ATARI ST

PROFI MAT ST

PROFIMAT ST est un outil de développement compact à l'usage du programmeur en langage assembleur. Précieux pour tous ceux qui désirent développer des programmes orientés machine avec à leur disposition un langage de haut niveau.

Caractéristiques :

- Assembleur avec éditeur intégré, débugger, désassembleur et réassembleur.
- Travail simplifié par la possibilité d'assemblage en mémoire ou sur disque.
- Exploitation totale des possibilités de GEM.
- Possibilité de générer du code relogeable.
- Création de macro-instructions avec multiples paramètres.
- Variables locales et redéfinissables.
- Assemblage conditionnel très convivial.
- Calculs sur 32 bits supportant toutes les opérations logiques, avec 32 niveaux de parenthèses.
- Création d'une liste de références croisées.
- Envoi des messages d'erreur vers un fichier ou gestion des erreurs par l'éditeur.
- Débugger avec trace pas à pas ou émulation du mode Single-Step du 68020.
- Menu d'aide affichant les différents modes d'adressage possibles à l'emplacement courant, ainsi que les paramètres pour les fonctions GEMDOS.
- Manuel d'utilisation détaillé.

Réf. ST 015 PRIX : 495 F.

EDITIONS MICRO APPLICATION
13 RUE SAINTE-CÉCILE 75009 PARIS / TÉL. : (1) 47 70 32 44

ATARI ST

PROFIMAT ST

MA

PROFI MAT ST

LOGICIEL ASSEMBLEUR, DÉSASSEMBLEUR, RÉASSEMBLEUR

EDITIONS MICRO APPLICATION

DATA BECKER

pas influencé par cette opération même si l'insertion s'effectue à l'intérieur de celui-ci.

Sup. marquage cette opération permet d'annuler le marquage d'un bloc.

2.9. LE CLAVIER

Le curseur de texte peut être déplacé à l'aide de la souris en plaçant à l'endroit voulu le curseur de celle-ci. Il faut alors appuyer brièvement sur le bouton gauche. N'oubliez pas que ceci ne fonctionne que si le mode MAJUSC./MINUSC. est hors service. Vous pouvez également déplacer le curseur de texte à l'aide des touches - curseur.

Touche	Normal	Shift	Control
gauche	1 caract	1 mot	début de ligne
droite	1 caract	1 mot	fin de ligne
haut	1 ligne	variable précédente	début de texte
bas	1 ligne	variable suivante	fin de texte

Légende :

1 mot : Le curseur est déplacé au début ou à la fin du mot suivant ou précédent. Un mot est défini par une succession de caractères comprise entre deux caractères espace.

Variable précédente/suivante : On entend par là la ligne précédente/suivante dans laquelle on trouve comme premier caractère '\' ou '_' (tous deux servant à définir une variable).

Le curseur peut aussi être déplacé à l'aide des touches suivantes :

CLR/HOME

Normal : Le curseur va à la fin de ligne. S'il s'y trouve déjà, il sera placé en début de ligne.

Shift : Le curseur est déplacé dans le coin haut à gauche de la fenêtre. S'il s'y trouve déjà, il sera placé dans le coin en bas à gauche de cette dernière.

Control : Le curseur est déplacé en début de texte. S'il s'y trouve déjà, il est placé à la fin du texte en colonne 1.

TAB

Place le curseur sur la prochaine position marquée par un tabulateur. Le déplacement s'effectue vers la droite. S'il n'existe plus de tabulateur, il est procédé à un test pour savoir si un tabulateur existe dans la première colonne de la ligne. Dans l'affirmative le curseur passe au début de la ligne suivante. Dans la négative il ne se passe plus rien.

INSERT

Normal : Insère un caractère vide à la position du curseur. Le texte situé à droite du curseur est décalé vers la droite d'un caractère.

Shift : Permet de passer du mode insertion au mode recouvrement.

Control : Insère un nouvelle ligne après la ligne où se trouve le curseur.

DELETE

Normal : Efface le caractère situé sous le curseur et décale vers la gauche le reste du texte.

Shift : Efface tous les caractères situés sous le curseur et à droite de celui-ci jusqu'à la fin du mot, et au plus loin jusqu'à la fin de la ligne. Comme d'habitude, la fin du mot est repérée par un caractère espace situé immédiatement après celui-ci.

Control : Efface tous les caractères si fin de la ligne.

Alternate : Efface la ligne où se trouve le curseur.

BACKSPACE

Normal : Efface le caractère à gauche du curseur et décale le reste de la ligne vers la gauche.

Shift : Efface tous les caractères à gauche du curseur jusqu'au prochain début de mot et au plus loin en début de ligne. Comme d'habitude le début de mot est repéré par un caractère espace.

Control : Efface tous les caractères situés à gauche du curseur jusqu'au début de la ligne.

Alternate : Efface la ligne dans laquelle se trouve le curseur.

UNDO

Remplace dans presque tous les cas, l'ancien état de la ligne, à la place de celui dernièrement défini à partir de la position du curseur (voir aussi le point 2.4. LA FONCTION UNDO).

HELP

Visualise la position des tabulateurs ou le dernier message d'erreur dans la ligne d'information.

ESC

La touche ESC permet déjà la saisie directe de commandes courtes sans pour autant être obligé de redéfinir une touche de fonction. La saisie est placée dans un tampon de commande spécial et reste disponible jusqu'à ce que ce dernier soit effacé ou remplacé par une nouvelle commande.

Normal : Efface le tampon spécial de commande et permet la saisie d'une nouvelle séquence de commande dans la ligne d'information. Après validation par la touche *RETURN* ou *ENTER*, cette dernière est exécutée. Si vous ne saisissez rien, il ne se passe rien.

Shift : Le tampon spécial de commande est affiché dans la ligne d'information à fin d'une éventuelle modification. Après validation, la séquence est exécutée.

Control : Exécute directement l'instruction contenue dans le tampon spécial de commande.

Touches de fonction

Normal : Exécute directement la séquence de commande correspondant à la touche sélectionnée.

Shift : Présente le commentaire de la touche correspondante dans la ligne d'information.

2.10. LES COMMANDES COURTES

Les commandes courtes vous permettent, sans passer par le détour des menus, de vous servir des fonctions de l'éditeur. Ceci est très intéressant si vous ne souhaitez pas utiliser la souris pour la saisie et l'édition de programmes.

L'appel d'une séquence de commande s'effectue soit par les touches de fonction prédéfinies soit à l'aide de la touche *ESC*.

Il est possible, comme pour un langage de programmation, d'exécuter plusieurs instructions les unes après les autres et/ou les faire exécuter plusieurs fois.

Survolons tout d'abord la gamme des commandes disponibles. Posons tout d'abord la convention suivante : pour obtenir l'équivalent de *^A*, il faut taper les touches *CONTROL* et *A* simultanément. A l'écran, vous obtiendrez un caractère graphique qui représente la commande saisie. Par exemple, *^A* sera représenté par une flèche vers le haut. Il en va de même pour *^B* ou *^D*. Il n'est fait aucune différence entre les lettres majuscules ou minuscules.

La plupart des commandes courtes se comportent de la même manière que celles auxquelles vous accédez avec la souris.

Si le comportement devait être différent pour ce qui est des commandes courtes, cela vous sera indiqué dans le descriptif suivant.

*) La ligne actuelle est la ligne à laquelle où se trouve le curseur.

2.10.1. Commandes de Curseur I

(^A) : Curseur une ligne plus haut.
(^A)A : Curseur en début de texte.
(^A)B : Curseur au début de bloc.
(^A)F : Curseur sur erreur précédente.
(^A)S : Cherche/remplace en arrière.

(^B) : Curseur une ligne plus basse.
(^B)B : Curseur en fin de bloc.
(^B)E : Curseur en fin de texte.
(^B)F : Curseur sur la prochaine erreur.
(^B)S : Cherche/remplace en avant.

(^C) : Curseur un caractère plus à droite.
(^C)B : Curseur en fin de bloc.
(^C)E : Curseur en fin de ligne.

(^D) : Curseur un caractère plus à gauche.
(^D)A : Curseur en début de ligne.
(^D)B : Curseur en début de bloc.

2.10.2. Commandes de bloc.

BA : Placer début de bloc à la position du curseur.
BD : Démarquer le bloc.
BE : Placer fin de bloc à la position du curseur.
BK : Copie du bloc à l'emplacement du curseur.
BL : Effacer le bloc.
BV : Déplace le bloc à l'emplacement du curseur.

2.10.3. Commandes de curseur II.

C(co,li) : Place le curseur en colonne 'co' et ligne 'li'.
Par exemple C(5,25) place le curseur en 5ème colonne, 25ème ligne.
CN : Place le curseur au début de la prochaine ligne.
CV : Place le curseur au début de la ligne précédente.

2.10.4. Commandes de fichier.

DL 'Nom' : Charge le fichier texte du sommaire actuel dans la mémoire de l'éditeur.

DS 'Nom' : Sauvegarde le contenu de la mémoire de l'éditeur sous le nom de fichier *'Nom'* dans le sommaire de la disquette en cours.

REGLE D'EMPLOI :

A la place de *'Nom'*, vous pouvez aussi écrire *"Nom"*. Sous le vocable *Nom*, on entend le nom véritable donné au fichier. Il n'a rien à voir avec celui qui figure dans le point FICHIER sous nom de fichier. Si vous n'indiquez pas de nom, c'est le dernier employé qui sera utilisé. Ainsi si vous chargez le texte *'Essai'* par *DL 'Essai'*, pour sauvegarder la version modifiée, il suffira de taper *DS*. Le nom de fichier doit être complet. Vous devez, le cas échéant, préciser l'extension (ex : *DL 'ESSAI.L'*).

2.10.5. Remplacement

E[?AVU]r'Chaîne 1' 'Chaîne 2':
Remplace la chaîne 1 par la chaîne 2.

Les paramètres suivants sont optionnels :

? : Place une question de sécurité avant le remplacement si ce dernier doit effectivement avoir lieu.
A : Remplacement de toutes les expressions.
V : Cherche les variables.
U : Fait la différence entre majuscules et minuscules.

Si aucune indication n'est donnée, il n'est pas fait de différence entre majuscules et minuscules, la recherche ne se cantonne pas seulement aux variables et seule la première expression est échangée.

L'ordre des paramètres indiqué importe peu.

r : Permet d'indiquer le sens de travail :

= ^A recherche arrière
= ^B recherche avant.

Il est possible d'utiliser un autre caractère séparateur que l'apostrophe, pourvu qu'il n'apparaisse pas dans la chaîne de caractères. Ainsi /, @, ou tout autre caractère conviennent très bien.

EXEMPLE :

EVA (^B) "OV" 'OVERFLOW' :
remplace toutes les variables OV par OVERFLOW de la position actuelle du curseur à la fin du texte.

EAV (^B) 'OV' >OVERFLOW> :
effectue le même travail que précédemment.

E(^A) /.W/ ## :
remplace le .W précédent par rien. Autrement dit le .W situé avant le curseur se trouve supprimé.

EA?(^B) -.W- @@ :
efface tous les .W du curseur à la fin du texte en posant une question de sécurité.

2.10.6. Commandes d'effacement

L(^A) : Efface tout ce qui se trouve entre la position du curseur du moment et le début du texte.

L(^B) : Efface tout ce qui se trouve à partir du curseur (celui-ci compris) et la fin du texte.

L(^C) : Efface tout ce qui se trouve à partir du curseur (celui-ci compris) jusqu'à la fin de la ligne.

L(^D) : Efface tout ce qui se trouve avant le curseur jusqu'en début de ligne.

LL : Efface un caractère vers la gauche à partir du curseur (comme pour un *BACKSPACE).*

LR : Efface le caractère se trouvant sous le curseur (comme pour *DELETE).*

LZ : Efface la ligne où se trouve le curseur.

2.10.7. Commandes de recherche

S[VU]r 'Chaîne' :
Effectue la recherche d'une chaîne de caractères. Les paramètres suivants peuvent être utilisés.

V : Recherche de variables.
U : Effectue la différence entre majuscules et minuscules.

Si aucun paramètre n'est indiqué, la recherche s'effectue sans pour autant faire la différence entre majuscules et minuscules, et ne portera pas sur une variable proprement dite.

La suite dans laquelle sont indiqués les paramètres importe peu. Un paramètre peu être utilisé autant que désiré. Mais le résulat ne dépassera pas la fonction obtenue. Vous pouvez aussi utiliser les paramètres ? et A. Mais ils n'auront aucune influence sur la procédure de recherche.

r : Permet d'indiquer le sens de travail :

= ^A recherche arrière
= ^B recherche avant.

Il est possible d'utiliser un autre caractère séparateur que l'apostrophe, pourvu qu'il n'apparaisse pas dans la chaîne de caractères. Ainsi /, @, ou tout autre caractère conviennent très bien.

EXEMPLE :

SV (^B) "OV" :
recherche la prochaine variable

SU (^A) #.W# :
recherche la chaîne '.W' précédent le curseur en effectuant la différence entre les majuscules et les minuscules.

S(^B) !""'! :
recherche la prochaine chaîne de caractères '"'.

2.10.8. Commandes de tabulation

TL : efface le tabulateur dans la colonne du curseur.
TS : place un tabulateur dans la colonne du curseur.

2.10.9. Commandes diverses

Z 'Chaîne' : insère la chaîne à la position actuelle du curseur.

ZN 'Chaîne' : insère la chaîne comme une nouvelle ligne, autrement dit, cette chaîne sera terminée par un *CRLF*. Cette insertion s'effectue après la ligne actuelle.

ZV 'Chaîne' : insère la chaîne comme une nouvelle ligne, autrement dit, cette chaîne sera terminée par un *CRLF*. Cette insertion s'effectue avant la ligne actuelle.

Il est possible d'utiliser un autre caractère séparateur que l'apostrophe, pourvu qu'il n'apparaisse pas dans la chaîne de caractères. Ainsi /, @, ou tout autre caractère conviennent très bien.

2.10.10. Formation de séquences de commande

Deux commandes différentes sont séparées par un deux point.

Il est tout à fait possible de faire suivre plusieurs commandes entre parenthèses comme par exemple :

(LZ:LZ:LZ)

Par la mise en place d'un nombre de 5 chiffres, en base dix, avant la séquence de commande, il est possible d'obtenir une réitération de celle-ci, correspondant à ce chiffre. Par exemple :

7(^C)

permet d'obtenir un déplacement de 7 colonnes vers la droite. Par la mise en place de la lettre 'W' avant une séquence de commande, on obtient la répétition de la commande jusqu'à ce qu'une erreur soit détectée ou si vous appuyez simultanément sur les deux touches SHIFT.

Ainsi la commande :

W(^B)

déplace le curseur vers la fin du texte.

Nous vous rappelons que l'appui simultané sur les deux touches *SHIFT* permet d'interrompre le processus en cours.

EXEMPLE :

(^A)A:WE(^B)'' ';'*
fait alors passer le curseur en début de texte, et remplace tous les caractères * par ;.

(^A)A:EA(^B)'' ';'*
à le même effet que la commande précédente.

(^D)A:128(TL:(^C))
efface tous les tabulateurs.

(^D)A:16(TS:8(^C))
place, toutes les 8 colonnes, un tabulateur.

Tous ceux qui travaillent avec *RCS* (ma compassion leur est acquise) trouveront une autre raison de satisfaction avec la touche de fonction F10. Ce programme convertit un fichier '.I', délivré sous *RCS*, en format assembleur. Alors que la définition de constantes possède en format .I le format suivant :

*Nom = Valeur ;@(*Commentaire**)*

elle demande, pour l'assembleur décrit dans le prochain chapitre le format :

Nom = Valeur

où le commentaire du programme de la touche de fonction sera effacé. En effet il y a de forte chance que vous n'en ayez pas besoin.

Essayez de concevoir un programme pour votre propre usage permettant d'attribuer une valeur aux touches de fonction. N'oubliez pas, une fois la programmation achevée, de placer la disquette assembleur et de cliquer le point *'Sauvegarde du préréglage'* pour sauvegarder le préréglage. Se faisant, celui-ci restera disponible lorsque vous rechargerez PROFIMAT ST.

Le chargement de ce préréglage s'effectuera automatiquement puisqu'il sera contenu dans le fichier PROFIMAT.INF. Ce fichier fait l'objet, à chaque mise en service de PROFIMAT-ST, d'un chargement automatique dans la mesure où il se trouve dans le même sommaire que le logiciel.

3. L'assembleur.

L'assembleur est certainement la clé de voûte de ce logiciel. Il s'agit, en fait d'un assembleur à 2 passes. Pour ceux d'entre vous qui ne savent pas ce à quoi correspond un tel mode opératoire, il faut savoir que la première passe sert à traiter les variables et la seconde à générer le programme en lui-même (en fait les codes objets).

Lorsque vous n'avez pas encore sauvegardé les codes obtenus, vous aurez droit à l'édition d'une mise en garde si vous désirez les effacer, en quittant PROFIMAT ST, par exemple. Vous n'obtiendrez pas de message de mise en garde si vous demandez un nouvel assemblage de programme, même si le dernier programme assemblé n'a pas été sauvegardé.

L'assembleur respecte en tout les conventions, format de saisie, instructions et mode d'adressage habituel aux processeurs de la classe des 68000 ce qui fait que ceux d'entre vous qui ont déjà programmé avec celui-ci, ne se sentiront nullement dépaysés. PROFIMA-ST est donc un outil de travail tout à fait adapté à tous les programmeurs, débutants et connaisseurs.

Voici un rappel des conventions utilisées en matière de programmation et de syntaxe :

(A1I...IAn) : Choix possibles dans l'espace des éléments allant de A1 à An.

(0I1) : Signifie que le choix se porte soit sur 0, soit sur 1.

[...] : Le contenu compris entre crochets forme des paramètres optionnels.

*(...)** : Le contenu entre les accolades peut être répété autant de fois que désiré. Une répétition nulle est aussi acceptée.

3.1. LA FENETRE D'ASSEMBLAGE

Comme d'habitude, la fenêtre d'assemblage peut être déplacée et il est possible de modifier sa taille. L'utilisation de celle-ci n'est propre qu'à l'assembleur en lui-même et permet de montrer ce qui est en cours de traitement (si le point *Editer nom de fichier* est coché) et, en fin d'assemblage, pour dénombrer les erreurs qui ont été détectées (si le point *FICHIER D'ERREUR* a été coché).

3.2. LA LIGNE D'INFORMATION

Vous trouverez différentes indications dans la ligne d'information :

LIBRE : Nombre d'octets qui restent libres dans la mémoire de l'assembleur. Dans cet espace, on retrouve toutes les variables et toutes les macros ainsi que les bibliothèques afférentes. Comme votre programme grandit en taille tout au long de sa conception, il est souhaitable de vérifier, après chaque assemblage, si vous disposez de suffisamment de place pour procéder au nouvel agrandissement de votre programme. C'est à cette seule condition que vous éviterez le message d'erreur : PAS ASSEZ DE MEMOIRE.

TEXTE : Longueur du segment de texte du dernier texte assemblé.

DATA : Longueur du segment DATA du dernier texte assemblé.

BSS : Longueur du segment BSS du dernier texte assemblé.

RESTE : Taille mémoire restant disponible par le système d'exploitation.

Si vous ne deviez pas encore connaître la signification des expressions TEXTE, DATA et BSS, n'en prenez pas ombrage, ils font l'objet d'explications plus loin dans ce livre.

3.3. CONSTRUCTION D'UNE LIGNE

Une ligne est contruite suivant le modèle suivant :

```
[Variable:][Instruction [opérande source [,opérande objet]]]
[;Commentaire]
```

Nous rappelons le caractère optionnel des éléments entre crochets. La variable doit toujours débuter en première colonne. Il ne peut y avoir plus d'une instruction par ligne. Elle doit prendre fin dans la même ligne. Elle ne peut donc pas s'étendre sur plusieurs lignes. Cet état de fait rend d'ailleurs le programme bien plus lisible.

Voici un exemple type :

```
LABEL_1:MOVE.L D0,A0; commentaire
```

Mais ceci n'est pas encore des plus lisibles. Il est possible d'insérer des caractères espace qui permettront une plus grande lisibilité :

```
LABEL_1:  MOVE.L  D0 , A0  ; commentaire
```

Dans notre exemple, ceci n'est pas déterminant, mais cela permet quand même de bien situer les différentes parties d'une ligne de commande. Il est aussi important de repérer les endroits où il est possible d'insérer des caractères espace, et là où il ne faut pas le faire. Il ne faut pas, par exemple, placer un caractère espace entre deux parties du nom d'une variable (LABEL et _1 par exemple). Par contre il vous est possible de sacrifier le double-point si vous insérez derrière la fin du nom de la variable un ou plusieurs caractères espace :

```
LABEL_1  MOVE.L  D0 , A0  ; commentaire
```

Là pourtant nous arrivons à un cas particulier. Si votre nom de variable se trouve seul et seulement suivi d'un commentaire, le double-point ou un ou plusieurs caractères espace sont facultatifs :

```
LABEL_1; commentaire
```

Si la ligne ne définit pas de variable, il faut au moins placer un caractère espace (par convention représenté ») avant l'instruction

pour permettre à l'assembleur de ne pas le considérer comme une variable :

```
»MOVE.L  D0 , A0
```

La longueur d'une ligne est limitée à 128 caractères, si on peut appeler ceci une limitation. Dans ces 128 caractères, le caractère de fin de ligne doit y être compris. Comme une ligne doit s'achever soit par CR, soit par CRLF, ce ne sont alors que 127, voir 126 caractères qui sont disponibles (nous vous rappelons que l'éditeur implémenté clôt une ligne par un CRLF : vous ne disposez donc que de 126 caractères dans la majeure partie des cas).

3.4. LES VARIABLES

Le nom d'une variable débute soit par une lettre, soit par un caractère _, ou un caractère \. Les caractères suivants sont alors des lettres, des chiffres, _, @ ou tout caractère dont le code ASCII est compris entre 128 et 240.

Variable::=('A'..'Z'|'_'|'\')[Fz]*

Fz::=('A'..'Z'|'0'..'9'|'_'|'@'|chr$(128)..chr$(240))

La longueur d'un nom de variable est comprise entre 1 et 125. La limite supérieure découle de la taille maximum autorisée qui est de 126 caractères moins un pour placer le double-point.

Le nom des variables est enregistré en entier, et non comme habituellement limité à 8 caractères. Vous pouvez donc bien

utiliser, pour vos sous-programmes, des noms évocateurs, comme par exemple CALCUL_DU_SINUS_DE_D0 sans avoir aucune crainte que l'assembleur ne le confonde avec CALCUL_ DU_ COSINUS_DE_D0. Mais n'oubliez tout de même pas que des noms courts utilisent moins de place en mémoire....

Ce qu'est un label ou une étiquette ne vous est certainement pas inconnu. Il s'agit de marques dans le texte-source qui sont utilisées, en règle générale, pour repérer un branchement, un sous-programme mais aussi des adresses, voir des données. Lorsque vous désirez faire appel à un sous-programme, il suffit de rappeler le label (on pourrait dire le nom) qu'il porte. C'est par ce dernier qu'il est défini. Un exemple :

```
»BSR SP_1

SP_1; sous-programme 1
```

ou bien

```
»MOVE POSITION_X , D0

POSITION_X: DC.W 0
```

On peut dire que les labels définissent des adresses.

Les constantes ressemblent aux variables. La différence provient du fait qu'elles figurent à un endroit précis du texte-source. La valeur d'une constante doit être explicite et c'est pourquoi on utilise le caractère = :

```
CONSTANTE=13
```

La syntaxe de la déclaration d'une constante peut être différente.

Ainsi les formes suivantes sont aussi acceptées :

```
CONSTANTE:=13
```

ou

```
CONSTANTE = 13
```

Ici aussi, vous pouvez utiliser autant de caractères espace que vous le désirez, si cela peut permettre une meilleure compréhension.

On utilise des constantes lorsqu'un chiffre revient régulièrement dans le programme et qu'il possède une signification particulière.

CR = 13
LF = 10

Pour des indices précis compris dans une valeur qui peut être modifiée, comme par exemple dans la structure d'une donnée ou le 5ème caractère joue un rôle déterminant de compteur, il vaut mieux utiliser à la place de la séquence 5(An) la définition d'une constante comme :

```
COMPTEUR=5
```

et saisir alors la valeur sous la forme : COMPTEUR(An), si An pointe sur la structure de donnée et que l'adresse se trouve en dehors du programme, comme par exemple des variables-système comme :

```
ETV_TERME=$408
```

Il faut impérativement définir les constantes en début de programme. En effet l'assembleur, lors de la passe 1, considère les variables inconnues comme des labels. Ceci peut conduire, pour des programmes relogeables, à des erreurs inutiles.

Si pour une raison bien précise, il vous semble nécessaire de placer une constante plus loin dans le programme, il faudra joindre à la fin de son nom le caractère % (voir chapitre 3.5.2.).

La différence fondamentale entre labels et constantes provient de l'adressage. Nous reviendrons sur cette notion un peu plus tard.

Les symboles sont quelque chose de totalement différent. On définit pour un symbole une chaîne de caractères à l'aide de la commande EQU :

```
SYMBOL EQU 12(A0,D0.L)
```

Pour des raisons de compréhension, il est conseillé, là aussi de placer au moins un caractère espace après la variable.

Si vous écrivez à présent ceci :

```
»MOVE.L SYMBOL,D1
```

l'assembleur le transformera en

```
»MOVE.L 12(A0,D0.L),D1
```


Il vous est donc possible, suivant le mode d'adressage ou autre chose, de donner un nom d'une façon détournée. Il est même possible d'y cacher toute une instruction :

```
SAVE_REGISTER EQU MOVEM.L D0-A6,-(SP)
```

Si vous écrivez à présent :

```
»SAVE_REGISTER
```

l'assembleur transforme ceci en

```
»MOVE.L D0-A6,-(SP)
```

Vous pourriez dire que nous sommes là en présence de mini-macros. Mais toute rose a ses épines. Ici il s'agit de ralentissement de la vitesse d'assemblage.

Comme des symboles peuvent être placés dans tout le texte-source, l'assembleur doit analyser la totalité des lignes. Ce type de recherche est, bien entendu, gourmand en temps, d'autant plus lorsque le programme est long.

Si vous ne définissez pas de symboles, ou si vous ne les utilisez que localement (patience, nous reviendrons sur cette notion) cela ne vous ralentira aucunement (pour les utilisations locales, ceci coûte le temps de la recherche, durant la validité du symbole).

Le dernier type de variables recouvre ce qui est appelé Macro. Comment définir une macro vous sera indiqué dans la partie réservée à l'explication de la commande qui s'y rapporte.

Il nous faut maintenant faire la différence entre les variables globales, locales ou redéfinissables.

Les variables globales debutent par une lettre ou un caractère _. Elles ne peuvent être definies qu'une seule fois et sont accessibles à tout moment.

Au contraire en ce qui concerne l'accessibilité, les variables locales ne peuvent être utilisées que si, entre le lieu où se situe leur appel et l'adresse où se trouve la variable, il n'y a pas de variables globales. Ce type de variable est défini par un caractère \ en début de définition.

Prenons un exemple :

```
GLOBALE_1:BSR.S \LOCALE
```

ici il est possible d'accéder à \LOCALE

```
»RTS
\LOCALE

»BNE.S \LOCALE

»RTS
GLOBALE_2:BSR.S \LOCALE
```

vous aurez droit ici à un message d'erreur car l'accès à \LOCALE vient d'être interdit par la définition d'une variable globale GLOBALE_2


```
GLOBALE_3

\LOCALE
»
```

vous pouvez réutiliser le nom de variable locale précédent BRA \LOCALE

```
\LOCALE = 2
»
```

vous aurez droit, ici, à l'édition d'un message d'erreur. En effet LOCALE a déjà été définie et ne peut l'être à nouveau.

Important :

Les intervalles marqués '.....' ne doivent pas comporter de définition de variables globales. L'utilisation des variables locales s'inspire des concepts de PASCAL. Ainsi est-il possible d'écrire :

```
BOUCLE_DE_TEMPS_DE_D0:MACRO ; boucle de temps
\LP:DBRA D0,\LP
»ENDM

»MOVEQ #64,D0
»BOUCLE_DE_TEMPS_DE_D0

»MOVEQ #19,D0
»LEA SOURCE,A0
»LEA CIBLE,A1
\LP:MOVE.B (A0)+,(A1)+ ;Transfert de 20 Octets
```

```
»DBRA D0, \lp

»MOVE.W #$7000,D0
»BOUCLE_DE_TEMPS_DE_D0
```

Comme vous le voyez, la variable locale \LP a été définie 3 fois de suite, sans pour autant provoquer de conflit de définition. En effet des variables définies dans une MACRO ne servent qu'à l'intérieur de celle-ci. Dès que la déclaration de macro est terminée par l'instruction ENDM, l'accès à la variable locale définie à l'intérieur de celle-ci s'achève aussi. Ceci est totalement indépendant du fait qu'il s'agisse d'un label, d'une constante ou d'un symbole. Cette remarque compte aussi pour ce qui concerne des macros imbriquées.

```
M1:MACRO
\LP:DBRA D0,\LP
»M2
»ENDM
M2:MACRO
\LP:DBRA D1,\LP
»ENDM


»M1
```

Des variables locales définies précédemment, peuvent aussi être utilisées dans une MACRO, si dans cette dernière aucune autre variable locale ne porte le même nom :


```
M1: MACRO
»MOVEQ #\VALEUR,D0
\LP: DBRA D0,\LP
»ENDM

\VALEUR=20
»M1
```

On pourrait expliquer ceci d'une autre manière, suivant le travail interne effectué. Si vous avez compris comment cela fonctionne, il n'est pas nécessaire de lire le paragraphe suivant.

A chaque variable locale est attribué un ordre de priorité. Lorsque vous démarrez le processus d'assemblage, le compteur des priorités est placé à 1. Ce compteur est incrémenté de 1 chaque fois que le programme bifurque vers une macro, ou si un texte-source extérieur est chaîné (*INCLUDE*). Il est décrémenté de un si une macro ou un texte-source extérieur (par *END* et *ENDM*) est achevé. La priorité est donc fonction de l'espace de priorité défini. Vous ne pouvez accéder qu'à la variable locale dont le rang est inférieur ou égal à l'espace de priorité du moment. Si plusieurs variables de même nom sont accessibles, c'est celle qui à le rang de priorité le plus fort qui sera utilisée.

Vous pouvez aussi définir une variable locale entre le début de programme et la première variable globale. De même une variable locale peut être définie entre la dernière variable globale et la fin de programme.

Les variables ne sont pas liées aux programmes *TEXTE*, *DATA* ou *BSS* de par leur segment mais bien plus par leur ordre d'introduction dans le texte.

D'ailleurs, lorsque vous introduisez un texte-source extérieur à l'aide de *INCLUDE*, le comportement est identique à celui obtenu lors de l'appel d'une macro. L'espace de priorité est incrémenté de 1. Vous ne pouvez donc travailler dans le texte-source rajouté qu'avec des variables locales pour éviter tout problème conflictuel avec le texte d'origine.

Mais attention ! les variables locales ne peuvent pas être définies plus d'une fois dans l'espace de priorité correspondant. C'est pourquoi il existe en plus des variables redéfinissables. Elles peuvent être tout aussi bien globales que locales et sont repérées par le signe '@'.

Ce type de variables peut donc être défini autant de fois que nécessaire. Elles auront pour valeur, la dernière définie. Un exemple d'utilisation est repris dans la partie traitant de la commande REPEAT.

Mais ne croyez pas pour autant que vous pourrez mélanger tout avec tout. Il existe tout de même des limites d'utilisation auxquelles il faudra prendre garde :

Les macros doivent être globales et non redéfinissables.

Les symboles peuvent être locaux mais non redéfinissables.

Seuls constantes et labels peuvent être locaux et/ou redéfinissables.

Nous voici au bout de cette partie un peu fastidieuse. Nous allons passer à quelque chose de plus facile:

3.5. EXPRESSIONS ARITHMETIQUES

3.5.1. Formats de nombres

Les formats de nombres suivants peuvent être compris par PROFIMAT ST :

- Décimal.
- Héxadécimal par l'adjonction, avant le nombre du caractère $ (taille maxi 8 chiffres).
- Binaire par l'adjonction, avant le nombre du caractère % (taille maxi 32 chiffres).
- Chaîne, entre guillemets ou apostrophes (taille maxi 4 caractères).

Pour cette partie, il en va de même, en ce qui concerne les majuscules et les minuscules : il n'est fait aucune différence entre les deux modes d'écriture. Tous les formats sont portés à 32 Bits. Vous aurez droit à un message d'erreur si un chiffre est édité en étant supérieur à l'équivalent de $FFFFFFFF (4294967295) ou inférieur à -$80000000 (-2147483648). PROFIMAT ST travaille pratiquement en deux formats numériques, en valeurs positives avec des chiffres non-signés sur 32 Bits, en valeurs négatives avec des chiffres signés sur 32 Bits. Si les puristes de la théorie des nombres trouvent cette explication un peu confuse, qu'ils m'excusent mais dans la pratique, c'est ce qui se passe !

Voici un exemple (comme d'habitude, se trouve entre parenthèses la valeur en base dix) :

```
12, 1029, 65536, 34651236274,

$A (10), $FFFF (65535), $10000000 (268435456), $AFFE (45054)

%101 (5), %11111111111111111111111111111111 (-1), %1001001100111
(4711)

'AFFE' (1095124549), " = )" (540876841), 'a' (97), 'ST' (21332)
```

3.5.2. Opérateurs.

Les opérateurs suivants sont disponibles pour tout calcul :

+, -, *, /	:	les 4 opérations élémentaires,
&	:	ET (AND) logique,
|	:	OU (OR) logique,
^	:	OU (XOR) exclusif logique,
-	:	(complément à deux ; signe négatif),
~	:	Non (NOT) logique (complément à 1)

Toutes les opérations sont exécutées sur 32 Bits. On utilise le point comme séparateur de décimal.

Les priorités sont ainsi définies :

priorité supérieure	:	- (signe)
		*,/
		+,- (opération)
		~
		&
priorité inférieure	:	|,^

3.5.3. Opérations composées

Les opérateurs cités ci-avant permettent de réaliser des opérations arithmétiques. Il va de soi que vous pouvez aussi utiliser des parenthèses et ce, sur 32 niveaux. Le tampon mémoire permettant le calcul est calculé de telle manière que vous ne devriez pas avoir de problèmes en ce qui concerne les expressions. Pour les fanatiques du calcul, il faut savoir que les expressions sont placées sur une pile et calculées dès que possible. La pile est calculée pour 20 opérandes, c'est-à-dire qu'après le 19 ème vous aurez droit à un message d'erreur.

Les labels dont le caractère qui suit est % sont traités comme des constantes. Ceci est significatif en matière d'adressage.

Avant et après chaque opérateur, il peut y avoir autant de caractères espace que désirés.

EXEMPLE :

```
1 + 2 * 3

4711 * (3 - (123 I %1101) ^ $F) / 2

'aBcD' & $FF00FF00 -1234
```

```
LABEL_1 - CONSTANTE_2

(0) * 32 / (((17 + 4) * 3) & %11110000)
```

3.6. LES MODES D'ADRESSAGE EFFECTIFS

Un mode d'adressage effectif correspond à la méthodologie utilisée pour accéder à une adresse mémoire ou un registre. On y distingue les groupes suivants :

N°	Description	Syntaxe	Exemple
1)	registre de données direct	Dn	D3
2)	registre d'adresses direct	An	A3
3)	registre d'adresses indirect	(An)	(A3)
4)	registre d'adresses indirect avec postincrément	(An)+	(A5)+ (SP)+
5)	registre d'adresses indirect avec prédécrément	-(An)	-(A5) -(SP)
6)	registre d'adresses indirect avec distance sur 16 bits	d (An) 16	$1234(A5)
7)	registre d'adresses indirect avec distance sur 8 bits	d (An,Rn) 8	$C0(A1,D1)

8)	absolu court	$xxxx.W	$3000
9)	absolu long	$x..x.L	$12345678
10)	immédiat	#"données"	#$0D
11)	relatif au compteur programme avec distance sur 16 bits	d (PC) 16	$1000(PC)
12)	relatif au compteur de programme avec distance sur 8 bits et index (registre)	d (PC,Rn) 8	$30(PC,D5)

pour lesquels nous utilisons les conventions suivantes :

Dn est un registre de données quelconque.
An est un registre d'adresses quelconque.
Rm est soit un registre de données, soit un registre d'adresses.
x est soit un opérande court (W) soit un opérande long (L)
n,m sont des chiffres entiers compris entre 0 et 7.
a16, a32, d, k sont des expressions arithmétiques.

Pour ce qui concerne les modes d'adressage du compteur programme, d(PC) et d(PC,Rm.x), il existe deux possibilités supplémentaires quant à l'utilisation de l'expression arithmétique 'd'.

Si d ne possède pas de label, elle est utilisée directement comme index. Autrement dit, il est procédé à un adressage de l'adresse mémoire située à une distance de 'd' octets du compteur programme actuel (pour la forme plus complexe d(PC,Rm.x) il faut encore y ajouter le contenu du registre Rm).

Mais si l'expression 'd' possède au moins un label, c'est l'adresse de ce label qui est utilisée comme adresse de travail. Le PC est décrémenté auparavant pour permettre la formation de l'index (ceci compte aussi pour la forme complexe).

EXEMPLE :

```
MOVE.W 2(PC),D0
```

Charge le code opération de l'instruction suivant ce MOVE dans le Registre D0

```
MOVE.L LABEL+2(PC), D0
```

Charge le contenu de l'adresse LABEL+2 dans D0.

```
MOVE.B LABEL(PC,D1.L), D0
```

Charge le contenu de la case mémoire qui se trouve à D1 octets après LABEL, dans le registre D0.

Pour récapituler, ce qui est primordial est de savoir s'il existe un label dans l'expression ou une variable, telle une constante par exemple.

D'autre part, les alternatives suivantes sont possibles pour les EA (adresses effectives) :

Pour A7	vous pouvez utiliser SP
Pour 0(An,Rm.x)	vous pouvez utiliser (An,Rm.x)
Pour d(An,Rm.W)	vous pouvez utiliser d(An,Rm)

Si vous désirez avoir un aperçu des modes d'adressage autorisés, vous cliquerez, dans le menu TABLEAU le point AE et en suivant les indications qui vous sont données, il vous est possible de générer l'instruction. Si vous ne comprenez pas les indications (un peu brèves, nous vous le concédons), il faudra attendre le moment où nous traiterons de ce point particulier des menus.

3.7. LES INSTRUCTIONS DU 68000

PROFIMAT ST est en mesure de comprendre toutes les instructions des processeurs 68000. La taille des opérandes utilisables pour chaque instruction vous sera indiquée dans le tableau AE dont nous venons de parler brièvement.

En général, on peut dire que pour toutes les instructions, dont la taille des opérandes est définie (par l'équipement-machine), il n'est pas nécessaire de les indiquer à la suite de l'instruction (comme pour *BCD* ou *MOVEQ*, par exemple).

Lorsque pour les instructions, on ignore la taille des opérandes, le traitement s'effectuera, en général, sur la base de traitement de mots (.W). Seul pour *EXT* et quelques autres instructions, il est nécessaire d'indiquer un opérande, pour éviter une erreur.

Mais comme PROFIMAT ST se tient au standard de *MOTOROLA*, vous ne devriez pas rencontrer de problèmes.

En complément de ceci, il est encore possible d'utiliser, pour certaines instructions, une alternative à ce standard *MOTOROLA*.

Standard	Alternative
ADDA,ADDI	ADD
SUBA,SUBI	SUB
CMPA,CMPI	CMP
ANDI	AND
EORI	XORI
EOR	XOR
ORI	OR
MOVEA	MOVE

Pour ce qui est de l'utilisation des instructions *Bcc*, *Scc* et *DBcc*, pour les paramètres devant être indiqués, vous pouvez, tout aussi bien utiliser HS (Higher or Same) à la place de CC et à la place de CS, LO (Lower).

L'emploi de ces écritures alternatives permet d'économiser un peu de temps d'écriture (instructions plus courtes), voire même de gagner du temps sur l'exécution (pour les paramètres). Mais c'est avant tout une affaire d'habitude d'utilisation.

3.8. LES INSTRUCTIONS D'ASSEMBLAGE

3.8.1. TEXT, DATA et BSS

Syntaxe ------> »SEGMENT

Segment peut être utilisé pour du texte (*TEXT*), des données (*DATA*) ou des blocs (*BSS*).

En fait, vous pouvez diviser votre programme en trois parties.

Dans le segment TEXT, vous placerez le programme en lui-même.

Dans le segment DATA, les données qui ont été initialisées sont stockées. Il s'agit des données qui sont nécessaires au fonctionnement du programme. On y trouvera, par exemple, les textes qui doivent être édités, les arborescences, etc...

Le segment BSS (*Block Storage Segment*) sert à stocker les données élaborées par le programme, et qui ne lui sont pas nécessaires.

Si on peut considérer les segments TEXT et DATA comme des fioritures, le segment BSS a une utilité tout à fait justifiée. En effet, cette partie ne sera pas sauvegardée et ne prendra donc pas de place sur la disquette.

Pour commuter sur le SEGMENT désiré, il faut en taper le nom, *TEXT*, *DATA*, ou *BSS* sur une ligne de commande. Toutes les instructions suivantes seront alors assemblées dans le segment désigné. Chaque segment débute à une adresse paire.

L'ordre dans lequel sont placés les segments en mémoire est le suivant. Tout d'abord on trouve la partie *TEXT*, puis *DATA* et enfin *BSS*. Cette disposition n'a pas d'importance, en ce qui concerne l'espace d'accès des variables locales. On pourrait utiliser la macro suivante en guise de définition :

```
PRINT: MACRO $\MESSAGE
»PRINTLINE \ADR
»DATA
\ADR: DC.B \MESSAGE,0
```

```
»ALIGN
»TEXT
»ENDM
```

Il suffira alors de l'appeler par :

```
»PRINT 'CECI EST UN TEXTE'
```

Ne vous formalisez pas si vous ne savez pas encore que *'$\MESSAGE '* est un paramètre de transmission du type *SYMBOL. PRINTLINE* est une macro, que vous trouverez dans la bibliothèque *TOS.Q*. Mais nous reviendrons sur ce point plus tard.

3.8.2. DC (Define Constant)

Pour déposer une donnée quelconque dans votre programme, vous utiliserez, la plupart du temps, l'instruction *DC*. La syntaxe générale de cette instruction est la suivante :

```
»DC.x Expression [,Expression]
```

où x correspond à B (*BYTE*), W (*WORD*) ou L (*Long Word*). Les données seront alors placées en mémoire comme octets, mots ou mots longs. On pourrait comparer cette instruction à la commande *POKE* que vous connaissez certainement du BASIC. La seule différence réside dans le fait qu'il n'y a pas d'adresse, mais par contre il est possible de traiter plusieurs données en même temps.

Si vous désirez définir des données à l'aide de *DCB*, vous pouvez remplacer l'expression arithmétique par une chaîne de caractères d'une longueur indéterminée comprise entre guillemets ou apostrophes.

EXEMPLE :

```
»DC.B ' CECI EST UN TEXTE',0

»DE.L 0, 0, 7

»DC.W 4711,'ST',0

»DC.B 0, 6, "CHAINE",0
```

Pour cette commande, il n'est pas possible d'omettre la taille des opérandes. Vous n'avez donc pas le droit de remplacer *DC.W* par *DC*. Pourtant il existe d'autres alternatives :

Texte	Alternative
DC.B	DEFB,DEFM
DC.W	DEFW
DC.L	DEFL

3.8.3. DS (Define Space)

```
Syntaxe ------> »DS.x nombre [, valeur de remplissage]
```

x doit contenir une des lettres B, W ou L. Nombre et valeur de remplissage sont des expressions arithmétiques. Cette instruction permet de définir un espace mémoire dont la taille dépend de la taille des opérandes. La réservation de mémoire porte sur le nombre d'octets, de mots ou de mots longs. Cet espace est rempli d'octets ayant pour valeur la valeur de remplissage. Si vous omettez de préciser cette dernière, l'espace sera rempli de zéros.

A la place de

```
»DS.x A,B
```

vous pouvez aussi utiliser A fois

```
»DS.x B
```

EXEMPLE :

```
»DS.B 20
;définit 20 octets

»DS.W 30,-1
;définit 30 mots (= 60 Octets) et les remplit avec la
valeur -1.

»DS.L 40, $1234
;définit 40 mots longs (=160 Octets) et les remplit avec la
valeur $1234
```

Pour cette commande, il faut là aussi, ne pas omettre d'indiquer les opérandes. Mais vous pouvez utiliser l'alternative *DEFS* à la place *DS.B*.

3.8.4. =

```
Syntaxe ------> Nom=valeur
```

où le nom est le nom de la variable et la valeur est une donnée arithmétique. Ceci permet d'attribuer à la constante *'Nom'* la valeur *'Valeur'*. Si vous deviez l'avoir déjà oublié, vous trouverez

au chapitre 3.4. relatifs aux variables, la façon de différencier les constantes, labels etc...

3.8.5. EQU

```
Syntaxe ------> Nom EQU chaîne
```

définit un symbole. Là aussi revoyez le chapitre 3.4..

Pour attribuer une valeur numérique à un nom, vous devriez utiliser continuellement les constantes. Vous devriez penser à ceci lorsque vous tapez un programme en provenance d'un livre ou d'une revue spécialisée et que le programme est en assembleur.

3.8.6. ORG

```
Syntaxe------> »ORG adresse
```

où adresse est une valeur arithmétique. Si vous écrivez un programme en codes absolus, autrement dit si vous savez exactement l'endroit en mémoire où il se trouvera, il est possible, à l'aide de cette commande, de définir l'adresse à partir de laquelle ce qui suit doit être assemblé. Le programme sera alors utilisable à cette seule adresse.

Cette commande n'a pas d'influence sur le lieu où l'assembleur place le programme pour le traiter. En règle générale, il n'est pas possible de tester un programme absolu à l'aide d'un *DEBUGGER*.

En règle générale, la commande *ORG* n'est utilisée qu'une seule fois et ceci au début du programme. Si vous divisez votre programme en segments, vous ne pourrez pas utiliser de commandes *ORG* dans les

parties *DATA* et *BSS*. En effet, elles sont placées l'une derrière l'autre. Les adresses de début de ces deux segments sont donc définies une fois pour toutes. Si vous désirez vous passer de cette forme de '*segmentation*', il vous sera possible d'insérer d'autres commandes *ORG* dans votre programme.

Ceci peut être utile, par exemple, si vous désirez programmer une *EPROM*. Il est vrai que vous ne pourrez pas étendre vraiment votre programme, c'est pourquoi il est de la plus haute importance, dans ce cas, de placer les données initialisées dans un autre espace mémoire.

EXEMPLE :

```
»ORG $FA0000
.....
```

Le programme qui suivra cette commande sera utilisable à partir de l'adresse $FA0000.

3.8.7. END

```
Syntaxe------> END
```

Il faut clore votre texte par un *END* pour signifier à l'assembleur, que rien ne suit cette instruction. Si vous oubliez de placer un *END*, vous aurez droit à un message d'erreur. Si votre programme se compose de plusieurs textes-source (de plusieurs modules), il faut que chacun d'entre eux se termine par l'instruction *END*. En effet, si vous utilisez un texte-source extérieur, à l'aide de la commande *INCLUDE*, l'assembleur a besoin d'un *END* pour pouvoir revenir au texte qui a fait appel à cette routine extérieure.

EXEMPLE :

```
; ici se trouve votre programme

.....

»END
; tout ce qui se trouve au-delà de cette commande n'est
plus pris en considération.
```

3.8.8. ALIGN

```
Syntaxe------> ALIGN.x
```

Seuls W et L peuvent être utilisés à la place de 'x'. Il faut se rappeler que pour un MC 68000, l'accès aux mots et mots longs ne peut s'effectuer qu'à des adresses paires. Il faut donc vous assurer que les mots ou mots longs se trouvent bien à de telles adresses. Pour être sûr que la prochaine instruction figure bien à cette adresse, il suffit de placer *ALIGN.W* dans la ligne précédant cette instruction.

Si pour une raison ou une autre, vous devez être sûr que l'instruction doit se trouver à une adresse divisible par 4, vous pouvez mettre devant l'instruction la séquence *ALIGN.L*. Mais cette instruction n'a de sens que pour un programme absolu.

La plupart du temps, vous utiliserez l'instruction *ALIGN* après une instruction *DC.B* pour avoir la certitude que l'instruction suivante figure bien à une adresse paire. Un exemple de ceci se trouve dans le paragraphe 3.8.1.

3.8.9. INCLUDE

Syntaxe-------> INCLUDE fichier

Il va de soi que par 'fichier' on entend le nom du fichier que vous désirez insérer. Il doit se trouver dans le sommaire (DIRECTORY) du moment. Mais vous pouvez aussi indiquer un nom de chemin complet comme par exemple B.:

```
»INCLUDE A:\TOS\TOS.L
```

Le nom de fichier ou de chemin peut être inclus de façon optionnelle entre apostrophes ou guillemets. Si le nom de fichier ne se trouve pas dans le sommaire du moment, l'assembleur effectuera tout d'abord ses recherches dans le sommaire global avant qu'un message d'erreur ne soit édité.

Il faut que le fichier considéré soit clôs par une commande *END*. L'assemblage sera dévié dans ce nouveau fichier et reviendra dans le programme d'origine après avoir rencontré la commande *END*.

Vous pouvez, par exemple, placer tous vos sous-programmes, dans un fichier, et dans un autre programme. Il faut alors placer judicieusement l'instruction *INCLUDE*, de préférence vers la fin, pour chaîner les deux textes-source.

Pourtant il est bien plus prudent, si votre programme atteint une taille assez importante, d'utiliser plusieurs textes-source. La recherche d'erreur se fera bien plus aisément, sans pour autant être obligé de charger tout le programme.

Si votre application est assez complexe et que votre programme emboîte plusieurs instructions *INCLUDE*, n'oubliez pas pour autant qu'il ne vous est pas possible d'en emboîter plus de 30 (ce qui est déjà très confortable).

La taille à partir de laquelle il vaut mieux utiliser des sous-programmes est une affaire de goût, donc affaire de personnes. Quelques chiffres vous aideront peut-être à vous faire une idée plus technique.

Les textes-source de PROFIMAT-ST ont une longueur de 300 à 1200 lignes (PROFIMAT est décomposé en 35 textes-source).

3.8.10. IBYTES

Syntaxe------> »IBYTES fichier [,longueur]

Le mot '*fichier*' représente bien entendu le nom du fichier, qui d'une façon optionnelle peut être placé entre guillemets ou apostrophes. Le mot '*longueur*' correspond à une expression arithmétique. Avec *IBYTES*, il est possible de chaîner les données en provenance d'une disquette. Le fichier doit se trouver dans le sommaire du moment.

S'il n'y est pas, l'assembleur cherche celui-ci dans le sommaire global avant d'éditer un message d'erreur.

Les données sont insérées dans le programme à la place où se trouve la commande et dans l'ordre où elles se trouvent sur la disquette. Si vous ne désirez pas insérer la totalité des données vous pouvez, à l'aide de l'option '*longueur*', indiquer le nombre d'octets que vous désirez voir figurer dans le programme.

EXEMPLE :

Vous désirez insérer dans votre programme une image que vous avez générée avec un programme *ad hoc* dans le but de la faire éditer par celui-ci. Si votre fichier image s'appelle *'IMA1.PIC'* il suffit de placer la séquence suivante :

```
»IBYTES IMA1.PIC
```

ou

```
»IBYTES IMA1.PIC,32000
```

puisqu'une image représente 32000 Octets.

3.8.11. SLABEL

```
Syntaxe------> »SLABEL fichier

»NOM_1
...
»NOM_n
»ENDS
```

Il est possible d'indiquer un grand nombre de noms de variables entre *SLABEL* et *ENDS*. Devant ceux-ci, il faut au moins un caractère espace et il faut prendre garde à ne mettre qu'une variable par ligne. A la suite des noms de variables peut figurer un commentaire séparé du nom par un point virgule. Vous pouvez aussi insérer des lignes ne comportant qu'un commentaire.

Le mot *'fichier'* représente le nom du fichier qui peut être, de façon optionnelle, compris entre guillemets ou apostrophes. Le fichier est inscrit dans le sommaire du moment.

Avec cette instruction, il vous est possible de créer votre propre bibliothèque dans laquelle seront stockées les variables nécessaires par la suite. Cette bibliothèque est accessible à volonté, pour un programme, à l'aide de la commande *ILABEL*.

Vous pouvez sauvegarder des labels, constantes, symboles et macros, en n'utilisant, bien entendu, que des variables globales. Avec cette instruction, on accède aussi à la bibliothèque jointe et qui traite des appels de routine du système d'exploitation. Le programme en question à pour nom *'TOS_SV.L'* et se trouve dans le dossier *TOS* sur la disquette PROFIMAT-ST.

EXEMPLE :

```
M1:MACRO

»ENDM
K1=10
S1 EQU D0-A6
L1:
»SLABEL BIBLIOTQ_L
;Label:
»L1
;Macros:
M1
;Constantes:
»K1
;Symbole:
»S1
»ENDS
```

3.8.12. ILABEL

Syntaxe------> »ILABEL fichier

'fichier' peut être, là aussi, de façon optionnelle, compris entre guillemets ou apostrophes. Le fichier doit comprendre des variables, il doit donc avoir été généré à l'aide d'instructions *SLABEL*. Si le fichier ne se trouve pas dans le sommaire du moment l'assembleur le cherchera dans le sommaire global avant d'éditer un message d'erreur.

Cette instruction permet, par exemple, de chaîner des bibliothèques comme *TOS.Q*, fournies sur la disquette.

```
»ILABEL TOS\TOS.Q
```

Le nom du chemin est nécessaire puisque le fichier se trouve dans le dossier *TOS*. Toutes les macros et constantes de TOS.Q peuvent alors être utilisées dans votre programme.

Lorsque vous sauvegardez des labels, si vous les chaînez à l'aide de *ILABEL*, la valeur de chaque LABEL est additionnée l'adresse où se trouve l'instruction *ILABEL*. Si, par exemple, vous avez sauvegardé un label se trouvant à l'adresse 20 et que la commande *ILABEL* se trouve, dans votre programme, à l'adresse 148, le label aura pour valeur 168. C'est ainsi qu'il est possible de travailler de concert avec l'instruction *IBYTES* pour chaîner un programme assemblé et fini.

EXEMPLE :

(Si vous vous comptez parmi les débutants, gardez cet exercice pour plus tard car il ne compte pas parmi les plus faciles à comprendre)

Vous avez écrit le programme suivant :

```
CONVERT_INT_TO_HEX
;Ce programme convertit un entier en une
;chaîne héxadécimale.

;Le programme doit encore être conçu
;par vos soins.

»RTS
CONVERT_INT_TO_DEC;Conversion en décimale


»RTS

»SLABEL CONVERT_L
»CONVERT_INT_TO_HEX
»CONVERT_INT_TO_DEC
»ENDS

END
```

Assemblez à présent ce programme de façon *'PC RELATIF'!* puis sauvegardez le code généré sous le nom *CONVERT.B*. Il est à présent possible d'utiliser dans un autre programme les deux routines en procédant de la manière suivante :

```
»ILABEL CONVERT.Q
»IBYTES CONVERT.B
```

L'ordre dans lequel figurent *ILABEL* et *IBYTES* est important, cela permet un report des variables correct. Les programmes que vous chaînez de cette manière doivent être écrits en code *PC RELATIF*, et fonctionner, bien entendu, correctement. Ils ne peuvent comprendre des routines ou des variables qui ne sont pas sauvegardées en même temps. A part cela le programme doit être sauvegardé avec l'extension '.B', donc sans amorce de programme.

3.8.13. MACRO

```
Syntaxe------> NOM:MACRO [paramètre[,paramètre]*]
.....;ici se trouve votre programme

»ENDM
```

Nom et paramètre sont des noms de variables. *'Nom'* est en fait le nom de la macro alors que *'paramètre'* représente les paramètres locaux de transmission. On peut y trouver les paramètres suivants :

$paramètre	:	symbole
%paramètre	:	constante
*paramètre	:	label
paramètre	:	label ou constante

La macro est simplement appelée par le nom qui lui a été donné :

```
PRINTLINE:MACRO $\ADR
»PEA \ADR
»MOVE #9, -(SP)
```



```
»TRAP #1
»ADDQ.L #6,SP
»ENDM

.....

»PRINTLINE TEXT

.....

TEXT:DC.B "Ceci est un texte",0
```

D'autres exemples de programmation de macros se trouvent dans les fichiers GEMDOS.L, BIOS.L, XBIOS.L, AES.L, VDI.L et TOS.L qui se trouvent tous dans le dossier *TOS* et le texte-source de la bibliothèque dans *TOS.Q*.

Le nombre des paramètres de transmission n'est pas limité. Ils doivent pourtant tous tenir sur une ligne dans laquelle figure aussi l'appel de la macro. Il ne reste donc que 28 possibilités emboîtées. Si le texte-source est appelé par un *INCLUDE* dans un autre texte-source, ce ne sont plus que 27 possibilités qui restent disponibles. Mais il semble peu probable qu'un programme normal atteigne ces sommets.

Par contre, il n'est pas possible de définir une macro à l'intérieur d'une autre macro.

3.8.14. IF (assemblage conditionnel)

```
Syntaxe------> IFcc expression, expression
```

```
.....

» [ELSE]

.....

»ENDIF
```

Pour 'cc' vous pouvez introduire une condition quelconque à l'exception de T, F et RA. *'Expression'* représente une expression arithmétique. Le test de la condition est réalisé comme pour l'instruction Bcc. Si la condition est remplie, on continue en séquence jusqu'à ce que l'assembleur rencontre l'instruction *ENDIF*. Si l'assembleur arrive à une instruction *ELSE*, l'assemblage ne se poursuit qu'après la commande *ENDIF*. Si la condition n'est pas remplie, la chose se poursuit normalement.

EXEMPLE :

```
»IFHI 0,3
»NOP; Ceci va être assemblé puisque 3>0
»ELSE
»DC.W 2
»ENDIF

»IFLT 0,1
»NOP
»ELSE
»DC.W 4; Ceci sera assemblé car non 1<0
»ENDIF
```


L'instruction IF ne devient intéressante que si des variables se trouvent dans l'expression arithmétique. Un exemple est repris pour la commande *REPEAT*.

Il n'est pas possible d'emboîter plus de 16 boucles IF en même temps.

3.8.15. REPEAT (assemblage répété)

```
Syntaxe------> REPEAT

.....

UNTILcc expression, expression
```

'cc' peut représenter une condition à l'exception de T, F et RA. 'expression' représente une expression arithmétique. Pour éviter des explications à vous couper le souffle, voici un exemple :

```
N@='0'
»REPEAT
»DC.B N@
»IFEQ N@,'9'
N@='A'
»ELSE
N@=N@+1
»ENDIF
»UNTILHI 'F',N@
```

Ce petit programme génère le même code que la ligne suivante :

```
»DC.B '0123456789ABCDEF'
```

Reprenez le 1er programme et analysez-le si vous ne l'avez pas bien compris. Ce qui peut paraître dérangeant, mais ne possède pas de solution, c'est que la constante redéfinissable doit toujours figurer dans la première colonne lorsque vous lui attribuez une valeur.

Pour ce qui est de l'instruction *REPEAT*, il n'est pas possible d'en utiliser plus de 8 imbriquées.

3.8.16. INPUT

```
Syntaxe------> »INPUT [message,]variable
```

où *'Message'* est constitué par une chaîne de caractères et *'variable'* par une variable. Il est très important, pour cette instruction qu'aucune variable ne la précède (ce qui n'était pas le cas pour les instructions précédentes). Le type de la variable peut être global, local ou redéfinissable.

On peut utiliser les types suivants :

$variable	:	symbol
%variable	:	constante
*variable	:	label
variable	:	label ou constante, suivant le cas si dans l'expression qui est transmise à la variable, il existe un label ou non.

Lorsque l'assembleur arrive à une instruction *INPUT*, vous verrez apparaître lors de la passe 1 (il ne se passe rien lors du passage en passe 2) un formulaire dans lequel se trouve le message. Si un tel message n'existe pas, vous verrez apparaître simplement un point d'interrogation. Vous devez alors saisir une valeur. Suivant votre désir de définir une constante ou un symbole, vous devrez introduire une expression arithmétique ou une chaîne de caractères.

L'instruction INPUT est identique à la définition normale d'une variable à l'aide de '=' (*constante*) ou 'EQU' (*symbole*). La seule différence provient du fait qu'il est possible de définir la valeur pendant l'assemblage. Vous pouvez donc remplacer :

```
CONSTANTE=3
```

par

```
»INPUT 'CONSTANTE =', CONSTANTE
```

et taper le chiffre 3 dans le formulaire d'interrogation.

A la réflexion, il est intéressant de travailler avec cette instruction, si vous désirez créer plusieurs versions d'un même programme, suivant une valeur donnée et variable. Vous pourrez alors générer toutes les versions avec le même texte-source et les personnaliser lors de l'assemblage. Vous placerez alors en début de programme la commande *INPUT* qui vous permettra de définir d'emblée la version que vous désirez réaliser.

3.8.17. START

```
Syntaxe------> START expression
```

Cette instruction permet de définir le début du segment *TEXTE*. L'expression arithmétique que vous indiquerez sera placée dans l'amorce du programme. Cette amorce (*HEADER*) à la structure suivante :

```
+ 0 : Mot       = $601A
+ 2 : Mot long  = Longueur du segment texte
+ 6 : Mot long  = Longueur du segment Data
+10 : Mot long  = Longueur du segment BSS
+14 : Mot long  = Longueur du tableau des symboles
+18 : Mot long  = 0
+22 : Mot long  = Début du segment texte
+26 : Mot       = drapeau de relogement.
```

3.8.18. PAGE, LIST et NOLIST

Ces instructions n'ont d'influence que sur l'édition (listage) si vous désirez obtenir un tel effet.

Elles ont les significations suivantes :

PAGE — permet d'obtenir l'avancée d'une page

NOLIST — permet d'interdire l'édition.

LIST — permet de rétablir l'autorisation d'édition si NOLIST a été employée.

Les 3 instructions n'apparaissent pas à l'édition. Si vous ne désirez pas préparer une édition, elles n'ont aucune utilité. Vous ne pourrez donc pas obtenir de listing, si vous n'avez pas prévu celui-ci dans le menu (même en utilisant *LIST* après).

3.9. LE MENU

Assembler :

Lorsque vous désirez assembler votre programme, il faut cliquer sur ce point du menu. Il faut pourtant vous assurer que tous les réglages situés dans ce menu correspondent à ce que vous désirez.

Vous trouverez les messages d'erreur possibles dans une annexe de ce livre. Le curseur de la souris est déconnecté pendant l'assemblage. Si vous désirez, pour quelque raison que ce soit, interrompre le processus d'assemblage, il suffit d'appuyer en même temps sur les deux touches *SHIFT*.

Vous obtiendrez alors le message d'erreur correspondant au moment de l'interruption et vous aurez la possibilité, soit de retourner dans le menu, soit de poursuivre l'assemblage.

Si vous n'avez pas choisi d'imprimer les messages d'erreur, ou de les éditer dans un fichier, vous obtiendrez, lors de l'apparition de l'erreur, un formulaire qui vous informera de la situation. (type d'erreur et ligne où elle s'est produite, ainsi qu'une flèche pointant sur l'endroit présumé de l'erreur).

Vous trouverez aussi le nom du fichier dont dépend cette ligne ainsi que son numéro. Une aide vous sera aussi fournie vous permettant d'accéder au tableau dont nous reparlerons un peu plus tard.

Corrigez l'erreur dans la mesure du possible. Si vous cliquez alors sur le point : *REFAIRE UN ESSAI*, l'assembleur effectuera un nouvel essai d'assemblage de la ligne. S'il réussit, il continuera son travail (jusqu'à la prochaine erreur s'il en existe encore).

Un bip est émis lorsque l'assemblage est terminé.

Code-Source :

Vous déterminez ici, si vous désirez assembler un texte actuellement en mémoire de l'éditeur ou stocké sur disquette (ou disque dur). Si vous souhaitez assembler un fichier stocké sur disquette ou disque, il vous faut cliquer sur "fichier" puis saisir son nom dans la plage de saisie, et valider votre choix en cliquant sur OK. Dans l'autre cas, cliquez sur "mémoire" et "OK".

Code-objet :

Ce point permet d'effectuer un choix similaire au point précédent mais en ce qui concerne le code-objet.

Si vous assemblez le programme directement sur disquette, cet assemblage s'effectuera d'abord en mémoire, à condition que la place nécessaire existe. Il sera alors sauvegardé sur disquette, à la fin de ce processus. Ceci constitue un facteur de gain de temps non-négligeable.

Editer nom de fichier :

Si vous désirez suivre les différentes étapes de l'assemblage, il vous faudra cliquer sur cette option. Dans la fenêtre de l'assembleur, vous verrez apparaître le numéro de la ligne assemblée ainsi que toutes les instructions qui composent les macros auxquelles vous avez fait appel dans votre programme.

Si vous attachez de l'importance à la vitesse d'exécution, il faudra éviter d'utiliser cette possibilité car elle double à peu près le temps d'exécution.

Si vous désirez générer en même temps une édition de votre programme, le nom des fichiers n'apparaîtra que lors de la passe 1. Lors de la passe 2 s'effectuera l'édition.

Optimisation arrière :

Pour un assembleur, il est toujours bon de pouvoir le laisser générer des codes optimisés. Il est vrai que lors de la deuxième passe d'assemblage, qui permet en plus la segmentation en trois parties, cette possibilité est relativement limitée. C'est la raison pour laquelle une seule possibilité demeure, très fiable.

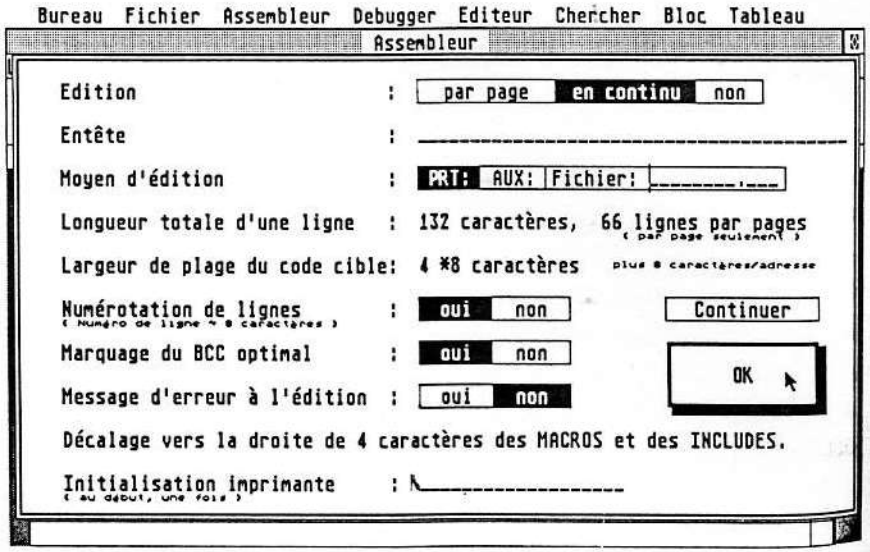
Si toutes les instructions Bcc, auxquelles l'extension '.S' n'a pas été indiquée, ont pour distance de sauts une valeur comprise entre - 128 et - 2, il faudra cocher ce point. Vous aurez alors affaire à des sauts courts assemblés, autrement dit vous économiserez deux octets.

Editer variables non définies :

Normalement, ce point du menu est coché, pour permettre la visualisation de toutes les variables non définies par l'édition d'un message d'erreur. Mais si vous ne désirez que tester votre programme du point de vue syntaxique, vous pouvez ne pas sélectionner ce point. Des instructions inexistantes comme des macros indéfinies, feront toujours l'objet d'une remarque, quelque soit l'état de ce point. En effet la taille du programme peut varier de façon considérable suivant ce type d'erreur.

Edition :

Ce formulaire gigantesque (voir figure 3) vous permet de choisir si une édition (listing) de votre programme doit être effectuée, avec certaines options ou non.



Voyons celui-ci par le menu.

La ligne supérieure (*EDITION*) permet de définir si une édition doit avoir lieu ou non (*NON*). Si vous désirez une édition, vous pouvez choisir entre une formule page par page ou en continu. Si vous avez sélectionné l'édition dans le menu Assembleur, ce point du menu sera précédé d'un check-mark

L'en-tête n'est imprimé qu'une seule fois, lors de l'édition en continu en début d'impression. Si vous avez choisi l'option page par page, cet en-tête figurera au début de chaque page. De même, vous trouverez dans la ligne où se trouve l'en-tête la disposition d'une ligne en adresse, code-source, numéro de ligne et texte-source. Pour l'édition page par page, les numéros de page sont incrémentés et imprimés automatiquement.

Vous pouvez définir le périphérique qui va recevoir l'édition. Le choix se porte entre le port imprimante (*PRT*), l'interface *RS 232 (AUX)* ou un fichier. La configuration imprimante peut être définie avec l'accessoire de bureau adéquat.

La largeur totale d'une ligne doit être, en tout état de cause, correctement définie pour éviter une mauvaise mise en page. En effet l'impression est orientée suivant les colonnes, autrement dit, des lignes trop longues seront formatées de telle sorte que les codes-source se trouvent sous les codes-source et les textes-source sous les textes-source.

Le nombre de lignes par page ne doit être indiqué que si vous avez choisi l'édition page par page. Dans le champ code-source, vous verrez apparaître la valeur du code en hexadécimal.

Dans la plage relative au code-cible, les valeurs indiquées figurent en hexadécimal. Si vous indiquez comme largeur de code-source la valeur 0, les valeurs des codes-source et les adresses de l'assemblage, ne seront plus éditées. Il va de soi que la somme des valeurs indiquées (pour les codes-source et les adresses) ne doit pas dépasser la largeur totale de la ligne fixée précédemment.

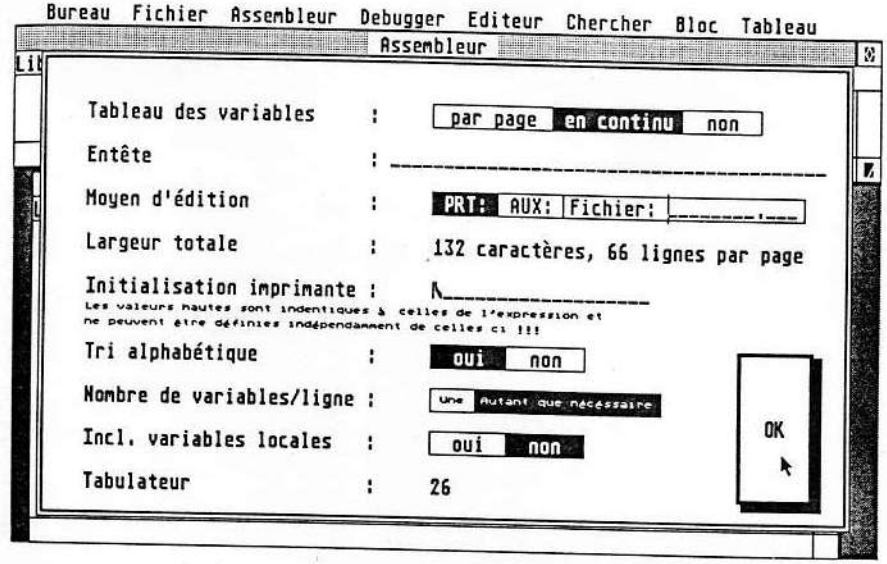
Pour ce qui est de la numérotation des lignes, vous pouvez définir si les numéros de lignes doivent être imprimés, préciser quels numéros de lignes correspondent au texte-source, et déterminer si les lignes doivent être numérotées en continu. Si vous ne vous rendez pas bien compte de la différence, nous ne pouvons que vous conseiller d'en faire l'essai à l'aide d'un petit programme qui, soit utilise plusieurs textes-source, soit utilise des macros.

Un autre réglage consiste à faire figurer toutes les commandes Bcc optimal précédées par le caractère '>'. On peut aussi obtenir l'impression des messages d'erreur, si vous ne désirez pas les voir apparaître à l'écran.

Pour une meilleure compréhension, vous pouvez demander le décalage des macros et des includes de 1 à 9 caractères vers la droite.

Vous disposez aussi d'une ligne d'initialisation de votre imprimante permettant de programmer jusqu'à 20 caractères qui seront envoyés à votre périphérique, dans lesquels figureront, bien entendu, des caractères *ESC*. Pour obtenir un caractère de ce type il suffit de taper sur les touches *CONTROL* ';'. Cette possibilité permet de sélectionner le type d'écriture (écriture resserrée, par exemple) ainsi que le registre de caractères (caractères accentués français, par exemple).

Tableau des variables :



Le formulaire est presqu'aussi grand que celui de la partie *EDITION*. Les premiers réglages sont pratiquement identiques à ceux vus dans le paragraphe précédent. Nous ne reviendrons pas sur ces points, reportez-vous au paragraphe précédent. La seule chose importante qu'il faut retenir, c'est que le réglage des points communs aux deux formulaires, est valable pour les deux tableaux. Il n'est donc pas possible de régler l'édition d'une façon et le tableau des variables d'une autre.

Si vous avez choisi pour l'édition, le mode page par page, et que dans le tableau des variables vous choisissez le mode continu, l'édition s'effectuera suivant ce dernier mode. Il en va de même pour ce qui concerne l'en-tête, le moyen d'édition, l'initialisation de l'imprimante, qu'il sera nécessaire de taper qu'une seule fois, la largeur totale et la longueur de la page.

Voyons les autres réglages qui sont propres au tableau des variables.

Vous pouvez définir si vous désirez voir figurer les variables suivant l'ordre dans lequel vous les avez créées, ou si vous préférez les voir dans l'ordre alphabétique. N'oubliez pas, au moment d'effectuer ce choix, que les variables locales apparaissent entre des variables globales et peuvent porter le même nom. Il est donc indéniable que dans ce cas, c'est plutôt l'ordre d'apparition qui doit prévaloir. Dans ce cas, il faudra répondre par oui, au point définissant l'affichage des variables locales.

Vous pouvez également définir si vous voulez voir apparaître une variable par ligne, ou plutôt si vous désirez les voir les unes après les autres sur une ligne, autant qu'il est possible d'en placer. La place utilisée par une variable dépend de sa taille et des tabulateurs. En effet, lorsqu'une variable figure dans une ligne, on place autant de caractères vides jusqu'à ce que la prochaine variable puisse être placée en entier suivant un tabulateur.

Seuls seront édités les labels et constantes, et ceci sous forme hexadécimale.

Fichier d'erreurs :

Si vous trouvez qu'il est fastidieux d'attendre que l'assembleur trouve une erreur et vous le fasse savoir par voie de message d'erreur à l'écran, voici un moyen de ne pas vous énerver.

Vous donnez tout simplement le nom d'un fichier, qui sera indiqué dans le menu et dans lequel les messages d'erreur seront enregistrés. Il s'agit, bien entendu, d'erreurs pouvant être corrigées et vous pourrez exploiter ce fichier.

A la fin de l'assemblage, vous verrez apparaître une fenêtre qui vous indiquera le nombre d'erreurs découvertes. Si votre programme comporte des erreurs, il vous faudra faire appel à l'éditeur et charger le fichier d'erreurs. Vous pouvez maintenant les corriger en toute quiétude.

Cette option est très intéressante, surtout pour un programme long. Il faut pourtant savoir que les erreurs ne pouvant être corrigées continuent d'être éditées à l'écran. Elles interrompent le processus d'assemblage. Vous trouverez à la fin de ce manuel, une annexe reprenant les erreurs et les messages correspondants. Il y sera aussi indiqué si une erreur donnée peut être rattrapée ou si elle est fatale.

PC Relatif

Si ce point est coché, l'assembleur génèrera des codes PC relatif (voir chapitre 3.10. LES CODES GENERES).

Relogeable :

Si ce point est coché, les codes générés seront relogeables. (voir chapitre 3.10. LES CODES GENERES).

Ligne originale :

Ce point permet de choisir entre l'édition de la ligne originale, de la ligne assemblée et celle du message d'erreur. Il faut encore savoir ce que cela signifie.

La ligne originale est identique à celle que vous avez tapée. Mais cette ligne ne peut être traitée telle quelle et les minuscules doivent être converties en majuscules (exception faite des commentaires). Le remplacement des éventuels symboles devra aussi être exécuté. Par exemple :

Vous avez défini un symbole ayant pour nom TOUS_REGISTRES

```
TOUS_REGISTRES:EQU D0-A6
```

la ligne originelle :

```
»MOVEM.L tous_registres,-(sp)
```

sera transformée de la manière suivante :

```
»MOVEM.L D0-A6,-(SP)
```

Il est encore important de savoir quel message d'erreur correspond à une ligne donnée. Mais, là aussi, c'est à vous de définir la version qui vous semble la plus intéressante.

Mémoire :

Ce formulaire permet de définir la taille de la mémoire de l'assembleur.

3.10. LES CODES GENERES

Comme vous le savez déjà, PROFIMAT-ST peut générer 3 sortes de code. En fait, plutôt que de 3 codes différents, il s'agit de 3 possibilités différentes d'interpréter l'adresse effective donnée.

3.10.1. Codes absolus

Il s'agit des codes standard de *MOTOROLA*. Tous les adressages effectifs sont transcrits de la manière prescrite par *MOTOROLA*.
Des codes absolus sont alors générés, pour peu qu'aucun des points PC-RELATIF et RELOGEABLE ne soit marqué.

3.10.2. Codes PC-RELATIF

Comme il n'est pas possible, avec le système d'exploitation de l'ATARI, de définir, en règle générale, où le programme va être chargé, vous devriez assembler votre programme de telle manière qu'il soit relogeable à n'importe quel endroit. Si vous avez écrit votre programme de manière absolue, il faudrait utiliser l'adressage d(PC) et placer derrière chaque label que vous désirez utiliser la mention (PC).

Mais comme ceci peut devenir très pénible, surtout pour des programmes longs et compliqués, vous pouvez assembler le programme de telle manière que l'assembleur pense être en permanence en présence de label muni de la mention (PC). Tout ce qui était valable en mode d'adressage a16 ou a32 sera traité comme d(PC), à condition qu'il y ait au moins un label dans l'expression arithmétique

3.10.3. Codes relogeables

Que veut dire reloger ? Toutes les adresses effectives, qui ont été spécialement marquées, sont modifiées de telle sorte que le programme puisse fonctionner à l'endroit où il a été chargé. Cela est rendu possible par la fonction EXEC (GEMDOS $48) qui place un drapeau particulier dans l'annexe du programme (en réalité, ce drapeau est mis à zéro).

L'assembleur marque, pour cela, toutes les adresses dont les expressions seront relogeables. On dit qu'une expression est relogeable lorsqu'elle répond aux conditions suivantes :

Représentez-vous un compteur que vous venez de mettre à zéro. Pour chaque label, qui est additionné à une expression, il faut incrémenter le compteur de un. De même pour chaque label que vous soustrayez, il faut décrémenter le compteur de un.

Lorsqu'après évaluation de l'expression, le compteur est à zéro, l'expression n'est pas à reloger. Par contre, si le compteur est à un, cette expression est réputée relogeable. Si le compteur a une valeur différente de 0 et de 1, il y a édition d'un message d'erreur.

Vous aurez droit, de même, à un message d'erreur, lorsque l'expression est relogeable mais que le format utilisé ne correspond pas à un mot long, ou si vous essayez de multiplier un label, ou toutes autres actions perverses de ce style.

Les adressages effectives a16, a32 et #k seront relogeables. Les expressions de commande DC, sont elles aussi testées en vue d'un relogement éventuel.

Si, par exemple, vous écrivez :

```
POINTEUR:DC.L UP_1

.....

UP_1:

.....
```

la case mémoire POINTEUR contiendra, après chargement, l'adresse effective du sous-programme UP_1. Par contre,

```
MOVE.L #UP_1-UP_2,D0
```

ne sera pas relogé car 'UP_1-UP_2' est une soustraction et le compteur dont nous avons fait état précédemment, est ramené à 0.

Dans la majeure partie des cas, lorsque vous tapez des programmes en provenance de journaux ou livres spécialisés, il faudra générer des codes relogeables, car ces programmes sont presque toujours écrits en assembleur qui ne sont pas à même de générer de tels codes.

L'avantage de ce type de programme tient du fait, entre autres, qu'il est aussi possible de donner comme cible d'une opération, la mémoire, ce qui ne peut être le cas de programmes en mode PC-RELATIF. Un programme relogeable, allie les avantages des programmes en codes PC-RELATIF et en codes absolus.

Mais il existe aussi un inconvénient aux programmes relogeables. Non seulement ils sont plus longs, mais il faut sauvegarder, en même temps, les informations nécessaires à la fonction EXEC. Elles se trouvent placées après le segment des datas et utilisent les codes suivants :

+0 Mot long= adresse du premier mot long à reloger dans le programme (relativement au début du programme)

+4 Octet = Index de la prochaine adresse à reloger.

.....

+n Octet = 0 (marquage de fin)

L'index est toujours pair. Si le contenu de l'octet d'index est égal à un, l'index se trouve dans l'octet suivant qui doit être différent de un. Pour chaque octet, qui est égal à un, il faut ajouter à l'index la valeur 254.

Si l'octet d'index est nul, ceci signifie que toutes les adresses ont été relogées. Si les deux premiers mots longs du segment de relogement sont égaux à 0, il n'existe pas d'adresse à reloger.

Si vous observez les datas de certains programmes Basic permettant de créer des fichiers exécutables (chargeur Basic), vous pourrez, en général, repérer le segment d'index car la majeure partie des assembleurs ne génèrent que des codes relogeables.

3.11. OU PLACER LE PROGRAMME ?

Lorsque vous avez écrit un programme, que vous désirez lancer à partir du BUREAU, il suffit de le sauvegarder en donnant, dans le point *FICHIER* du menu FICHIER, le nom sous lequel vous désirez que votre programme soit stocké. Vous choisirez alors le point *SAUVEGARDER* qui vous fera apparaître une boîte de dialogue, dans laquelle il faudra cliquer sur une extension qui sera adjointe au nom du fichier (il s'agit de *PRG, TOS* ou *TTP).*

Mais que faire si vous ne désirez pas faire démarrer votre programme à partir du bureau mais plutôt à partir du BASIC. Il faut alors assembler votre programme en PC RELATIF et le sauvegarder sous le même nom, mais avec l'extension .B. Il sera alors possible de faire appel à ce programme à partir de BASIC grâce aux commandes prévues à cet effet (*BLOAD* et *CALL).*

La même remarque doit être faite pour un programme devant être chaîné à un autre qui est écrit dans un autre langage. Mais seul le manuel d'utilisation livré avec ce langage pourra vous dire si ce *'mariage'* est possible.

Vous pouvez également placer votre programme à un endroit précis de la mémoire. Mais n'oubliez pas que l'ordinateur aura toujours le dernier mot en ce qui concerne la gestion de la mémoire. Il ne reste donc guère que l'espace mémoire situé au-dessus de la mémoire d'écran qui reste possible. Mais cet espace pèche par son peu de place (768 octets).

Il est indéniable que la manière la plus élégante est d'utiliser un Vecteur *TRAP*. Vous écrirez le programme de telle manière que le

vecteur *TRAP* pointe vers le début de la routine désirée et permette de revenir au bureau à l'aide de *KEEPPROC (GEMDOS $31)* et non avec *TERM!* Cette routine peut alors être appelée avec une instruction *TRAP*. Si vous désirez appeler plusieurs routines, il est possible de placer dans les routines du système d'exploitation une valeur sur la pile ou dans un registre.

Il faut que l'initialisation soit effectuée une fois, par exemple lorsque vous placez le programme dans le dossier AUTO de la disquette BOOT. Votre programme (d'aide) sera disponible aussi longtemps que vous ne mettrez pas l'ordinateur hors tension ou que vous n'appuyez pas sur le bouton *RESET*.

Pour illustrer la manière d'initialiser ce type de routine, reportez-vous au SPOOLER d'imprimante ou au RAM-disque de l'ouvrage *TRUCS et ASTUCES* de l'ATARI ST (Editions Micro Application) ou aux lignes suivantes :

```
»ILABEL A:\TOS\TOS.Q
»MOVE.L 4(SP), A5
»MOVE.l #256,D7          ;taille de la page de base
»ADD.L 12(A5),D7         ;taille du SEGMENT texte
»ADD.L 20(A5),D7         ;taille du SEGMENT data
»ADD.L 20(A5),D7         ;taille du SEGMENT BSS


; il n'est pas nécessaire de réserver de la place
; pour la pile puisqu'elle sera mise à disposition
; par le programme appelé.

;passons au détournement, par exemple
; TRAP#15 (vecteur47)
```



```
»SETEXEC ROUTINE, #47

; et retour au bureau (DESKTOP)

»KEEPPROC #0,D7; pas d'erreur

ROUTINE; ceci est la routine désirée.
»MOVEM.L D0-A6,-(SP); sécuriser une première fois
; le registre
; Concevoir à partir d'ici le programme.
; Prenez garde au fait que toutes les erreurs soient
; prises en compte.

»MOVEM.L (SP)+,D0-A6; Retour registre

; et retour au programme d'appel

»RTE
»END
```

Il va de soi que cet exemple doit encore être étoffé par le programme que vous allez élaborer, par exemple, si vous désirez exploiter la valeur traitée.

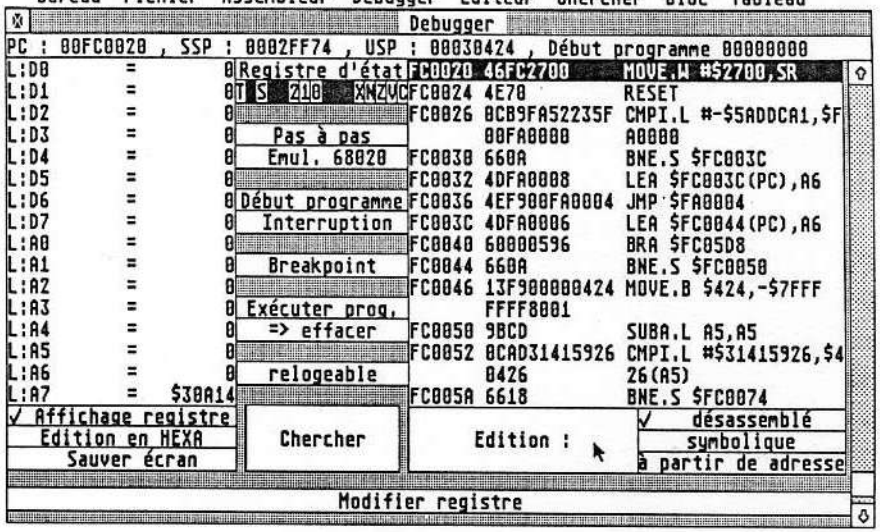
4. LE DEBUGGER.

Un debugger est un outil de programmation permettant de rectifier les erreurs survenues lors de la programmation. Il peut aussi servir à révéler les derniers secrets du système d'exploitation.

Pour y accéder, il faut cliquer sur le point de menu *debugger* dans le menu 'DEBUGGER'. La fenêtre d'assemblage est alors agrandie à sa taille maximum pour mieux servir la partie DEBUGGER. PROFIMAT-ST ne se sert que de deux fenêtres, de telle manière que le programme que vous désirez traiter puisse lui aussi disposer de deux fenêtres. Pour quitter le DEBUGGER il suffit de cliquer sur le petit carré en haut à gauche de la fenêtre.

4.1. LA FENETRE DU DEBUGGER

La fenêtre du debugger est assez touffue (voir figure 5) et c'est pour cela qu'il est difficile de la réduire ou de la déplacer. Mais ne vous laissez pas impressionner par la multitude d'indications qui y figurent.



4.1.1. La plage d'édition

Dans la grande plage, qui occupe la moitié droite de la fenêtre, vous verrez apparaître le contenu de la mémoire. Cette plage d'édition comprend 16 lignes où apparaît le contenu de la mémoire suivant deux modes :

- en mode DUMP HEX/ASCII
- en désassemblé.

4.1.1.1. Edition en mode DUMP HEX/ASCII

Lorsque vous demandez l'édition suivant le mode HEX/ASCII le format de la ligne sera le suivant :

ADRESSE - CONTENU EN HEX. - CONTENU EN CARACTERES ASCII

Dans la configuration de caractères ASCII, un octet nul est représenté par un zéro digital. L'adresse qui se trouve toujours sur les 3 premiers octets de la ligne, est toujours indiquée sous format hexadécimal. Par ligne, 8 octets sont indiqués, ce qui revient à dire que ce ne sont pas moins de 128 octets qui sont édités. En même temps vous verrez apparaître un curseur que vous pourrez déplacer à l'aide des touches de curseur ou de la souris dans toute la fenêtre d'édition.

Vous pouvez modifier le contenu de la mémoire, en plaçant le curseur à la place désirée, et en frappant la nouvelle valeur. Vous pouvez faire ceci aussi bien dans la plage réservée aux valeurs hexadécimales que celle des valeurs ASCII. L'attribution des touches, et en particulier celles dont la valeur ASCII est supérieure à 127 est absolument identique à celle de l'éditeur. Vous ne devriez donc pas avoir de problèmes.

Il est vrai, pourtant, que vous ne pourrez pas écrire dans tous les espaces mémoire. Les deux premiers mots longs qui contiennent les vecteurs du RESET et de la pile sont, eux aussi, tabous. Tabou est aussi l'espace mémoire qui contient PROFIMAT-ST, la ROM, et l'espace protégé entre RAM et ROM.

La modification est effectuée en mode *SUPERVISEUR*, ce qui fait qu'il est aussi possible d'écrire dans les tableaux de vecteur. Mais ceci peut aussi conduire à un plantage magistral. La modification n'est transcrite qu'au moment où le curseur quitte la ligne. Aussi longtemps que le curseur se trouve encore dans la ligne il est possible de restaurer l'ancienne valeur en appuyant sur la touche *UNDO*. Si des modifications portent sur plusieurs lignes, seule la dernière ligne peut voir l'ancienne valeur restaurée.

Si vous désirez examiner un autre espace mémoire, il suffit, soit d'indiquer directement l'adresse désirée, soit en utilisant les combinaisons de touches suivantes :

SHIFT (curseur haut)	—> 128 Octets en arrière
SHIFT (curseur bas)	—> 128 Octets en avant
CONTROL (curseur haut)	—> 2048 Octets en arrière
CONTROL (curseur bas)	—> 2048 Octets en avant

4.1.1.2. Edition en mode désassemblé

Si vous désirez voir apparaître le contenu de la mémoire sous forme désassemblée, vous disposez de deux possibilités : effectuer ceci sous forme symbolique ou non. La différence n'existe pas seulement dans la disposition des lignes,

Normal : ADRESSE - CODE HEX. -INSTRUCTION DESASSEMBLEE

Symbolique : [LABEL:] INSTRUCTION DESASSEMBLEE

mais aussi la disposition des chiffres. En effet si vous avez assemblé un programme précédemment, les labels et constantes qui ont été définis dans celui-ci, seront aussi utilisés par le debugger. En particulier les adresses qui ont été repérées comme labels dans le programme assembleur seront annotées du label correspondant.

Il est vrai qu'il n'existe pas que des avantages à la représentation symbolique, et notamment en ce qui concerne la clarté du programme affiché lors d'un emploi intensif de constantes. Dans ce cas, il vaut mieux utiliser l'édition normale, comme cela est représenté figure 5.

Pour l'édition désassemblée il existe aussi un curseur, qui, il est vrai, est dépendant de la valeur du compteur de programme (PC). Si la ligne qui est pointée par le compteur de programme est affichée dans la fenêtre d'édition, elle apparaît en inversion vidéo. Vous pourrez vérifier ceci dans la ligne d'information ou le PC est constamment affiché.

Vous pouvez placer le PC sur n'importe quelle ligne. Pour ce faire, pointez le curseur de la souris sur la ligne en inversion vidéo. Maintenez la touche gauche de souris enfoncée et déplacez cette ligne où vous le désirez. Ensuite, relâchez le bouton de la souris. Le compteur ordinal est pointé alors sur la ligne que vous avez choisie.

Si le PC pointe sur une ligne qui n'est pas affichée à l'écran, appuyez simultanément sur les touches CONTROL et sur le bouton gauche de la souris.

Avec la touche curseur, il est possible de modifier la position de la fenêtre d'édition dans la mémoire. Les combinaisons de touches suivantes vous permettront ce type de déplacement :

curseur haut	—> une ligne en arrière
curseur bas	—> une ligne en avant
SHIFT (curseur haut)	—> 16 lignes en arrière
SHIFT (curseur bas)	—> 16 lignes en avant (1 page)
CONTROL (curseur haut)	—> 1024 Octets en arrière
CONTROL (curseur bas)	—> 1024 Octets en avant

Comme vous le voyez, la configuration est plus restreinte qu'en mode HEX/ASCII.

Les colonnes de mouvements présentent dans les deux modes, la position de la fenêtre. Si vous désirez examiner la partie de la mémoire à partir de l'adresse zéro il faudra monter le curseur tout en haut. Pour passer à la ROM, il faut descendre le curseur assez bas.

Vous pouvez examiner d'autres espaces-mémoire à l'aide de ce curseur. Il est vrai qu'il est difficile d'effectuer un placement exact avec l'ascenseur situé à droite de la fenêtre. En effet, la plage d'adressage étant de 16 Moctets, cet ascenseur est très sensible. Toutefois, vous pouvez vous déplacer dans la fenêtre de l'édition en cliquant sur les flèches (haut et bas) situées au même niveau que l'ascenseur.

Pour indiquer une position exacte, il est possible de cliquer sur le point *'A PARTIR DE L'ADRESSE'*. Vous verrez alors apparaître dans la partie *'MODIFIER REGISTRE'* une ligne de saisie ayant l'aspect suivant :

*=$_____

Il vous est possible, là aussi, de saisir une expression arithmétique, qui sera alors traitée comme adresse d'édition.

Si l'édition doit être effectuée sous forme désassemblée ou en HEX/ASCII, elle peut être définie en cliquant sur le champ *'DESASSEMBLE'*. Le mode est alors précédé d'un check mark (désassemblé) ou non (HEX/ASCII).

Pour une édition désassemblée, vous pouvez choisir entre la possibilité d'édition symbolique ou normale. Ce choix est effectué à l'aide de la plage *'SYMBOLIQUE'* précédée d'une marque ou non.

4.1.2. L'édition de nombres

Vous pouvez déterminer si l'édition doit s'effectuer avec des nombres hexadécimaux, ou des nombres décimaux. Pour cela, il faut encore que vous sachiez que des nombres pouvant être égaux ou inférieurs à 9 devraient être édités en décimal et que ceux égaux ou supérieurs à 1000000 devraient être édités en héxadécimal, ceci pour des raisons de lisibilité. La commutation s'effectue à l'aide de la plage *'EDITION EN....'* qui se trouve en bas à gauche de la fenêtre.

4.1.3. La plage des registres

La commutation de l'édition des nombres influe aussi sur l'édition des registres dans le grand champ qui se trouve à gauche de la fenêtre. Lorsque le point *'AFFICHAGE REGISTRE'* est coché, le contenu des registres est affiché en hexadécimal ou en décimal.

Mais l'affichage ne se limite pas seulement aux registres, mais aussi à n'importe quelle adresse. Mais voyons tout d'abord comment est construite une telle ligne :

Taille:AE = Contenu

où taille indique si le contenu est un octet, un mot ou un mot long et correspond donc aux lettres B, W ou L.

Pour AE, vous pouvez indiquer une adresse quelconque, hormis *CCR*, *SR* ou *USP*. Si vous ne désirez pas voir afficher le contenu d'une adresse effective, mais plutôt l'adresse elle-même, la ligne aura la physionomie suivante :

AE ^ Adresse

L'adresse est toujours indiquée en format *'mot long'*. Toutes les adresses effectives qui sont autorisées pour l'instruction *PEA* peuvent être utilisées. Il ne faut donc pas indiquer *Dn, An, (An)+, -(An), #k, CCR, SR* et *USP*.

Pour ce qui est du contenu et de l'adresse, le registre A7 ne doit apparaître nulle part. La raison provient du fait que *USP* et *SSP* ont un contenu et une adresse en provenance de PROFIMAT-ST et non du programme testé. Ceci provoque, une fois de plus, par l'intermédiaire d'un contenu erroné de *USP* (à une exponentiation) et de *SSP* (double erreur de bus), une erreur qui ne peut être détectée.

Lorsque vous avez lancé PROFIMAT-ST et que vous regardez de plus près le champ des registres, vous serez certainement étonné de voir figurer le registre A7. Mais il s'agit là du pointeur de pile interne de PROFIMAT-ST qui est indiqué pour la circonstance, si jamais vous en aviez besoin.

Pour modifier l'adresse effective d'une ligne, il suffit de double-cliquer sur celle-ci. Vous obtiendrez alors un formulaire dans lequel vous verrez apparaître l'AE et où vous pourrez définir le contenu ou l'adresse, et si ce sont des octets, des mots ou mots longs qui doivent être édités.

L'accès aux contenus et aux adresses s'effectue suivant le mode choisi et pour lequel le drapeau du superviseur a été placé. Les erreurs de bus et d'adressages seront indiquées. En plus vous pourrez utiliser des variables pour des adresses effectives, si vous avez assemblé un programme précédemment dans lequel elles ont été définies.

4.1.4. Sauvegarde du contenu de l'écran

Le point *'SAUVER ECRAN'* qui se trouve sous le champ des registres, permet de disposer d'une seconde page écran. Vous pouvez y faire appel en cliquant dessus. Le point sera alors coché, et n'importe quel mouvement pas à pas ou un saut provoquera le passage de PROFIMAT-ST sur le deuxième écran. Mais il n'y aura pas d'influence entre PROFIMAT-ST et le programme à tester pour ce qui concerne l'édition à l'écran.

Si vous désirez voir la seconde page à partir du debugger, il suffit d'appuyer sur la touche *ESC*. Un nouvel appui vous permet de revenir à la page du debugger. Si vous n'avez plus besoin de la seconde page (32 KOctets) il suffit de cliquer une nouvelle fois sur le point *'SAUVER ECRAN'*. L'espace mémoire ainsi utilisé est restitué.

4.1.5. Indication du registre d'état

En haut, au milieu de la fenêtre, vous verrez apparaître la mention 'Registre d'état'. Son contenu est affiché en binaire. Si un bit est nul, il apparaît en blanc (normal). S'il est égal à un, vous le verrez apparaître en noir. Si vous cliquez sur un bit avec la souris, son état s'en trouvera modifié. Mais le bit TRACE ne peut être modifié comme cela. En effet sa modification peut faire effectuer un pas unique ou même démarrer le programme.

4.1.6. Pas de programme

Si vous cliquez sur le point *'PAS A PAS'*, l'instruction pointée par le compteur ordinal est exécuté.

Si la partie à tester représente un long passage, cette méthode peut être fastidieuse. Vous pouvez alors utiliser une émulation du 68020 pas à pas. Ceci peut être obtenu en cliquant sur le point 'émul. 68020'.

Si celui-ci est marqué, PROFIMAT-ST effectue un branchement vers le DEBUGGER devant une des instructions suivantes :

Bcc, BSR
DBcc
JMP, JSR
TRAP
RTE, RTR et RTS

Comme il s'agit d'une émulation, il va de soi que la vitesse d'exécution s'en trouve ralentie. Mais l'avantage de cette méthode, fait que le programme n'est interrompu qu'au moment d'arriver à un branchement. Ceci permet d'éviter un trop grand nombre de manipulations de la souris.

Il faut pourtant montrer la plus grande prudence en effectuant du pas à pas dans le système d'exploitation. En effet, le contenu des registres qui ne sont pas modifiés n'est pas sauvegardé sur la pile, mais dans un espace appelé *'REGISTER SAVE AERA'*. Le pointeur du début de cette aire n'est actualisé que lorsque tous les registres sont sauvegardés. Mais comme PROFIMAT utilise aussi le système d'exploitation, il utilise, lui aussi cet espace mémoire ! Il risque donc d'écraser des valeurs placées là-bas après votre mode pas à pas, avant d'arriver à la commande qui réactualise le pointeur. Si vous désirez évoluer dans les profondeurs du système d'exploitation, il vaut mieux travailler avec des breakpoints.

4.1.7. Faire démarrer un programme.

Le point 'LANCER PROGRAMME' permet de lancer le programme à partir de l'adresse indiquée dans le PC. Mais il faut, bien entendu, avoir la certitude que le programme s'achève soit avec une instruction TERM (GEMDOS 0,$31 ou $4C), soit qu'un breakpoint ait été placé. Le non-respect de ceci conduirait à ne plus pouvoir entrer dans le debugger.

Une autre possibilité pour revenir dans cet utilitaire consiste à cliquer sur le point *'INTERRUPTION'*. S'il est coché, il est possible d'interrompre le programme démarré à l'aide de la fonction précédente en appuyant simultanément sur les deux touches SHIFT et de revenir dans le DEBUGGER.

Comme la vitesse du programme ne souffre pas trop de ce réglage, l'interrogation peut être conservée, car cette possibilité ne peut être, bien entendu, obtenue qu'en faisant fonctionner le mode programme pas à pas. Pour des raisons de simplicité de l'interrogation, deux conditions doivent être remplies :

- le programme ne doit pas inhiber l'INTERRUPT LEVEL 6, interruption qui s'occupe de l'interrogation du clavier.

- le statut du clavier doit être déposé à l'adresse $E1B de votre version TOS. C'est le cas pour la version ROM. Si vous êtes en possession d'une autre version vous pouvez définir le caractère d'interruption en utilisant, par exemple, une boucle sans fin programmée comme celle qui suit :

```
\L:BRA.S \L
```

Bien entendu vous pouvez aussi essayer d'atteindre votre but à l'aide de la routine KBSHIFT en mode pas à pas ou réassembler votre version TOS.

4.1.8. BREAKPOINTS

La façon la plus sûre de sortir d'un programme en cours de fonctionnement est d'utiliser les points d'arrêt. On peut aussi les utiliser pour des branchements à des points spécifiques vers un debugger. Le breakpoint est, en fait, une instruction illégale. Pour placer un point d'arrêt, placez le curseur de la souris sur "breakpoint". Cliquez sur le bouton gauche de la souris sans le relâcher. Déplacez alors le curseur sur la ligne où vous désirez voir apparaître un breakpoint et relâchez le bouton de la souris.

Dans la ligne d'édition vous verrez alors apparaître la mention breakpoint.

Vous pouvez aussi déplacer un breakpoint déjà placé. Vous faites ceci de la même manière que pour le déplacement du PC. Si vous faites figurer un point d'interruption et un PC, le déplacement n'affectera que le breakpoint. Pour supprimer un breakpoint, autrement dit pour rétablir la ligne de commande dans son intégralité, il suffit d'inverser le processus de placement.

Vous cliquez sur le point d'interruption que vous désirez supprimer, sans relâcher le bouton gauche de la souris. Vous amenez le curseur de la souris sur "breakpoint" et relachez la touche. Le point d'arrêt disparaît.

Vous pouvez placer au maximum 16 points d'arrêt. Si vous avez oublié les endroits où se situent les différents points d'interruption, il suffit de double-cliquer sur la plage BREAKPOINT pour que la plage d'édition saute sur la ligne du premier point d'arrêt. Chaque double-clic suivant permet de passer au prochain point d'arrêt. Après le dernier placé, vous sauterez à nouveau au premier.

Lorsque vous effacez le programme en mémoire à l'aide du point *EFFACER* du menu FICHIER, les breakpoints placés sont, eux aussi effacés. Ceci ne compte pas pour la fonction du DEBUGGER '=>EFFACER', car la longueur du programme est inconnue pour celle-ci. L'utilitaire ne sait donc pas jusqu'où il doit chercher des points d'arrêt.

4.1.9. Charger un programme

Pour utiliser des programmes GEM ou TOS, il est conseillé d'employer le point *'Exécuter prog.'* et non *'CHARGER'* du menu 'FICHIER'.

Ce chargement sera exécuté par la fonction EXEC (GEMDOS $4B) qui reloge le programme et met la mémoire nécessaire à disposition, comme cela se passe lorsque vous lancez le programme à partir du bureau *(desktop)*.

Dans le formulaire que vous obtenez en cliquant sur cette plage, vous pourrez définir la ligne de commande et l'environnement désiré. Vous pourrez aussi y définir si le programme doit être seulement chargé ou s'il doit être lancé après cette opération. La ligne de commande correspond à la ligne que vous avez tapé pour le programme *.TTP avant le lancement. Après la confirmation de vos

données, vous verrez apparaître la fenêtre permettant la sélection de fichiers. Si le point *'SAUVER ECRAN'* est coché, c'est la deuxième page graphique qui se trouve sélectionnée.

Si vous faites démarrer le programme automatiquement, vous ne vous retrouverez dans le debugger qu'à la fin de celui-ci, ou si une erreur a été détectée. Si le programme s'achève magistralement avec l'instruction TERM, il est automatiquement effacé. Dans le cas contraire, ceci peut être fait *'à la main'*, en appuyant sur la touche CTRL et en cliquant sur le point '=>EFFACER'.

4.1.10. Reloger

Si vous avez chargé un programme relogeable avec la fonction *'CHARGER'* du menu 'FICHIER' ou si vous avez assemblé un tel programme, il faut tout d'abord le reloger avant de passer dans le debugger.

Si vous cliquez sur la plage *'RELOGER'*, le programme qui se trouve en mémoire est replacé à partir de l'endroit où il a été sauvegardé.

Il ne faut pas oublier, qu'un programme relogé de cette manière, ne peut plus être sauvegardé. En effet celui-ci n'est fonctionnel qu'à une adresse particulière.

La meilleure façon de procéder consiste à mettre en mémoire le programme à l'aide du point *'Exécuter prog.'* même s'il vient seulement d'être assemblé.

Dans cette dernière éventualité il faudra alors tout d'abord le sauvegarder.

4.1.11. Chercher

Si vous cliquez sur la plage *'CHERCHER'*, vous obtiendrez un formulaire, dans lequel vous pourrez indiquer l'octet, le mot ou mot long à rechercher. La syntaxe est identique à celle de l'instruction DC.

```
EXPRESSION [,EXPRESSION]
```

Le mot *expression* correspond, là aussi, à une expression arithmétique. Si vous recherchez une succession d'octets, vous pouvez indiquer une suite d'octets aussi longue que nécessaire. Celle-ci sera comprise entre apostrophes ou guillemets. Voici quelques exemples :

```
.B BYTE:'MICRO APPLICATION'

.W MOT:"A", "aA", 65535, %101001

.L MOT LONG: 12, 'abcd', LABEL -2, $6152, %100100101001010
```

Comme vous le voyez, vous pouvez aussi utiliser des variables, pour le peu qu'elles figurent encore en mémoire après le dernier assemblage. La recherche porte sur l'adresse qui correspond à la ligne supérieure de la plage d'édition. Si vous avez mis en service le mode HEX/ASCII, le curseur se retrouvera au début de la séquence recherchée, pour le peu que la recherche soit couronnée de succès.

Vous obtiendrez le message *'NON TROUVE'*, lorsque le debugger passe dans un espace-mémoire protégé et que la séquence recherchée n'a pas été retrouvée. Vous devrez alors appuyer sur la touche ENTER pour confirmer.

4.1.12. Modifier un registre

Si vous cliquez sur ce champ, vous verrez apparaître une ligne de saisie, dans laquelle vous pourrez modifier le contenu de registre suivant la syntaxe indiquée ci-dessous :

```
REGISTRE=VALEUR
```

Cette fois-ci, il ne vous est pas autorisé de placer de caractère espace. *'VALEUR'* est une expression arithmétique et les registres possibles sont ceux qui suivent :

D0...D7	:	Registre de données
A0...A6	:	Registre d'adresse
USP, SSP	:	Pointeur de Pile
PC	:	Compteur ordinal
*	:	Adresse de la ligne supérieure de la plage d'édition.

Le caractère '*' devrait vous être encore connu du point *'A PARTIR DE ADRESSE'*. En réalité il s'agit d'un cas particulier de la fonction *'MODIFIER REGISTRE'*.

Si vous n'introduisez pas de données dans la plage de saisie, rien ne sera modifié. La validation de la saisie par *ENTER ou RETURN* entraîne la réédition de la plage du registre.

4.2. LA LIGNE D'INFORMATION

La ligne d'information vous donne constamment des valeurs sous forme hexadécimale :

PC	:	Contenu du compteur ordinal
SSP	:	Contenu de la pile superviseur
USP	:	Contenu de la pile utilisateur
DEBUT PROGRAMME	:	Adresse de début du dernier programme assemblé.

4.3. UTILISATION DU CLAVIER

Certaines fonctions du DEBUGGER peuvent aussi être atteintes par la frappe de touches du clavier :

^E	:	pas unique de programmation
^S	:	lancement du programme
^Q	:	à partir de l'adresse
^R	:	Modification d'un registre

Et naturellement la touche *ESC* qui permet de passer à la deuxième page écran.

4.4. LES EXCEPTIONS

Certaines exceptions sont repérées par le DEBUGGER :

- des erreurs de BUS,
- des erreurs d'adressages,
- des instructions illégales,
- la division par zéro,
- la commande CHK,
- la commande TRAPV
- les modifications PRIVILEG

En plus, le vecteur TRACE est utilisé. Vous pouvez dévier tous les vecteurs à l'exception des 4 vecteurs ci-dessous, même ceux qui ne sont pas nommés, pourvu qu'ils ne soient pas utilisés par le système d'exploitation, comme par exemple l'exception LINE-F de vos propres routines. Et voici les exceptions à la règle :

- Erreur de bus,
- Erreur d'adressage
- Instruction illégale (pour BREAKPOINT)
- TRACE (pour pas à pas).

Vous devrez prendre garde au fait que les vecteurs soient replacés en position initiale, en fin de fonctionnement du programme.

4.5. MISE EN CONCORDANCE DU SYSTEME D'EXPLOITATION

Cette mise en concordance ne touche que le point *'INTERRUPTION'*, car il s'agit là de la seule fois où l'on accède réellement au système d'exploitation. Pour adapter PROFIMAT-ST à votre version du système d'exploitation, il suffit de modifier deux endroits de la mémoire. Mais si *'INTERRUPTION'* fonctionne sur votre ordinateur, vous pouvez passer au chapitre suivant.

Il faut, en premier lieu, définir l'endroit où votre système d'exploitation place le status du clavier. Débutez vos recherches, à l'aide du debugger dans la partie TRAP 13 dans laquelle il est possible d'appeler la fonction *KBSHIFT (BIOS 11)*.

Copiez le programme PROFIMAT-ST sur une autre disquette en guise de sécurité, si jamais quelque chose devait mal se passer.

Pour effectuer la modification, vous chargez PROFIMAT-ST en cliquant sur le point *CHARGER* du menu FICHIER. Vous choisirez dans le formulaire correspondant le point *PRG* et réaliserez le chargement effectif en choisissant PROFIMAT dans la fenêtre de sélection des fichiers.

Activez à présent le debugger et laissez défiler la mémoire à partir du début du PROGRAMME (voir ligne d'information). Cliquez sur *'CHERCHER'* et effectuez cette opération pour un mot contenant $E1B. Comme ce mot n'existe que deux fois dans PROFIMAT-ST, vous devriez arriver à bon port sans difficulté. Si c'est le cas vous devriez voir afficher à l'écran le programme :

```
BTST #0,$E1B
BEQ.S SUITE
BTST #1,$E1B
BNE.S ARRET
SUITE: RTE
ARRET:
```

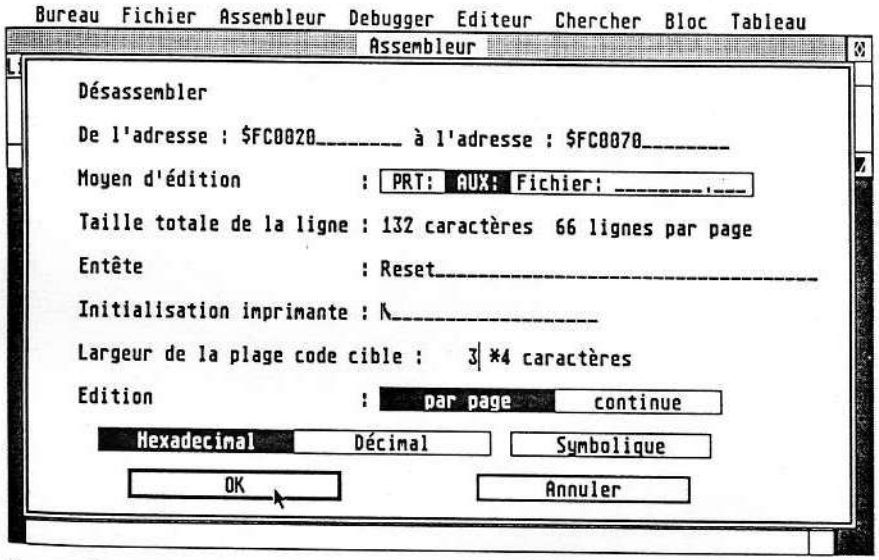
Il vous reste à indiquer en lieu et place de $E1B l'adresse mémoire où votre système d'exploitation place le STATUS du clavier. Il va de soi que cette indication doit être faite sous format HEX/ASCII.

Maintenant que vous avez effectué ce que l'on appelle un PATCH, il reste à sauvegarder PROFIMAT-ST en cliquant le point *'SAUVEGARDER'* du menu FICHIER et dans le formulaire obtenu utiliser l'appendice PRG.

Vous êtes, à présent en mesure de vérifier si la modification effectuée atteint le but escompté. Dans le cas où cela ne fonctionne pas, ayez une pensée pour SENEQUE qui dit que l'ERREUR EST HUMAINE et reprenez cette modification du début.

5. Désassembleur et Réassembleur

Les points DESASSEMBLEUR et REASSEMBLEUR peuvent être appelés à partir du menu du DEBUGGER. Vous n'obtiendrez ici qu'un formulaire et non une nouvelle fenêtre. Dans celui-ci, il sera possible de fixer les différents paramètres. Il est similaire à celui que vous avez déjà vu pour l'édition. En fait les deux fonctions ont un but similaire : éditer le programme.



Le code est désassemblé suivant les mnémoniques standard de MOTOROLA. Les instructions non implémentées sont transcrites par DC.W .

5.1. DESASSEMBLEUR

Le formulaire pour le désassemblage est repris ci-avant. Il permet seulement de transcrire les codes compris entre l'adresse de début (DE L'ADRESSE) et l'adresse de fin (A L'ADRESSE), sans se soucier s'il s'agit d'un programme ou de données.

Comme toujours, vous pouvez indiquer des expressions arithmétiques pour les adresses de début et de fin. Si l'adresse de fin est située avant l'adresse de début, il ne se passera rien. Il faudra prendre garde au fait qu'aucun espace mémoire protégé ne soit compris entre l'adresse de début et l'adresse de fin. Le non-respect de ceci conduit le désassembleur à arrêter son travail dès que cet espace est atteint.

La ligne suivante, sur le formulaire, permet de définir le moyen d'édition pour ce désassemblage. Le MOYEN D'EDITION comprend l'imprimante (PRT), l'interface RS 232 (AUX) et un fichier, qui se trouvera alors dans le sommaire du moment.

Vous pourrez ensuite définir la largeur totale d'une ligne, et, si vous avez choisi d'éditer page par page, le nombre de lignes sur l'une d'elle.

L'en-tête figurera, suivant le mode d'édition, soit une fois, en tête de fichier (Edition continue), soit à chaque début de page (Edition page par page). L'initialisation de l'imprimante n'est réalisée qu'une seule fois, en début d'édition.

Comme lors du désassemblage le code est, lui aussi, indiqué, il vous est possible de régler la largeur de la plage d'édition de

celui-ci (LARGEUR DE LA PLAGE-CODE). Avant le code, vous verrez figurer, en hexadécimal, l'adresse où figure celui-ci.

C'est maintenant qu'apparaît la ligne qui permet de définir le mode d'édition, page par page ou en continu, si celle-ci a lieu en chiffres décimaux, ou hexadécimaux ou si la notation symbolique doit être utilisée (autrement dit si les variables doivent être elles aussi éditées).

5.2. REASSEMBLEUR

Le formulaire de réassemblage est identique au précédent. La tâche du réassembleur est de restaurer un texte-source. Les réglages suivants sont donc inutiles :

- LIGNES PAR PAGE
- EN-TETE
- INITIALISATION IMPRIMANTE
- LARGEUR DE LA PLAGE-CODE
- EDITION : PAR PAGE/EN CONTINU
- SYMBOLIQUEMENT

Le réassemblage s'effectue de manière continue, sans en-tête et initialisation d'imprimante, car ceci ne ferait que compliquer la tâche du réassembleur. Comme les codes ne sont pas édités, le réglage de la plage d'édition devient tout à fait inutile. Le restant des paramètres est absolument identique à ceux décrits pour le désassemblage. Nous n'y reviendrons pas.

Le réassemblage s'effectue toujours de manière symbolique. Si une instruction tend vers une adresse, un label est immédiatement défini, qui pointera vers cette adresse. Comme le réassembleur ne brille pas par son vocabulaire, il attribuera un nom ayant la forme Lxxxxx où xxxxx est un nombre hexadécimal attribué dans l'ordre croissant.

Si une instruction utilise une constante, la nomination de celle-ci s'effectuera de la même manière, en utilisant le préfixe K (Kxxxxx). Une instruction DC.W ne génère pas de constantes.

Le texte-source ainsi généré se terminera par une instruction END. Le réassembleur essayera de diviser le programme en instructions et en données.

C'est la raison pour laquelle la transcription s'effectue en deux passes. La première permet au réassembleur de définir les adresses de début des sous-routines qui seront adressées ainsi que les adresses finales de celles-ci. Il définit aussi celles qui utilisent un return et celles qui utilisent un saut inconditionnel. Les branchements sont eux aussi détectés.

Fort de ces renseignements, la séparation entre programme et données est entreprise. Mais comme avec cette méthode, les sauts indirects restent non-détectés, le réassembleur va vous demander d'autres adresses de départ, après avoir validé le formulaire. C'est maintenant que vous pouvez indiquer des adresses de sous-programmes qui seront appelées par adressage indirect.

Il faut, tout d'abord être en possession de ces adresses. C'est pourquoi il est plus que recommandé de désassembler tout d'abord le programme et de l'analyser. C'est après ceci que vous réassemblerez

le programme d'une façon claire. Les adresses de départ doivent être validées par OK. Si vous ne voulez pas entrer d'autres adresses de début, il suffit de cliquer le point FIN.

ATTENTION ATTENTION ATTENTION

La dernière adresse doit être validée par OK et non par FIN.

Il nous semble encore important de vous indiquer que le réassembleur ne définit de labels que pour ceux qui figurent en adresse paire. Si le programme tombe sur un label situé à une adresse impaire, il le repèrera comme AE 'Lxxxxx+1'.

Pas de LABEL défini

START défini présentement

LABEL non autorisé

Nom illégal

APPENDICE II

Instructions de l'assembleur

Dans cet appendice nous avons classé les instructions de l'assembleur en notant la syntaxe devant être utilisée :

Constante=Expression

ALIGN.y

BSS

DATA

*DC.x Expression{,Expression}**

*DEFB Expression{,Expression}**

*DEFL Expression{,Expression}**

*DEFM Expression{,Expression}**

DEFS Nombre[,Valeur de remplissage]

*DEFW Expression{,Expression}**

DS.x Nombre[,Valeur de remplissage]

ELSE => ENDIF

END

ENDIF

ENDM

ENDS

Symbole EQU Chaîne de caractères

IBYTES Nom de fichier[,Nombre]

IFcc Expression,Expression => [ELSE =>] ENDIF

ILABEL Nom de fichier

INCLUDE Nom de fichier

INPUT [Message,]Variable

LIST

Nom:MACRO [Paramètre(,Paramètre)] => ENDM*

MERGE Nom de fichier

NOLIST

ORG Expression

PAGE

REPEAT => UNTILcc

SLABEL Nom de fichier => ENDS

START Expression

TEXT

UNTILcc Expression,Expression

avec:

y::=(.W|.L)
x::=(.B|.W|.L)
Expression, Nombre, F Valeur de remplissage::=Expression arithmétique.
Constante, Symbol, Variable, Nom, Paramètre::=Nom de variable
Chaîne de caractères::=Suite de caractères suivant désir
Nom de fichier::=Nom de fichier désiré;option encadrée par des ' ou "
cc::=Utilisation comme l'instruction Bcc, mais sans T, F ou RA.
Message::=Chaîne de caractères entre apostrophes ou guillemets

APPENDICE VI

Instructions courtes de l'éditeur

Vous trouverez à la suite de ceci une vue d'ensemble des instructions courtes de l'éditeur et de la syntaxe correspondante :

Commandes du curseur I :

(^A)(|A|B|F|S)	:	haut
(^B)(|B|E|F|S)	:	bas.
(^C)(|B|E)	:	droite.
(^D)(|A|B)	:	gauche.

avec A=DEBUT, B=BLOC, F=ERREUR, E=FIN et S=RECHERCHE

Commandes de bloc :

BA	:	Marquage début.
BD	:	Démarquage.
BE	:	Marquage Fin.
BK	:	Copie.
BL	:	Effacer.
BV	:	Déplacer.

Commandes de curseur II. :

C(x,y) : Place le curseur en position (x,y).
CN : Curseur au début de la ligne suivante.
CV : Curseur au début de la ligne précédente.

Commandes de fichier :

DL Nom de fichier : Chargement.
DS Nom de fichier : Sauvegarde.

Remplacement :

E{(A|V|U|?}*)(^A|^B) String1 String2

avec A=Toutes, V=Variables, U=Différence entre Majuscules et Minuscules et ?=interrogation avant le remplacement.

Effacer :

L(^A) : Effacer jusqu'au début du texte
L(^B) : Effacer jusqu'à la fin du texte
L(^C) : Effacer jusqu'à la fin de la ligne
L(^D) : Effacer jusqu'en début de ligne
LL : Efface à gauche (BACKSPACE)
LR : Efface à droite (DELETE)
LZ : Effacer la ligne

Rechercher :

S{(V|U)}*(^A|^B) String

avec V=Variables et U=Différence entre Majuscule et Minuscule.

Commandes de tabulations :

TS : Placer.
TL : Effacer.

Commandes diverses :

Z String : Insérer suite de caractères
ZN String : Insérer ligne à la suite de celle du moment
ZV String : Insérer ligne avant celle du moment.