Logiciel pour
Apple IIe et Apple IIc

ProCODE

ASSEMBLEUR 65C02

Éditeur pleine page

Système ProDOS

Présentation

Vous possédez à présent, un manuel de programmation en assembleur pour Apple *//e* et Apple *//c*, ainsi que trois disquettes :

— Le programme ProCODE
— La sauvegarde du programme ProCODE
— /UTIL, qui vous permettra :
- de formatter vos disquettes vierges
- de convertir des fichiers DOS en ProDOS
- d'avoir accès à la configuration de ProCODE
- d'accéder aux fichiers documents que nous vous avons préparés :

. QUICK.COPIE
. BIBLIO.MACRO
qui sont des sources éditables par ProCODE
et :
. RWTS.C
. GO.QUICK.COPIE
qui sont des fichiers binaires permettant d'utiliser QUICK.COPIE.

Le but premier de l'ouvrage, n'est pas de vous apprendre le langage assembleur mais comment utiliser ProCODE. Grâce aux nombreux exemples que nous y avons intégrés, nous vous donnons la possibilité de l'utiliser le plus aisément possible.

Tous les chapitres du manuel concernent aussi bien l'Apple *//e* que l'Apple *//c* :
— le chapitre 1 est une description sommaire des particularités de ProCODE.

— le chapitre 2 vous permet de démarrer avec ProCODE.

— le chapitre 3 vous enseigne comment configurer ProCODE.

— le chapitre 4 détaille toutes les fonctions de ProCODE auxquelles vous pouvez accéder à partir de son Menu.

— le chapitre 5 décrit l'éditeur de ProCODE.

— le chapitre 6 développe en détail tous les points de l'assembleur ProCODE.

Vous trouverez également, en annexe, la récapitulation des commandes, la table de référence des codes ASCII et une bibliographie.

Introduction

De quoi s'agit-il ?

Nous vous proposons un programme dont l'objectif principal est de vous faciliter la programmation en assembleur sur Apple *//e* et Apple *//c*. En effet tant d'un point de vue pratique que technique, ProCODE interviendra pour vous aider dans vos démarches de programmation.

En fait, nous pouvons définir ProCODE comme un éditeur assembleur pleine page en 80 colonnes. Il accepte plusieurs sources simultanément en mémoire, ceci pour une plus grande souplesse d'écriture et de correction des erreurs pendant l'assemblage.

Cet assembleur reconnait les mnémoniques 6502 et 65C02 ; il est donc utilisable pour l'Apple *//e* et l'Apple *//c*.

Quels sont les points forts de ProCODE ?

— Les Macro-instructions, qui ont une grande analogie avec les sous-programmes que vous pouvez avoir dans les langages évolués, enrichissent cet éditeur assembleur.

— Les instructions de chaînage vous permettent d'assembler des fichiers de grande taille.

— L'assemblage conditionnel vous donne la possibilité de préparer un programme très général et d'en assembler différentes versions selon ce que vous désirez changer.

— Il reconnait les mnémoniques et les modes d'adressage de l'assembleur 65C02.

— Il vous permet d'utiliser des labels locaux, redéclarables par instruction, et, d'avoir des labels non limités en nombre de caractères.

— Vous pourrez assembler un source en mémoire sans accès disque.

— Page par page, vous pourrez connaitre le nom du fichier assemblé.

— Même dans le cas de fichiers chaînés, ProCODE peut se positionner automatiquement sur la ligne qui a provoqué l'erreur.

De plus, ProDOS comme système d'exploitation, c'est la rapidité ainsi que la possibilité de se connecter à ProFILE !!!

Note : Dans ce document, toutes les touches du clavier sur lesquelles vous devez appuyer,sont symbolisées de la manière suivante : [ ]
Lorsque vous devez appuyer sur plusieurs touches simultanément, elles seront symbolisées ainsi :
Ex : [Ctrl] - [RETURN]

Comment démarrer ProCODE

Introduisez la disquette ProCODE dans le lecteur principal (lecteur 1) et allumez votre ordinateur.
On dit que vous mettez en marche votre système (to boot en anglais).
Assez rapidement le logo ProCODE masque l'écran, puis le lecteur de disquettes continue à tourner pendant quelques secondes, et le Menu ProCODE apparaît.

Après le chargement de ProCODE, il est recommandé d'ôter la disquette ProCODE, et de la remplacer par une disquette sur laquelle vous travaillerez ensuite. Si cette disquette est vierge, vous devrez la formater.

Suivez avec attention la démarche que nous vous décrivons ci-dessous.

a) Initialisation d'une disquette de travail

— Booter la disquette ProDOS ou /UTIL (1)
— Taper -FILER puis (RETURN)
— Choisissez «V» (Volume commands)
— Retirer la disquette ProDOS ou /UTIL
— Insérer une disquette vierge (2)
— Taper «F» (Format a volume)
— Après Slot répondez : (6) puis (RETURN), et après Drive répondez : (1) puis (RETURN)
— à côté de «New volume name», répondez par exemple : /BIBLIO1 puis (RETURN).

(1) Si vous possédez un Apple *IIc*, n'utilisez pas votre Utilitaire Système, mais bien cette disquette /UTIL fournie avec ProCODE.

(2) Si la disquette n'est pas vierge, un message vous l'indiquera : DESTROY «/BI»? (Y/N).
L'ordinateur vous demande si vous désirez détruire la disquette dont le préfixe est /BI.
Si vous tapez (Y) (oui), alors ce qui était préalablement sur votre disquette sera détruit.
Si vous tapez (N) (non), vous pourrez conserver votre disquette intacte.

Note : /BI est un exemple de nom de volume.

b) Booter ProCODE

Si vous avez travaillé auparavant avec un autre programme et que vous en êtes sorti, il n'est pas nécessaire d'éteindre votre ordinateur puis de le rallumer après avoir introduit ProCODE dans le lecteur principal. Il vous suffit d'appuyer sur Ctrl - Ⓐ - Reset et de les relâcher (Reset en premier).

Dès que le système a parcouru le ou les lecteurs de disquettes, le Menu ProCODE apparait à l'écran.

A présent, vous retirez ProCODE, et introduisez votre disquette de travail à laquelle vous aurez donné un nom, par exemple : /BIBLIO1

Tapez 1 pour indiquer le préfixe (nom) du volume sur lequel vous désirez travailler.

Votre système va chercher le volume nommé /BIBLIO1, et dès qu'il l'aura trouvé, il vous présentera à nouveau le menu de ProCODE.

Note : Il se peut que lors du BOOT, d'autres messages apparaissent :

— «Il faut un Apple *//e* 80 colonnes».

Ce message apparait si vous essayez de charger ProCODE, sur un Apple II+ ou sur un Apple *//e* non muni d'une carte 80 colonnes Apple.

— «je ne peux pas charger ProCODE».

Vérifiez votre unité de disquettes, réinsérez et rechargez votre disquette ProCODE. Si ce message persiste à l'écran essayez alors avec la sauvegarde fournie avec votre programme. Si le message réapparaît, contactez votre revendeur.

— «recharger ProCODE».

Ce message peut se présenter même lors d'une utilisation normale de ProCODE. En effet, ProCODE vérifie si le programme en mémoire n'a pas été détérioré.

Si vous utilisez ProCODE pour la première fois, il faut lui indiquer la composition de votre système. Reportez vous au Chapitre 3 CONFIGURATION.

Configuration

Le programme Configuration, est nécessaire pour adapter votre logiciel assembleur aux différentes configurations existantes :

— selon le type d'imprimante,
— selon le type d'interface,
— selon la taille mémoire disponible (64 ou 128 k).

Pour indiquer à ProCODE la configuration avec laquelle il va travailler, il faut que vous exécutiez le programme Configuration.

Nous allons donc vous expliquer, le moyen de faire reconnaitre à ProCODE votre configuration.

L'entrée dans le programme configuration

a) Mettez la disquette /UTIL dans le lecteur de disquettes et mettez en marche votre système.

(/UTIL : c'est le nom du volume de la disquette comportant les utilitaires et les programmes de service. Cette disquette est livrée avec le programme ProCODE).

Apparaissent à l'écran quelques indications, dont l'occupation des connecteurs de votre Apple *//e* ou de votre Apple *//c*.

Otez la disquette /UTIL du lecteur, et remplacez la par le programme ProCODE.

b) Tapez : PREFIX /PROCODE (avec un espace entre PREFIX et /PROCODE)

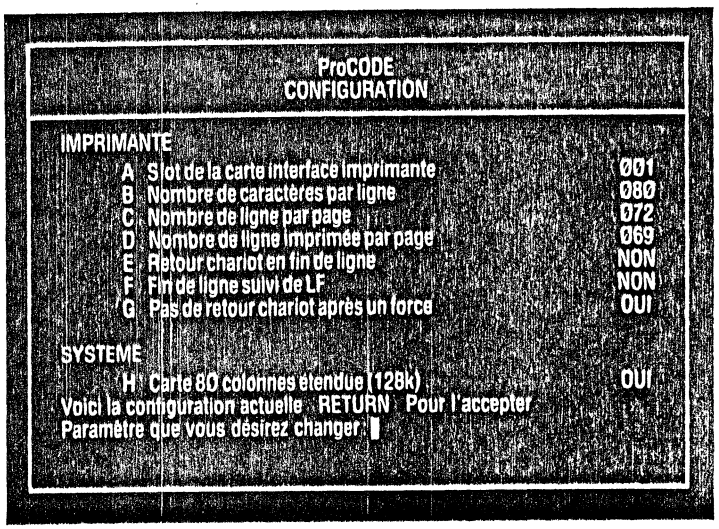
Remarque : PROCODE, est le nom du volume occupé par ProCODE.

Tapez RETURN

Le lecteur lit votre disquette programme afin de reconnaitre son volume.

c) Tapez : RUN CONFIGURATION puis RETURN

Apparait à l'écran :



Si l'environnement standard vous convient.

Tapez (RETURN) pour annoncer que vous acceptez la configuration :

— Apparait à l'écran : On enregistre cette configuration

Tapez (O) pour OUI

— Apparait à l'écran : Retour au basic systeme

Tapez (O) pour OUI
Tapez (N) pour NON ; dans ce cas, il ne vous reste plus qu'à revenir au menu de ProCODE, en appuyant simultanément sur (Ctrl) - (C) - Reset

Si l'environnement standard ne vous convient pas.

Vous sélectionnez les paramètres, en tapant au clavier la lettre qui leur correspond (colonne de gauche).
Dès que vous voulez modifier un de ces paramètres, le curseur se place automatiquement sur le premier caractère de la définition de celui que vous avez choisi (colonne de droite).
Notez que les modifications se font automatiquement en mode recouvrement.

(A) : Répondez le numéro du connecteur dans lequel se trouve l'interface de l'imprimante, généralement 001.

(B) : En standard, vous avez 80 caractères par ligne.

(C) : En standard, vous avez 72 lignes par page.

(D) : En standard, vous pouvez avoir 66 lignes imprimées par page.

(E) : En général, vous demandez un retour chariot en fin de ligne.

(F) : En général vous demandez un saut de ligne.

(G) : Si vous possédez une imprimante, qui dès qu'une ligne est remplie, pratique un retour chariot automatique, alors vous devez répondre NON.
Si votre imprimante ne saute pas de ligne (le papier n'avance pas dans le vide), alors vous répondez OUI.

(H) : Votre réponse dépend de la carte que vous possédez.

Notez que vous pouvez modifier autant de paramètres que vous le désirez.

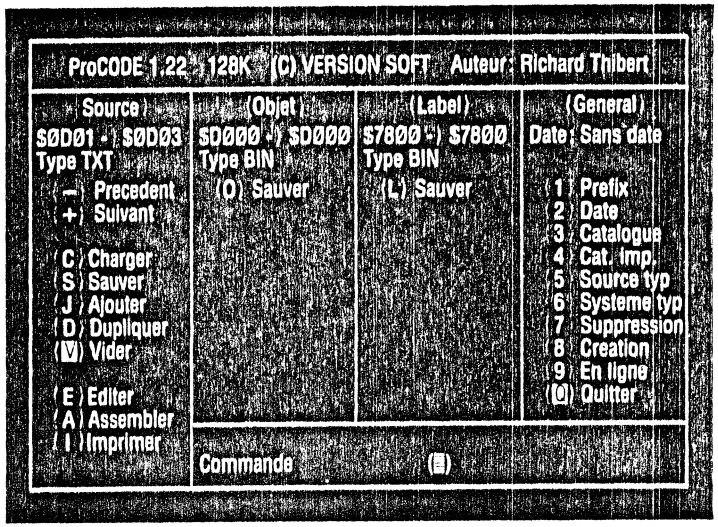
Dès que vous en avez terminé, vous devez enregistrer cette nouvelle configuration. Pratiquez comme dans le cas précédent.

Note : Si vous ne voulez pas enregistrer la configuration, répondez (N) pour NON.
Dans ce cas, apparaît à l'écran :

Retour au basic système. Répondez (O) ou (N) .

Le menu

Comme nous vous l'avons précisé précédemment, l'accès au menu ProCODE est direct. En effet, vous le voyez apparaître dès que vous chargez le programme.
Il se présente sous la forme d'un tableau dont certains indices changeront lorsque vous utiliserez ProCODE, comme par exemple les adresses, le type, la date, etc.
De même, les noms des sources, des labels, du fichier code, apparaîtront à l'écran dès que vous les indiquerez.



Utilité du menu

Le menu vous donne accès à toutes les fonctions du programme ProCODE, et, vous permet ainsi de charger, sauver, assembler, éditer...

Pour une plus grande clarté, les commandes ont été regroupées en fonction du fichier sur lequel elles agissent : source, objet ou label.

Les commandes qui n'agissent sur aucun fichier en mémoire sont dites «générales», elles peuvent seulement modifier le contenu du disque, la date, le type des fichiers...

Chaque commande vous est accessible, simplement en appuyant sur une touche du clavier.
Par exemple, (S) pour sauver, (1) pour préfix, etc.

▲ Attention

Quelle que soit la fonction que vous ayez sélectionnée et où que vous soyez, vous pouvez toujours revenir au menu en tapant sur (Esc).

Organisation de la mémoire

ProCODE vous autorise à avoir plus d'un source en mémoire au même moment. Vous pouvez en avoir exactement trois simultanément en mémoire ; il est possible de naviguer de l'un à l'autre, de tous les assembler, les éditer, les sauver...

Exemple :

Lorsque l'on veut revenir à un programme autre que celui sur lequel on travaille, il n'est pas nécessaire de sauver la mémoire, de charger ce programme, de l'éditer, puis de recharger le programme en cours, pour continuer la frappe. Il suffit de charger le programme désiré et de l'éditer, puis on revient immédiatement au source actuellement en composition.
Vous pouvez également travailler sur plusieurs fichiers au même moment (plusieurs versions, plusieurs fichiers constituant un même programme...).

Comment s'organise la mémoire pour nous permettre ces facilités ? :

— les sources s'empilent dans un espace réservé de 46 k (version 128 k) ou plus petit dans la version 64 k.

— à chaque source sont attribués un numéro (1, 2 ou 3), et le nom du fichier disque. Si vous éditez le source 2 (que nous pouvons appeler «programme» par exemple), alors, le source 1 descend au bas de la mémoire, au-dessus vient le source 2 et le source 3 se place tout en haut de la mémoire ; on réserve ainsi la place maximum pour le source édité.
Ceci vous explique, que l'adresse de début et de fin d'un même fichier peuvent varier sans que l'on ait travaillé avec ce source.

Nous allons voir à présent ce qu'il en est du fichier code et des labels. Deux cas se présentent, suivant que vous ayez une version 64 k ou une version 128 k :

version 64 k

Imaginez une tour dont les pierres peuvent bouger (monter ou descendre), se développer (par entrée de texte) et s'amoindrir (suppression de texte). A la base de la tour vous avez les sources, au-dessus vous trouverez les labels puis le fichier code.
Tout cela tient dans un même espace de la mémoire donc, plus le code et plus les labels ont peu de place réservée et plus la place disponible pour les sources est grande. De même, plus les sources sont petits et plus la place réservée pour le code et les labels peut être importante. En fait tout ceci est négligeable quand on travaille avec des sources peu étendus, et quand le code généré n'est pas très important (inférieur à $AOO).
Il faut concilier toutes ces réalités, en partageant au mieux la mémoire (utilisez les instructions FXLBL et FXCD) et ou, en découpant les sources.

Notons que l'espace réservé pour le fichier code, les labels et les sources, est de $6DOO soit environ 28 k.

version 128 k

On a deux fois plus de place en mémoire, et il est possible d'assembler des sources au moins trois fois plus grands.

— Pour les 3 sources, on réserve 46 k en mémoire auxiliaire, de $DOO à $COOO.

— Pour le fichier code, on réserve $3000 octets, soit 12 k en banque commutable de la mémoire auxiliaire, de $DOOO à $FFFF.

— Pour les labels, on réserve $4800 octets en mémoire principale, de $7800 à $BFFO.

Il n'y a pas de paramétrage (comme dans la version 64 k) de l'espace réservé au fichier code et aux labels. On peut ainsi assembler très vite et sans problème, de grands sources.

Vous vous demandez sans doute à quoi ces informations techniques peuvent vous servir effectivement, puisque vous pouvez très bien les ignorer et utiliser ProCODE. Néanmoins, il est recommandé d'avoir une idée, même très générale, de l'organisation de la mémoire. En effet, une bonne maîtrise de l'assembleur, vous permettra de mieux comprendre que son organisation joue un rôle important dans l'utilisation de certaines instructions (FXLBL, FXCD, PUT, DSK...).

La présentation du menu

Le menu de ProCODE est découpé en 4 grandes parties :
— le source
— l'objet (fichier code)
— le label
— le général

Nous définirons ci-après, les commandes spécifiques de chaque menu.

Les colonnes fichier code, labels et sources sont très semblables. Vous avez effectivement dans chaque colonne :
— le nom (source, fichier code, label, général)
— l'adresse de début et de fin dans la zone (sauf pour la colonne «général»)
— le type du fichier (TXT, BIN...) sauf pour la colonne «général».

De plus :
— la colonne «général» affiche la date (si aucune date n'a été rentrée auparavant, elle affiche l'intitulé «sans date»).

En bas à droite, un cadre est réservé au dialogue «programmeur - menu». Vous y verrez apparaître des messages et des demandes d'indications lors des options du menu que vous sélectionnerez. A ce propos, notez que :

— Pour vous déplacer vers la droite, vous utilisez (→)
— Pour vous déplacer vers la gauche, vous utilisez (←) ou (Del)
— (🍎) - (→), vous permet d'aller en fin de chaîne
— (🍎) - (←), vous permet d'aller en début de chaîne.

Toutes les commandes sont présentées à l'écran, précédées par la lettre qui vous permet de les appeler au clavier.

Les commandes menu

La colonne source

- (-) précédent et (+) suivant

Pour sélectionner un programme source parmis d'autres en mémoire, vous pouvez utiliser les commandes (-) et (+), qui orienteront vos recherches vers la zone inférieure (−) ou supérieure (+) de la mémoire.

Exemple :

Si on a trois sources en mémoire :

Source 1 $D01 -> $F10
Source 2 $F11 -> $5400
Source 3 $5401 -> $5673

Si à l'écran dans la colonne source, on a le Source 1, alors en appuyant sur (+) apparaîtra le Source 2 à l'écran et, si on appuie de nouveau sur (+) le Source 3 s'affichera; en appuyant ensuite sur (-) le Source 2 revient, etc.

La commande (-) , n'a aucun effet si on est dans le Source 1.

Si on est dans le Source 3, la commande (+) n'a aucun effet.

- (C) charger

Grâce à cette commande, il est possible de charger des sources à partir du disque.
Si vous chargez un fichier non sauvé par ProCODE, il faut l'éditer avant de l'assembler. L'éditeur se chargera de modifier ce source pour qu'il soit compatible avec ProCODE (compatibilité de forme: texte en ASCII haut terminé par $8D puis $3F).
Si vous avez des fichiers PUT, et s'ils n'ont pas été sauvés par ProCODE, il faut les charger et les éditer pour les sauver. Cela vous évitera des problèmes pendant l'assemblage.

▲ Attention

Si le source actif n'est pas vide, il sera effacé avant le chargement du nouveau source.

Remarque : ProCODE accepte pratiquement tous les types de sources et peut éditer des sources d'à peu près n'importe quel assembleur.

- (S) sauver

Cette commande vous permet d'écrire le source actif sur le disque. ProCODE vous demande le nom du fichier que vous avez créé ou mis à jour.

▲ Attention

Vous ne pouvez sauver que le source actif.

- (J) ajouter

Cette commande est identique à (C) (charger), mais le fichier est ajouté à celui qui était déjà écrit.

- (D) dupliquer

A l'aide de cette commande, vous pouvez copier un source en mémoire sur un autre source.

Exemple :

Imaginez que vous vouliez faire une copie du source 2 dans le source 3. Placez-vous dans le source 3 et demandez (D) ; à la question : quel source copier ?, répondez (2) .

Remarque : On ne peut écrire que dans le source actif.

- (E) éditer

  Par cette commande, vous appelez l'éditeur (vous éditez le fichier actif).

- (A) assembler

  Vous assemblez le fichier sur l'écran.

- (I) imprimer

  Vous assemblez le fichier sur une imprimante.

- (Ctrl) - (V) vider

  Vous effacez totalement de la mémoire le source actif.

▲ Attention

ProCODE ne vous demandera pas de confirmation ((Ctrl) - (V) ne se fait pas par hasard) et un source vide ne peut pas être récupéré.

Sur la colonne objet

- (O) sauver

Grâce à cette commande, vous sauvez le fichier code sur le disque. Le fichier est sauvé à son adresse d'assemblage (définie par ORG) de manière à pouvoir être utilisé par BRUN.

Sur la colonne des labels

- (L) sauver

Avec cette commande, vous sauvez la table des labels sur le disque (ou sur tout autre volume en ligne).

Remarque : Il est très pratique de pouvoir utiliser cette table indépendamment de l'assembleur, afin de la lister différemment. Pour faire des recherches rapides, vous pouvez créer des programmes utilisant avantageusement cette option.

Commandes générales

- (1) préfixe

  Vous l'utilisez afin d'indiquer le préfixe (nom) du volume sur lequel vous travaillez (comme dans le basic system).

- (2) date

  L'utilisation de cette fonction est intéressante pour connaître la dernière version d'un programme.
  Si vous avez une carte horloge reconnue par ProDOS, la date s'affichera automatiquement, sinon, il vous faudra l'entrer vous-même.

  Pour entrer votre date, suivez l'exemple décrit ci-après :
  — vous désirez indiquez que nous sommes le 30 Septembre 1984
  — vous tapez (2), et « date » apparaît à l'écran
  — vous tapez : 30/SEP/84 qui est de la forme JJ/MMM/AA
  (les majuscules sont indispensables)
  Les mois reconnus sont : JAN, FEV, MAR, AVR, MAI, JUN, JUL, AOU, SEP, OCT, NOV, DEC.

- (3) catalogue

  Cette commande vous permet de lister à l'écran, le contenu d'un volume défini par son préfixe (le format du catalogue est très proche de celui du basic system).

- (4) cat.imp.

  Avec cette commande, vous pouvez lister sur imprimante, le contenu d'un volume défini par son préfixe.

- (5) typ.source

  Type des fichiers sources écrits par ProCODE (par défaut type.txt).

  Types reconnus :

  bad, txt, bin, dir, cmd, bas, var, rel, sys
  (il est amusant mais inutile de choisir des sources de type dir ou bad... auquels on ne peut plus accéder directement ensuite).
  Les types usuels sont txt ou bin.

- (6) typ.syste.

  Cette option vous permet de définir le type des fichiers code et des labels.

  Il est très pratique de créer directement ces fichiers sys, par exemple afin que le programme soit exécuté immédiatement lors du BOOT.

- (7) suppression

  La commande Suppression, vous permet d'effacer un fichier qui ne vous intéresse plus ou que vous avez déjà ailleurs. Il vous suffit de donner son nom.

Elle ressemble au «delete» du basic.system de ProDOS.

- (8) créer

  En sélectionnant cette fonction, vous pouvez créer un sous-catalogue sur le volume. ProCODE vous demandera alors le nom que vous lui donnez.

- (9) en ligne

  En choisissant cette commande, vous affichez à l'écran le nom des différents volumes en ligne auxquels vous pouvez avoir accès.

- (Ctrl) - (Q) quitter

  En appuyant simultanément sur ces deux touches du clavier, vous confirmez votre désir de sortir de ProCODE. Vous êtes à présent en mode moniteur (call-151 en basic).
  Pour revenir dans ProCODE sans effacer les fichiers en mémoire, tapez simultanément sur (Ctrl) - (Reset) .

L'éditeur

L'édition d'un programme doit être réalisée avant son assemblage ; les modifications et la mise à jour seront facilitées grâce à l'éditeur pleine page de ProCODE.

Il existe deux manières d'accéder à cet éditeur :

— Par l'instruction «E» du menu.
— Par l'instruction «C», lorsque l'assembleur trouve une erreur, et que vous demandez sa correction immédiate.

Pour quitter l'éditeur, il suffit d'appuyer sur (Esc) .

L'introduction de texte

Pour créer un texte ou modifier un fichier source, il suffit d'utiliser votre clavier, aussi bien en QWERTY qu'en AZERTY, puisque les messages sont étudiés pour les deux modes.

Tous les caractères normaux (c'est-à-dire lisibles à l'écran) sont autorisés.

Une ligne peut contenir jusqu'a 180 caractères, mais, comme l'écran en affiche au plus 80, les lignes défilent automatiquement. ProCODE utilise un mode de défilement original : en effet, le curseur peut bouger sur une même ligne et la ligne elle-même peut défiler.
Vous apprécierez sans doute ce procédé, qui permet de voir le maximum de caractères autour du curseur.
Il est très pratique de pouvoir écrire sur une même ligne plus de 80 caractères, surtout pour la zone des commentaires et, également si votre imprimante vous permet une impression de plus de 80 caractères.

Les tabulations

Elles existent lors de la frappe de votre texte, et ce que vous voyez correspond au listing fait par l'assembleur.

Il existe quatre tabulations :

— le début de la ligne au caractère numéro 0
le premier espace tapé déclenchera la tabulation suivante.

— le début du mnémonique au caractère numéro 10
le premier espace tapé déclenchera la tabulation suivante.

— le début de l'opérande au caractère numéro 15
le premier «;» est tabulé comme le commentaire.

— le commentaire est tabulé au caractère numéro 25.

Remarques :
— Les tabulations sont des artifices de présentation, vous ne pouvez donc pas placer le curseur sur des espaces qui servent aux tabulations.

— Si vous débutez une ligne par «★», aucun caractère ne provoquera de tabulations.

— Si vous commencez une ligne par «;», alors ce caractère sera tabulé à la colonne numéro 25.

Modification de texte

Pour modifier un texte, pour corriger des erreurs de frappe, vous avez plusieurs commandes possibles :

(←) : vous revenez sur le dernier caractère frappé, vous ne supprimez pas le caractère sous le curseur. Si vous êtes sur le premier caractère de la ligne, vous ne bougez pas.

(→) : vous passez au caractère suivant le curseur; s'il n'y en a pas, ou, si vous y êtes déjà, vous ne bougez pas.

(↑) : vous remontez d'une ligne et vous vous placez sur le premier caractère ; si vous êtes sur la première ligne du programme, rien ne se passe.

(↓) : vous descendez au début de la ligne suivante ; si vous êtes à la fin du programme, vous y restez.

(Del) : le curseur avance vers la gauche de votre écran, et vous effacez tout caractère se trouvant à sa gauche. Si vous appuyez sur cette touche alors que vous êtes sur un caractère, celui ci se déplacera avec le curseur.

(⇥) ou (Ctrl) - (I) : vous passez directement à la tabulation des commentaires ; si ce champs n'existe pas, ProCODE rajoute des espaces et un point virgule à la fin de la ligne, pour marquer le début du champs des commentaires.

Avec la touche maintenue enfoncée :

(↵) : vous revenez directement en début de ligne.

(→) : vous placez le curseur en fin de ligne.

(↑) : votre curseur se place sur le premier caractère de la page précédente ou sur celui de votre page si c'est la première.

(↓) : vous descendez au début de la page suivante ; si vous êtes à la fin du programme vous y restez.

(Del) : la ligne se déplace vers la gauche, et vous supprimez le caractère sous le curseur.

Avec la touche ⌘ maintenue enfoncée :

(←) : vous déplacez la tabulation 0 d'un caractère vers la gauche (vous affichez donc le 81e caractère, et le premier disparaît).

(→) : vous déplacez la tabulation 0 d'un caractère vers la droite.

> **Remarque :** Les deux instructions précédentes, n'agissent pas sur la ligne où est votre curseur.

(↑) : vous remontez au début de la première ligne du programme; si vous y êtes déjà, vous y restez.

(↓) : vous descendez au début de la dernière ligne de votre programme; si vous êtes déjà sur cette ligne, vous ne bougez pas.

Fonctions utilitaires de l'éditeur

- (Ctrl) - (B) : met en mode inverse la ligne où se situe le curseur (on la distingue mieux à l'affichage). Pour revenir en mode normal, vous retapez (Ctrl) - (B) .

- (Ctrl) - (L) : supprime la ligne sur laquelle vous placez le curseur.

- (Ctrl) - (C) : vous permet de copier une partie de texte à un autre endroit de votre programme.
  Vous définissez la première ligne à copier en vous plaçant dessus, puis vous validez par (RETURN) ; la ligne apparaît en inverse. Vous faites de même pour la dernière ligne (s'il s'agit d'un paragraphe). Puis, à l'aide des flèches, vous vous positionnez sur la ligne destination, et vous faites un (RETURN) .
  Cette commande n'efface aucun caractère du programme.

- (Ctrl) - (F) : cette commande s'utilise pour rechercher une chaîne de caractères dans le programme.

  a) l'ordinateur vous demande tout d'abord la chaîne à rechercher.
  Si vous mettez des «?» dans cette chaîne, ProCODE considèrera que n'importe quel caractère convient.

  b) puis il vous demande dans quel champ est cette chaîne :

  (A) Label, (B) Mnémonique, (C) Opérande, (D) Commentaire, (E) Partie utile, (F) Ligne

  vous répondez en appuyant sur la lettre caractérisant l'option choisie.

> **Remarques :**
> — L'option (F) correspond à un «chercher standard» : on recherche la chaîne n'importe où dans le texte.
> — «partie utile» correspond à tous les champs sauf celui des commentaires.
> — Les champs tabulés par des espaces ne possèdent pas l'espace qui les tabule, mais le «;» qui tabule le champ des commentaires en fait partie.

c) ProCODE vous demande ensuite la situation de la chaîne dans le texte :

(A) isolé, (B) entouré, (C) non précédé, (D) non suivi

Une chaîne non précédée est soit en début de ligne, soit précédée par un caractère autre qu'un chiffre ou une lettre (de même pour suivi et isolé).

d) Pour finir, ProCODE affiche :

(A) importance des minuscules, (B) convertit tout en majuscule.

Si vous répondez (B), vous devez rentrer votre chaîne en majuscule.

ProCODE recherche la chaîne à partir du curseur et le place dessus. Pour rechercher une autre chaîne correspondant aux mêmes critères, utilisez l'instruction (Ctrl) - (S) («S», comme suite).

- (Ctrl) - (R) : c'est un «chercher remplacer». ProCODE vous posera les mêmes questions qu'avec le «chercher» simple, mais avec en plus, celle se rapportant à la chaîne qu'il faut substituer à celle que l'on cherche («chaîne» est remplacé par :).
  Dès que ProCODE trouve la première chaîne qu'il doit remplacer il vous demande :

  (A) on remplace, (B) on poursuit, (C) on remplace tout.

> **Remarque** : Lorsque vous choisissez l'option (C) :
>
> — vous ne verrez pas chaque chaîne être remplacée l'une après l'autre.
>
> — le curseur se place en fin de texte, après les modifications effectuées.

- (Ctrl) - (V) : vous fait passer du mode insertion au mode recouvrement et vice versa.
  En mode recouvrement le texte est inséré en fin de ligne sinon il remplace le texte déjà existant.

- (Ctrl) - (T) : vous permet de répéter à l'infini la dernière commande sélectionnée.

- (RETURN) descend le curseur à la ligne suivante en l'insérant dans le programme.

Conclusion sur l'éditeur

Dès à présent, vous pouvez vous rendre compte, qu'il sera plus facile pour vous d'utiliser pendant un court moment le programme, plutôt que de lire des dizaines de pages d'explications.

Règles facilitant la mémorisation des touches :

— (Ctrl) - (C) comme copier
— (Ctrl) - (F) comme FIND (ou trouver en français)
— (Ctrl) - (S) comme suite
— (Ctrl) - (R) comme remplacer
— (Ctrl) - (L) comme ligne

On sait que la mémoire est pleine quand on ne peut plus entrer de caractères.
On sait que le nombre maximum de caractères sur une ligne est atteint quand on ne peut plus en rajouter.

En avant dernière ligne de l'écran, s'affichent plusieurs nombres :

— le premier est le numéro de la ligne actuelle où se trouve votre curseur .

— le second est le nombre de lignes de votre texte.

— le troisième est le nombre de caractères après le curseur.

— en dernier lieu, vous pouvez voir indiqué, le mode dans lequel vous travaillez (recouvrement ou insertion).

L'assembleur

Vous pouvez accèder à l'assembleur directement à partir du menu, en tapant (A) ou (I) :

— (A) provoque l'assemblage avec édition à l'écran.
— (I) provoque l'assemblage avec édition sur imprimante.

Prise en main de l'assembleur

Si vous choisissez de lister un programme sur imprimante, ProCODE vous demande son nom. Vous le verrez affiché en haut de chaque page, précédé par le nom du fichier assemblé (cela est utile si le programme est divisé en plusieurs morceaux appelés par l'instruction PUT).

Lorsque le programme s'assemble, le message «Assemblage par ProCODE» vous indique que votre fichier est en cours d'assemblage. Pour interrompre l'assemblage, il vous suffit d'appuyer sur (Esc), et, dès lors, le fichier code et la table des labels qui devaient être générés sont remis à 0.
Si vous interrompez le programme pendant que les labels sont listés, le fichier code et la table des labels ayant été entièrement générés, l'arrêt par (Esc) ne réinitialise pas ces fichiers.

Quand vous demandez un listing de programme (pas de LST OFF), vous pouvez immobiliser l'affichage en appuyant sur (ESPACE). Ensuite, chaque fois que vous appuierez sur cette touche, vous provoquerez l'assemblage d'une ligne. Pour reprendre le défilement normal, appuyez sur n'importe quelle touche autre que (ESPACE) et (Esc).
Lorsque l'assembleur ne comprend pas une ligne de texte, il imprime un message d'erreur. Il indique également dans quel fichier a lieu cette erreur, en quelle ligne de ce fichier, il la décrit et vous propose trois choix différents :

(A) : vous continuez l'assemblage en ignorant l'erreur. Une fois l'assemblage terminé, vous n'aurez tout de même pas le fichier code ni la table des labels.

(B) : vous interrompez l'assemblage et revenez au menu.

(C) : vous éditer immédiatement l'erreur et vous vous placez dans l'éditeur sur la ligne où elle se situe.

A la fin du listing, ProCODE imprime la table des labels pour vous permettre de retrouver rapidement leurs adresses.
Devant chaque label listé il peut y avoir plusieurs signes :

— «?» indique que le label est déclaré mais jamais utilisé.
— «S» le label est redéclarable par instruction.
— «R» le label est redéclarable.
— «M» le label est une définition de MACRO.

A noter, que cette table est listée 2 fois :
la première fois par ordre alphabétique et la seconde fois par ordre numérique.

Avant de vous expliquer en détail le fonctionnement de l'assembleur ProCODE, voici ses caractéristiques comparées aux autres assembleurs.

ProCODE permet :

— les MACRO instructions
— l'assemblage conditionnel
— les mnémoniques et les modes d'adressage 65C02
— l'utilisation de labels locaux, redéclarables par instruction
— de travailler avec 128k, d'assembler, sans accès disque, des fichiers sources de 46K
— d'avoir des labels non limités en nombre de caractères
— par ses instructions de chaînage, d'assembler d'importants fichiers
— de connaître page par page le nom du fichier assemblé
— une correction immédiate des erreurs.

De plus :

— de nombreux messages accompagnent les erreurs
— vous pouvez réutiliser des programmes sources créés par d'autres assembleurs avec peu de modifications.

L'assembleur

Des trois parties constituant ProCODE (menu, assembleur et éditeur), l'assembleur est la plus importante (ProCODE est fondamentalement un assembleur!!).
Sa tâche consiste à traduire un source compréhensible pour l'homme, en une suite d'octets, intelligibles pour le processeur.
Il faut comprendre, que le texte source est écrit dans un véritable langage. Il suit des règles strictes que le programmeur doit respecter s'il veut que son programme s'assemble sans erreur.
Ce qui va suivre, ne vous apprendra pas le langage «assembleur» (ce n'est pas le but premier de ce manuel). Nous allons essentiellement vous expliquer, comment utiliser ProCODE pour pouvoir faire simplement des programmes très complexes ,et pour vous montrer que la difficulté réside dans la programmation elle-même et non dans l'utilisation de l'assembleur.

Structure générale de l'assembleur

L'assembleur se compose de plusieurs parties,et l'on distingue principalement :

— l'analyse d'une ligne de programme
— les instructions assembleur
— la gestion des erreurs
— la gestion de la vidéo ou de l'imprimante.

Dans la traduction du source, (fichier écrit par le programmeur en utilisant l'éditeur) ces quatre parties travaillent ensemble ou séparément pour créer deux nouveaux fichiers :

— le fichier code : qui est la traduction exécutable par le 6502 du source compréhensible par l'homme.
— la table des labels : qui est directement utilisée par l'assembleur. Un fichier contient sous forme codée, la liste de tous les labels utilisés au cours de l'assemblage.

Sachez que l'assembleur procède en 2 passages : un pour créer la table des labels, et l'autre pour transcrire véritablement le fichier source.
L'assembleur analyse le fichier source ligne par ligne ,et s'il ne comprend pas ce qui est écrit, il imprime un message d'erreur, indiquant ce qui ne va pas dans la ligne.

Syntaxe reconnue par l'assembleur

Définition des termes

Un label

C'est un nom mnémonique représentant des nombres compris entre 0 et 65535.

régles de constitution d'un label

■ 1re lettre pour un label normal :

— lettres majuscules ou minuscules
— signes «ç» ou «'» ou «^» ou «—»

■ 1re lettre pour un label local :

— signe «§»

■ lettres suivantes :

— lettres majuscules ou minuscules
— chiffres
— signes «§», «^» et «.»

Note : Le nombre de caractères dans un label n'est pas limité.
L'écriture en majuscule ou en minuscule d'un label ne change en rien ce label.

Exemples :

— COUT23
— KEYIN
— keyin.1 ou KEYIN.1 ou kEYIn.1
— A23ç..
— §BOUCLE

Une donnée

C'est un nombre compris entre 0 et 65535 ou $FFFF et pouvant être écrit de plusieurs manières :

■ une série de chiffres
la donnée est de forme décimale.

■ «$» suivi par une série de chiffres ou de «A» , «B» , «C» , «D» , «E» , «F»
la donnée est de forme hexadécimale.

Exemples :

— $FDED
— $1000
— $1A98
— $1abc (les minuscules sont autorisées)

■ «%» suivi par une série de «1» ou «0»
la donnée est de forme binaire.

Exemples :

— %10101111 codé : $AF
— %11 codé : $03

■ « (guillemet) suivi par une lettre ou un signe autre que «^» la donnée est sous la forme ascii haut (le 7e bit est à 1).

Exemples :

— «A» codé : $C1
— «1» codé : $B1

■ « (guillemet) suivi par «^» puis par une lettre la donnée est sous forme de controle acsii haut (le bit de poids fort ou 7e bit est à 1).

Exemples :

— «^C» codé : $83
— «^D» codé : $84

■ ' (apostrophe) suivi par une lettre ou un signe autre que «^» la donnée est sous forme ascii bas (le 7e bit est à 0).

Exemple :

— 'A codé : $41

■ ' (apostrophe) suivi par «^» puis suivi par une lettre.
La donnée est sous forme de controle acsii bas (le bit de poids fort ou 7e bit est à 0).

■ label.
La donnée est un nombre sous forme de variable.

■ «★»
Le nombre représentant la donnée est l'adresse actuelle de l'assemblage.

■ «&»
Le nombre représenté est le numéro de la ligne actuelle.

Un opérateur

C'est un signe ou un symbole placé entre deux données, et, indiquant le type d'opération effectuée entre ces deux opérandes.

■ «+» addition
■ «–» soustraction
■ «★» multiplication
■ «/» division
■ «!» ou exclusif

Une valeur

C'est un nombre compris entre 0 et 65535.
Si le premier caractére est :

■ «<» : on prend la partie basse du nombre généré, qui sera comprise entre 0 et 255.

■ «>» : on prend la partie haute du nombre généré, qui sera comprise entre 0 et 255.

Autres caractères de la valeur :

■ une donnée ou pas de donnée suivie par un opérateur.

Exemples :

— $34 codé $0034
— >$34 codé $00
— <$1234 codé $34
— 12+4–8 codé $0008

Une chaîne

■ 1er caractère :

« ou '

■ autres signes :

Tous les signes, sauf le ;

■ Dernier caractère :

« ou '

délimitation des champs

Exemple :

Nous avons la ligne d'instructions suivante :

PLOT LDA £$00 ; initialise A

(1) (2) (3) (4)

Nous pouvons distinguer quatre champs, notés ici 1, 2, 3 et 4.

PLOT	LDA	£$00	; initialise A
champ des labels (1)	champ des mnémoniques (2)	champ des opérandes (3)	champ des commentaires (4)

Comme sur tous les assembleurs, la ligne générant du code, répond à des caractèristiques standards, imposées par le langage 6502.

On sépare la ligne en 4 champs :
le champ des labels
le champ des mnémoniques
le champ des opérandes
le champ des commentaires

Comme vous pouvez le voir sur l'exemple, une ligne commence par le champ label, ceci jusqu'au 1er espace. Ensuite vient le champ mnémonique jusqu'au 2e espace, puis le champ opérande et ceci jusqu'au « ; » que vous trouvez, qui annonce l'entrée dans le champ des commentaires.

Certaines lignes ont une structure spéciale :
— une ligne qui commence par « ★ » n'a qu'un champ commentaire, affiché sans tabulation.
— une ligne qui commence par « ; » n'a qu'un champ commentaire tabulé.

Structure d'une ligne

Le champ des labels

Le champ des labels sert à affecter des labels, c'est-à-dire à définir le nombre compris entre 0 et 65535 qu'ils vont représenter.
Normalement, quand le label d'une ligne est dans le champ des labels, c'est l'adresse où est assemblée cette ligne qui lui est affectée.

ProCODE possède plusieurs types de labels.
Pour le programmeur, il est souvent plus important de savoir par exemple, qu'un label est local, que de connaître sa fonction.

Les divers labels

On en distingue 4 types.

Les labels locaux :

Les labels locaux, permettent d'éclaircir les sources et de diminuer l'espace mémoire occupé. Ils peuvent être déclarés plusieurs fois. De plus, sachez q'un label local, n'est reconnu que si on ne doit pas croiser la déclaration d'un label général pour le trouver (ce qui revient à dire, que la zone de reconnaissance d'un label local, est placée entre deux déclarations de labels généraux).
On peut avoir ainsi, plusieurs labels locaux différents, placés entre deux labels généraux. Il est également possible de redéfinir les mêmes labels locaux à des emplacements différents dans le programme.
On utilise généralement les labels locaux pour les branchements intérieurs à un sous-programme.

Exemple :

```
EFFACE    LDX   £$00
          LDA   £$00
§BOUCLE   STA   $200,X
          INX
          BNE   §BOUCLE
          RTS
REMPLI    LDX   £$00
          LDA   £$FF
§BOUCLE   STA   $200,X
          INX
          BNE   §BOUCLE
          RTS
```

Les labels généraux :

Les labels généraux sont définis une fois pour toute. Ainsi,au cours d'un source, il n'est pas possible de les affecter à une autre adresse que celle qui leur a été initialement fixée.

Les labels redéclarables par SET

(c'est-à-dire affectés par l'instruction SET) :

Pouvant être affectés de nombreuses fois dans un même programme, ils ne représentent pas toujours une seule valeur mais plusieurs.
Un label affecté par l'instruction SET à une adresse «X», représentera l'adresse «X», tant que ce même label n'est pas affecté à une autre adresse.
En fonction de l'endroit où l'on se trouve, il représente donc la dernière valeur qui lui a été affectée.

Exemple :

```
MOVE  SET  $1000
      LDA  MOVE      (ici, MOVE vaut $1000)
      PLA
      TAY
MOVE  SET  $2000
      LDA  MOVE      (ici, MOVE vaut $2000)
```

Les labels redéclarables :

Il est possible de les utiliser comme des labels généraux, et il est également possible de les redéclarer plusieurs fois.
On définit un label comme redéclarable par l'instruction MDF.

Exemple :

```
RE  MDF
RE  EQU  $357
    LDA  £30
    STA  RE      ; RE représente l'adresse $357
    NOP
RE  EQU  $666
    STA  RE      ; RE représente l'adresse $666
```

Le champ des mnémoniques

Le champ des mnémoniques contient le nom de la fonction 65C02 dans le cas d'une ligne de codes, sinon, on trouve le nom de l'instruction assembleur ou le nom d'une MACRO.

Divers mnémoniques autorisés :

Liste des mnémoniques assembleur 65C02 :

ADC	BRK	DEC	LDX	PLX	STX
AND	BVC	DEX	LDY	PLY	STY
ASL	BVS	DEY	LSR	ROL	STZ
BCC	CLC	EOR	NOP	ROR	RAX
BCS	CLD	INA	ORA	RTI	TAY
BEQ	CLT	INC	PHA	RTS	TRB
BIT	CLV	INX	PHP	SBC	TSB
BMI	CMP	INY	PHX	SEC	TSX
BNE	CPX	JMP	PHY	SED	TXA
BPL	CPY	JSR	PLA	SEI	TXS
BRA	DEA	LDA	PLP	STA	TYA

Sont également autorisées, les instructions assembleur répertoriées pages 34 à 50.

Le champ des opérandes

Le champ des opérandes (1) contient, dans le cas d'une ligne de fichier code, les formes ci-après :

```
£valeur
valeur,x
valeur,y
```

(valeur),y
(valeur,x)
(valeur)
valeur

Ceci détermine le mode d'adressage à utiliser pour l'instruction.

(1) : une opérande est une information utilisée par l'instruction assembleur et qui en général la complète.

Le champ des commentaires

Il contient des indications sur le programme qui, même si elles ne vous paraissent pas indispensables, sont très utiles. En effet une autre personne peut être amenée à consulter votre programme pour des raisons sérieuses et les commentaires que vous aurez mis lui seront d'un grand secours pour une bonne compréhension du programme et lui faciliteront le travail.
Tous les caractères sont autorisés, et la longueur de ce champs n'est pas limitée.

Néanmoins, certains champs sont optionnels :

— si aucun label n'est dans le champ des labels, l'adresse d'assemblage ne sera pas affectée.
— s'il n'y a rien dans le champ des opérandes, l'adressage sera inhérent.
— le champ des commentaires est facultatif.

Conclusion :

Récapitulation de la syntaxe

On peut schématiser une ligne de la manière suivante :

label – mnémonique – opérande – ;commentaire
(Les espaces ont été symbolisés par « – »)

Les espaces sont tous très importants, à l'exception de celui placé avant le «;» qui est optionnel.
Mais, comme certains champs sont optionnels ont peut également avoir :

– mnémonique – ;commentaire
ou
– mnémonique
ou
;commentaire
etc.

— le «label» est affecté à l'adresse actuelle de la ligne.

— le «mémonique» doit être :
- une instruction 65C02
- une instruction assembleur
- une MACRO définie auparavant

— l'«opérande» doit correspondre à l'instruction employée.
  - pour une instruction 65C02, elle comprend le mode d'adressage et l'adresse, sous forme définie comme <valeur>.
  - pour une instruction assembleur, elle doit suivre la syntaxe précise de cette instruction.
  - pour une MACRO, elle doit contenir les paramètres.

— le «commentaire» renseigne sur ce que fait la ligne de programme, ajoutant une clarté indispensable quand on travaille en assembleur.

Instructions reconnues par l'assembleur

Contrôle de l'assemblage

Les instructions de contrôle d'assemblage permettent au programmeur d'agir directement sur l'assembleur et indirectement sur le code.
Elles ont la particularité de ne pas générer de code mais d'affecter des pointeurs internes à l'assembleur.

Note :
— Dans ce qui va suivre, tout ce qui est entre ( ), est obligatoire, et, ce qui est entre <> est facultatif.

— Les espaces sont symbolisés par –.

EQU ou =

(label)–EQU–(valeur)<;commentaire>
(label)–=–(valeur)<;commentaire>

Affecte le label mis au début de la ligne, à la valeur spécifiée par l'opérande. Le label est de type simple standard et utilisable dans le champs des opérandes. Il est possible de définir ainsi, des labels locaux.
Le champs des opérandes ne doit pas contenir de labels non déclarés précédemment dans le source.
L'instruction est active si do est à 1; elle est listée.

Exemples :

```
COUT    =       $FDED
IN      EQU     $200
```

SET

(label)–SET–(valeur)<;commentaire>

Cette instruction est semblable à EQU mais il est possible de redéfinir plusieurs fois le même label (contrairement à EQU). En effet il est souvent très pratique lors de la conception d'un long programme, de pouvoir utiliser le même nom de label dans plusieurs parties mais sans qu'il soit affecté à la même adresse, ceci est possible grâce à SET.
Il est impossible de redéclarer par SET un label déjà déclaré par EQU.
L'instruction est active si do est à 1; elle est listée.

Exemple :

```
RE   SET  $453
     PHA
     PLA
     LDA  RE   ;L'adresse chargée dans l'accumulateur est $453
RE   SET  $999
     NOP
     NOP
     LDA  RE   ;L'adresse chargée dans l'accumulateur est $999
```

MDF

(label)–MDF<–;commentaire>

L'instruction MDF permet de définir un label comme redéclarable, mais sans lui affecter de valeur.
Cette instruction nous permet de garder (avec précaution) le même nom pour plusieurs adresses, définies par EQU, ou, simplement avec un label dans une ligne de codes.
Il faut utiliser cette instruction avec précaution, car tout label défini ainsi peut apparaître plusieurs fois dans le champs des labels.
L'instruction est active si do est à 1; elle est listée.

Exemple :

```
MI   MDF
MI   EQU  $566
     PHA
     PLA
     LDA  MI   ;on charge dans l'accumulateur l'adresse $566
MI   EQU  $999
     NOP
     NOP
     LDA  MI   ;on charge dans l'accumulateur l'adresse $999
```

ORG

–ORG–(valeur)<;commentaire>

Définit l'adresse de la prochaine instruction assemblée.
Le code généré se chargera à l'adresse spécifiée.
Un label placé avant l'instruction ne sera pas affecté.
La valeur peut contenir des labels mais ils doivent être déclarés auparavant dans le source.
L'instruction est active si do est à 1 elle est listée.

Exemple :

```
     ORG  $786
FA   NOP        ;le label FA est affecté a l'adresse $786
```

LST OFF

-LST-OFF

Après l'exécution de cette instruction, le listing sur imprimante ou à l'écran est interrompu; il ne reprendra qu'après une instruction LST ON. L'instruction est active en passe 2; elle n'est pas listée.

LST ON

-LST-ON

Rétablit l'affichage du listing après une instruction LST OFF.
L'instruction est active en passe 2; elle n'est pas listée.

LMC OFF

-LMC-OFF

Il peut s'avérer nécessaire de ne pas lister le code généré par la MACRO (gain de place et compréhension optimale) ; cette instruction indique à l'assembleur de lister uniquement l'instruction comprenant la MACRO, mais de ne pas lister chaque instruction de cette MACRO.
L'instruction est active en passe 2; elle n'est pas listée.

LMC ON

-LMC-ON

Rétablit le listage complet des MACRO après une instruction LMC OFF.
L'instruction est active en passe 2; elle n'est pas listée.

SOUND

-SOUND<-;commentaire>

Qaund un fichier met un certain temps à être assemblé, on est souvent tenté de faire autre chose ; on quitte alors l'écran des yeux ou même parfois on quitte la pièce !
Si l'assembleur trouve une erreur, il émet un petit bruit. Dans le cas où l'on est pas à côté du système, ce n'est toutefois pas suffisant. En plaçant au début du programme une instruction SOUND, ce n'est plus un petit bruit qui est émis par le haut-parleur, mais une véritable sirène qui retentit, aussi longtemps que l'on n'a pas appuyé sur une touche du clavier.
L'instruction est toujours active ; elle est listée.

FXLBL

-FXLBL-(valeur)<;commentaire> Pour la version 64K

Il arrive que la place réservée pour les labels soit insuffisante, il faut alors déplacer le début de la table des labels à une nouvelle adresse, cette instruction le permet : la valeur devient l'adresse du début de la

table des labels, ProCODE vérifie avant d'exécuter cette instruction que l'adresse est possible et qu'elle ne détruit ni un source, ni le code, ni les routines ProCODE.
L'instruction est active en passe 1; elle est listée.

Exemple :

FXLBL $5800

Une fois ce programme assemblé, le menu général affiche dans la colonne <LABEL> $5800 -> $5800

FXCD

-FXCD-(valeur)<;commentaire> Pour la version 64K

Il arrive que la place réservée pour le code soit insuffisante, il faut alors déplacer le début de la zone réservée au code à une nouvelle adresse, cette instruction le permet : la valeur devient l'adresse du début de la zone code, ProCODE vérifie avant d'exécuter cette instruction que l'adresse est possible et qu'elle ne détruit ni un source, ni la table des labels, ni les routines ProCODE.
L'instruction est active en passe 1; elle est listée.

Vous pouvez vous référer à l'exemple précédent.

Instructions de présentation

Les instructions de présentation permettent au programmeur d'avoir un listing clair et facilement compréhensible.
Elles limitent le nombre d'octets nécessaires pour réaliser une bonne présentation.
Elles ne génerent pas de codes, et la plupart ne sont pas listées, car ce n'est pas l'instruction qui est intéressante mais son résultat sur le listing.

CHR

-CHR-(valeur)<;commentaire>

La valeur définit le caractère répété par REP (voir ci-dessous).
L'opérande ne doit pas contenir de labels non déclarés précédemment dans le source.
L'instruction est active en passe 2 si LST est à 1; elle n'est pas listée.

REP

-REP-(valeur)<;commentaire>

Cette instruction repète «valeur» fois le caractère défini par CHR.
L'opérande ne doit pas contenir de labels non déclarés auparavant dans le source.
L'instruction est active en passe 2 si LST est à 1; elle n'est pas listée.

Exemple :

```
CHR  «=»
REP  5      ;Sur le listing vous aurez =====
```

Vous aurez donc, 5 fois le signe =

SKIP

-SKIP-(valeur)<;commentaire>

On saute «valeur» lignes en exécutant cette instruction.
L'opérande ne doit pas contenir de labels non déclarés précédemment dans le source.
L'instruction est active en passe 2 si LST est à 1; elle n'est pas listée.

PAG

-PAG-<;commentaire>

Le papier avance jusqu'en haut de la page suivante.
Cela permet par exemple, de définir les labels sur la ou les premières pages et de commencer le code en haut d'une page vierge.
L'instruction est active en passe 2 si LST est à 1; elle n'est pas listée.

AST

-AST-(valeur)<;commentaire>

Cette instruction envoie «valeur» astérisques dans le listing, ce qui permet de faire des cadres en un minimum d'octets.
L'instruction est active en passe 2 si LST est à 1; elle n'est pas listée.

Exemple :

```
AST  2      ;sur le listing apparaitra **
```

PTC

-PTC-(valeur)<,valeur><...><;commentaire>

Quand on liste un programme sur une imprimante, la plupart du temps, on ne la paramètre pas.
Il est pourtant pratique de pouvoir le faire dans certaines conditions, en lui envoyant une chaîne de caractères de contrôle ou tout autre caractère. ProCODE le permet, grâce à l'instruction PTC suivie par tous les codes des caractères à envoyer séparés par des virgules.
L'instruction est active en passe 2; elle n'est pas listée.

Exemple :

```
PTC  «A»,«B»  ;Les caracteres AB apparaîtront lors de
              l'assemblage
```

TAB

-TAB-(valeur)<,valeur><...><;commentaire>

Il n'est pas toujours pratique d'avoir les tabulations prédéfinies par ProCODE. Cette instruction vous permet d'en définir des nouvelles. Chaque valeur indique un paramètre. Il n'est pas nécessaire de tous les indiquer, seuls ceux que vous décrirez y seront.

Le numéro du caractére où est aligné le texte, définit une tabulation :

— la 1re valeur définit le nombre d'octets code écrits par ligne ; la valeur par défaut est de 3. Une valeur de 0 aura comme effet de ne pas lister les codes et ne fera apparaître que les adresses.

— la 2e valeur définit la tabulation, pour l'affichage du numéro de ligne (il prend toujours 5 caracteres).

— la 3e valeur définit la tabulation du premier caractère du label, éventuellement de la ligne.

— la 4e valeur définit la tabulation du premier caractère du mnémonique.

— la 5e valeur définit la tabulation du premier caractère de l'opérande.

— la 6e valeur définit la tabulation du premier caractère (;) du commentaire.
L'instruction est active en passe 2 si LST est à 1 ; elle n'est pas listée

PAU

-PAU<-;commentaire>

Cette instruction n'est pas réellement «de présentation» car elle ne change pas le listing. Elle permet d'arrêter l'assemblage jusqu'à ce que l'utilisateur ait appuyé sur une touche.
Elle se révèle très pratique pour attirer l'attention sur un passage précis du programme, ou pour réaliser un listing sur plusieurs types de papiers.
Quand l'assembleur attend à cause d'une instruction PAU, un curseur apparaît comme premier caractère d'une ligne et le listing est interrompu.

L'instruction est active en passe 2 si LST est à 1 ; elle n'est pas listée.

Instructions génératrices de code

Il est indispensable de pouvoir placer dans un code, des tableaux, des suites de données, des messages d'écran...
Ceci est possible grâce aux instructions d'assemblage générant un code.

DS

(label)–DS–(valeur1)<,valeur2><;commentaire>

Cette instruction permet de placer un nombre important d'octets à la même valeur ; la valeur 1 indique le nombre d'octets à placer dans le code, la valeur 2 est optionnelle (par défaut elle est égale à 0) et permet de définir la valeur écrite dans les octets réservés.
Une utilisation un peu différente peut être d'un grand intérêt. On remarque en effet assez souvent, que lors de la définition des variables utilisées par un programme, ce ne sont pas les adresses qui nous interessent, mais uniquement le fait qu'elles ne se chevauchent pas. Dans ce cas,l'instruction EQU devient très lourde à utiliser, obligeant à avoir en opérande le dernier label déclaré plus le nombre d'octets qu'il occupe. C'est alors que DS combiné à ORG peut rendre de grands services :
L'adresse du premier label est définie par ORG, puis, tous les autres labels seront définis par l'instruction DS, qui contiendra en opérande, le nombre d'octets réservés pour ces labels.
L'instruction est active si do est à 1 ; elle n'est pas listée.

Exemple :

```
        ORG   $70
ACC1    DS 4
ACC2    DS 4
VAR     DS 1
VAR2    DS 1
        ORG   $1000
DEBUT   LDA   £$00
```

Dans ce cas ACC1 vaut $70
ACC2 vaut $74
VAR vaut $78
VAR2 vaut $79
DEBUT vaut $1000

Une fois, la ou les définitions terminées, le programme débutera par ORG, remettant le code inutilement généré à 0.

L'opérande ne doit pas contenir de labels non définis auparavant.
L'instruction est active si do est à 1 ; elle est listée.

DFB

(label)–DFB–(valeur)<,valeur><,valeur><...><;commentaire>

Pour chaque valeur, DFB génere un octet de code qui correspond à la partie basse de cette valeur.
Ainsi, on peut par exemple définir :

— des tableaux de constantes
— des tables contenant les parties basses d'adresses de sous-programmes, auxquels on accède indirectement.

C'est l'instruction de base pour l'insertion de données dans un programme. Toutes les autres instructions, comme DA, DDB,... peuvent être remplacées par une ou plusieurs instructions DFB correctement utilisées.
On peut ainsi définir un grand nombre de valeurs. Des labels déclarés n'importe où dans le source sont autorisés.
L'instruction est active si do est à 1; elle est listée.

Exemple :

```
DFB   64,$10+1    ;le code généré est : $40,$11
```

HBY

(label)–HBY–(valeur)<,valeur><,valeur><...><;commentaire>

Comme pour DFB, mais c'est la partie haute qui est codée.
L'instruction est active si do est à 1; elle est listée.

Exemple :

```
HBY   $4523,256   ;le code généré est : $45,$1
```

DA

(label)–DA–(valeur)<,valeur><,valeur><...><;commentaire>

Instruction semblable à DFB, mais chaque valeur génère deux octets : en premier la partie basse de la valeur, puis sa partie haute.
L'instruction est active si do est à 1; elle est listée.

Exemple :

```
DA    $1234,$5678   ;le code généré est : $34,$12,$78,$56
```

DDB

(label)–DDB–(valeur)<,valeur><,valeur><...><;commentaire>

Instruction semblable à DA, mais la partie haute est insérée avant la partie basse.
L'instruction est active si do est à 1; elle est listée.

Exemple :

```
DDB   $1234,$5678   ;le code généré est : $12,$34,$56,$78
```

HEX

(label)–HEX (0 à FF)<0 à FF><...><;commentaire>

C'est une instruction différente des précèdentes, car son opérande n'est pas semblable.
HEX est utilisé uniquement pour créer des tables de données identiques aux DATA du Basic.
C'est une instruction qui simplifie la définition d'un grand nombre de données comparée à l'utilisation de DFB ou DDB.

L'opérande est constituée par une suite de chiffres ou de lettres comprises entre A et F. Chaque série de deux signes définit un octet par son contenu hexadécimal.

Exemple :

```
HEX   A6B7EF56   ;le code généré est : $A6,$B7,$EF,$56
```

ASC

(label)-ASC-(chaîne)<;commentaire>

Grâce à cette instruction, il est possible d'insérer du texte dans le fichier code.
Chaque caractère de la chaîne est codé en ASCII et stocké comme code généré.
L'instruction est active si do est à 1; elle est listée.

Exemple :

```
ASC   «ABC»   ;le code généré est : $C1,$C2,$C3
ASC   'ABC'   ;le code généré est : $41,$42,$43
```

DCI

(label)-DCI-(chaîne)<;commentaire>

Instruction semblable à ASC, mais le dernier caractère de la chaîne a son bit 7 à l'opposé des 7** bits des caractères précédents.
L'instruction est active si do est à 1; elle est listée.

Exemple :

```
DCI   «ABC»   ;le code généré est : $C1,$C2,$43
DCI   'ABC'   ;le code généré est : $41,$42,$C3
```

PCI

(label)-PCI-(chaîne)<;commentaire>

Instruction semblable à ASC, mais le premier caractère de la chaîne a son bit 7 à l'opposé des 7** bits des caractères suivants.
L'instruction est active si do est à 1; elle est listée.

Exemple :

```
PCI   «ABC»   ;le code généré est : $41,$C2,$C3
PCI   'ABC'   ;le code généré est : $C1,$42,$43
```

STR

(label)–STR–(chaîne)<;commentaire>

Instruction semblable à ASC, mais le nombre de caractères de la chaîne est placé avant le codage des caractères.
Cette instruction est très utile sous prodos pour définir des noms de fichiers.
L'instruction est active si do est à 1; elle est listée.

Exemple :

```
STR  «ABC»   ;le code généré est : $03,$C1,$C2,$C3
STR  'ABC'   ;le code généré est : $03,$41,$42,$43
```

INV

(label)–INV–(chaîne)<;commentaire>

Instruction semblable à ASC mais :
— si la chaîne est en ASCII haut (bit 7 à 1) le codage la met en inverse : affichage noir sur fond blanc.
— si la chaîne est en ASCII bas (bit 7 à 0) le codage la met en flash : affichage clignotant.
Il faut prendre garde à l'utilisation des minuscules dans la chaîne.
L'instruction est active si do est à 1; elle est listée.

Exemple :

```
INV  «ABC»   ;le code généré est : $01,$02,$03
INV  'ABC'   ;le code généré est : $41,$42,$43
```

Instructions de chaînage

Souvent, lorsque l'on travaille sur de longs sources, l'espace mémoire est une donnée critique. ProCODE vous permet, grâce aux instructions dites de chaînage ou d'accès relatif, de diminuer considérablement les problèmes causés par une place mémoire limitée.
C'est ainsi qu'il devient possible d'assembler des fichiers d'une taille supérieure à ce qui est chargeable en mémoire, ceci grâce à l'utilisation du disque.

DSK

–DSK–(nom du fichier)

On rencontre parfois des situations très critiques, où les fichiers assemblés sont de grande taille; on a alors recours, sur la version 64 k, aux instructions FXLBL et FXCD. Sur la version 128 K, l'espace diponible permet d'assembler sans problème des fichiers comparables.

Si malgré ces instructions l'assemblage n'est pas possible et qu'un message «code généré trop long» apparait sur l'écran, ne capitulez pas. Il suffit d'insérer au début du programme une instruction DSK, suivie du nom du fichier, qui sera écrite sur le volume et sur le directory en ligne.
L'espace en mémoire ne permettant pas au code de tenir entièrement, il sera écrit à chaque débordement, sur le disque, créant ainsi un fichier code sans passer par la commande éditeur «O».

Cette instruction peut également être simplement utilisée pour écrire le code sur le disque sans avoir à s'en préoccuper.
L'assembleur crée deux fichiers, les labels et le code. S'il est primordial que la table des labels réside en mémoire (temps de recherche très court) il n'en est pas de même pour le code.
En effet le code est écrit mais jamais relu. On comprend alors très bien l'intérêt de ne pas avoir tout le code en mémoire mais seulement une partie, suffisante pour servir de tampon, le reste étant écrit sur le disque.
Si plusieurs instructions DSK sont placées dans un fichier, seule la première est prise en compte.
L'instruction est active en passe 2 ; elle n'est pas listée.

Exemple :

```
DSK   FICHIER.C   ; dès que le code remplit la mémoire, il est
                    sauvé.
```

PUT

-PUT-(nom du fichier)<;commentaire>

Que faire lorsque de longs sources entrent en mémoire et qu'il ne reste plus suffisamment de place pour les labels. (Ces conditions sont toutefois assez rares, car le plus souvent un long programme est divisé en plus petits fichiers ; de plus sur la version 128 K, un source peut occuper 46 K et être sans aucune difficulté assemblable).

On a recours à l'instruction PUT de ProCODE, qui permet la cohabitation en mémoire, à un même instant, de trois sources. (explication plus détaillée dans le chapitre réservé au menu).

Lorsque l'assembleur rencontre une instruction PUT, il charge le fichier dont le nom est spécifié par l'opérande et l'assemble comme s'il se trouvait écrit à la place de l'instruction PUT. Après l'avoir assemblé il revient à la suite du programme principal. Tout se passe en fait comme si à la place de «PUT fichier», on avait eût le source «fichier» écrit en toutes lettres.
Si dans un fichier on a recours successivement à plusieurs PUT, l'espace occupé sera celui du plus long des fichiers; en effet les fichiers ne cohabitent pas dans la mémoire mais y passent les uns après les autres au fil de l'assemblage. On peut donc assembler des fichiers démesuremment grands comparés à la place disponible en mémoire.

Exemple :

```
PUT  FICHIER1
PUT  FICHIER2
PUT  FICHIER3
```

Tout se passe comme si on demandait d'assembler un unique fichier composé de l'adjonction de 3 fichiers.

Assemblage conditionnel

L'assembleur est utilisé lorsque l'on nécessite d'une grande rapidité ou du contrôle de fonctions d'e/s, ce qui serait irréalisable dans un autre langage.
Les routines (petits programmes) assembleurs ont des usages très spécifiques et même parfois très limités. Cependant elles doivent pouvoir s'adapter très facilement à un usage précis leur permettant alors de s'intégrer totalement au programme qui les utilise.
On peut imaginer de faire une multitude de routines, chacune possédant des caractéritiques propres. Cette solution est peu pratique : grande place d'archivage pour peu de différences entre les diverses variantes d'un même programme.
On a alors recours à un outil très puissant : l'assemblage conditionnel, un nom barbare, désignant une souplesse inhabituelle pour des langages de bas niveau.
Grâce aux instructions «conditionnelles», il est possible avec le même fichier source d'assembler plusieurs variantes d'un même programme.

DO

-DO-(valeur)<;commentaire>

En fonction de l'opérande, le drapeau «do» (1), interne à l'assembleur, est modifié ou non : si l'opérande est différente de 0 le drapeau «do» n'est pas modifié, par contre, si l'opérande est égale à 0, le drapeau «do» passe à 0.

Mais que permet donc ce drapeau ?
Il contrôle la génération des octets constituant le fichier code. S'il est à 1, toutes les instructions générant du code sont actives, par contre, s'il est à 0, le code n'est plus généré, les labels ne sont plus affectés, la syntaxe n'est plus vérifiée, seules certaines instructions sont encore exécutées. (Il est spécifié quand les instructions sont actives, on peut donc savoir si «do» intervient).
L'opérande ne doit pas contenir de labels non définis précédemment.
L'instruction est active si do est à 1.

(1) drapeau : indicateur pouvant prendre deux états.
état actif ou ON (actif 1)
état passif ou OFF (passif 0)

Les drapeaux les plus utilisés en assembleur sont :

```
LST.  MAC
LST
DO
```

ELSE

-ELSE<-;commentaire>

Cette instruction inverse l'état du drapeau «do» : s'il était à 1 il passe à 0 et réciproquement, s'il était à 0 il passe à 1.
On peut donc assembler des parties de programme, différentes en fonction de certaines constantes, qui figurent dans l'opérande du DO.
L'instruction est toujours active.

FIN

-FIN<-;commentaire>

Cette instruction rétablit le drapeau «do» à la valeur 1.
Après cette instruction, le code est toujours généré.
Elle termine les parties de programmes assemblées conditionnellement.
L'instruction est active si do est à 0.

Les MACRO instructions

Il est très fréquent que dans un même programme des suites d'instructions reviennent plusieurs fois, sans qu'il soit possible de les mettre en sous programmes. De plus, la manipulation de pointeurs 16 bits implique des incrémentations, des décrémentations, des additions... enfin, plusieurs calculs plus ou moins complexes.
Comme il n'existe pas d'instruction 6502 ou 65C02 permettant ces manipulations, on définit souvent de nouvelles instructions appelées MACRO. Elles sont constituées d'une suite d'instructions et exécutent une fonction utilisée plusieurs fois dans le programme.
Le langage d'assemblage devient bien plus facile à manipuler, et l'on peut (fictivement) rajouter des instructions au processeur.

Exemples de MACRO :

a) Avec LMC OFF

```
        DO    0        ;On interrompt la génération du code
ICC     MAC            ;La macro débute ici
        INC   .0       ;Incrémentation de la partie basse
        BNE   §FIN
        INC   .0+1     ;Incrémentation de la partie haute
§FIN    EOM            ;Termine la macro
        FIN
        ORG   $20
VAR     DA    0
        ORG   $1000
DEBUT   LDA   £00
        STA   VAR
        STA   VAR+1
        ICC   VAR      ;On incrémente la variable sur deux
        RTS            octets
```

b) Avec LMC ON

```
        DO    0
ICC     MAC
        INC   .0
        BNE   §FIN
        INC   .0+1
§FIN    EOM
        FIN
        ORG   $20
VAR     DA    0
        ORG   $1000
DEBUT   LDA   £00
        STA   VAR
        STA   VAR+1
ICC     MAC
        INC   .0
        BNE   §FIN
        INC   .0+1
        ICC   VAR
        RTS
```

Les MACRO peuvent se définir ainsi :

— ce sont des instructions propres à l'assembleur. La place mémoire utilisée dans le code par une MACRO est équivalente à la place mémoire des instructions qu'elle remplace.
— ce ne sont donc pas des sous programmes car en plus, elles peuvent être assemblées différemment en fonction des paramètres passés.

Paramètres dans les MACRO instructions

On voit ici une ressemblance avec les sous-programmes auxquels on passe des paramètres ; mais que l'on ne se trompe pas la différence est énorme. Les paramètres passés en MACRO, ne sont pas des données passées par les registres ou par des variables, ce sont des artifices existant uniquement dans l'assembleur et qui modifient le code généré. Comme l'opérande d'une instruction modifie le code de cette instruction, les paramètres d'une MACRO modifient le codage de cette MACRO en 6502.
IL est bien difficile d'expliquer à quoi correspondent ces paramètres. Ils se rapprochent des variables locales ou redéclarables mais possèdent un nombre important de différences liées à la récursivité des MACRO (nous verrons cela en détail un peu plus tard).
Un paramètre de MACRO est une variable contenant un nombre codé sur 16 bits, comme n'importe quelle variable.

Syntaxe de cette variable

«.0»

— le point indique que c'est un paramètre MACRO
— il est suivi par un nombre décimal compris entre 0 et 255 (en limite avec la place disponible dans la pile des paramètres).

Ce nombre indique le numéro du paramètre; ainsi .0 contient une valeur différente de .1 elle même contenant une valeur différente de .2 (si elles ont été affectées à des valeurs différentes).

Utilisation de cette variable

Elle n'est utilisable que dans la MACRO pour laquelle elle a été définie.
Une fois sorti de la MACRO, cette variable n'est plus définie.
Elle ne peut pas être affectée à une adresse ou à une constante ailleurs que dans l'appel même de la MACRO.
Elle est utilisable comme n'importe quelle variable dans des instructions, des calculs...

Définition des MACRO

On définit une MACRO par l'instruction MAC, précédée dans le champ label, du nom de la MACRO définie.

▲ Attention

On ne peut pas définir de MACRO avec un nom déjà réservé, aussi bien dans un label que dans les instructions 65C02.
Si vous essayez de définir une MACRO avec un nom correspondant à un label, un message d'erreur vous avertira. Par contre, si vous définissez une MACRO du même nom q'une instruction assembleur, ce sera toujours l'instruction assembleur qui sera prioritaire.
Les labels locaux prenant référence à partir des labels généraux, il faut faire attention à ce qu'il existe au moins un label général entre deux appels d'une même macro sinon, lors de la deuxième macro, ProCODE affichera : label double...

Syntaxe de définition :

— La génération de code doit être arrêtée («do» mis à 0 par une instruction DO 0).

— Le nom de la MACRO est défini, suivi par l'instruction MAC.

— La MACRO peut contenir des labels utilisés dans le source; si elle définie des labels, ils doivent être locaux car si la même MACRO est utilisée plusieurs fois il y aura des labels doubles.
Une MACRO ne peut pas être définie dans un fichier PUT (fichier non résident mais chargé lors de son assemblage), un message d'erreur vous l'indiquera.
Elle peut utiliser des paramètres définis (ou affectés comme vous préférez) lors de l'appel de cette MACRO.
Elle peut même appeler une autre MACRO et ceci jusqu'à 10 appels imbriqués.
Pour finir, elle peut s'appeler elle-même, formant une MACRO dite récursive.

— Elle doit se terminer par l'instruction EOM.
On ne doit pas oublier à la fin des définitions, de rétablir la génération du code par l'instruction FIN.

Utilisation des MACRO

En effet, à quoi servirait de pouvoir définir des MACRO, de pouvoir leur passer des paramètres si on ne pouvait pas les utiliser.
Une MACRO définie, peut être utilisée n'importe où après dans le source, une ou plusieurs fois.
L'assembleur remplace la MACRO (qui occupe une seule ligne) par la suite d'instructions qui lui a été attribuée. Ces instructions sont assemblées et si des variables paramètres MACRO sont utilisées elles sont remplacées par la valeur qui leur a été affectée à l'appel de la MACRO.
Une MACRO s'appelle simplement en utilisant comme mnémonique le nom sous lequel elle a été définie.
Les paramètres sont placés dans le champs des opérandes.

Syntaxe du passage des paramètres :

— Ils sont dans le champs des opérandes de la MACRO appelée
— ils sont séparés par une virgule
— un paramètre est une valeur (variable, calcul, constante...)
Le 1er paramètre est stocké dans la variable «.0».
Le suivant sera écrit dans «.1» puis «.2» et ainsi de suite.

Remarque : Quand une MACRO en appelle une autre (ou la même), elle peut lui donner un de ses paramètres MACRO (ceci est montré en exemple). Normalement le code généré par une MACRO n'est pas listé, seule la MACRO apparait et le code affiché est inexistant, ceci permet (quand on utilise des grosses MACRO) d'optimiser la compréhension. Pourtant, il s'avère très utile (déjà pour bien comprendre ce qu'est et ce que fait une MACRO) de pouvoir lister totalement la MACRO ainsi que le code qu'elle génère, pour cela, utilisez l'instruction LMC ON.

Récapitulation sur les MACRO

Une MACRO instruction remplace une suite d'instructions; elle est assemblée comme si ces instructions étaient écrites (réellement) dans le source (d'où diminution de la taille du source).
La définition des MACRO doit être dans un fichier résident (pas dans un fichier appelé par l'instruction PUT).
Le nom de la MACRO ne doit pas être un label déjà défini ni une instruction reconnue par l'assembleur.

Les MACRO doivent : être définies avec DO à 0
débuter par MAC
terminer par EOM.

Une MACRO ne peut affecter que des labels locaux.

Une MACRO peut utiliser des labels définis dans le source ou dans la MACRO, des MACRO définies avant, des paramètres définis lors de son appel.
Une MACRO peut être appelée autand de fois qu'on le désire. Après avoir été définis, tous les paramètres dont elle a besoin doivent être passés.

MAC

(label) – MAC< – ;commentaire>

Cette instruction définit une MACRO. Le nom de la MACRO est un label.
La MACRO va remplacer les instructions suivantes jusqu'au EOM :
L'instruction est active si DO est à 0, elle est listée.

EOM

(label) – EOM< – ;commentaire>

Cette instruction détermine la fin de la MACRO. Le label sera affecté à la 1re instruction après la MACRO.
L'instruction est active si DO est a 1 et que l'on est dans une MACRO, elle est listée.

(MACRO)

(label) – (MACRO) – <valeur 0><,valeur 1><...><;commentaire>

Si le nom ,ici représenté par MACRO, n'est pas une instruction assembleur, ceci met dans le fichier code la MACRO définie sous le nom MACRO et on lui passe les paramètres <valeur 0> dans «.0» <valeur 1> dans «.1»...
Un label peut être affecté à la 1ere instruction de la MACRO.

Exemple :

```
        DO    0       ;On interrompt la génération du code
SOL     MAC           ;La macro commence ici
        INC   .0      ;Incrémentation de la partie basse
        BNE   §1
        INC   .0+1    ;Incrémentation de la partie haute
§1      EOM           ;Termine la macro
        FIN
VAR     EQU   $20
        LDA   VAR
        ICC   VAR     ;On incrémente la variable sur 2 octets
        LDA   VAR
```

Messages d'erreurs

Lors de la description de ProCODE, nous vous avons annoncé qu'il existait de nombreux messages d'erreurs. Aussi, nous consacrons un chapitre pour vous les énumérer le plus exhaustivement possible en précisant leur type : assembleur ou menu.

Ces messages étant tous très parlant, il ne nous a pas semblé indispensable de les expliquer en détail.

Codes des erreurs assembleur

«Décimal trop grand»
«Hexa trop grand»
«Binaire trop grand»
«Multiplication trop grande»
«Division par 0»
«Opération inconnue»
«Opérande fausse»
«Instruction inconnue»
«Opérande non en page 0»
«Code non existant»
«Label non légal»
«Label double»
«Saut trop long»
«Label non déclaré»
«Label non déclaré avant»
«HEX mal utilisé»
«Label de type incompatible»
«Table des labels trop grande»
«Code généré trop grand»
«Label déclaré comme non redéfinissable»
«Pas de fichier libre pour PUT»
«MDF avec un label dejà déclaré»
«Mauvaise adresse pour FX»
«Macro dans un PUT»
«Trop de MACRO imbriquées»

Codes des erreurs menu

«Erreur invalide» : ne doit jamais apparaitre à l'écran
«Erreur» : erreur générale
«E/S erreur» : erreur du disque par exemple
«Volume absent»
«Ecriture impossible»
«Nom mauvais»

«Fichier non trouvé»
«Fichier double»
«Volume plein»
«Catalogue plein»
«Type mauvais»
«Accès protègé»
«Mémoire pleine»

Récapitulation des commandes

Récapitulation des commande du menu

Concernant la colonne source

(+) Précédent : Recherche d'un programme dans la zone supérieure de la mémoire.

(-) Suivant : Recherche d'un programme dans la zone inférieure de la mémoire.

(C) Charger : Charge des sources à partir du disque.

(S) Sauver : Ecrit le source actif sur le disque.

(J) Ajouter : On charge, en ajoutant un fichier à celui déjà écrit.

(D) Dupliquer : Copier un source en mémoire dans le source actif.

(E) Editer : Appel de l'éditeur. Edition du fichier actif.

(A) Assembler : Assemble le fichier à l'écran.

(I) Imprimer : Assemble le fichier sur imprimante.

(Ctrl) - (V) Vider : Efface le source actif de la mémoire.

Concernant la colonne objet ou code

(O) Sauver : Sauve le fichier code sur le disque.

Concernant la colonne des labels

(L) Sauver : Sauve la table des labels sur le disque.

Concernant le général

(1) Préfixe : Nom du volume (disquette ou disque dur).

(2) Date : Permet de dater les différentes versions de vos programmes.

(3) Catalogue : Liste le contenu d'un volume à l'écran.

(4) Cat.imp. : Liste le contenu d'un volume sur imprimante.

(5) Typ.source : Donne le type des fichiers sources écrits par ProCODE.

Les types usuels sont : txt ou bin

(6) Typ.syste. : Pour définir le type des fichiers code et des labels.

(7) Suppression : Effacer un fichier de la mémoire.

(8) Créer : Création d'un sous-catalogue sur le volume.

(9) En ligne : Liste les préfixes des volumes en ligne.

(Ctrl) - (Q) Quitter : Sortie de ProCODE.

Pour revenir au menu tapez sur (Esc)

Récapitulation des commandes éditeur

(E) : A partir de cette instruction du menu ProCODE, vous accédez directement à l'éditeur.

(C) : Au cours de l'assemblage, si vous avez des corrections à faire, en appuyant sur cette touche, vous revenez à l'éditeur.

Commandes générales :

(→) : Déplace le curseur vers la droite.

(←) : Déplace le curseur vers la gauche.

(↑) : Remonte le curseur d'une ligne, et le place sur le premier caractère.

(↓) : Descend le curseur d'une ligne, et le place sur le premier caractére.

(→|) ou (Ctrl) - (I) : Le curseur se place à la tabulation du champ des commentaires. (s'il n'existe pas, ProCODE rajoute des espaces et écrit un ;)

Avec la (🍎) maintenue enfoncée :

(→) : Le curseur va directement en fin de ligne.

(←) : Le curseur va se placer au début de votre ligne.

(↑) : Le curseur se retrouve en haut du programme si vous n'avez qu'une seule page, ou en haut de la page précédente.

(↓) : Le curseur va se placer au début de la page suivante, ou, à la fin de votre programme si vous n'avez qu'une seule page.

Avec la (⇧) maintenue enfoncée :

(→) : La tabulation 0 se déplace d'un caractère vers la droite.

(←) : La tabulation 0 se déplace d'un caractère vers la gauche.

▲ Attention

Ces deux commandes n'agissent pas sur la ligne où est le curseur.

(↑) : Le curseur se place au début de votre programme.

(↓) : Le curseur se place au debut de la dernière ligne de votre programme.

Modification de texte

(Del) : Effacement du caractère se trouvant à gauche du curseur.

(♦) - (Del) : La ligne se déplace d'un caractère vers la gauche, et vous effacez le caractère situé sous le curseur.

Fonctions utilitaires de l'EDITEUR

(Ctrl) - (B) : Met la ligne en mode inverse. Même commande pour revenir en mode normal.

(Ctrl) - (L) : Supprime la ligne sur laquelle est le curseur.

(Ctrl) - (C) : Copie une partie de texte ailleurs dans le programme.

(Ctrl) - (F) : Recherche une chaîne de caractères dans le programme.

(Ctrl) - (S) : Recherche les chaîne de caractères correspondant aux critères définis par l'instruction (Ctrl) - (F).

(Ctrl) - (V) : Passage du mode insertion au mode recouvrement et vice versa.

(Ctrl) - (T) : Répete la dernière commande.

(Ctrl) - (R) : Comme la commande (Ctrl) - (F), mais vous pouvez remplacer la chaîne de caractères recherchée par une autre.

(RETURN) : Descend le curseur à la ligne suivante.

Pour quitter l'Editeur, appuyez sur (Esc).

Récapitulation des commandes ASSEMBLEUR

Pour accéder à l'assembleur, vous avez deux possibilités :
(A) : l'assemblage se fera avec une édition à l'écran.
(I) : l'assemblage se fera avec une édition sur imprimante.

(ESPACE) vous permet d'arrêter le déroulement d'un programme à l'écran. Pour le reprendre, appuyez sur n'importe quelle touche sauf (Esc) et (ESPACE).

Contrôle de l'assemblage

Instructions listées :

EQU ou =

(label)–EQU–(valeur)<;commentaire> :
Affecte le label à la valeur spécifiée par l'opérande.

SET

(label)–SET–(valeur)<;commentaire> :
Comme EQU, mais un même label peut être défini plusieurs fois.

MDF

(label)–MDF<–;commentaire> :
Définit un label comme redéclarable, sans lui affecter de valeur.

ORG

–ORG–(valeur)<;commentaire> :
Définit l'adresse de la prochaine instruction assemblée.

SOUND

–SOUND<;commentaire> :
Fait retentir un signal sonore prévenant d'une erreur lors de l'assemblage.

Les deux instructions suivantes, ne sont utiles que pour la version 64 k :

FXLBL

–FXLBL–(valeur)<;commentaire> :
Déplace le début de la table des labels à une autre adresse.

FXCD

–FXCD–(valeur)<;commentaire> :
Déplace le début de la zone des codes à une nouvelle adresse.

Instructions non listées :

LST OFF

–LST–OFF
Arrêt du listing sur imprimante et à l'écran.

LST ON

–LST–ON
Rétablit l'affichage du listing et l'impression, après un LST OFF.

LMC OFF

-LMC-OFF
Permet de lister, uniquement l'instruction comprenant la MACRO.

LMC ON

-LMC-ON
Rétablit le listage complet du contenu des MACRO après une instruction LMC OFF.

Instructions de présentation :

CHR

-CHR-(valeur)<;commentaire> :
La valeur définit le caractère répété par REP.

REP

-REP-(valeur)<;commentaire> :
Cette instruction répète «valeur» fois le caractère défini par CHR.

SKIP

-SKIP-(valeur)<;commentaire> :
On saute «valeur» ligne.

PAG

-PAG-<;commentaire> :
Le papier avance jusqu'au début de la première page, ou au début de la page suivante.

AST

-AST-(valeur)<commentaire> :
Envoie «valeur» astérisques dans le listing.

PTC

-PTC-(valeur)<,valeur><...><;commentaire> :
Donne la possibilité de paramétrer votre imprimante.

TAB

-TAB-(valeur)<,valeur><...><;commentaire> :
Permet de définir des tabulations autres que celles prédéfinies par ProCODE.

PAU

-PAU<;commentaire> :
Permet de stopper l'assemblage.

Instructions génératrices de code :

Ces instructions sont toutes listées

DS

(label)–DS–(valeur1)<,valeur2><;commentaire> :
Permet de placer un nombre important d'octets à la même valeur.

DFB

(label)–DFB–(valeur)<,valeur><...><;commentaire> :
Chaque valeur génère un octet de code, correspondant à sa partie basse.

HBY

(label)–HBY–(valeur)<,valeur><...><;commentaire> :
Chaque valeur génère un octet de code, correspondant à sa partie haute.

DA

(label)–DA–(valeur)<,valeur><...><;commentaire> :
Chaque valeur génère deux octets, sa partie basse puis sa partie haute.

DDB

(label)–DDB–(valeur)<,valeur><...><;commentaire> :
Chaque valeur génère deux octets, sa partie haute puis sa partie basse.

HEX

(label)–HEX–(0 à FF)<0 à FF><...><;commentaire> :
Utilisé pour créer des tables de données.

ASC

(label)–ASC–(chaîne)<;commentaire> :
Permet d'insérer du texte dans le fichier code.

DCI

(label)–DCI–(chaîne)<;commentaire> :
Comme ASC, mais le dernier caractère de la chaîne a son bit 7 à l'opposé des 7^es^ bits des autres caractères.

PCI

(label)–PCI–(chaîne)<;commentaire> :
Comme ASC, mais le premier caractère de la chaîne a son bit 7 à l'opposé des 7^es^ bits des autres caractères.

STR

(label)–STR–(chaîne)<;commentaire> :
Comme ASC, mais le nombre de caractères de la chaîne est placé avant le codage des caractères.

INV

(label)–INV–(chaîne)<;commentaire> :
Comme ASC, mais : si la chaîne est en ASCII haut le codage sera en inverse.
si la chaîne est en ASCII bas le codage sera clignotant.

Instructions de chaînage :

DSK

–DSK–(nom du fichier)<;commentaire> :
S'utilise lorsque le message «code généré trop long» apparaît à l'écran. Introduit au début de votre programme, vous pourrez assembler de longs fichiers.

PUT

–PUT–(nom du fichier)<;commentaire> :
Permet une cohabitation de trois sources en mémoire et donc d'assembler de très longs sources.

Instructions d'assemblage conditionnel :

DO

–DO–(valeur)<;commentaire> :
Cette instruction autorise l'asemblage si l'opérande est diférente de 0. Sinon, il n'y aura pas assemblage, jusqu'au prochain ELSE ou FIN.

ELSE

–ELSE<–;commentaire> :
Cette instruction inverse l'état du drapeau «do».

FIN

–FIN<–;commentaire> :
Instruction qui rétablit le drapeau «do» à la valeur 1.

Instructions MACRO :

MAC

(label)–MAC<–;commentaire> :
Définit une MACRO.

EOM

(label) – EOM< – ;commentaire> :
Détermine la fin d'une MACRO.

Syntaxe des labels

1re lettre

Pour un label normal :

— lettre majuscule ou minuscule

— (ç) ou (.) ou (—)

Pour un label local :

— (§)

Lettres suivantes

— lettre majuscule ou minuscule

— chiffre

— (§) ou (.) ou (.)

Syntaxe des données

— série de chiffres

— ($) suivi d'une série de chiffres ou de lettres de A à F

— (%) suivi par une série de (1) ou de (0)

— (") suivi par une lettre ou un signe autre que (.)

— (") suivi par (.) puis par une lettre

— (') suivi par une lettre ou par un signe autre que (.)

— (') suivi par (.) puis par une lettre

— sous la forme d'une variable

— (★)

— (&)

Syntaxe des opérateurs

— (+) l'addition

— (–) la soustraction

— (★) la multiplication

— (/) la division

— (|) le ou exclusif

Syntaxe des valeurs

Première valeur :

— (>) on prend la partie haute du nombre généré

— (<) on prend la partie basse du nombre généré

Autres valeurs :

Une donnée ou non, suivie par un opérateur

Syntaxe d'une chaîne

Le premier caractère

— (") ou (')

Autres caractères

— tous les signes sauf le (;)

Dernier caractère

— (") ou (')

Syntaxe des Macros

Paramètre de Macro

«.X» avec X qui est un nombre décimal compris entre 0 et 255.

Passage des paramètres

— ils sont dans le champs des opérandes de la Macro appelée.
— ils sont séparés par une virgule (,).

Table de référence des 128 codes ASCII :

HX	DEC	OCT	BINARY	ASCII
00	0	000	0000000	NULL
01	1	001	0000001	SOH
02	2	002	0000010	SRX
03	3	003	0000011	ETX
04	4	004	0000100	EOT
05	5	005	0000101	ENQ
06	6	006	0000110	ACK
07	7	007	0000111	BEL
08	8	010	0001000	BS
09	9	011	0001001	HT
0A	10	012	0001010	LF
0B	11	013	0001011	VT
0C	12	014	0001100	FF
0D	13	015	0001101	CR
0E	14	016	0001110	SO
0F	15	017	0001111	SI
10	16	020	0010000	DLE
11	17	021	0010001	DC1
12	18	022	0010010	DC2
13	19	023	0010011	DC3
14	20	024	0010100	DC4
15	21	025	0010101	NAK
16	22	026	0010110	SYN
17	23	027	0010111	ETB
18	24	030	0011000	CAN
19	25	031	0011001	EM
1A	26	032	0011010	SUB
1B	27	033	0011011	ESC
1C	28	034	0011100	FS
1D	29	035	0011101	GS
1E	30	036	0011110	RS
1F	31	037	0011111	US
20	32	040	0100000	SP espace
21	33	041	0100001	! point d'exclamation
22	34	042	0100010	" guillemet
23	35	043	0100011	£ la livre (équiv. au dièse)
24	36	044	0100100	$ dollar
25	37	045	0100101	% pourcentage
26	38	046	0100110	& esperluète
27	39	047	0100111	' apostrophe
28	40	050	0101000	(parenthèse ouverte
29	41	051	0101001	) parenthèse fermée
2A	42	052	0101010	* astérisque
2B	43	053	0101011	+ plus
2C	44	054	0101100	, virgule
2D	45	055	0101101	– moins
2E	46	056	0101110	. point
2F	47	057	0101111	/ barre de fraction
30	48	060	0110000	0 zéro
31	49	061	0110001	1 un
32	50	062	0110010	2 deux
33	51	063	0110011	3 trois
34	52	064	0110100	4 quatre
35	53	065	0110101	5 cinq
36	54	066	0110110	6 six
37	55	067	0110111	7 sept
38	56	070	0111000	8 huit
39	57	071	0111001	9 neuf
3A	58	072	0111010	: deux-points
3B	59	073	0111011	; point-virgule
3C	60	074	0111100	< inférieur
3D	61	075	0111101	= égal
3E	62	076	0111110	> supérieur
3F	63	077	0111111	? question
40	64	100	1000000	à
41	65	101	1000001	A
42	66	102	1000010	B
43	67	103	1000011	C

HX	DEC	OCT	BINARY	ASCII
44	68	104	1000100	D
45	69	105	1000101	E
46	70	106	1000110	F
47	71	107	1000111	G
48	72	110	1001000	H
49	73	111	1001001	I
4A	74	112	1001010	J
4B	75	113	1001011	K
4C	76	114	1001100	L
4D	77	115	1001101	M
4E	78	116	1001110	N
4F	79	117	1001111	O
50	80	120	1010000	P
51	81	121	1010001	Q
52	82	122	1010010	R
53	83	123	1010011	S
54	84	124	1010100	T
55	85	125	1010101	U
56	86	126	1010110	V
57	87	127	1010111	W
58	88	130	1011000	X
59	89	131	1011001	Y
5A	90	132	1011010	Z
5B	91	133	1011011	°indice
5C	92	134	1011100	ç
5D	93	135	1011101	§
5E	94	136	1011110	^accent circonflexe
5F	95	137	1011111	soulignement
60	96	140	1100000	`accent grave
61	97	141	1100001	a
62	98	142	1100010	b
63	99	143	1100011	c
64	100	144	1100100	d
65	101	145	1100101	e
66	102	146	1100110	f
67	103	147	1100111	g
68	104	150	1101000	h
69	105	151	1101001	i
6A	106	152	1101010	j
6B	107	153	1101011	k
6C	108	154	1101100	l
6D	109	155	1101101	m
6E	110	156	1101110	n
6F	111	157	1101111	o
70	112	160	1110000	p
71	113	161	1110001	q
72	114	162	1110010	r
73	115	163	1110011	s
74	116	164	1110100	t
75	117	165	1110101	u
76	118	166	1110110	v
77	119	167	1110111	w
78	120	170	1111000	x
79	121	171	1111001	y
7A	122	172	1111010	z
7B	123	173	1111011	é
7C	124	174	1111100	ù
7D	125	175	1111101	è
7E	126	176	1111110	¨
7F	127	177	1111111	DEL

Bibliographie

— ANDRIEUX Alain et CREUZET Gérard. La conduite de l'Apple. II[e] partie. Ed. Radio. Septembre 1984.

— BREAUD-POULIQUEN Nicole et JEAN-DAVID Daniel. La pratique de l'Apple II. Volume 3 : Langage machine et assembleur du 6502. Ed. du P.S.I. 1983.

— Apple *IIe* Reference Manual. Cupertino, California. Apple computer. Inc. 1982.

— Apple *IIc* Reference Manual Cupertino, California Volume 1 et Volume 2. Apple computer. Inc. 1984.

Limitation de la garantie et de la responsabilité :

Malgré les essais effectués sur le logiciel décrit dans ce manuel, et le contrôle de son contenu, Version Soft ne peut garantir expressément ou implicitement, ce manuel ou le logiciel décrit dans celui-ci, ni leur qualité, performances, vendabilité ou adaptation à des fins particulières. Ce logiciel et ce manuel sont donc vendus «tels quels», et c'est vous l'acheteur, qui en assumez le risque du point de vue de leur qualité et de leurs performances. Version Soft ne sera en aucun cas responsable des dommages directs ou indirects résultant d'un quelconque défaut du logiciel ou du manuel, même si on l'a prévenue de la possibilité de tels dommages. En particulier, elle ne saurait être responsable des programmes ou données stockées dans les produits Version Soft ou utilisés avec ceux-ci, y compris du coût de récupération ou de reproduction de ces programmes ou de ces données.

Droit de reproduction :

Révision des produits :

Version Soft ne peut garantir que vous soyez informé des révisions opérées sur le logiciel décrit dans ce manuel. Il vous est donc recommandé de vous informer périodiquement auprès de votre revendeur habituel.

Auteur de ProCODE : **Richard THIBERT**