bytes de données affichés à partir de $00 vous permettront d'observer l'état des registres du SWEET 16 après qu'un déplacement ait été effectué et, de ce fait, de comprendre comment le programme SWEET 16 travaille.

Dans l'article "Sweet 16 : la machine de rêve du 6502", il est rappelé que SWEET 16 peut établir des registres de 16 doubles bytes à partir de $00. Cela veut dire que SWEET 16 peut utiliser les 32 premières adresses en page zéro.

Les "événements" qui arrivent dans ce programme de démonstration peuvent être étudiés dans les 4 premiers registres de SWEET 16 et de ce fait, l'affichage de 8 bytes démarrant en $0000 est suffisant pour cet usage. Ces 4 registres sont établis en R0, R1, R2 et R3 :

```
R0   $0000   &   0001    - accumulateur du SWEET 16
R1   $0002   &   0003    - adresse source
R2   $0004   &   0005    - adresse destination
.
.
.
R14  $001C   &   001D    - registre résultat antérieur
R15  $001E   &   001F    - compteur du programme SWEET 16
```

Vous pourrez trouver de plus amples informations dans "La notice de WOZ.text". Notez que le H.O.B de R14 (à l'adresse $1D) contient $06, ce qui est la spécification du registre double (3*2 = $06). R15, le compteur du programme SWEET 16 contient l'adresse de l'opération suivante (de même que pour chaque pas de programme) qui était $0312 quand le programme s'est terminé.

Pour essayer simplement le programme, entrez-le en Basic Integer comme montré dans le listing 1. Evidemment, les REM peuvent être omis et la ligne 10 n'est nécessaire que si on veut sauver le code macchine sur une disquette.

Le listing 2 doit aussi être entré en $300.

Notez que le listing de désassemblage du 6502 ne ressemble pas au listing 3, mais le programme de désassemblage SOURCEROR générera un désassemblage correct.

Entrez RUN et RETURN

Entrez 12 et tapez RETURN (A$-A$ chaîne données)
Entrez 8 et tapez RETURN (byte haut de destination)

L'affichage doit apparaître comme suit :

```
$0800-C1 40 00 10 08 B1 B2 1E  (SOURCE)
$0A00-C1 40 00 10 08 B1 B2 1E  (DESTINATION)
$0000-1E 00 08 08 08 0A 00 00  (SWEET 16)
```

Notez que les 8 bytes rangés en $0A00 sont identiques aux 8 bytes démarrant en $0800, ce qui indique que que le déplacement des 8 bytes a été correctement réalisé. Nous verrons qu'ils ont été déplacés byte par byte en commençant par C1 et en terminant par E1 (si on a déplacé mois de 8 bytes, les données suivant le déplacement existeront de toute façon à ces adresses avant le déplacement).

Les bytes ont la signification suivante : (le $ est omis)

C1	40	00	10	08	B1	B2	1E
VN		DSP		NVA	DATA	DATA	chaîne fin

Les registres du SWEET 16 se présentent comme suit :

	Bas	Haut	Bas	Haut	Bas	Haut	Bas	Haut
$0000	1E	00	08	08	08	0A	00	00
	registre R0 (acc)		registre R1 (source)		registre R2 (destination)		registre R3 (£ bytes)	

Le byte bas de R0, l'accumulateur SWEET 16, est à $1E, ce qui était le dernier byte déplacé (le huitième).

Le byte bas du registre de la source, R1, démarre à $00 et a été incrémenté 8 fois, une fois par déplacement d'un byte de données.

Le byte haut de du registre de destination, R2, contient $0A dans lequel on a mis la variable "10" et rangé dans le code SWEET 16. Le LOB de R2 a été incrémenté exactement comme R1.

Finalement, le registre R3, qui contient le nombre de bytes qui doivent être déplacés, a été poussé à 8 et décrémentés 8 fois. Donc il contient $0000 au bout du compte.

En entrant une chaîne de caractères et en faisant varier le nombre de bytes à déplacer, les registres SWEET 16 peuvent être observés.

En travaillant avec ce programme de démonstration, et en étudiant la matière du texte, vous serez vite capable d'écrire des programmes SWEET 16 qui permettent des manipulations 16 bits. Les codes-opération mentionnés dans l'article de WOZ présentent une opportunité très intéressante de les manipuler.

SWEET 16, comme un langage ou un outil, élargit l'horizon des possesseurs d'Apple II de façon permanente et sans dépenser un sou.

Pour les utilisateurs d'Apple qui désirent apprendre la programmation en langage machine, SWEET 16 peut être utilisé comme point de départ. Possédant moins de mnémoniques, il peut être efficace très rapidement.

Pour ceux qui ne possède pas le Basi Integer, SWEET 16 est fourni sur cette disquette.

LISTING 1

```
1   D$="":REM CTRL-D
10  PRINT D$;"BLOAD SWEET"
20  CALL -936:DIM A$(10)
30  INPUT "ENTREZ LA CHAINE A$ ";A$
40  INPUT "ENTREZ LE NOMBRE DE BYTES "; B
50  IF NOT B THEN 40 : REM AU MOINS 1
60  POKE 778,B : REM POKE LA LONGUEUR
70  INPUT "ENTREZ LA DESTINATION "; A
80  IF A>PEEK (203)-1 THEN 70
90  IF A<PEEK (205)+1 THEN 70
100 POKE 776,A : REM POKE LA DESTINATION
110 M=B : GOSUB 160 : REM AFFICHAGE
120 CALL 768 : REM GOTO $0300
130 M=A : GOSUB 160 : REM AFFICHAGE
140 M=0 : GOSUB 160 : REM AFFICHAGE
150 PRINT : PRINT: GOTO 30
160 POKE 60,0 : POKE 61,M
170 CALL 605 : RETURN
```

LISTING 2

Entrez le code comme suit :

```
                                        A       B

300:20  B9   F6   11  00  0B  12  00  00  13  00  00  41  51  52
    F3  07FB 00  60
```

LISTING 3

```
SWEET 16

$300  20  B9  F6  JSR        $F689
$303  11  00  08  SET        R1      ADRESSE SOURCE
$306  12  00  00  SET        R2      ADRESSE DESTINATION
                  A
$309  13  00  00  SET        R3      LONGUEUR
                  B
$30C  41          LD         @R1
$30D  52          ST         @R2
$30E  F3          DCR        R3
$30F  07  FB      BNZ        $30C
$311  00          RTN
$312  60          RTS
```

Les données seront transférées depuis le programme en Basic Integer :

"A" depuis la ligne 100
"B" depuis la ligne 60

2. SWEET 16 : UN MICROPROCESSEUR PSEUDO 16 BITS

a. Description

En écrivant "APPLE BASIC pour le microprocesseur 6502", j'ai rencontré à plusieurs reprises une variante de "MURPHY'S LAW" (la loi de Murphy). Enoncée brièvement, n'importe quelle routine opérant sur une donnée 16 bits requière au moins 2 fois le code. Les programmes utilisant intensément les pointeurs 16 bits (comme les compilateurs, les éditeurs et les assembleurs) sont compris dans cette catégorie. Dans mon cas, même l'addition de quelques instructions double byte sur le 6502 aurait seulement allégé supperficiellement le problème. Ce dont j'avais réellement besoin, était un "6502/RCA 1800 HYBRID", un puissant manipulateur de données 8 bits avec, en complément, un processeur facile à utiliser, contenant beaucoup de registres 16 bits et des excellentes possibilités "pointeurs". Ma solution était d'implémenter un processeur 16 bits non existant par software, style interpréteur, que j'ai appelé SWEET 16.

SWEET 16 est basé sur 16 registres 16 bits (de R0 à R15), en fait 32 adresses mémoire. R0 comme accumulateur du SWEET 16 (ACC), R15 comme "programme counter" (compteur ordinal) et R14 comme le registre d'état. R13 contient le résultat des instructions de comparaison et R12 est la sous-routine pointeur de pile si les sous-routines SWEET 16 sont utilisées. Tous les autres registres SWEET 16 sont à la disposition de l'utilisateur.

Les instructions SWEET 16 tombent dans les catégories registres et non-registres. Les opérations registre spécifient un des 16 registres à utiliser aussi bien comme élément donnée ou comme pointeur de données en mémoire; cela dépend de l'instruction spécifique. Par exemple, INR R5 utilise R5 comme donnée et ST àR7 utilise R7 comme pointeur de données rangées en mémoire. Excepté pour l'instruction SET, les opérations registre prennent un byte de code chacune. Les opérations non registres sont primitivement du style 6502 reliées à second byte spécifiant un déplacement de plus ou moins 127 bytes relatif à l'adresse de l'opération suivante. Dans le cas où le résultat de l'opération registre précédente rencontre un branchement spécifié, le déplacement est ajouté au compteur ordinal (PC) SWEET 16 effectuant un branchement.

SWEET 16 est destiné à être utilisé avec le 6502 et non comme un processeur indépendant. Un programme 6502 passé en mode SWEET 16 par un appel sous-routine et les codes suivants sont interprétés comme des instructions SWEET 16. L'opération on registre RTN renvoie le programme utilisateur au mode 6502 après avoir rétabli le contenu des registres internes (A, S, Y, P, et S). L'exemple suivant illustre la façon d'utiliser SWEET 16 :

```
300  B9  00  02            LDA  IN,Y        prendre un caractère
303  C9  CD                CMP  £"M"        "M" pour le déplacement
305  D0  09                BNE  NOMOVE      non : saut à déplacement
307  20  89  F6            JSR  SW16        oui : appel à SWEET 16
30A  41          BOUCLE M  LD   àR1         R1 contient l'adresse source
30B  52                    ST   àR2         R2 contient l'adresse destin.
30C  F3                    DCR  R3          décrémenter la longueur
30D  07  FB                BNZ  BOUCLE M    boucler jusqu'à la fin
30F  00                    RTN              retourner au mode 6502
310  C9  C5       NOMOVE    CMP  £"E"        caractère "E" ?
312  D0  13                BEQ  EXIT        oui, on sort
314  C8                    INY              non, on continue
```

Note : les registres A, X, Y, P et S ne sont pas perturbés par SWEET 16.

b. Description des instruction

Le listing des mnémoniques de SWEET 16 est court et facile. Excepté pour les déplacements par branchement relatif, l'assemblage à la main est trivial. Tous les codes-opération registre sont formés de la combinaison de 2 chiffres hexadécimaux, un pour le code opération et l'autre pour spécifier un registre. Par exemple, les codes-opération 15 et 45 spécifient tous les deux le registre R5 tandis que les codes 23, 27 et 29 sont tous des opérations ST. La plupart des opérations registre sont assignées en paire complémentaire pour faciliter leur mémorisation. Donc, LD et ST sont les codes-opération 2N et 3N tandis que LDà et STà sont les codes 4N et 5N.

Les codes allant de 0 à C (hex) sont assignés aux treize opérations non-registre. Excepté pour RTN (code-opération), BK (0A) et RS (0B), les opérations non-registre sont des branchements du style 6502. Le second byte d'une instruction de branchement contient une valeur de déplacement de plus ou moins 127 bytes (sous la forme complément à 2) relative à l'instruction suivant immédiatement le branchement. Si une condition de branchement spécifié est rencontrée par le résultat de l'opération du précédent registre, le déplacement est ajouté au compteur ordinal (PC) effectuant un branchement. Excepté pour BR (Branch Always = brancher toujours) et BS (Branch to Subroutine = brancher à sous-routine), les codes-opération de branchement sont assignés en paires complémentaires, de telle façon qu'ils sont plus facile à mémoriser pour l'assemblage à la main. Par exemple Branch if Plus (brancher si plus) et Branch if Minus (brancher si moins) sont les codes-opération 4 et 5 tandis que Branch if Zero (brancher si zéro) et Branch if NonZero (brancher si différent de zéro) sont les codes-opération 6 et 7.

c. Résumé des codes-opération SWEET 16

OPERATIONS REGISTRE

1n	SET	Rn	Set (mettre constante)
2n	LD	Rn	Load (charger)
3n	ST	Rn	Store (ranger)
4n	LD	àRn	Load Indirect (chargement indirect)
5n	ST	àRn	Store Indirect (rangement indirect)
6n	LDD	àRn	Load Double Indirect (chargement indirect double)
7n	STD	àRn	Store Double Indirect (rangement indirect double)
8n	POP	àRn	POP Indirect
9n	STP	àRn	Store POP Indirect (ranger POP indirect)
An	ADD	Rn	Add (ajouter)
Bn	SUB	Rn	Sub (soustraire)
Cn	POPD	àRn	POP Double Indirect (POP indirect double)
Dn	CPR	Rn	Compare (comparer)
En	INR	Rn	Increment (incrémenter)
Fn	DCR	Rn	Decrement (décrémenter)

00	RTN		Return to 6502 mode (retourner au mode 6502)
01	BR		Branch Always (brancher toujours)
02	BNC	ea	Branch if No Carry (brancher si pas de retenue)
03	BC	ea	Branch if Carry (brancher si retenue)
04	BP	ea	Branch if Plus (brancher si plus)
05	BM	ea	Branch if Minus (brancher si moins)
06	BZ	ea	Branch if zero (brancher si zéro)
07	BNZ	ea	Branch if Not Zero (brancher si différent de zéro)
08	BM1	ea	Branch if Minus 1 (brancher si moins de 1)
09	BNM1	ea	Branch if Not Minus 1 (brancher si pas moins de 1)
0a	BK		Break
0B	RS		Return from Subroutine (revenir de sous-routine)
0C	BS	ea	Branch to Subroutine (brancher à sous-routine)
0D			Non assigné
0E			Non assigné
0F			Non assigné

d. Instructions registre

SET Rn,Constante 1n Bas Haut Set

La constante 2 bytes est chargée dans Rn (n=0 à F, hex) et les conditions de branchement sont mises en fonction de cette constante. La retenue est annulée.

LD Rn 2n Load

L'accumulateur R0 est chargé avec Rn et les conditions de branchent sont mises en fonction des données transférées. La retenue est annulée et le contenu de Rn n'est pas affecté.

ST Rn 3n Store

Le contenu de l'accumulateur est rangé en Rn et les conditions de branchement sont mises en fonction des données transférées. La retenue est annulée et le contenu de l'accumulateur n'est pas affecté.

LD àRn 4n Load Indirect

Le byte bas de l'accumulateur est chargé avec le conenu de la mémoire dont l'adresse est spécifiée dans Rn et le byte haut est annulé. Les conditions de branchement reflètent le contenu final de l'accumulateur qui sera toujours positif et jamais à moins 1. La retenue est annulée. Après le transfert, Rn est incrémenté de 1.

Exemple :

```
15  34  A0      SET     R5,$A034      L'accumulateur est chargé avec le contenu de la
45              LD      àR5           mémoire $A034
                                      R5 est incrémenté à $A035
```

ST àRn 5n Store Indirect

Le byte bas de l'accumulateur est rangé dans la mémoire dont l'adresse est spécifiée dans Rn. Les conditions de branchement reflètent le contenu des 2 bytes de l'accumulateur. La retenue est annulée. Après le transfert, Rn est incrémenté de 1.

Exemple :

```
15  34  A0     SET    R5,$A034     Charger les pointeurs R5 et R6 avec
16  22  90     SET    R6,$9022     $A034 et $9022
45              LD     àR5          Déplacer le byte de $A034 en $9022
56              ST     àR6          Les 2 pointeurs sont incrémentés
```

LDD àRn 6n Load double-byte indirect

Le byte bas de l'accumulateur est chargé avec le contenu de la mémoire dont l'adresse est spécifiée dans Rn et Rn est incrémenté de 1. Le byte haut de l'accumulateur est chargé avec le contenu de la mémoire est spécifié dans Rn incrémenté et Rn est à nouveau incrémenté de 1. Les conditions de branchement reflètent le contenu final de l'accumulateur. La retenue est annulée.

Exemple :

```
                                    Le byte bas est chargé de la mémoire $A034,
15  34  A0     SET    R5,$A034     le byte haut est chargé de la mémoire $A035
45              LDD    àR5          R5 est incrémenté à $A036
```

STD àRn 7n Store double-byte indirect

Le byte bas de l'accumulateur esr rangé dans la mémoire dont l'adresse est spécifiée dans Rn et Rn est incrémenté de 1. Le byte haut de l'accumulateur est rangé dans la mémoire dont l'adresse est spécifiée par Rn incrémenté et Rn est à nouveau incrémenté de 1. Les conditions de branchement reflétent le contenu de l'accumulateur qui n'est pas altéré. La retenue est annulée.

Exemple :

```
15  34  A0     SET    R5,$A034     Charger les pointeurs R5 et R6
16  22  90     SET    R6,$9022     avec $A034 et $9022
65              LDD    àR5          Déplacer le double byte de
76              STD    àR6          $A034-$A035 en $9022-$9023
                                    Les 2 pointeurs sont incrémentés de 2
```

POP àRn Rn Pop indirect

Le byte bas de l'accumulateur est chargé avec le contenu de la mémoire dont l'adresse est spécifiée par Rn après que Rn ait été décrémenté de 1 et le byte haut de l'accumulateur est annulé. Les conditions de branchement reflètent le contenu final des 2 bytes de l'accumulateur qui doit toujours être positif et jamais moins de 1. La retenue est annulée. Du fait que Rn est décrémenté avant de charger l'accumulateur, les piles simple byte peuvent être implémentées avec les opérations ST àRn et POP à Rn (Rn est le pointeur de pile).

Exemple :

```
15  34  A0    SET   R5,$A034    Initialiser le pointeur de pile
10  04  00    SET   R0,4        Charger 4 dans l'accumulateur
55            ST    àR5         Pousser 4 sur la pile
10  05  00    SET   R0,5        Charger 5 dans l'accumulateur
55            ST    àR5         Pousser 5 sur la pile
10  06  00    SET   R0,6        Charger 6 dans l'accumulateur
55            ST    àR5         Pousser 6 sur la pile
85            POP   àR5         Faire sauter 6 de la pile dans l'accumulateur
85            POP   àR5         Faire sauter 5 de la pile dans l'accumulateur
85            POP   àR5         Faire sauter 4 de la pile dans l'accumulateur
```

STP àRn 9n Store POP indirect

Le byte bas de l'accumulateur est rangé dans la mémoire dont l'adresse se trouve dans Rn après que Rn ait été décrémenté de 1. Les conditions de branchement reflètent le contenu des 2 bytes de l'accumulateur qui ne sont pas modifiés. STP àRn et POP àRn sont utilisés ensemble pour déplacer des blocs de données commençant à l'adresse la plus haute et allant en décroissant. De plus, les piles simple byte peuvent être implémentées avec l'opération STP àRn.

ADD àRn An Add

Le contenu de Rn est ajouté au contenu de l'accumulateur, (R) et les 16 bits bas de la somme sont replacés dans l'accumulateur. Le 17e bit de la somme devient la retenue et les autres conditions de branchements reflètent le contenu final de l'accumulateur.

SUB Rn Bn Substract

Le contenu de Rn est soustrait du contenu de l'accumulateur en faisant une addition en complément à 2 :

ACC = ACC + Rn + 1

Les 16 bits bas résultant de la soustraction sont replacés dans l'accumulateur et le 17e bit devient la retenue. Les autres conditions de branchement reflètent le contenu final de l'accumulateur. Si la valeur des 16 bits non-signés contenus dans l'accumulateur est plus grande ou égale à la valeur des 16 bits non-signés contenus dans Rn, alors la retenue est mise sinon elle est annulée. La valeur de Rn n'est pas affectée.

Exemple :

```
10  34  76     SET     R0,$7634    Initialiser R0 (acc)
11  27  42     SET     R1,$4227    et R1
            SUB     R1          Soustraire R1 (diff. = $340D avec retenue)
            SUB     R0          Annuler l'accumulateur (R0)
```

POPD àRn Cn Pop Double-byte indirect

Rn est décrémenté de 1 et le byte haut de l'accumulateur est chargé avec le contenu de l'adresse mémoire dont l'adresse se trouve dans Rn. Rn est à nouveau décrémenté de 1 et le byte bas de l'accumulateur est chargé avec le contenu de la mémoire correspondante. Les conditions de branchement reflètent le contenu final de l'accumulateur. La retenue est annulée. Du fait que Rn est décrémenté avant de charger les deux moitiés de l'accumulateur, les piles double-byte peuvent être implémentées avec les opérations STD àRn et POPD àRn (Rn est le pointeur de pile).

Exemple :

```
15  34  A0     SET     R5,$A034    Initialiser le pointeur de pile
10  12  AA     SET     R0,$AA12    Charger $AA12 dans l'accumulateur
75             STD     àR5         Pousser $AA12 sur la pile
    34  BB     SET     R0,$BB34    Charger $BB34 dans l'accumulateur
75             STD     àR5         Pousser $BB34 sur la pile
C5             POPD    àR5         Faire sauter $BB34 de la pile
C5             POPD    àR5         Faire sauter $AA12 de la pile
```

CPR Rn Dn Comoare

Le contenu de l'accumulateur (R0) est comoaré à Rn en faisant la soustraction binaire 16 bits ACC = Rn et les 16 bits bas de la différence sont rangés dans R13 pour des tests de branchement ultérieurs. Si le contenu des 16 bits non-signés de l'accumulateur est plus grand ou égal au contenu des 16 bits non-signés de Rn, la retenue est mise, sinon elle est annulée. Aucun autre registre que l'accumulateur et Rn n'est perturbé.

INR Rn En Increment

Le contenu de Rn est incrémenté de 1. La retenue est annulée et les autres conditions de branchement reflètent la valeur incrémentée.

DCR Rn Fn Decrement

Le contenu de Rn est décrémenté de 1. La retenue est annulée et les autres conditions de branchement reflètent la valeur décrémentée.

e. Instructions non-registre

RTN 00 Return to 6502 mode

Le contrôle retourne au 6502 et l'exécution du programme continue à l'adresse suivant immédiatement l'instruction RTN. Les registres 6502 et les conditions d'état sont restaurées à leur contenu original (comme avant d'entrer dans le mode SWEET 16).

BR ea 01 d Branch always

Une adresse effective (ea) est calculée en ajoutant le byte de déplacement signé (d) au compteur ordinal (PC). Le compteur ordinal contient l'adressse de l'instruction suivant immédiatement l'instruction BR, ou l'adresse de l'opération BR plus 2. Le déplacement est une valeur en complément à 2 signée comprise entre -128 et +127. Les conditions de branchement ne sont pas changées.

Notez que le calcul de l'adresse effective est identique aux branchements relatifs du 6502. Les caractéristiques d'addition et de soustraction hexadécimales du moniteur Apple II peuvent être utilisées pour calculer ces déplacements.

```
d = $80   ea = PC + 2 - 128
d = $81   ea = PC + 2 - 127

d = $FF   ea = PC + 2 - 1
d = $00   ea = PC + 2 + 0
d = $01   ea = PC + 2 + 1

d = $7F   ea = PC + 2 + 126
d = $7F   ea = PC + 2 + 127
```

Exemple :

```
$300:  01  50      BR $352
```

BNC ea 02 d Branch if No carry

Un branchement est effectué à l'adresse effective uniquement si la retenue est annulée, sinon, l'exécution continue normalement à l'instruction suivante. Les conditions de branchement ne sont pas modifiées.

BC ea 03 d Branch if Carry set

Un branchement est effectué selement si la retenue est mise. Les conditions de branchement ne sont pas modifiées.

BP ea 04 d Branch if Plus

Un branchement est effectué seulement si le résultat précédent (ou la donnée la plus récemment transférée) était positif. Les conditions de branchement ne sont pas modifiées.

Exemple : (vider la mémoire de A034 à A03F)

```
15  34  A0              SET     R5,$A034    Initialiser le pointeur
14  3F  A0              SET     R4,$A03F    Initialiser la limite
B0              BOUCLE3 SUB     R0
55                      ST      @R5         Vider le byte mem
                                            Incrémenter R5
24                      CPR     R4          Comparer la limite
D5                      CPR     R5          au pointeur
04  FA                  BP      BOUCLE3     Boucler jusqu'à ce que soit fait
```

BM ea 05 d Branch if minus

Un branchement est effectué seulement si le résultat précédent était moins (MSB = 1). Les conditions de branchement ne sont pas modifiées.

BZ ea 06 d Branch if Zero

Un branchement est effectué seulement si le résultat précédent était zéro. Les conditions de branchement ne sont pas modifiées.

BNZ ea 07 d Branch if NonZero

Un branchement est effectué seulemen si le résultat précédent était différent de zéro. Les conditions de branchement ne sont pas modifiées.

BM1 ea 08 d Branch if Minus 1

Un branchement est effectué seulement si le résultat précédent était mois 1 ($FFFF hex.). Les conditions de branchement ne sont pas modifiées.

BRK 0A Break

Une instruction 6502 BRK est effectuée. SWEET 16 peut être réentrée sans rien détruire après avoir rétabli le pointeur de pile à la valeur qui était la sienne avant d'effectuer le BRK.

RS 0B Return from Sweet 16 subroutine

RS termine l'exécution d'une sous-routine SWEET 16 et retourne au programme appelant SWEET 16 qui poursuit son exécution (dans le mode SWEET 16). R12, qui est le pointeur de pile de retour de sous-routine dans SWEET 16, est décrémenté 2 fois. Les conditions de branchement ne sont pas modifiées.

BS ea 0C d Branch to Sweet 16 subroutine

Un branchement est réalisé à l'adresse effective (PC + 2 + d) et l'exécution se poursuit dans le mode SWEET 16. Le compteur ordinal actuel (PC) est poussé sur une pile d'adresse de retour de sous-routine SWEET 16 dont le pointeur est R12 et R12 est incrémenté de 2. La retenue est annulée et les conditions de branchement sont mises pour indiquer le contenu actuel de l'accumulateur.

Exemple : (appeler une sous-routine de déplacement de mémoire de A034-A03B en 3000-3007)

```
15  34  A0          SET     R5,$A034    Initialiser pointeur 1
14  3B  A0          SET     R4,$A03B    Initialiser limite 1
16  00  30          SET     R6,$3000    Initialiser pointeur 2
0C  15              BS      DEPLAC      Appeler la sous-routine deplac.

45          DEPLAC  LD      àR5         Déplacer un '
56                  ST      àR6         byte
?4                  LD      R4
_5                  CPR     R5          Test si c'est fait
04  FA              BP      DEPLAC
08                  RS                  Return
```

f. Théorie des opérations

Le mode exécution SWEET 16 commence par un appel sous-routine à SW 16. L'utilisateur doit s'assurer que le 6502 est en mode hexadécimal au moment de l'entrée. Tous les registres 6502 sont sauvés à ce moment pour pouvoir être rétablis quand une instruction SWEET 16 RTN rend le contrôle au 6502.

Après avoir sauvé les registres 6502, SWEET 16 initialise son comoteur ordinal (R15) avec l'adresse de retour de sous-routine issue de la pile 6502. Le compteur ordinal SWEET 16 affiche l'adresse précédant l'instruction suivante à exécuter. Après les appels sous-routines, se trouvent des instructions SWEET 16 1, 2 et 3 bytes rangées à des adresses mémoire d'ordre croissant, comme les instructions 6502. La boucle principale en SW 16B appelle de façon répétitive la routine d'"exécution d'instruction" en SW 16C qui examine le code-opération pour brancher le programme sur la sous-routine appropriée et l'exécuter.

La sous-routine SWEET 16C incrémente le compteur ordinal (R15) et va chercher le code-opération suivant qui peut être soit une opération registre de la forme OP REG compris avec OP entre 1 et 15, soit une opération non-registre de la forme 0 OP avec OP compris entre 1 et 13. Dans le cas d'une opération registre, la spécification du registre est doublée pour calculer pour les registres SWEET 16 3 bytes et placée dans le registre X pour indexation. Puis le type d'instructiion est déterminé. Les opérations registre place une spécification registre double dans le byte haut de R14 indiquant le registre résultat précédent pour les instructions de branchements ultérieures. Les opérations non registre traitent la spécification du registre (demi byte droit) comme leur code-opération, incrémentent le compteur ordinal SWEET 16 au byte de l'endroit de déplacement des instructions de branchement, chargent le registre A avec l'index du registre du précédant résultat pour tester les conditions de branchement et annulle le registre Y.

g. Quand un RTS est-il vraiment un JSR ?

Chaque type d'instruction a une sous-routine correspondante. Les points d'entrée de la sous-routine sont rangés dans une table dont l'index est directement mis à jour par le code-opération. En assignant toutes les entrées à une page commune, on ne doit ranger qu'un seul byte d'adresse par routine. Le saut indirect 6502 aurait pu être utilisé comme suit pour transférer le contrôle à la sous-routine appropriée :

```
LDA     £ADRH       Byte haut
STA     IND+1
LDA     OPTBL,X     Byte bas
STA     IND
```

Pour sauver le code, l'adresse d'entrée de sous-routine (moins 1) est poussée sur la pile, le byte haut pour commencer. Un RTS 6502 (Return from Subroutine) est utilisé pour faire sauter l'adresse hors de la pile dans le compteur ordinal du 6502 (après avoir été incrémenté de 1). Le résultat est que la sous-routine désirée est atteinte en exécutant une instruction de retour de sous-routine.

h. Les sous-routines code-opération

Les routines opération registre utilisent les modes d'adressage indexés par X et par X indirect en page zéro 6502 pour accéder aux registres spécifiés et aux données indirectes. Le résultat de la plupart des opérations registre sont à gauche dans le registre spécifié et peuvent être pressentis par des instructions de branchement ultérieurs du fait que la spécification du registre est sauvée dans le byte haut de R14. Cette spécification est changée pour renseigner R0 (Acc) pour les instructions ADD et SUB et R13 pour les instructions CPR.

L'instruction SET incrémente 2 fois le compteur ordinal, en puisant les bytes de données dans les registres spécifiés. En accord avec les conventions 6502, le byte de données bas précède le byte haut.

La plupart des opérations non-registre sont des branchements relatifs. Les sous-routines correspondantes déterminent si oui ou non le résultat précédent rencontre les conditions de branchement spécifiées et dans ce cas, met à jour le compteur ordinal SWEET 16 en ajoutant la valeur du déplacement (de -128 à +127 bytes).

L'opération RTN rétablit le contenu des registres 6502, fait sauter la pile de retour de sous-routine et fait un saut indirect dans le compteur ordinal SWEET 16. Cela transfère le contrôle au 6502 à l'instruction suivant immédiatement l'instruction RTN.

L'opération BK exécute réellement l'instruction BRK 6502, transférant le contrôle au controleur d'interruption.

N'importe quel nombre de niveaux de sous-routine peuvent être implémentés dans le code SWEET 16 via les instructions BS et RS. L'utilisateur doit initialiser, puis ne plus modifier, R12 si la capacité de la sous-routine SWEET 16 est utilisée comme elle est utilisée dans le pointeur de pile de retour de sous-routine automatique.

i. Allocation mémoire

Le seul rangement en mémoire qui doit être assigné pour les variables SWEET 16 sont les 32 adresses consécutives en page zéro pour les registres SWEET 16, 4 adresses pour sauver le contenu des registres 6502 et quelques niveaux de la pile de retour d'adresse de retour de sous-routine 6502. Si vous n'avez pas besoin de préserver le contenu des registres 6502, effacez les sous-routines SAVE et RESTORE et les appels sous-routine correspondants. Cela libèrera les 4 adresses page zéro ASAV, XSAV, YSAV et PSAV.

j. Modifications utilisateurs

Vous pouvez vouloir ajouter certaines de vos propres instructions à cette implémentation de SWEET 16. Si vous utilisez les codes-opération inutilisés $0E etD $0F, souvenez-vous que SWEET 16 les traite comme des instructions 2 bytes. Vous pouvez vouloir considérer l'instruction break comme un appel SWEET 16, sauvant 2 bytes de code chaque fois que vous passez au mode SWEET 16. Vous pouvez aussi vouloir utiliser l'opération SWEET 16 BK comme un call "caractère out" dans une interruption. Vous pouvez réaliser des sauts absolus dans SWEET 16 en chargeant l'accumulateur (R0) avec l'adresse à laquelle vous voulez sauter (moins 1) et exécuter une instruction ST R15.

G. PROGRAMMES UTILITAIRES

Le premier groupe de 3 programmes sont des utilitaires superbes du langage d'assemblage écrits par Steve Wozniak et Allen Baum et sont encore trouvés dans la ROM F$ du Basic Integer. Ils sont fournis en format code source sur cette disquette pour les utilisateurs d'Apple II plus qui n'ont pas le Basic Integer.

1. LE MINI-ASSEMBLEUR

C'est un assembleur simple mais maniable qui n'accepte pas des étiquettes ou des insertions de lignes. Chaque instruction et opérande est assemblée au moment de son entrée et sa syntaxe est également vérifiée à ce moment. Son but principal est une entrée rapide de programmes courts. Le curseur est un "!" et les instructions pour son utilisation peuvent être trouvées à la page 66 de l'"Apple II reference manual".

2. LES ROUTINES DE VIRGULE FLOTTANTE

Ce sont des routines de virgule flottante simple précision qui peuvent être interfacées à un programme en Basic ou en langage d'assemblage. Des informations à leur sujet peuvent être trouvées dans le WOZPAK.

3. ROUTINES DE MULTIPLICATION ET DE DIVISION

Ces routines visent à être utiliSées comme sous-routines dans les programmes en langage d'assemblage, fournissant un résultat de multiplication ou de division su 4 bytes. De brèves informations concernant leur utilisation peuvent être trouvées dans "The Apple II Monitor Peeled" et une démonstration de multiplication réalisée par Dave Garson est incluse sur cette disquette.

4. PRDEC

C'est une des sous-routines les plus utilisées dans le système ROM Basic Integer. Elle est appelée par presque toutes les routines qui nécessitent la sortie d'un nombre entier dans le champ 0-65535. Elle est intégrée facilement dans les programmes en langage d'assemblage. Pour l'utiliser, charger l'accumulateur avec le byte haut du nombre, chargez X avec le byte bas et appelez PRDEC. Alternativement, rangez le byte haut dans $F3 et le byte bas dans $F2 et appelez PRDEC+4.

5. MSGOUT

C'est une sous-routine pour sortir des chaînes ASCII d'un programme en langage d'assemblage. Si les pseudo-opérations BIG MAC INV ou FLS sont utilisées en connection avec cette sous-routine, le ORA£$80 doit être enlevé et tous les ASCII normaux doivent avoir le bit haut mis. Dans le même fichier source, il y a aussi 2 sous-routines simples pour lire les caractères ASCII et hexadécimaux entrés par l'utilisateur.

6. UPCON

Cet utilitaire est fourni pour les utilisateurs qui ne possèdent pas un chip d'affichage vidéo des minuscules. Il cherche les commentaires fichier source commençant soit par un "*" ou un ";" et convertit les caractères minuscules en majuscules. Chargez le fichier source avec BIG MAC, puis faites BRUN UPCON. Ce programme est intégré dans la version carte langage de BIG MAC

7. FIX

Cet utilitaire enlève les espaces excédentaires des fichiers source en mémoire.

H. DRIVER D'IMPRIMANTE GAME PADDLE

Au début du développement des Apple II, il n'y avait pas de carte d'interface d'imprimante. Plus tard, les utilisateurs d'Apple II ont éprouvé le besoin d'avoir des copies sur papier de leurs routines de développement et de ce fait, Randy Wiggington et Steve Wozniak ont écrit un driver de télétype primitif pour leurs besoins. Cela a été publié dans le fameux manuel d'instruction "livre rouge", le second pour l'Apple II. Par la suite, de nombreuses cartes interface de toutes sortes ont été développées pour l'Apple. Faire tourner une imprimante série sur le port d'entrée/sortie des paddles est une façon d'économiser l'achat d'une interface et de libérer un slot. Le driver télétype a déjà été adapté pour des imprimantes telles que Integral Data, Base 2, Heath H-14 et d'autres.

En dernier lieu, Glen Bredon a ajouté certaines améliorations au driver : il peut imprimer des listings Basic formattés sur toutes les largeurs de colonnes, il peut sortir des données avec ou sans le vidéo et le vidéo peut être laissé allumé même quand on imprime sous 40 colonnes, ce que la plupart des interfaces ne peuvent pas faire. Ces fonctions sont prises en charge par la commande Basic "POKE" sur les flags à la fin du programme.

Une information complète et les instructions peut être trouvée dans le fichier source inclus dans cette disquette. Naturellement, c'est entièrement compatible avec BIG MAC et commandé avec les commandes utilisateur BIG MAC. C'est installé quand on fait BRUN pour la première fois.

De plus, le code source est bien commenté de telle façon qu'il puisse servir comme guide quand on écrit des routines de commande pour diverses applications.

QUATRIEME PARTIE

UTILISATION DE BIG MAC

Notes et démonstrations pour des programmeurs BIG MAC débutants

A. INTRODUCTION

Le but de cette section n'est pas de fournir des explications sur la programmation en langage d'assemblage mais d'introduire BIG MAC aux programmeurs débutant dans le langage d'assemblage en général ou dans BIG MAC en particulier.

Beaucoup de commandes et de fonctions BIG MAC sont très similaires. Cette section ne vise pas à présenter la démonstration de toutes les options de commande. L'objectif est de clarifier et de présenter des exemples des opérations les plus communes, suffisantes pour acquérir une base de travail.

Note d'éclaircissement :

Dans le manuel BIG MAC, on parle des temes "mode" et "module".

Dans cette section, "module" se rapporte à un composant du programme ordinateur du système BIG MAC. Il y a 4 modules :

1. L'EXECUTIVE
2. L'EDITEUR
3. L'ASSEMBLEUR
4. Le GENERATEUR DE LA TABLE DES SYMBOLES

Chaque module est groupé sous un ou deux MODES DE CONTROLE : (1) L'EXECUTIVE, abrégé en EXEC est indiqué par le curseur "%" ou (2) L'EDITEUR indiqué par le curseur ":".

MODE DE CONTROLE D'EXECUTION	MODE DE CONTROLE D'EDITION
Module executive	Module Editeur
	Module assembleur
	Module génération table des symboles

Le terme "mode" peut être utilisé pour indiquer soit le mode de contrôle actuel (comme indiqué par le curseur) ou quand on est en mode contrôle et après une commande d'entrée, le système est dit en mode "de commande d'entrée". Par exemple, quand on tape dans un programme en sortant de la commande ADD, le système se trouve en mode ADD.

Quand on termine le mode "entrée commande", le système retourne au mode contrôle.

B. INPUT

Les programmeurs familiarisés avec un langage d'assemblage et des langages de haut-niveau verront la nécessité de formatter les entrées comme les étiquetttes, les codes-opération, les opérandes et les commentaires qui doivent être tapés dans des champs spécifiques pour être reconnus par l'assembleur.

Dans BIG MAC, l'opérateur TABS fournit une possibilité de formattage semi-automatique.

Quand on entre un programme, il faut se rappeler que chaque espace dans le code source provoque une tabulation un champ de tabulation suivant. A titre de démonstration, entrons la petite routine suivante:

à partir du tout début :

1. BRUN BIG MAC

2. Quand le curseur "%" apparaît au sommet du menu du mode EXEC, tapez "E" ou "Z". Cela vous placera automatiquement dans le mode contrôle éditeur. Si on est entré par "Z", les arrêts de tabulation seront mis à zéro.

3. Comme nous allons entrer un nouveau programme, utilisez la commande suivante après que le curseur ":" soit apparu : tapez "A" et pressez RETURN. Un "1" apparaît une ligne plus bas sur la droite et le curseur est automatiquement tabulé à un espace sur la droite du numéro de ligne. Le "1" et tous les numéros de ligne suivants qui apparaissent après un RETURN servent grossièrement au même usage que les numéros de ligne en BASIC, sauf que, dans le code source d'assemblage, les numéros de ligne ne sont pas référencés pour des sauts à des sous-routines ou comme dans les instructions GOTO.

4. A la ligne 1, entrez un "*". Une astérisque comme premier caractère dans n'importe quelle ligne est similaire à l'instruction REM en Basic; elle signale à l'assembleur que c'est une ligne de commentaires et que tout ce qui se trouve derrière l'astérisque doit être ignoré. Pour confirmer cela, tapez "PROGRAMME DE DEMONSTRATION NO 1" et tapez RETURN.

5. Après RETURN, le curseur saute à nouveau une ligne, un "2" apparaît et le curseur saute un espace.

6. Frappez une fois la barre d'espacement et tapez "OBJ", à nouveau un espace, tapez "$300" et frappez RETURN. Notez que dans la plupart des cas, "OBJ" n'est ni requis ni désirable.

7. A la ligne 3, tapez la même séquence : espace, tapez "ORG", espace, tapez "$300", RETURN.

8. A la ligne 4, ne frappez pas d'espace après le numéro de ligne. Tapez "CLOCHE", espace, "EQU", espace, "$FBDD", RETURN.

9. Ligne 5 : tapez "DEPART", espace, "JSR", espace, "CLOCHE", espace,";", "FAIRE SONNER CLOCHE", RETURN.

10. Ligne 6 : "FIN", espace, "RTS", RETURN

11. Le programme est entré, mais le système est toujours en mode ADD. Pour sortir de ce mode, frappez simplement RETURN ou tapez CONTROL X, RETURN. Le curseur ":" réapparaît à gauche de l'écran, indiquant qu'on est revenu au mode contrôle.

12. L'écran doit maintenant apparaître comme suit :

```
A:
 1  *PROGRAMME DE DEMONSTRATION NO 1
 2   OBJ $300
 3   ORG $300
 4  CLOCHE EQU $FBDD
 5  DEPART JSR CLOCHE ;FAIRE SONNER CLOCHE
 6  END RTS
 7
```

13. Tapez maintenant "L", la commande pour "LIST" et RETURN

```
L
 1  *PROGRAMME DE DEMONSTRATION NO 1
 2           OBJ  $300
 3           ORG  $300
 4  CLOCHE   EQU  $FBDD
 5  DEPART   JSR  CLOCHE      FAIRE SONN
ER LA CLOCHE
 6  END      RTS
```

NOTE : tous les listings de cette section sont montrés en format 40X24 standard. Mais, si les tabulations ont été mises à zéro par la commande TABS, le "FAIRE SONNER LA CLOCHE" tiendrait sur une seule ligne.

A droite de la colonne de numéros de ligne se trouve le code source formatté. Comparez-le à la source entrée. Vous pourrez remarquer que chaque chaîne de caractères a été déplacée dans un champ spécifique. Il y a 4 champs qui, à partir de la droite sont :

Champ 1 : il est réservé aux étiquettes. CLOCHE, DEPART et FIN sont des exemples d'étiquettes.

Champ 2 : il est réservé aux mnémoniques comme les pseudo-mnémoniques BIG MAC OBJ, ORG et EQU et les codes-opération 6502 JSR et RTS.

Champ 3 : il est réservé aux opérandes comme $300, $FBDD et dans ce cas CLOCHE.

Champ 4 : il contient les commentaires.

Il doit apparaître dans cette leçon qu'il n'est pas besoin d'introduire des espaces supplémentaires dans le fichier source dans le but de formatter.

En partant de la colonne des numéros de lignes :

Pas d'espace pour une étiquette

Un espace après une étiquette (ou, s'il n'y a pas d'étiquette, un espace après le numéro de ligne) pour le mnémonique.

Un espace après le mnémonique pour l'opérande.

Un espace après l'opérande pour le commmentaire. S'il n'y a pas d'opérande, entrez 2 espaces après le mnémonique. Le premier espace après le mnémonique provoquera une tabulation jusqu'au champ commentaires où un ";" est nécessaire s'il n'y a pas d'opérande.

C. COMMANDES SYSTEME ET D'ENTREE .

BIG MAC possède un éditeur puissant et complexe. Complexe dans le nombre d'opérations possibles mais très pratique et remarquablement facile à utiliser.

Expliquer en détails chaque option de commande d'édition et chaque entrée système nécessiterait un manuel séparé. Les définitions qui ont été données dans les parties précédentes devraient suffire pour expliquer ces opérations et de ce fait, les paragraphes suivants ne contiennent que quelques clarifications et des petites démonstrations de ces commandes.

Toutes les commandes système et d'entrées sont utilisées dans le mode contrôle éditeur immédiatement après le curseur ":".

CONTROL X ou RETURN comme premier caractère d'une ligne provoque une sortie du mode actuel et renvoie le système en mode contrôle quand on est en mode ajout (ADD) ou insertion (INSERT). CONTROL C sort du mode d'édition et renvoie le système au mode contrôle après avoir édité des lignes.

Les autres commandes système et d'entrée se termine en frappant simplement RETURN.

Insérer et effacer des lignes dans le code source sont des opérations simples. L'exemple suivant insère 3 nouvelles lignes entre les lignes 4 et 5 existantes :

1. Après le curseur, tapez I pour INSERT, le chiffre 5 et pressez RETURN. Les lignes insérées prendront place avant la ligne dont le numéro est spécifié dans la commande.

2. Entrez une astérisque et pressez RETURN. Notez qu'on n'est pas sorti du mode INSERT.

3. Refaites une fois le point 2.

4. Entrez un espace, tapez "TYA" et RETURN.

L'écran doit apparaître comme suit :

```
:I5
     5 *
     6 *
     7           TYA
  le mode contrôle, entrez "D5,6" et RETURN
```

2. Listez la source. Les lignes 5 et 6 de l'exemple précédent qui contenaient la ligne avec l'astérisque seule et le mnémonique TYA ont été effacées et les lignes suivantes ont été renuméro $FBDD

```
     5 *
     6 *
     7              TYA
     8 DEPART       JSR  CLOCHE    FAIRE SONN
ER LA CLOCHE
     9 FIN          RTS
```

Les nouvelles lignes (5,6 et 7) ont été insérées et les lignes suivantes du fichier source original ont été renumérotées (8 et 9).

Utiliser da commande DELETE (effacer) est également aisé :

1. Dans le mode contrôle, entrez "D5" et RETURN. Rien de nouveau n'apparaît sur l'écran.

2. Listez le code source. Le listing source est plus court d'une ligne, une des lignes où une astérisque seule apparaissait a disparu et les lignes suivantes ont été renumérotées.

Il est possible d'effacer un champ de lignes en une seule étape :

1. Dans le mode contrôle, entrez "D5,6" et RETURN

2. Listez la source. Les lignes 5 et 6 de l'exemple précédent qui contenaient la ligne avec l'astérisque seule et le mnémonique TYA ont été effacées et les lignes suivantes ont été renumérotées. Le listing est identique à celui du chapitre "INPUT" opération No 13.

Cette renumérotation automatique implique que, quand on efface des lignes, il est <u>important de commencer par les lignes de numéros les plus élevés et d'aller en décroissant</u>.

La commande d'édition possède plusieurs sous-commandes accessibles avec les caractères de contrôle. Pour montrer cela, utilisons notre routine CLOCHE :

1. Après le curseur ":", entrez "E" (la commande d'édition) et un numéro de ligne (6 pour cette démonstration) et pressez RETURN. Une ligne plus bas, à droite, la ligne mentionnée apparaît formattée :

```
6 FIN    RTS
```

   et le curseur se trouve sur le F de FIN.

2. Tapez CONTROL D et le caractère qui se trouvait sous le curseur disparait. Tapez à nouveau CONTROL D, puis une troisième fois. "FIN" a été effacé et le curseur est positionné à gauche du mnémonique.

3. Pressez RETURN et lister le programme. A la ligne 6 du code source, il ne reste que le numéro de ligne et le mnémonique.

4. Refaites la commande du point 2.

5. Cette fois, tapez CONTROL I. Ne déplacez pas le curseur avec la barre d'espacement ou les flèches. Tapez le mot "FIN" et RETURN.

6. Lister le programme. La ligne 6 a été rétablie

Si vous éditez une seule ligne, le fait de presser RETURN vous renvoie au mode contrôle. Dans le point 1 ci-dessus, si vous aviez spécifié un champ de lignes (ex. : E3,6) après la commande EDIT, le fait de presser RETURN aurait édité la ligne suivante du champ. Au fur et à mesure que les lignes apparaissent, vous pouvez ou les modifier à l'aide des sous-commandes d'édition, ou passer à la ligne suivante en pressant RETURN, ou encore sortir du mode édition en utilisant CONTROL C. Notez que le fait de presser RETURN enregistre la ligne où vous vous trouviez sous la forme qui apparaît à l'écran.

Les autres sous-commandes accessibles par les caractères de contrôle dans le mode d'édition sont d'un usage semblable. Lisez la définition de ces fonctions dans la deuxième partie et essayez-en quelques unes.

ASSEMBLAGE

L'étape suivante dans l'utilisation de BIG MAC est l'assemblage du code source en un code objet.

Après le curseur ":", tapez la commande système du module d'édition ASM et pressez RETURN. Sur votre écran, la ligne suivante apparaitra :

```
UPDATE SOURCE Y/N ?
(mettre la source à jour oui/non ?)
```

Tapez "N" et pressez RETURN et vous verrez le listing d'assemblage tel qu'il apparait à la page 67 de l'édition anglaise.

Si, lors de l'assemblage un message d'erreur apparaît, notez le numéro indiqué dans le message jusqu'à ce que le message "... BYTES ..." apparaisse. Référez-vous alors au chapitre "INPUT" et comparez le listing avec celui du point 13. Regardez spécialement les éléments qui pourraient se trouver dans des champs incorrects.

Si tout s'est bien passé, à droite de la colonne des numéros dans la moitié inférieure de l'écran, se trouve le code source.

A gauche de la colonne des numéros, commençant en ligne 5, il y a une série de caractères numériques et alphabétiques. C'est le code objet, c'est-à-dire les mnémoniques et les opérandes assemblés en leur traduction hexadécimale langage machine.

Notez que l'étiquette DEPART n'est pas assemblée en un code objet. Il en est de même pour les commentaires, les remarques et les pseudo-mnémoniques OBJ et ORrce
système vous demandera un nom de fichier. Tapez "DEMO1" RETURN. Après que le programme ait été sauvé, le cont pas d'utilité pour l'ordinateur et de ce fait, ils ne sont pas traduits en langage machine.

Les 2 bytes suivants (chaque paire de caractères constitue un byte) à la ligne 5 présentent une curieuse ressemblance avec le dernier groupe de caractères de la ligne 4. Regardez : à la ligne 4 du programme source, nous avons dit à l'assembleur que l'étiquette CLOCHE était égale (EQU) à l'adresse $FBDD. A la ligne 5, quand l'assembleur rencontre CLOCHE comme opérande, il lui substitue l'adresse spécifiée. La séquence des bytes haut et bas a été inversée, ce qui est une convention du 6502.

Le reste des informations se comprend de lui-même. Le nombre total d'erreurs rencontrées dans le fichier source est de zéro et quatre bytes de code objet ont été générés (comptez les bytes suivant les adresses).

E. SAUVER ET FAIRE TOURNER DES PROGRAMMES

L'étape finale dans l'utilisation de BIG MAC est de faire tourner le programme. Avant cela, il serait bon de sauver le programme source.

1. Retourner au mode contrôle si nécessaire et tapez "Q" RETURN. Le système quitte le mode éditeur et revient au mode EXEC. Si la disquette BIG MAC se trouve encore dans le drive, retirez-la et insérez une disquette de travail initialisée.

   Après que le curseur "%" soit apparu, tapez S pour sauver le code source. Le système vous demandera un nom de fichier. Tapez "DEMO1" RETURN. Après que le programme ait été sauvé, le curseur "%" réapparait.

2. Tapez "C" (pour CATALOG) et regardez le catalogue du disque. Le code source a été sauvé comme un fichier binaire appelé "DEMO1.S". Le suffixe .S est une convention d'étiquettage de fichier qui indique que ce fichier est un code source. Ce suffixe est ajouté automatiquement au nom du fichier par la commande "S" qui sert à sauver le fichier source.

3. Pressez n'importe quelle touche pour revenir au mode EXEC et entrez "O" pour sauver le code objet. Le fichier objet est sauvé sous le même nom que celui spécifié précédemment pour le fichier source. Il n'y a aucun danger d'interférence car aucun suffixe n'est ajouté au nom du fichier objet.

Avant de pouvoir sauver le code objet, il faut que l'assemblage ait été réalisé sans erreur. En écrivant les fichiers sur le disque, BIG MAC affiche également le paramètre des adresses et calcule et affiche la longueur de ce paramètre. C'est une bonne habitude d'en prendre note.

Retournez en mode éditeur et tapez "MON" RETURN, ce qui vous envoie dans le moniteur (curseur : "*"). Entrez "300G" RETURN. On entend un son de cloche. Cela est réalisé par notre programme. Donc, il fonctionne.

Vous pouvez à présent retourner au mode EXEC en tapant CONTROL Y RETURN.

CINQUIEME PARTIE

SUPPLEMENT CARTE LANGAGE

CARACTERISTIQUES SPECIALES ET DIFFERENCES PAR RAPPORT A LA VERSION STANDARD

A. MODE EXEC

R(ead)

Cette commande permet de lire des fichiers texte dans BIG MAC. Elle travaille comme c'est décrit dans "READER" sauf que c'est chaque fois un "APPEND". Donc, tapez "NEW" dans l'éditeur si vous ne voulez pas un "APPEND". S'il n'y a pas de fichier en mémoire, le nom donné deviendra le nom du fichier par défaut. Un "READ" avec "APPEND" ne le fait pas. Si le fichier contient des lignes de plus de 255 caractères, elles seront divisées en 2 ou plus par la routine "READ". Le fichier sera lu jusqu'à ce qu'il atteigne HIMEM et un message d'erreur sera produit s'il va plus loin.

W(rite)

Cette commande écrit un fichier BIG MAC dans un fichier texte au lieu de l'écrire dans un fichier binaire. Elle travaille comme c'est écrit dans "WRITER". La vitesse d'exécution des routines "READ" et "WRITE" est approximativement la même que celle des commandes "BLOAD" et "BSAVE". La routine "WRITE" exécute un "VERIFY" après l'écriture. Si vous entrez un espace ou n'importe quel caractère plus petit que "à" (moins de $C0 dans le code ASCII) avant le nom du fichier, ce caractère et le préfixe ".T" ne seront pas envoyés au DOS par les commandes "READ" et "WRITE".

CONTROL C

Tapée après "COMMAND:" de la commande C (pour Catalog), cette commande envoie le curseur "%" du mode EXEC et accepte n'importe quelle commande du mode EXEC comme par exemple un "LOAD". Elle permet d'effectuer une commande EXEC quand le catalogue est affiché à l'écran. Cette caractéristique existe également dans la version standard de BIG MAC mais a été ajoutée trop tard que pour pouvoir apparaître dans la documentation. De plus, si CONTROL C est tapé à un endroit de pause du catalogue, l'impression du reste du catalogue est supprimée.

Les fonctions de contrôle du mode édition peuvent être utilisées à la fois dans le mode INSERT et ADD, sauf que CONTROL R n'a aucun effet, et la sortie du mode ADD travaille comme on l'a décrit. Par exemple, le fait de presser RETURN accepte la ligne telle qu'elle apparait à l'écran, et CONTROL Q tronque la ligne à la position du curseur.

Pendant une édition, si une ligne devient plus grande que 255 caractères, elle sera tronquée et une nouvelle ligne blanche sera créée derrière.

CONTROL L travaille dans tous les modes comme c'est décrit dans le mode ADD, c'est-à dire un verrouillage de clavier. Pour faire passer un mot de majuscules en minuscules (ou le contraire), vous pouvez taper CONTROL L puis passer sur le mot à l'aide de la flèche droite.

Conversion hexadécimale-décimale

En mode commande, si vous tapez un nombre décimal positif ou négatif, vous obtenez la conversion hexadécimale. Si vous tapez un nombre hexadécimal précédé de "$", vous obtenez la conversion décimale. Toutes les commandes acceptent les nombres hexadécimaux, ce qui est principalement utile pour les commandes "HIMEM" et "SYM".

Entrée de caractères supplémentaires

CONTROL O, en plus de permettre d'entrer des caractères de contrôle, permet aussi d'entrer des caractères qui ne sont normalement pas disponibles sur le clavier APPLE. Pour cela, tapez CONTROL O suivi du caractère de la première colonne de la table de conversion suivante pour obtenir le caractère correspondant de la deuxième colonne :

Tapez	pour obtenir
<	CTRL -
>	CTRL
K	
L	
M	
N	
O	
k	
l	
m	
n	
o	

Notez que si vous utilisez une modification "shift", alors, suivant ce que vous avez, "shift M" peut donner la lettre M majuscule et vous devrez utiliser "CONTROL O"M pour obtenir le bon crochet.

STRIP

Cette commande efface tous les commentaires (lignes commençant par une "*" ou suivant un ";") du fichier source. Elle est utilisée comme dernier recours pour réduire des très gros fichiers, mais avec la grande souplesse de version carte langage de BIG MAC, vous ne devrez jamais l'utiliser.

UPPER

Cette commande convertit (de façon permanente) les textes de commentaires de minuscules en majuscules. (Voyez "UPCON"). Comme avec la commande "STRIP", les caractéristiques différentes de la version carte langage ne laissent que peu d'utilité à cette commande, sauf peut-être pour convertir des fichiers pour quelqu'un qui n'a pas la version carte langage et n'a pas les minuscules.

TEXT

Cette commande convertit tous les espaces d'un fichier source en espaces inverses. Le but de cette commnande est d'être utilisée pour des fichiers texte de telle façon que vous n'ayez pas à vous souvenir de mettre les tabulations à zéro avant d'imprimer un tel fichier. Cette conversion n'a d'effets que sur la tabulation.

FIX

Cette commande annule l'effet de TEXT. C'est aussi un substitut au programme FIX (qui ne peut pas être utilisé dans la version carte langage). Donc, il est recommandé d'utiliser la commande FIX sur tous les fichiers source externes, après quoi, le fichier doit être sauvé. Cette commande n'est pas tout à fait la même que le programme FIX de la version standard dans le sens qu'elle en fait plus, mais ses effets sont essentiellement les mêmes. (Notez que les routines TEXT et FIX sont écrites en SWEET 16 et de ce fait sont relativement lentes. Plusieurs minutes peuvent être nécessaires pour l'exécution de fichiers importants.). FIX ou une édition tronque toutes les lignes de plus de 255 caractères.

SYM

La version carte langage place la table des symboles dans la carte langage elle-même (au rang 1 de $D000-$DFFF). Cet espace est suffisant pour des fichiers importants. Dans le cas où cet espace est quand même rempli, la commande SYM vous donne la possibilité de dire à l'assembleur de continuer la table des symboles à un autre endroit. Si, par exemple, vous tapez SYM $9000, et assemblez le programme, si la table des symboles utilise tout l'espace disponible normal, elle continuera de $9000 au HIMEM du Basic. Vous devez vous souvenir que la commande SYM est annulée par une commande HIMEM ou par une sortie en mode EXEC et une rentrée; donc, établissez HIMEM avant d'établir une adresse SYM. L'adresse SYM doit être au-dessus de HIMEM et en-dessous de HIMEM BASIC. Si la table des symboles dépasse l'espace alloué, vous obtiendrez un message d'erreur au premier passage de l'assembleur.

VIDeo

Cette commande est utilisée pour sélectionner ou "désélectionner" la carte 80 colonnes. La condition par défaut peut être sélectionnée en utilisant le programme de configuration. Ceci est similaire à l'utilisation de PR£ en Basic. N'utilisez pas la commande PR£ pour sélectionner la carte 80 colonnes dans le slot 3 (par exemple); ceci peut être obtenu en tapant "VIDEO 3" depuis l'éditeur. Cette commande est annulée par un "VIDEO 0" ou "VIDEO $10" qui peut être suivi d'un RESET. Les 2 dernières commandes sélectionnent l'écran standard Apple, mais "VIDEO 0" convertira les minuscules en majuscules à l'écran, sauf les minuscules du fichier source qui seront converties en majuscules clignotantes; (la sortie vers l'imprimante n'est jamais convertie). Donc, si vous avez un adapteur minuscules, vous utiliserez la commande VIDEO $10 (ou VIDEO 16) à la place de VIDEO 0 quand vous sélectionnerez l'écran Apple. Si votre carte 80 colonnes accepte un switch écran software par la touche ESCAPE, vous pouvez passer en 40 colonnes en utilisant cette commande dans l'éditeur (ou PR£0 RESET). Cette commande peut devoir être suivie d'un CONTROL X. Ce switch software fait un "VIDEO $10" et de ce fait doit être suivi d'un "VIDEO 0" si vous n'avez pas un adaptateur minuscules.

Pseudo-mnémoniques

TR

Cette commande a le même effet dans l'assembleur et fait la commande éditeur TR. Donc, TR ou TR ON limite l'impression du code objet à 3 bytes par ligne et TR OFF remet l'impression du code objet en entier.

CHK

Cette commande place un byte checksum (test) dans le code objet à l'adresse spécifiée par le mnémonique CHK (habituellement à la fin du programme).

PUT

PUT NOM DU FICHIER (les paramètres drive et slot sont acceptés dans la syntaxe standard du DOS) provoquera la lecture du fichier dont le nom est spécifié (avec le préfixe T. joint) et "l'insérera" à l'adresse spécifiée par le mnémonique. (Note : "insérer" se réfère à l'effet de l'assemblage sur la source. Le fichier lui-même est placé juste après le programme source principal.) Des fichiers texte sont requis pour cette facilité afin que la protection de la mémoire soit assurée. Vous obtiendrez une erreur mémoire si un fichier PUT va au-delà de HIMEM. On ne trouve qu'un seul de ces fichiers à la fois en mémoire et de ce fait, un programme très important peut être assemblé en utilisant la facilité PUT. Il existe 2 restrictions à un fichier PUT. Il ne peut pas y avoir de définitions de MACROS dans un tel fichier (elles doivent se trouver dans le programme source principal) et un fichier PUT ne peut pas en appeler un autre (c'est-à-dire pas de chaînage). Evidemment, un chaînage peut être simulé en ne mettant dans le programme principal que les définitions de MACROS et en appelant à tour de rôle les autres programmes par le mnémonique PUT. N'importe quelle variable peut être utilisée comme variable locale. Les variables locales usuelles 1 à 8 peuvent être utilisées à cet effet en utilisant le mnémonique VAR.

La facilité PUT fournit un moyen simple d'incorporer des sous-routines souvent utilisées, comme MSGOUT ou PRDEC, dans un programme.

VAR

Cette commande est juste une facilité pour mettre les variables 1 à 8 en équation. "VAR 3;$42;ETIQUETTE" mettra les variables 1 = 3, 2 = $42, 3 = ETIQUETTE. C'est tout désigné pour utiliser juste avant un PUT. Si un fichier PUT utilise les variable 1 à 8 (sauf dans les lignes >>> pour appeler les macros), il doit y avoir une déclaration préalable de ces variables.

KBD

Cette commande autorise l'équation d'une étiquette à partir du clavier pendant l'assemblage. Sa syntaxe est : ETIQUETTE KBD.

SAV NOM DE FICHIER (les paramètres slot et drive sont acceptés) sauve le code objet courant sous le nom spécifié. C'est exactement la même commande que dans le mode EXEC, mais elle peut être réalisée plusieurs fois pendant l'assemblage.

Ce pseudo-mnémonique vous donne le moyen de sauver des portions d'un programme qui a plus d'un ORG. Elle permet également l'assemblage de fichiers extrèmement grands. Après un sauvetage, l'adresse objet est remise à la dernière spécification de OBJ ou à HIMEM par défaut.

La commande SAV met l'adresse du fichier sauvé à son valeur correcte. Par exemple, si votre programme contient 3 commandes SAV, il sera sauvé en 3 parties. Quand, plus tard, vous ferez un BLOAD, il se placeront à l'adresse correcte, le troisième après le deuxième et le deuxième après le premier.

Utilisés ensemble, les mnémoniques PUT et SAV rendent possible l'assemblage de fichiers extrèmement grands.

Une incompatibilité mineure

Dans la version carte langage, les opérandes multiples du mnémonique PMC (ou >>>) doivent être séparées par un point virgule à la place d'un virgule comme on le fait dans la version standard. Le programme CONVERT permet de convertir les vieux fichiers source en leur nouvelle version (avec votre programme source en mémoire, tapez simplement BRUN CONVERT après la commande EXEC C(atalog). Cette modification a été apportée parce que les MACROS acceptent désormais les données littérales. Donc, l'assembleur accepte désormais le type suivant d'appel macro :

```
        DO 0
MUV     MAC
        LDA  1
        STA .2
        <<<
        FIN
        >>> MUV.(PNTR),YDEST
        >>> MUV.£3FLAG,X
```

Il acceptera également :

```
        DO 0
PRINT   MAC
        JSR SENDMSG
        ASC  1
        BRK
        <<<
        FIN
        >>> PRINT.!"GUILLEMETS"!
        >>> PRINT.'CECI EST UN EXEMPLE'
        >>> PRINT."COMPRENEZ-VOUS ?"
```

RESTRICTION : si de telles chaînes contiennent des espaces ou des points virgules, elles doivent être délimitées par des guillemets simples ou doubles. Les expressions littérales comme >>>WHAT."A" doivent avoir le délimiteur final (c'est seulement vrai dans les appels macros ou les commandes VAR, mais c'est mieux de s'habituer à l'utiliser dans tous les cas.

Macros emboîtées

Dans la version carte langage, des macros peuvent être emboîtées sur une profondeur de 15. Pour être emboîtées, les macros doivent être définies avec la condition DO sur OFF.

Voici un exemple de macros emboîtées dans lequel la définition elle-même est emboîtée (cela ne peut être réalisé que quand la fin de la définition des 2 macros se trouve au même endroit).

```
TRDB  MAC
      >>> TR. 1+1 2+1
 TR   MAC
      LDA  1
      STA  2
      <<<
```

Dans cet exemple, >>> TR.LOC;DEST assemblera en :

```
      LDA LOC
      STA DEST
```

et >>> TRDB.LOC;DEST assemblera en :

```
      LDA LOC+
      STA DEST+1
      LDA LOC
      STA DEST
```

Une forme plus commune d'emboîtement est illustrée par la définition de ces 2 macros (où CH = $24) :

```
POKE  MAC
      LDA £ 2
      STA  1
      <<<

HTAB  MAC
      >>> POKE.CH 1
      <<<
```

Adressage en page zéro

Les mnémoniques qui peuvent accepter soit les opérandes page zéro soit les non page zéro comme LDA ADRS peuvent maintenant être forcés à utiliser la forme non-zéro page en ajoutant simplement n'importe quel caractère (les 2 point par exemple) autre qu'un "D" au mnémonique. Donc, LDA: ADRS assemblera en non-zéro page. C'est parfois utile.

ORG multiples

Désormais, des ORG multiples ne dérangent plus le code objet quand il est sauvé; le programme sera chargé lors d'un LOAD à l'adresse utilisée par le premier ORG. Donc, par exemple, si vous désirez un programme qui se charge en $1000 mais est écrit pour tourner en $9000, vous pouvez commencer par :

```
ORG $1000
ORG $9000
```

et il le fera. C'est utile pour des programmes qui doivent "faire le ménage" avant d'être déplacés à leur adresse finale et il existe d'autres usages à cette possibilité.

D. REMARQUES GENERALES

Rentrer après être sorti vers le Basic est réalisé par la commande "ASSEM". Un BRUN BIG MAC ou un boot de la disquette effectue également une entrée à chaud et ne recharge pas BIG MAC s'il est déjà en mémoire. Cela peut être fait par un BRUN BOOT ASM qui constitue une entrée à froid, détruisant les fichiers se trouvant en mémoire.

Après une sortie vers le moniteur par la commande éditeur MON, la rentrée peut s'effectuer par CONTROL Y, CONTROL C ou CONTROL B (les deux derniers du fait qu'Apple croit que BIG MAC est du Basic). Une rentrée directe dans l'éditeur est possible en tapant 0G RETURN. Cette rentrée, à la différence des autres, utilise les pointeurs page zéro à la place de ceux sauvés lors de la sortie, donc, vous devez vous assurez qu'ils n'ont pas été altérés.

L'organisation de la mémoire est un peu différente dans la version carte langage. Bien que pour des fichiers de taille normale, ça ne concerne pas l'utilisateur, il est important de connaître les contraintes de manipulation de fichiers importants. HIMEM (dont la valeur par défaut est de $8000) est une limite supérieure pour le fichier source. C'est également une limite supérieure pour les fichiers PUT. Si, au cours de l'assemblage, une erreur mémoire apparait indiquant une ligne PUT, cela signifie que le fichier PUT dépasse HIMEM et que vous devez augmenter HIMEM. Les adresses par défaut de ORG et OBJ mette la valeur actuelle de HIMEM en équation (à la place de $8000 dans la version standard). C'est illégal, dans la version carte langage, de spécifier une adresse OBJ plus petite que HIMEM, sauf en page 3 (si vous utilisez une adresse OBJ en page 3, vous devez faire attention que le fichier n'écrive pas sur les JUMP DOS en $30D-$3FF car l'assembleur ne le vérifie pas pour vous). Si, pendant l'assemblage, le code objet dépasse le HIMEM BASIC (ou l'adresse SYM si une telle adresse a été spécifiée), le code ne sera pas écrit en mémoire, mais l'assemblage continuera et ses résultats seront envoyés à l'écran ou à l'imprimante. La seule indication vous permettant de savoir que c'est arrivé, si ce n'est pas intentionnel de votre part, est que, dans ce cas, la commande de sauvetage du code objet est inutilisable. Donc, si vous désirez un listing pour un très grand fichier sans vraiment créer le code, vous pouvez assembler sur le DOS et plus haut.

La source est placée en $901 (partiellement parce qu'un boot disquette écrit en page 8). L'éditeur et l'assembleur écrivent également en page 8 pour différents usages.

Lors d'une sortie vers le Basic ou vers le moniteur, les pointeurs en $A-$F sont sauvés en $E00A-$E00F.

E. UTILISATION AVEC LES MODIFICATIONS SHIFT

La version carte langage admet toutes les modifications SHIFT hardware. Le programme de configuration établit les modifications que vous voulez apporter. BIG MAC est assez habile pour savoir si la modification que vous apportez existe réellement sur l'Apple que vous utilisez et annule les modifications si elles ne sont pas correctes. Donc, il peut être utilisé sur une autre machine sans reconfiguration.

F. UTILISATION AVEC LA CARTE 80 COLONNES

La plupart, mais pas toutes les cartes 80 colonnes peuvent être acceptées par la version carte langage. Vous pouvez utiliser la commande VIDEO pour sélectionner la carte 80 colonnes. Pour sélectionner cette carte lors du boot, utilisez le programme de configuration. (Vous pouvez alors repasser en 40 colonnes en agissant comme c'est décrit à la commande VIDEO).

Si votre écran n'accepte pas les inverses, les caractères CONTROL de la source seront affichés en lettres capitales ordinaires à la place de lettres inverses comme c'est le cas avec les écrans qui l'acceptent. Vous pouvez utiliser la commande éditeur FIND pour rechercher des caractères CONTROL particuliers pour vérifier s'ils sont présents ou non ou simplement passer en écran Apple normal.

Si, par exemple, votre copie de BIG MAC a été configurée pour accepter une carte 80 colonnes dans le slot 3 et qu'il n'y a pas de carte dans ce slot, BIG MAC s'en rendra compte et ne cherchera pas à passer en 80 colonnes. Il n'y a donc pas besoin de le reconfigurer pour l'utiliser sur un autre ordinateur.

Quand il se trouve dans l'éditeur, BIG MAC prend pratiquement le contrôle total des entrées et des sorties. De ce fait, un caractère CONTROL frappé au clavier aura pour effet celui décrit dans ce manuel et non pas celui décrit dans le manuel de la carte 80 colonnes. Par exemple, CONTROL L ne nettoie pas l'écran mais est le verrouilleur de clavier. CONTROL A, qui agit comme un verrouilleur de clavier sur beaucoup de cartes 80 colonnes, ne le fera pas dans l'éditeur BIG MAC mais produira simplement un CONTROL A dans la ligne du fichier.

G. PROGRAMME DE CONFIGURATION ASM

Ce programme vous permet de faire plusieurs modifications mineures aux conditions par défaut de BIG MAC. Il vous permet d'abord de changer le caractère de mise à jour de la source (UPDATE SOURCE) cherché pour une entrée dans l'assembleur, le caractère "wild card" de l'éditeur et le nombre de champs de symboles imprimés dans le listing de la table des symboles. Il permet également de spécifier si vous désirez pouvoir accepter une carte 80 colonnes et dans ce cas, dans quel slot elle doit se trouver.

Il vous permet de spécifier une modification hardware de SHIFT. Toutes les modifications sont acceptées à ce niveau. Toutefois, si votre modification est du type de celles qui permettent une entrée directe des minuscules (comme avec le rehausseur de clavier VIDEX), à la place de fournir une adresse mémoire à tester (comme avec la modification "game paddle 2"), alors, la valeur par défaut au début de chaque ligne sera une minuscule plutôt qu'une majuscule et CONTROL L agira comme un verrouilleur de clavier.

Vous pouvez spécifier si vous avez un adaptateur minuscules. Cela affectera la condition au moment du boot si vous n'avez pas décidé d'avoir une carte 80 colonnes sélectionnée à ce moment. (Cela peut toujours être annulé depuis l'éditeur en utilisant la commande VIDEO, donc c'est seulement pour sélectionner les conditions de départ.)

Finalement, vous pouvez sauver la version configurée sur la même disquette ou sur une autre. Il n'y a pas de raison de garder la version originale comme vous pouvez, de toute façon, toujours y revenir en le reconfigurant.

A la fin du programme de configuration, il vous est donné la possibilité de transférer le programme de boot BIG MAC sur une autre disquette. C'est juste une facilité de transfert de programme. Vous pouvez aussi utiliser FID pour copier BIG MAC ou ASM.OBJ (le programme principal). N'essayez pas de faire BSAVE ou BLOAD BIG MAC depuis le clavier; ça ne marche pas! Quand il a été booté avec le DOS normal, COPYA ou n'importe quel autre programme de copie standard peut être utilisé pour copier la totalité de la disquette.

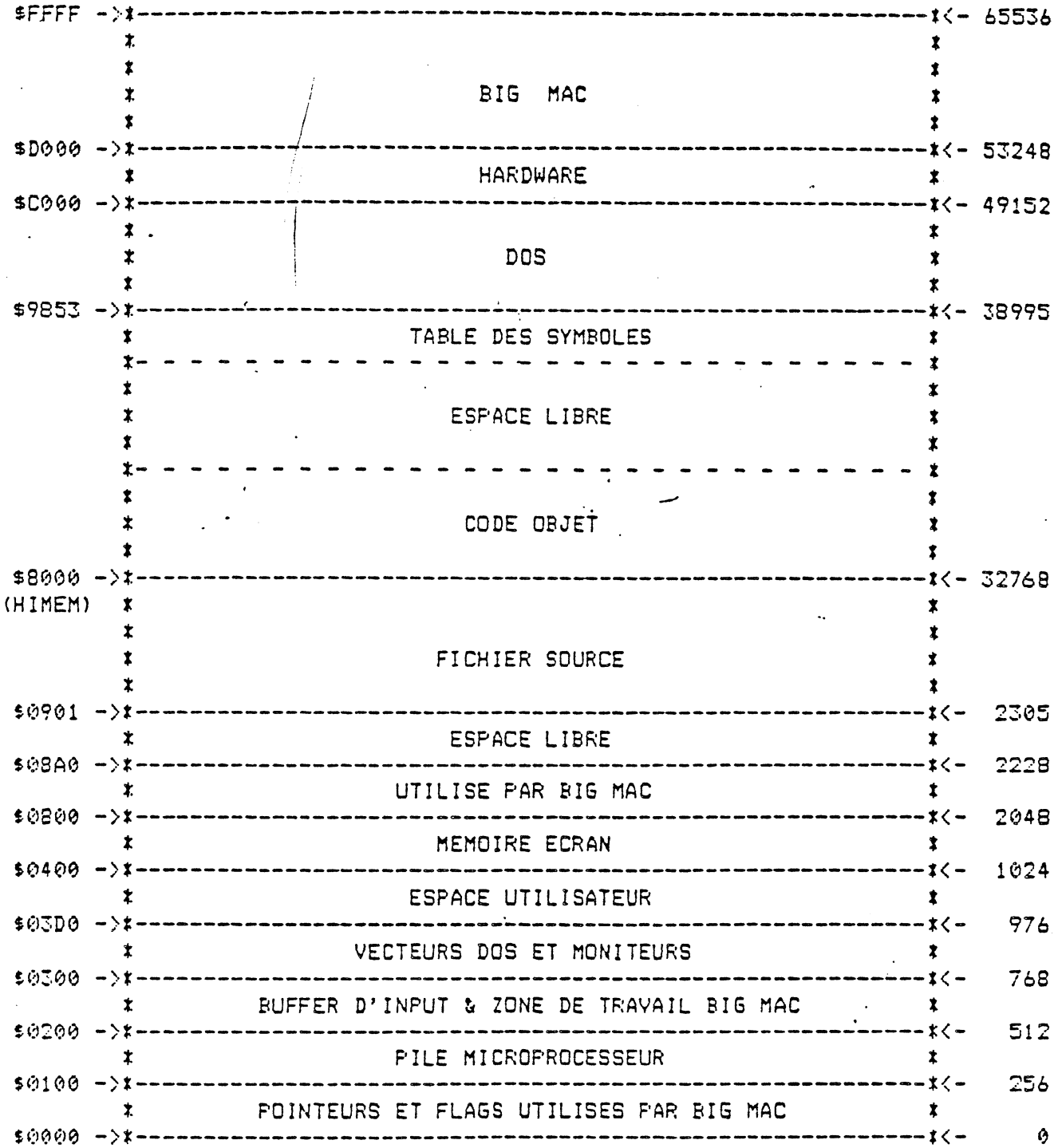
Le programme de configuration ne doit être lancé que quand l'écran Apple standard est utilisé.

H. CONSIDERATIONS SPECIALES

Le petit programme appelé BIG MAC sur la disquette peut être lancé par un BRUN. Il est toutefois désigné pour être un programme de boot. Dans cette optique, le DOS sur la disquette est modifié pour faire tourner un programme binaire lors d'un boot. Pour mettre une copie de ce DOS sur une autre disquette, vous devez faire ce qui suit : RENAME BIG MAC,XXX, rebooter la disquette, insérer une disquette vierge et taper INIT BIG MAC, puis DELETE BIG MAC et finalement RENAME XXX,BIG MAC sur la disquette originale. Vous devez alors utiliser le programme de configuration ou FID pour transférer les fichiers BIG MAC et ASM.OBJ sur la nouvelle disquette.

Le programme de boot BIG MAC fait plusieurs choses. D'abord, il vérifie s'il y a une carte langage. Ensuite, il effectue quelques changements mineurs au DOS qui seront requis par le programme principal (en particulier, la commande INIT sera inhibée) pui, il vérifie si ASM.OBJ est déjà là. Dans le cas contraire, ASM.OBJ est chargé dans la carte et il y saute. Le résultat est qu'un boot est toujours un démarrage à chaud, c'est-à-dire qu'il n'efface pas le fichier source se trouvant en mémoire (pour cette raison, il est impératif que vous ne convertissiez pas la disquette en disquette master; laissez-la telle qu'elle est). Si, pour une raison ou une autre, vous voulez forcer la carte langage à être rechargée, vous devez arrêter l'ordinateur ou faire BRUN BOOT ASM. Le dernier programme est identique à BIG MAC sauf qu'il charge toujours ASM.OBJ et est désormais un départ à froid qui efface n'importe quel fichier source.

I. CARTE DE LA MEMOIRE



J. RENVOI DES SYMBOLES (par Dale Waddel)

Une caractéristique qui manque à BIG MAC est la possibilité d'imprimer un listing de renvoi à une étiquette. BIG MAC a la possibilité d'imprimer la table des symboles par ordre alphabétique et numérique. Les tables des symboles sont utiles pour déterminer si une étiquette existe dans le programme ou si une étiquette a été référencée au moins une fois. Si vous devez déterminer comment ou où une étiquette est utilisée, la seule possibilité est d'utiliser la fonction FIND dans l'éditeur.

Normalement, je place un "LST OFF" à la fin du programme et de ce fait, la table des symboles n'est pas imprimée. Je trouve que le listing de renvoi fournit tout dans la table des symboles, sauf la location numérique de chaque étiquette. Pour ceux d'entre vous qui utilisent les macros, la table des symboles BIG MAC montre le nom des macros et les étiquettes définies dans une macro. Mon système de renvoi traite les noms de macros et de variables comme si elles étaient des étiquettes.

Pour faire tourner le programme de renvoi, suivez simplement les étapes suivantes :

- assurez-vous que BIG MAC est actif
- assurez-vous que le fichier source est en mémoire
- si un fichier objet est en mémoire, assurez-vous qu'il a été sauvé avant de faire tourner le programme de renvoi. Ce programme utilise toute la mémoire entre le fichier source et le DOS.
- allumez l'imprimante
- passez en mode EXEC et pressez C pour "Catalog"
- après affichage du catalogue, faites BRUN BIG MAC.XREF
- après que BIG MAC.XREF ait collecté toutes les informations à imprimer, il demande à l'utilisateur de répondre au message suivant :

RESUME DES COMMANDES

Commande	Description	Page
	Mode EXEC	
C CATALOG	Affiche catalogue; permet commandes DOS	3
L LOAD	Charge un fichier source de la disquette	3
R RD	Lit un fichier texte de la disquette	66
S SAVE	Sauve un fichier source sur la disquette	3
W WRITE	Ecrit un fichier texte sur la disquette	66
A APPEND	Charge fichier source après fichier en mém.	3
D DRIVE	Passe du drive 1 au drive 2	4
E EDIT	Entre dans le mode EDITEUR/ASSEMBLEUR	4
Z ZERO TABS	Idem ci-dessus avec tabulations a zéro	4
O OBJ SAVE	Sauve code objet si assemblage réussi	4
Q QUIT	Sort vers le BASIC	4
	EDITEUR	
	Mode Commande	
HImem	Etablit limite supérieur fichier source	5
NEW	Efface source présenté; remet HIMEM	5
PRE	Mêmes fonctions qu'en BASIC	5
USER	Exécute routine utilisateur en $3F5	5
TABS	Met arrêts tab. pour listing édition	6
LENgth	Donne nombre bytes fichier source	6
N	Donne adresse No ligne spécifié	6
Monitor	Sort vers Moniteur. Retour par CTRL-Y	6
TRunc ON	Omet commentaires préfixés	6
TRunc OFF	Remet flag troncation à valeur par défaut	7
STRIP	Efface commentaires du fichier source	68
Quit	Sort vers le mode EXEC	7
SYM	Etablit aire table symboles utilisateur	69
ASM	Commence l'assemblage	7
Delete	Efface No ligne, champ, liste de champs	7
Replace	Comme ci-dessus, mais passe en mode insert	7
List	Liste fichier source avec No lignes	8
Print	Liste fichier source sans No lignes	8
/	Continue lister depuis dernier No ligne	8
Find	Trouve chaîne spécifiée	9
Change	Remplace chaîne 1 par chaîne 2	9
COPY	Copie champ au-dessus ligne spécifiée	9
MOVE	Idem ci-dessus mais efface lignes originales	9
Edit	Edition No ligne ou champ spécifié	9
	Mode ajout/insertion	
Auto	Entre dans le mode entrée	10
Insert	Entre en mode entrée au-dessus ligne spécif.	10
CTRL-L	Verrouilleur de clavier	10
CTRL-O	Entrée caractères absents au clavier	10
CTRL-X	Sort du mode entrée	10
	Mode Edition	
CTRL-I	Insère caractère(s)	11
CTRL-O	Insère caractère de contrôle	11
CTRL-D	Efface caractère(s)	11
CTRL-F	Trouve 1er caractère spécifié après CTRL-F	11
CTRL-B	Place curseur en début de ligne	12
CTRL-N	Place curseur en fin de ligne	12
CTRL-L	Transforme caractère (maj/min) sous curseur	11
CTRL-R	Rétablit la ligne sous sa forme originale	11
CTRL-C	Sort du mode édition	12
CTRL-Q	Efface caractères à droite du curseur	11
RETURN	Entre la ligne telle qu'elle apparaît	12

Commande	Description	Page
	EXPRESSION D'ASSEMBLAGE	
+	Addition	14
-	Soustraction	14
*	Multiplication	14
/	Division	14
!	Ou exclusif	14
.	Ou	14
&	Et	14
	Données décimales	14
$	Données hexadécimales	14
%	Données binaires	14
<	Données immédiates (byte bas de l'expr.)	15
<(	Byte bas de l'expression	15
<)	Byte haut de l'expression	15
</	Syntaxe optionnelle pour ci-dessus	15
	PSEUDO MNEMONIQUES ASSEMBLEUR	
	Directives	
EQU	Egalise étiquette à adresse ou étiquette	17
=	Syntaxe optionnelle pour égaliser	17
VAR	Egalise les variables spéciales	70
ORG	Etablit l'adresse de 'RUN'	17
OBJ	Etablit adresse stockage assemblage altern.	17
SAV	Sauve code objet à cet endroit	71
PUT	Insère fichier pendant assemblage	70
CHK	Place 'checksum' dans la source	70
END	Signifie la fin de la soure à assembler	17
	Formattage	
LST ON	Envoie assemblage à écran ou imprimante	18
LST OFF	Annule les sorties durant l'assemblage	18
EXP ON	Imprime tout le code pendant l'assemblage	18
EXP OFF	N'imprime pas code objet des macros	18
TR ON	Imprime max. 3 bytes obj. pendant assembl.	70
TR OFF	Remet flag troncation à sa val. par défaut	70
PAU	2e passe assemblage attend presser touche	18
PAG	Envoie saut de page à imprimante	18
AST	Envoie n astérisques au list. d'assemblage	18
SKP	Envoie n retours chariot au list. d'assemb.	18
	Chaîne	
ASC	Entre chaîne ASCII délimitée ds c.objet	19
DCI	Idem ci-dessus avec bit haut dern. car co.	19
INV	Entre chaîne délimitée en inverse	19
FLS	Entre chaîne délimitée en flash	19
	Données et allocations	
DA	Entre une valeur 2 bytes le bas d'abord	20
DDB	Idem ci-dessus mais le haut d'abord	20
DFB	Entre multiples bytes données sép. par ','	20
HEX	Idem ci-dessus;pas de ',';données hex. seul.	20
DS	Réserve un espace de n bytes	20
	Conditionnels	
DO	Si 0, arrête assemblage	21
ELSE	Inverse condition assemblage mise par DO	21
FIN	Annule le dernier DO	21
	Macros	
MAC	Commence définition d'une macro	22
EOM	Fin définition macro	22
<<<	Syntaxe alternative pour ci-dessus	22
PMC	Assemble macro à l'adresse présente	22
>>>	Syntaxe alternative pour ci-dessus	22

TABLE DES MATIERES

TROISIEME PARTIE : AUTRES PROGRAMMES

QUATRIEME PARTIE : UTILISATION DE BIG MAC

CINQUIEME PARTIE : SUPPLEMENT CARTE LANGAGE