ATARI

CONVIVIAL,
PUISSANT...
UN OUTIL UNIQUE!

GFA ASSEMBLEUR

GFA

EDITIONS MICRO APPLICATION



CONVIVIAL,
PUISSANT...
UN OUTIL UNIQUE!

GFA ASSEMBLEUR

EDITIONS MICRO APPLICATION

Distribué par :

MICRO APPLICATION
58, rue du Faubourg Poissonnière
75010 PARIS

© Reproduction interdite sans l'autorisation de
MICRO APPLICATION

'Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de **MICRO APPLICATION** est illicite (Loi du 11 Mars 1957, article 40, 1er alinéa).

Cette représentation ou reproduction illicite, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

La Loi du 11 Mars 1957 n'autorise, aux termes des alinéas 2 et 3 de l'article 41, que les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à l'utilisation collective d'une part, et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration'.

© 1988 **GFA Systemtechnick GmbH**
Heerdter Sandberg 30
D-4000 Düsseldorf 11

Traduction française assurée par Pascal Hausman

© 1988 **Micro Application**
58, rue du Faubourg Poissonnière
75010 PARIS

Collection dirigée par Mr Philippe OLIVIER
Edition réalisée par Frédérique BEAUDONNET

MS-DOS est une marque déposée de Microsoft Corporation
Atari ST et TOS sont des marques déposées de Atari Corp.
GEM et GEM Desktop sont des marques déposées de Digital Research Inc.

SUPPORT PRODUIT

Seules les personnes retournant la carte client dûment remplie, en incluant bien nom, adresse, nom du produit et numéro de série, seront enregistrées comme client Micro Application et pourront bénéficier du support produit.

Nous rappelons que le support produit est effectué par l'équipe technique de Micro Application.

Les horaires sont :

Du Lundi au Jeudi :	14 h 30 à 17 h 30
Le Vendredi :	14 h 30 à 16 h 30

Toute personne n'ayant pas retourné chaque carte spécifique à chaque produit se verra refuser tout support.

GFA

Assembleur

Sommaire

Partie A : Introduction 7

Partie B : L'éditeur 11

1. Sélection de fichier 13

2. L'écran d'édition 17
2.1. Contenu de l'écran d'édition 18
2.2. Définition des touches dans l'écran d'édition 19
2.3. Les fonctions de la souris sur l'écran d'édition 26
2.4. Fonctions macro 27

3. Les menus 33
3.1. Le menu 'Fichier' 34
3.1.1 Etat
(F1 : Fichier puis F1 : Ecran d'état) 35
3.1.2. Sauver un texte
(F1 : Fichier F2 : Sauver) 41
3.1.3. Sauvegarder le texte sous un autre nom
(F1 : Fichier F3 : Sauver sous...) 41
3.1.4. Recharger texte
(F1 : Fichier F4 : Restaurer) 41
3.1.5. Charger nouveau texte
(F1 : Fichier F5 : Charger) 42
3.1.6. Charger bloc
(F1 : Fichier F6 : Charger Bloc...) 42
3.1.7. Sauvegarder bloc
(F1 : Fichier F7 : Sauver Bloc) 42
3.1.8. Supprimer fichier
(F1 : Fichier F8 : Effacer Fichier) 42
3.1.9. Formater une disquette
(F1 : Fichier F9 : Formater disque) 42
3.1.10. Quitter le GFA Assembleur
(F1 : Fichier F0 : Quitter) 43
3.2. Le menu Editeur 43
3.2.1. Paramètres d'édition
(F2 : Editeur F1 : Paramètres) 43
3.2.2. Mode assembleur
(F2 : Editeur F2 : Sc.en ASM...) 45
3.2.3. Mode de texte source général
(F2 : Editeur F3 : Source en ASCII...) 51

3.2.4. Caractères spéciaux
(F2 : Editeur F4 : Caractères spéciaux) 51
3.2.5. Tabulations
(F2 : Editeur F5 : Marquer tabs) 52
3.2.6. Déployer tabulations
(F2 : Editeur F6 : Expanser tabs...) 52
3.2.7. Sauver instructions
(F2 : Editeur F7 : Sauver commandes) 53
3.2.8. Charger commandes
(F2 : Editeur F8 : Charger commandes) 54
3.2.9. Comparaison de textes
(F2 : Editeur F9 : Comparer Fichier) 54
3.3. Le menu 'Bloc' 54
3.3.1. Fixer le début du bloc
(F3 : Bloc F1 : Marquer début) 55
3.3.2. Fixer la fin du bloc
(F3 : Bloc F2 : Marquer Fin) 55
3.3.3. Copier bloc
(F3 : Bloc F3 : Copier Bloc) 55
3.3.4. Déplacer bloc
(F3 : Bloc F4 : Déplacer) 56
3.3.5. Supprimer bloc
(F3 : Bloc F5 : Effacer Bloc) 56
3.3.6. Saut au début du bloc
(F3 : Bloc F6 : Début) 56
3.3.7. Saut à la fin du bloc
(F3 : Bloc F7 : Fin) 56
3.3.8. Annuler marquage
(F3 : Bloc F8 : Enlever Marques) 56
3.3.9. Déclarer un bloc comme global
(F3 : Bloc F9 : Bloc → Global) 56
3.3.10. Insérer bloc global
(F3 : Bloc F0 : Copier Global) 56
3.4. Le menu 'Recherche' 57
3.4.1. Placer marque
(F4 : Cherch. F1 : Placer Marques) 57
3.4.2. Aller à une marque
(F4 : Cherch. F2 : Aller à une marque) 58
3.4.3. Chercher chaîne de caractères
(F4 : Cherch. F3 : Chercher une chaîne) 58
3.4.4. Continuer la recherche
(F4 : Cherch. F4 : continuer Recherche) 60
3.4.5. Chercher et remplacer
(F4 : Cherch. F5 : Chercher & Remplacer) 61
3.4.6. Sauter au début du texte
(F4 : Cherch. F6 : Début du texte) 61
3.4.7. Sauter à la fin du texte
(F4 : Cherch. F7 : Fin du texte) 62

3.4.8. Sauter à la dernière position
(F4 : Cherch. F8 : Dernière Position) 62
3.5. Le menu 'Imprimante' 62
3.5.1. Paramètres d'impression
(F5 : Imprim. F1 : Paramètres...) 62
3.5.2. Imprimer le texte complet
(F5 : Imprim. F2 : Imprimer tout) 65
3.5.3. Imprimer bloc
(F5 : Imprim. F3 : Imprimer bloc) 65
3.5.4. Imprimer avec numéros de ligne
(F5 : Imprim. F4 : Avec numéros) 65
3.5.5. Imprimer sans numéros de ligne
(F5 : Imprim. F5 : Sans numéros) 65
3.5.6. Imprimer toutes les pages
(F5 : Imprim. F6 : Toutes pages) 65
3.5.7. Imprimer uniquement les pages impaires
(F5 : Imprim. F7 : Pages impaires). 66
3.5.8. Imprimer uniquement les pages paires
(F5 : Imprim. F8 : Pages paires) 66

Partie C : Emploi de l'assembleur 67

1. Mode d'emploi avec le menu 69
1.1. Paramètres (F6 : Assembleur F1 : Paramètres...) 69
1.2. Assembleur (F6 : Assembleur F2 : Assembleur) 73
1.3. Références croisées 83

2. Directives du texte source 85
2.1. Pseudo-codes opération 85
2.1.1. Instructions de structuration du programme 85
2.1.2. Macros et Include 92
2.1.2.1. Définition de macro 93
2.1.2.2. Appel de macro 95
2.1.2.3. Fonctions spéciales dans le corps de la macro 96
2.1.2.4. Insertion de texte source supplémentaire 106
2.1.3. Assemblage conditionnel 106
2.1.4. Assemblage répété 110
2.1.5. Instructions d'initialisation de la mémoire 112
2.1.5.1. Dans TEXT et le segment de données initialisé (.DATA) 112
2.1.5.2. Dans le segment de données non initialisé (.BSS) et les structures .ABS 114
2.1.6. Instructions de gestion de listing
(Protocoles d'assemblage) 114
2.2. Calculs, symboles, Operateurs 121
2.2.1. Définition de symboles 121
2.2.2. Appel de symboles 124
2.2.3. Les calculs 124

2.3. Variables alphanumériques 126
2.4. Fonctions de chaînes de caractères 127

3. Le linker 129

4. Gestion de bibliothèques 137

Partie D : Charger les programmes - Appeler les débogueurs 145

1. Exécuter programme 147

2. Appel du débogueur résident 153

Partie E : Annexes 155

A : Tables ou formats d'entrée de l'éditeur 157
A1 : Emploi des champs d'entrée 157
A2 : Le format des tables d'instructions 158
A3 : Réglage standard de l'imprimante 159
A4 : Format des tables de configuration de l'imprimante .. 159
A5 : Particularités survenant lorsque la mémoire est pleine 161

B : Formats de fichier de l'assembleur et du linker 163
B1 : Format des programmes exécutables 163
B2 : Format des modules objet (Format Digital Research Inc) 164
B3 : Format de fichier ABS 167
B4 : Format de fichier IMG 167
B5 : Structure des bibliothèques 168
B6 : Fichiers d'index (pour les bibliothèques) 169
B7 : Format tokenisé du texte source 170

C : Messages d'erreur de l'assembleur et du linker 171

D : Manipulations des textes en mémoire 175

Index 177

Partie A
Introduction

Ce nouveau produit GFA, le GFA Assembleur est inhabituel à bien des égards. La tendance étant de plus en plus, ces derniers temps, à écrire des programmes en partie ou en totalité en assembleur, nombreux seront certainement les utilisateurs à se réjouir de pouvoir maintenant faire aussi l'apprentissage d'un langage de programmation aussi proche de la machine (et aussi puissant).

Les programmes utilisateur écrits pour tout ou partie en assembleur se distinguent par une vitesse d'exécution plus élevée et une taille des programmes beaucoup plus réduite puisque la mémoire peut être employée de façon plus efficace.

Le GFA Assembleur a été développé par des programmeurs en assembleur pour des programmeurs en assembleur. C'est pourquoi il dispose de nombreuses fonctions et possibilités grâce auxquelles il se prête à merveille à un développement efficace des programmes.

Qui plus est, de par sa conception même, il peut être recommandé sans réserve aussi bien aux débutants en assembleur qu'aux experts en programmation. Nous voudrions indiquer ici rapidement quelques-unes de ses caractéristiques essentielles :

L'assembleur et l'éditeur figurent dans un seul et même programme, ce qui présente plusieurs avantages :

➥ **Détection des erreurs dès l'entrée** : lorsque vous venez d'entrer une ligne, l'éditeur contrôle immédiatement si elle contient une erreur, auquel cas il place automatiquement un message d'erreur à la suite de l'instruction incriminée.

➥ **Pas de temps perdu** : dès que vous avez fait une erreur et que vous tentez d'assembler, vous êtes aussitôt conduit à l'emplacement du listing où figure l'erreur. Si l'assembleur et l'éditeur étaient séparés, vous seriez cha-

que fois obligé de recharger le texte source, de le corriger, de le re-sauvegarder, après quoi seulement vous pourriez à nouveau tenter d'assembler le programme.

➥ **"Tokenisation" automatique des entrées** : les instructions assembleur et les modes d'adressage ne sont pas stockés en clair mais sont convertis en codes beaucoup moins longs. Les espaces superflus ne sont pas sauvegardés avec le reste du programme, ce qui permet un important gain de place mémoire sur l'ordinateur ainsi que sur la disquette et réduit, du même coup, les temps de chargement et de sauvegarde de vos programmes (un texte de programme codé a une longueur de seulement 50 à 60 % environ de celle du texte équivalent en clair).

➥ **Formatage automatique des entrées** : toutes les entrées sont immédiatement formatées en fonction des tabulateurs, que vous pouvez fixer librement. Cela vous permet de gagner le temps de l'entrée et améliore, en même temps, la clarté du texte.

➥ **Assemblage très rapide** : malgré sa puissance considérable et la diversité de ses fonctions, l'assembleur est extrêmement rapide.

L'éditeur du GFA Assembleur peut aussi être employé pour le développement de programmes en langages évolués. Il offre alors toutes les possibilités d'un éditeur de texte source convivial et rapide. Sa faculté de charger d'autres programmes à la suite et de transmettre ce texte source directement (c'est-à-dire sans passer par la disquette) en fait un outil également très pratique pour travailler avec des compilateurs. Il faut dans ce cas, être en mode "Source en ASCII".

Le débogueur puissant, qui permet de tester les programmes, est un programme indépendant. On évite ainsi que l'éditeur/assembleur ne soit trop lent, cela permet aussi que le débogueur soit utilisé seul (il peut évidemment être malgré tout chargé à la suite de l'assembleur). Le débogueur est symbolique et orienté écran, c'est-à-dire que le curseur peut être déplacé librement sur l'écran.

☞ Convention :

Dans la suite de ce manuel, la description de certaines instructions s'accompagne des caractères { et }. Ceci ne veut pas dire que ces caractères sont nécessaires à la syntaxe de l'instruction, mais seulement que les paramètres de cette commande englobés par ces caractères sont optionnels.

Partie B
L'éditeur

L'éditeur du GFA Assembleur possède une interface utilisateur indépendante très semblable à celle des programmes GEM mais plus rapide et qui autorise souvent une commande aussi bien avec la souris qu'au clavier.

Le GFA Assembleur peut être lancé en haute résolution (écran monochrome), en résolution moyenne ou en faible résolution (écran couleur). Si, toutefois, il est lancé en faible résolution, il commute automatiquement la résolution moyenne. Il en résulte que les messages d'erreur sortis directement par le système d'exploitation apparaissent incorrectement formatés. C'est pourquoi il est préférable de ne jamais lancer le GFA Assembleur en faible résolution.

Pour apprendre à vous servir de l'éditeur, veuillez suivre les instructions suivantes, qui seront écrites en retrait.

Pour pouvoir travailler de la façon la plus pratique avec le GFA Assembleur, veuillez déclarer les extensions '.IS', '.BEL' ainsi, éventuellement, que celles des textes source sur lesquels vous voulez travailler avec le GFA Assembleur, comme applications GFA Assembleur (cliquez sur GFA-ASM.PRG puis, dans le menu "Options" du GEM, cliquez sur Installer une application, entrez l'extension voulue, cliquez le bouton 'Confirmer'. Le mieux est alors de cliquer 'Sauvegarder le bureau'. Ces définitions seront alors sauvegardées et seront à nouveau à votre disposition lors de la prochaine mise sous tension de l'ordinateur ou à la suite d'un Reset). Si vous lancez le système à partir d'une disquette, vous devez placer dans le lecteur A, pour 'Sauvegarder le bureau', la disquette que vous utilisez pour "booter".

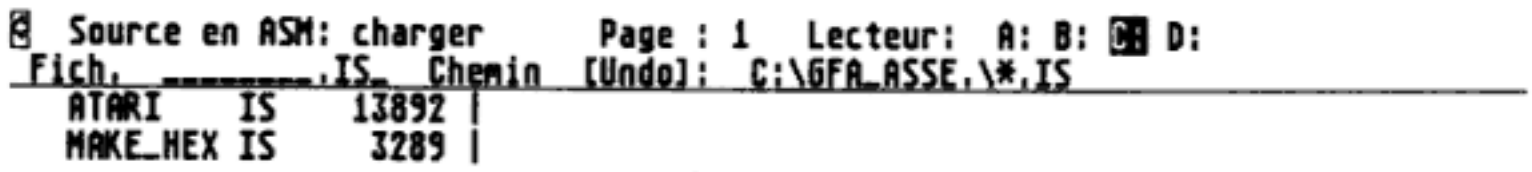
Si vous cliquez par la suite un des fichiers déclarés, le GFA Assembleur sera automatiquement chargé et chargera alors le fichier cliqué dans la mémoire de texte.

Veuillez réaliser une copie de sécurité de votre disquette originale et mettre votre disquette originale de côté. Chargez alors le programme 'GFA-ASS.PRG' à partir de la copie de sécurité.

Chapitre 1

Sélection de fichier

Peu après le lancement de GFA Assembleur apparaît un écran qui vous invite à sélectionner un fichier à charger :



En haut à gauche de la première ligne figure la 'flèche Escape'. Elle s'appelle ainsi parce qu'en la cliquant avec la souris vous obtenez le même résultat qu'en appuyant sur la touche Escape. Il ne se passe rien pour le moment car il faut toujours commencer par ouvrir un texte (ouvrir un texte signifie charger un texte ou en créer un nouveau). Si un ou plusieurs textes figurent déjà dans la mémoire, Escape permet de sortir de l'écran de sélection de fichier.

A côté de la flèche Escape est indiqué quel type d'action est engagé ainsi que le fichier en cause.. Pour le moment, par exemple, vous voyez 'Source en ASM : Charger'. Si vous sélectionnez maintenant un fichier qui n'est pas un source assembleur, ce texte se transformera, pendant le chargement, en 'Source en ASCII : Charger'.

Au centre de la première ligne est indiquée quelle page du répertoire de la disquette est actuellement affichée. Si le répertoire actuel comporte plus de 69 fichiers, des symboles de flèche, que vous pouvez cliquer avec le bouton gauche de la souris, apparaissent à côté du numéro de page. Vous pouvez aussi actionner les touches curseur 'Cursor up' et 'Cursor down'. Ceci vous permet de "Feuilleter" les répertoires trop longs pour tenir sur une seule page. En tenant la touche Shift enfoncée pendant que vous appuyez sur l'une des touches curseur, vous pouvez feuilleter le répertoire colonne par colonne.

Un peu plus à droite sont affichées les lettres code des lecteurs connectés. Le lecteur actuel est affiché en inversion vidéo.

Vous pouvez changer de lecteur en cliquant une autre lettre avec la souris ou bien en tenant la touche Control enfoncée et en entrant directement au clavier la lettre du lecteur voulu.

Tout à droite est affichée la quantité de mémoire encore libre (s'il en reste suffisamment) sur la disquette. Si le lecteur actuel est un disque dur ou un disque virtuel, la place mémoire libre n'est affichée que lorsque vous cliquez cette zone avec le bouton gauche de la souris.

A gauche, sur la deuxième ligne, est affiché le fichier sélectionné, et un peu plus à droite, le chemin menant au fichier.

Vous pouvez modifier le chemin de la façon suivante :

- Pour aller vers un répertoire immédiatement subordonné : cliquer le répertoire voulu.

- Pour aller vers le répertoire immédiatement au-dessus : actionner la touche Undo ou cliquer le mot '[Undo]' au milieu de la deuxième ligne.

- Pour modifier le masque d'extension : soit cliquer le masque d'extension (à l'extrémité droite du chemin), entrer la nouvelle extension et appuyer sur la touche Re-

turn, soit, si aucun nom de fichier n'a encore été entré, entrer simplement la nouvelle extension comme extension de fichier et appuyer sur la touche Return (ex : A:*.*). Vous pouvez utiliser dans le masque les caractères joker '*' et '?'. Si vous vous trouvez dans le répertoire racine, [Undo] sélectionnera le masque d'extension '.*'.

Le nom de fichier peut être entré de la manière suivante :

- En cliquant le fichier voulu dans le répertoire.
- En entrant au clavier le nom et éventuellement l'extension.

Les champs du nom et de l'extension sont séparés par un point. Pour passer de l'un à l'autre, vous pouvez utiliser les touches curseur 'Cursor left' et 'Cursor right' ou le point.

Pour charger des programmes, vous pouvez employer les symboles joker '*' et '?' dans le nom et dans l'extension. Le nom de fichier sera alors automatiquement complété.

Entrez maintenant le nom d'un fichier figurant sur la disquette ou bien entrez un nouveau nom de texte. Appuyez ensuite sur 'Return' (ou 'Enter'). Ou bien effectuez un double clic sur le nom de fichier voulu. L'ordinateur charge maintenant le texte dans la mémoire.

Si vous avez entré un nom n'existant pas dans ce répertoire, un nouveau texte sera créé sous le nom spécifié. GFA Assembleur tient compte pour cela du chemin actuel.

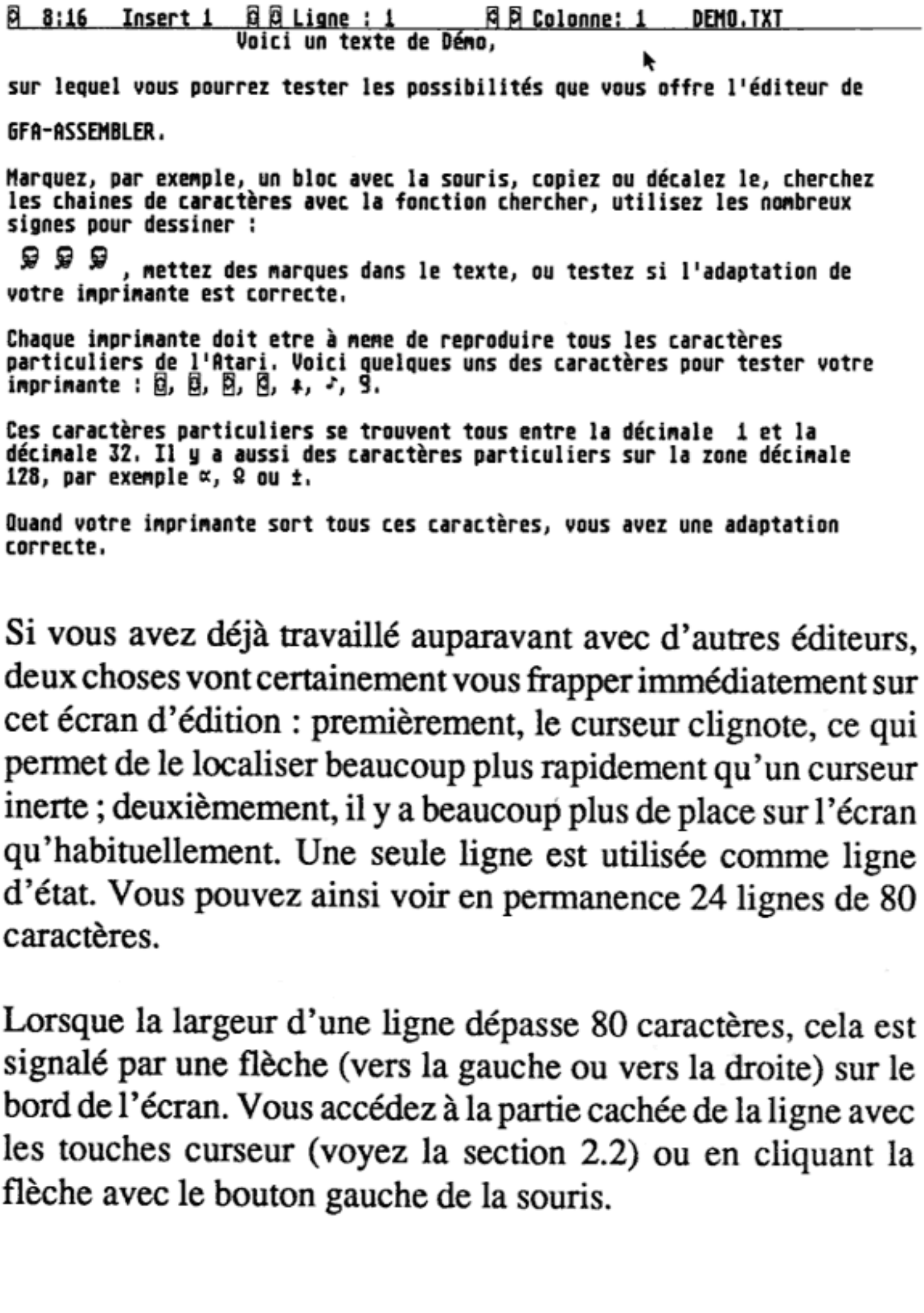
Par conséquent, lorsque vous sauvegarderez plus tard ce texte avec la fonction 'Sauver' (Menu 1,2) ou 'Sauver modifications' sur l'écran d'état (menu 1,1) (voyez la section 3.1.1), le texte sera sauvegardé dans le répertoire actuellement affiché.

Chargez à partir du répertoire le texte de démonstration 'DEMO.TXT' !

Chapitre 2

L'écran d'édition

Quelques instants après apparaît l'écran d'édition :



Si vous avez déjà travaillé auparavant avec d'autres éditeurs, deux choses vont certainement vous frapper immédiatement sur cet écran d'édition : premièrement, le curseur clignote, ce qui permet de le localiser beaucoup plus rapidement qu'un curseur inerte ; deuxièmement, il y a beaucoup plus de place sur l'écran qu'habituellement. Une seule ligne est utilisée comme ligne d'état. Vous pouvez ainsi voir en permanence 24 lignes de 80 caractères.

Lorsque la largeur d'une ligne dépasse 80 caractères, cela est signalé par une flèche (vers la gauche ou vers la droite) sur le bord de l'écran. Vous accédez à la partie cachée de la ligne avec les touches curseur (voyez la section 2.2) ou en cliquant la flèche avec le bouton gauche de la souris.

2.1. Contenu de l'écran d'édition

La ligne d'état sert à afficher des informations importantes, par exemple dans quelle ligne et quelle colonne vous vous trouvez actuellement, quel est le texte actuellement visible (le nombre de textes pouvant être gérés par l'éditeur n'est limité que par la mémoire disponible).

En haut à gauche, dans la ligne d'état, figure à nouveau la 'flèche Escape' mais elle est cette fois dirigée vers la droite. Cela signifie que vous accédez à la barre de menus en touchant la flèche Escape avec la flèche de la souris ou en la cliquant avec le bouton gauche de la souris. Pour quitter la barre des menus, vous devez cliquer à nouveau la flèche Escape, cliquer une fois avec le bouton gauche de la souris en dehors de la ligne d'état ou encore appuyer sur la touche Escape.

A côté est incrustée une montre. Cette montre reçoit l'heure du système d'exploitation. Elle est donc automatiquement réglée sur la bonne heure si vous avez entré l'heure dans le champ de contrôle ou bien si vous possédez un module de montre.

Pour régler vous-même la montre, vous devez la cliquer avec la souris ou appuyer sur 'F1'. Entrez alors l'heure et appuyez sur la touche 'Return'.

A côté de la montre est affiché le mode d'écriture actuel. Trois modes sont possibles :

Insert 1 : Les caractères que vous entrez sont insérés dans l'emplacement du curseur. Avec 'Return', vous faites passer le reste de la ligne (à partir de la position du curseur) sur la ligne suivante.

Insert 2 : Comme Insert 1, si ce n'est que 'Return' a seulement pour effet d'amener le curseur au début de la ligne suivante.

Overwrite : Recouvre Les caractères que vous entrez remplacent les caractères qui figuraient auparavant dans cet emplacement. 'Return' fait simplement passer le curseur au début de la ligne suivante.

Vous pouvez passer d'un mode à un autre en cliquant cette zone ou en actionnant 'F2'.

La combinaison de touches Shift + Insert commute les modes Insert 1 et Recouvre (Overwrite).

Suivent deux flèches ('⇧' et '⇩'). En cliquant ces flèches, vous pouvez feuilleter le texte en avant ou en arrière. Vous obtenez le même résultat en appuyant sur les touches curseur 'Vers le haut' et 'Vers le bas' tout en tenant la touche Control enfoncée.

Si vous cliquez la zone du numéro de ligne, vous pouvez (exactement comme avec 'F5') entrer un numéro de ligne. Au lieu d'un numéro de ligne vous pouvez aussi entrer un nom de label ou un nom de macro de 8 caractères au maximum.

Le texte sera alors examiné et dès qu'une ligne commençant par ce label ou contenant la définition de la macro voulue sera identifiée, le curseur sera immédiatement placé au début de cette ligne.

2.2. Définition des touches dans l'écran d'édition

Le clavier conserve la définition des touches correspondant aux inscriptions portées sur les touches. Cependant de nombreuses touches reçoivent en plus de nouvelles fonctions.

Les touches suivantes servent à la gestion du curseur :

Vers le haut : Curseur vers le haut avec, si nécessaire défilement du texte d'une ligne vers le bas.

Vers le bas : Curseur vers le bas avec, si nécessaire défilement du texte d'une ligne vers le haut.

A gauche : Curseur vers la gauche d'un caractère. Si le curseur figure à l'extrémité gauche de la ligne, il saute à la fin de la ligne précédente.

A droite : Curseur vers la droite d'un caractère. Si le curseur figure à l'extrémité droite de la ligne, il saute au début de la ligne suivante.

Shift + Vers le haut :
Curseur dans le coin supérieur gauche de l'écran actuel.

Shift + Vers le bas :
Curseur dans le coin inférieur gauche de l'écran actuel.

Shift + A gauche :
Curseur au début de la ligne.

Shift + A droite :
Curseur à la fin de la ligne.

Home : Curseur dans le coin supérieur gauche de l'écran actuel.

Shift + Home :
Curseur au début du texte.

Control + Home :
Curseur à la fin du texte.

Autres fonctions nouvelles :

Control + Vers le haut :
Page précédente.

Control + Vers le bas :
Page suivante.

Control + A gauche :
Mot précédent.

Control + A droite :
Mot suivant.

Delete : Supprimer le caractère sous le curseur. Tout ce qui était à droite du curseur est décalé d'une position vers la gauche.

Shift + Delete :
Comme Delete, sauf si le curseur figure à la fin de la ligne, auquel cas la ligne suivante vient s'accrocher à la suite de la ligne actuelle.

Control + Delete :
Supprimer le mot sous le curseur (y compris les espaces à la suite du mot).

Shift + Control + →
Supprimer le reste de la ligne à partir de la position du curseur.

Shift + Control + ←
Supprimer le reste de la ligne jusqu'à la position du curseur.

Backspace :
Supprimer le caractère à gauche du curseur. Tout ce qui était sous le curseur et à droite du curseur est décalé d'une position vers la gauche.

Shift + Backspace :
Comme Backspace, sauf lorsque le curseur figure en début de ligne, auquel cas la ligne actuelle va s'accrocher à la suite de la ligne précédente.

Return : En mode 'Insert 1' : le reste de la ligne, à partir du curseur, est inséré sous la ligne actuelle, comme nouvelle ligne. En modes 'Insert 2' et 'Overwrite' : le curseur saute au début de la ligne suivante.

Enter : Comme Return.

Insert : Une ligne vide est insérée au-dessus de la ligne actuelle.

Shift + Insert :
Commutation des modes d'écriture 'Insert 1' et 'Overwrite'.

TAB : Des espaces sont insérés à partir de la position du curseur et jusqu'à la prochaine tabulation.

F1 : Régler la montre (voyez la section 2.1).

F2 : Commutation 'Insert 1' - 'Insert 2' - 'Recouvre' (voyez la section 2.1).

F3 : Décaler l'écran dans le texte d'une ligne vers le bas, le curseur restant sur la même ligne.

F4 : Décaler l'écran dans le texte d'une ligne vers le haut, le curseur restant sur la même ligne.

F5 : Entrer un nouveau numéro de ligne.

F6 : Afficher la boîte de caractères spéciaux (voyez la section 3.2.4).

F7 : Continuer la recherche (voyez la section 3.4.4)

F8 : Supprimer ligne.

F9 : Traitement de macro/Fin de protocole.

F10 : Sauvegarder le fichier assemblé en arrière-plan.

Undo : Si votre dernière action a consisté à modifier ou à supprimer une ligne ou bien à supprimer un bloc, une boîte apparaît dans laquelle on vous demande si vous voulez annuler l'effet de cette opération.

Control + Help :
La zone de texte actuelle, (figurant sur l'écran), est imprimée après confirmation.

Si vous avez plusieurs textes en mémoire (voyez aussi 3.1.1) :

Control + ':' :
Changement de texte en arrière.

Control + '=' :
Changement de texte en avant.

Si 2 textes sont affichés (voyez aussi 3.1.1) :

Control + 'M' :
Changement de texte vers le texte de gauche.

Control + 'ù' :
Changement de texte vers le texte de droite.

Control + '$' :
Réduire la fenêtre de texte.

Control + '^' :
Agrandir la fenêtre de texte.

Escape : Afficher la barre des menus.

Pavé numérique :

En tenant la touche Shift enfoncée, vous pouvez entrer, sur le pavé numérique (le bloc de touches numériques à droite du clavier), le code ASCII d'un caractère (en décimal). Ce caractère sera sorti dans l'emplacement du curseur dès que vous relâcherez la touche Shift. Vous pouvez ainsi intégrer dans vos textes tous les caractères ASCII de 1 à 255.

Chiffres 1 à 9 :

En tenant la touche Alternate enfoncée et en appuyant sur une des touches de chiffre 1 à 9, vous pouvez placer jusqu'à 9 marques dans chaque texte. Ces marques peuvent être appelées en tenant la touche Control enfoncée et en appuyant sur la touche de chiffre voulue (voyez aussi les sections 3.4.1 Placer marques et 3.4.2 Aller à une marque).

Lorsque l'option 'Fonctions d'éditeur' est valide sur la ligne 'Control + lettre' de l'écran des paramètres d'édition (Menu : Editeur, Option : Paramètres), vous obtenez les fonctions supplémentaires :

Control + 'Q' : Curseur un mot sur la gauche.

Control + 'F' : Curseur un mot sur la droite.

Control + 'I' : Curseur sur la prochaine tabulation (sans insérer d'espaces).

Control + 'T' : Supprimer mot sous le curseur à partir du cu seur.

Control + 'C' : Page suivante.

Control + 'R' : Page précédente.

Control + 'Y' : Supprimer ligne.

Control + 'Z' : Curseur sur la dernière position.

Control + 'S' : Sauvegarde avec Backup.

Control + 'U' : Charger texte par dessus.

Control + 'X' : Sortir du GFA Assembleur.

Control + 'B' : Fixer début de bloc.

Control + 'K' : Fixer fin de bloc.

Control + 'H' : Annuler marquage de bloc.

Control + 'V' : Déplacer bloc.

Control + 'W' : Sauvegarder bloc.

Control + 'P' : Imprimer texte tout entier.

Control + 'D' : Imprimer zone du bloc.

Control + 'L' : Rechercher.

Control + 'E' : Rechercher et remplacer.

2.3. Les fonctions de la souris sur l'écran d'édition

➥ Simple clic avec le bouton droit de la souris : incrus- tation de la barre des menus. Elle disparaît à nouveau après un simple clic avec le bouton gauche de la souris sur la flèche Escape ou en dehors de la ligne d'état.

➥ Pour les fonctions de la souris dans la ligne d'état, veuillez vous reporter à la section 2.1.

En dehors de la ligne d'état :

➥ Appuyer une fois brièvement sur le bouton gauche de la souris : pour fixer le curseur sur cette position. Si la flèche de la souris se trouve après la fin du texte, le clic est ignoré. Si la flèche de la souris figure sur une colonne dans laquelle on n'a pas encore écrit, c'est-à-dire qu'elle désigne une colonne dépassant la longueur de la ligne actuelle, le curseur est fixé à la fin de la ligne actuelle.

➥ Double clic sur le bouton gauche de la souris : sauter directement à une ligne, un label ou une macro. Tout se déroule comme pour l'entrée d'un numéro de ligne d'un label ou d'une macro dans la ligne d'état (voyez la section 2.1.), si ce n'est qu'il n'y a, ici, rien à entrer puisque le numéro de ligne, le label ou le nom de macro voulu figure directement dans le texte.

➥ Tenir le bouton gauche de la souris enfoncé : marquer un bloc. Vous pouvez ainsi marquer très rapidement des blocs apparaissant sur l'écran. Vous pouvez aussi bien aller du point de départ au point final qu'inversement. Le point de départ et le point final ne doivent pas nécessairement être situés au début ou à la fin d'une ligne. Vous pouvez les sélectionner librement en n'importe quel endroit du texte. Lorsque vous êtes en train de marquer un bloc, vous ne pouvez faire sortir la flèche de la souris du texte actuel.

➥ Tenir le bouton droit de la souris enfoncé et appuyer une fois brièvement sur le bouton gauche de la souris : copie de bloc. Si un bloc a été marqué, il est copié dans l'emplacement de la flèche de la souris.

➥ Tenir le bouton droit de la souris enfoncé et effectuer un double clic sur le bouton gauche de la souris : déplacement de bloc. Si un bloc a été marqué, il est déplacé vers l'emplacement de la flèche de la souris.

Suivant le réglage effectué dans l'écran des paramètres d'édition, F2 Editeur puis F1 Paramètres (voyez la section 3.2.1.), l'éditeur réagira également lorsque la flèche de la souris pénètre sur le bord supérieur ou sur le bord inférieur de l'écran de la façon suivante :

➥ Avec la fonction 'Défilement', le curseur sera déplacé très rapidement dans la direction correspondante.

➥ Avec la fonction 'Ligne de menu', la barre des menus apparaît lorsque la flèche de la souris pénètre sur le bord supérieur de l'écran et la boîte du menu correspondant se déroule. Le mode d'emploi est assez semblable à celui d'un programme GEM.

L'assemblage en cours peut être interrompu en appuyant simultanément sur les deux boutons de la souris.

2.4. Fonctions macro

Les fonctions macro du GFA Assembleur permettent d'attribuer une signification différente à chaque touche du clavier. Il est possible de définir sur une touche jusqu'à 4095 frappes de touche et clics du clavier. Chaque touche peut être redéfinie huit fois au maximum (en combinaison avec les touches de commutation, SHIFT, ALT, CTRL, etc.). Cette redéfinition permet de déclencher les combinaisons de touches fréquemment utilisées à l'aide d'une seule touche.

Toutes les touches peuvent être utilisées pour la redéfinition. Afin que les macros puissent être employées même à l'intérieur des sélecteurs d'objet, il est également possible d'utiliser le clic sur le bouton gauche de la souris.

Le déclenchement de l'assemblage constitue un excellent exemple d'application des macros. Si vous voulez par exemple pouvoir déclencher l'assemblage à l'aide de la touche Help, vous pouvez affecter à cette touche : <Esc>, <6>, <2> et <Return>. Il vous suffira alors d'appuyer sur la touche Help pour que cette séquence de touches soit reproduite, ce qui lancera l'assemblage. Les macros pourraient aussi vous permettre de reprogrammer le bloc des touches numériques de façon à ce qu'il se comporte comme un bloc de contrôle du curseur (comme sur les claviers des ordinateurs compatibles avec l'IBM PC).

Les macros du clavier peuvent s'appeler mutuellement jusqu'à 16 fois. Si vous avez par exemple défini la combinaison <Shift>+ <1> pour charger un texte déterminé et la combinaison <Shift>+ <2> pour lancer l'assemblage, vous pourrez définir ces deux touches sur la touche <Shift>+ <3> pour pouvoir charger et assembler le texte voulu en frappant une seule touche.

Il résulte toutefois de cette possibilité qu'il ne faut pas affecter une touche à elle-même car cela produirait une récursion sans fin. Si vous tentez de le faire, un avertissement sonore retentit et la touche n'est pas prise en compte dans le protocole.

Enregistrer macro

Pour redéfinir une touche déterminée, on active le mode de protocole de macro et on indique au GFA Assembleur ce qu'il devra faire plus tard lorsque cette touche sera actionnée. On appuie à cet effet sur la touche de fonction F9, ce qui fait apparaître une boîte de sélection avec différentes fonctions macro. Dans cette boîte, il faut cliquer le bouton 'Enregistrer macro', ce qui fait apparaître une nouvelle boîte, dans laquelle vous pouvez entrer la touche qui doit être redéfinie. Dès qu'une touche a été

actionnée, on se retrouve sous l'éditeur et le mot 'Macro' apparaît en haut à droite de la ligne d'état. Les touches voulues sont alors imprimées dans l'ordre dans lequel elles devront plus tard être reproduites par le GFA Assembleur. Il faut ensuite appuyer à nouveau sur la touche F9 pour mettre fin au protocole.

Le protocole d'une macro ne peut être obtenu qu'à l'intérieur de l'écran d'édition. Il peut par contre être arrêté à n'importe quel endroit.

Exemple :

La touche <Help> devra déclencher l'opération d'assemblage :

<F9>	->	Appeler le sélecteur des fonctions macro, cliquer 'Protocole'
	->	Sortir le protocole de la nouvelle définition d'une touche
<Help>	->	En l'occurrence, la touche <Help>

On programme maintenant la nouvelle définition :

<Esc>	->	Passer dans la ligne du menu
<6>	->	Entrer dans la boîte de menu 'Assembleur'
<2>	->	Appeler la boîte de l'assembleur
<Return>	->	Lancer l'assembleur
<Return>	->	Revenir à l'écran d'édition immédiatement après l'assemblage
<F9>	->	Mettre fin au protocole

Remarques :



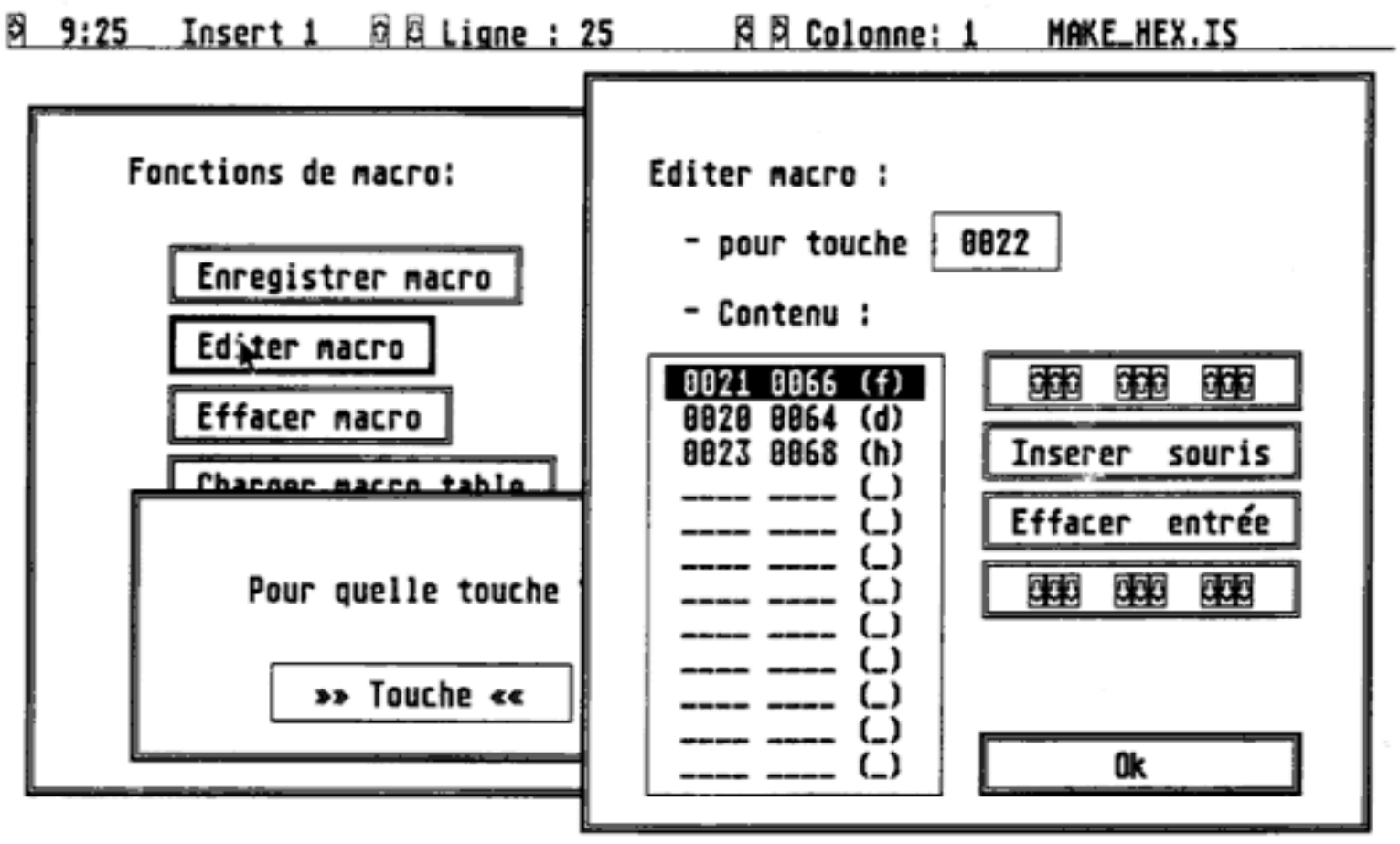
- La fonction macro permet en principe de redéfinir n'importe quelle touche. Toutefois, si des touches importantes, telles que <Esc> ou <Return> par exemple, sont redéfinies, le programme peut devenir inutilisable.
- Chaque touche physique peut être redéfinie séparément. Le <1> du bloc de touches numériques peut donc, par

exemple, être défini de façon différente de celle du <1> sur le clavier des lettres.

Editer macro

La fonction 'Editer macro' permet de modifier après coup la redéfinition d'une touche. On peut ainsi corriger des erreurs en liaison avec le protocole mais on peut aussi compléter les macros de façon à ce qu'ils puissent aussi être lancés à partir d'autres écrans que celui de l'éditeur.

Pour éditer une macro, cliquez le bouton correspondant. Une boîte apparaît alors qui vous demande d'entrer la touche dont la définition doit être éditée. Après que vous ayez entré ici une touche ayant déjà été redéfinie, une nouvelle boîte apparaît dans la moitié droite de l'écran, qui affiche la définition actuelle de la touche et vous offre différents boutons de fonction :



;

La boîte d'édition d'une définition de touche

Dans le haut de la boîte est affiché le code clavier de la touche qui a été redéfinie. Le premier caractère est toujours 0. Le second indique l'état des touches de commutation d'après les conventions suivantes :

0 : Pas de touche de commutation

3 : Shift
4 : Control
8 : Alternate
7 : Shift + Control
B : Shift + Alternate
C : Control + Alternate

F : Shift + Control + Alternate

Les troisième et quatrième caractères donnent, ensemble, le code clavier de la touche elle-même.

En dessous, à gauche figure une fenêtre avec les touches et clics de souris reproduits par le protocole. Chaque entrée de touche correspond à la sortie de la routine BIOS 'Cconin'. Toutes les valeurs sont exprimées en hexadécimal. Dans le mot du haut figure à nouveau le code clavier avec les touches de commutation, le mot du bas indiquant le code ASCII correspondant. Le caractère ASCII lui-même est affiché à la suite entre parenthèses. Pour les entrées de clic de la souris, les coordonnées X à gauche et Y à droite sont affichées en hexadécimal. A la suite apparaît le mot 'souris' pour faciliter la distinction avec les touches.

L'une de ces entrées est toujours affichée en inversion vidéo pour indiquer à quel endroit une insertion ou une suppression est possible.

A droite de cette fenêtre se trouvent les boutons permettant de modifier l'endroit où s'appliquent les insertions ou suppressions, c'est-à-dire où vous pouvez insérer ou supprimer un clic de la souris ou une touche.

Pour insérer une touche à l'endroit actuellement affiché en inversion vidéo, appuyez simplement sur la touche voulue. Pour insérer un clic de la souris, cliquez le bouton 'Insérer souris', amenez la flèche de la souris à l'endroit voulu et cliquez alors

sur le bouton gauche de la souris. Au fur et à mesure du déplacement de la souris, sa position actuelle est affichée en permanence. Après qu'une touche ou un clic de la souris ait été inséré, l'entrée suivante apparaît en inversion vidéo, vous pouvez ainsi effectuer des entrées à la chaîne.

Les boutons pour Déplacement vers le haut, vers le bas, et Supprimer entrée sont dotés d'une fonction de répétition automatique.

Une fois que la macro présente le format voulu, quittez la boîte d'édition avec le bouton 'OK'. Il n'est pas possible d'actionner pour cela la touche <Return> ou la touche <Esc> car ces touches sont reprises dans la macro si vous les actionnez.

Effacer macro

Cette fonction annule une redéfinition.

Charger table

Lors du lancement du GFA Assembleur, la table 'STANDARD.MON' est recherchée et chargée si elle est trouvée. Si vous avez besoin au cours de votre travail d'une autre table de macros, vous pouvez la charger à l'aide de ce bouton.

Sauvegarder table

Cette fonction permet de sauvegarder une table de macros créée ou modifiée.

Imprimer table

Imprime la table de macros en mémoire en tenant compte des paramètres d'impression fixés. Si ces paramètres prévoient une largeur trop petite, rien ne sera imprimé.

Chapitre 3

Les menus

La ligne de menu du GFA Assembleur fonctionne comme les barres de menus auxquelles vous êtes habitué sur le bureau de GEM et commandées aussi bien avec la souris qu'avec le clavier.

A partir de l'écran d'édition, vous accédez à la ligne de menu en appuyant sur la touche Escape, en touchant la flèche Escape avec la souris, en appuyant une fois sur le bouton droit de la souris ou encore, si la fonction 'Ligne de menu' a été activée dans l'écran des paramètres d'édition (voyez la section 3.2.1.), en amenant la flèche de la souris sur le bord supérieur de l'écran.

Dans la ligne de menu, la flèche Escape est pointée vers la gauche, ce qui signifie que vous pouvez sortir de la ligne de menu en cliquant cette flèche ou en appuyant sur la touche Escape.

Vous pouvez faire dérouler les différents menus en amenant la flèche de la souris sur le point de menu voulu ou bien en actionnant la touche de fonction indiquée avant ce point du menu ou encore en actionnant la touche numérique correspondante. C'est ainsi, par exemple, que pour faire apparaître la boîte du menu 'Fichier', vous devrez amener la flèche de la souris sur le mot 'Fichier' ou appuyer sur la touche 'F1' ou sur la touche numérique '1'.

Peu importe pour cela que vous appuyez sur une touche de la rangée des touches numériques en haut du bloc de lettres ou sur une touche du pavé numérique.

Lorsqu'une boîte de menu est affichée, La flèche Escape est dirigée vers le haut, cela signifie que vous pouvez en sortir en appuyant sur la touche Escape ou en touchant la flèche Escape avec la souris.

Vous pouvez accéder directement à un autre menu, exactement comme dans les programmes GEM, en amenant la flèche de la souris sur le point de menu voulu. De même, pour sélectionner les différentes options d'un menu, vous pouvez cliquer l'option voulue avec la souris ou la sélectionner à l'aide d'une touche de fonction ou de la touche numérique correspondante.

Les menus 'Fichier' et 'Bloc' comportent dix options. Pour sélectionner la dixième option à l'aide du clavier, vous appuyez sur la touche 'F10' ou sur la touche numérique '0'.

Le 'caractère ATARI'

Le menu 'ATARI' occupe une position particulière à l'intérieur du système de menus car on ne peut y accéder qu'avec la souris. Dans ce menu, vous pouvez activer (√ devant) ou désactiver (pas de √ devant) le point 'Sélection de fichier GEM' (GEM Fileselect). Lorsque ce point est activé, la boîte de sélection de fichier de GEM apparaîtra au lieu de l'écran de sélection de fichier du GFA Assembleur.

Dès que vous sortez de cette boîte de menu avec la souris, elle disparaît à nouveau pour éviter que le menu du 'caractère ATARI' ne soit appelé par mégarde.

Attention

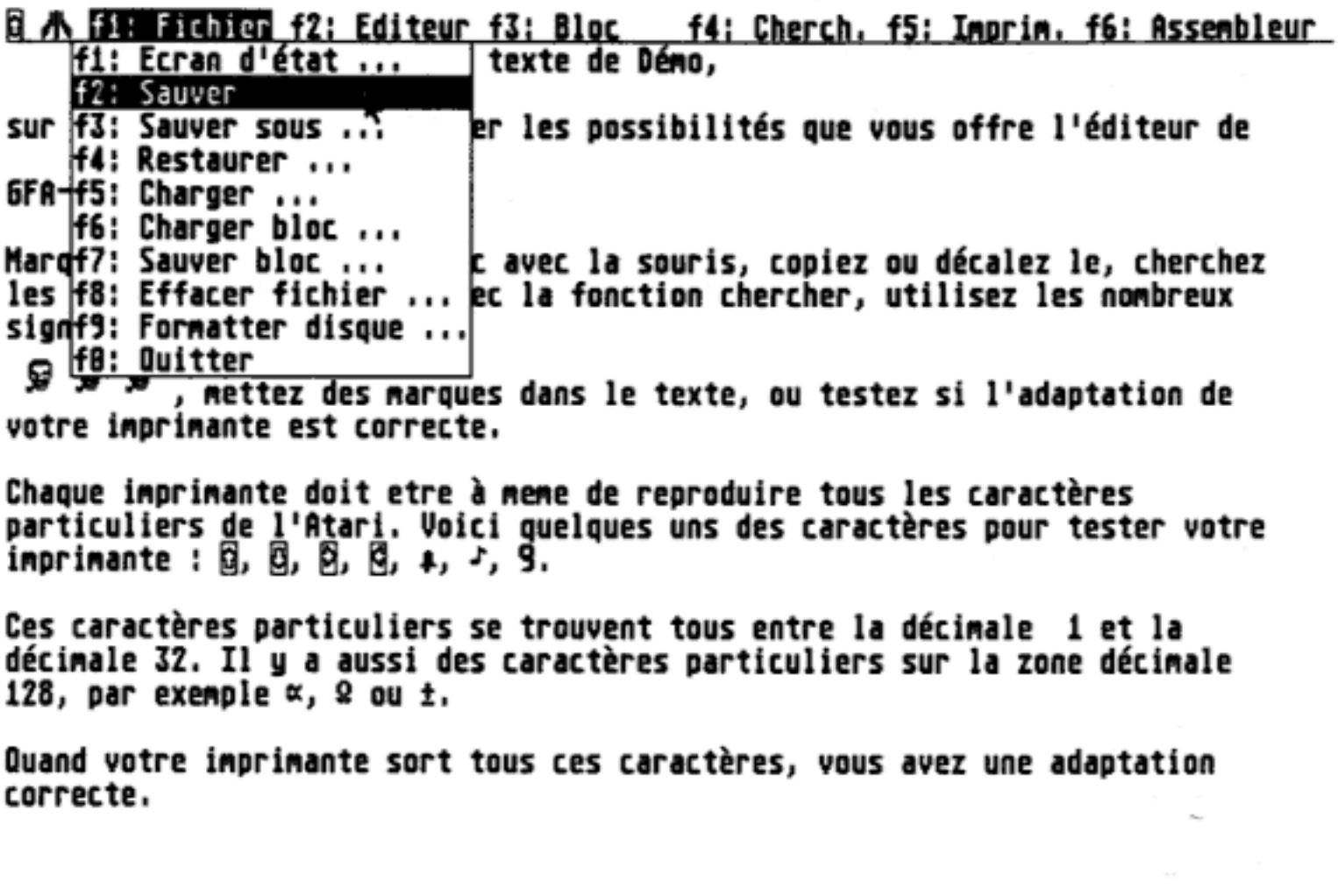


Si vous renommez l'assembleur en GFA_ASSE.TOS, il ne faut plus utiliser cette boîte car le GEM n'est pas accessible.

3.1. Le menu 'Fichier'

Le menu 'Fichier' permet d'appeler des fonctions servant à gérer, d'une part, les textes chargés et, d'autre part, l'échange de données entre le GFA Assembleur et les lecteurs de disquette,

les disques RAM et les disques durs. Lorsque vous sélectionnez le point 'Fichier' du menu, la boîte suivante apparaît sur l'écran :



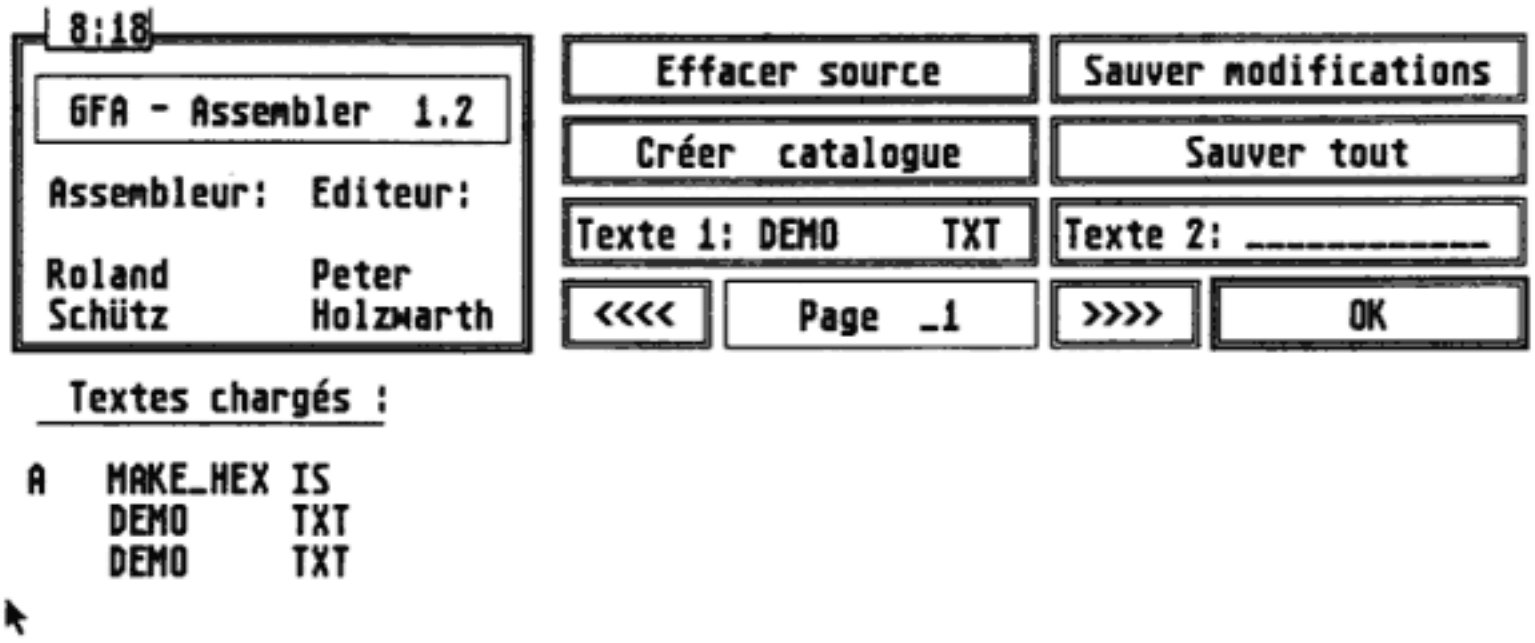
3.1.1 Etat (F1 : Fichier puis F1 : Ecran d'état)

L'écran d'état vous renseigne sur les textes figurant dans la mémoire de l'ordinateur. Il vous permet de passer à d'autres textes ou de demander que deux textes soient affichés simultanément.

Pour pouvoir essayer les fonctions de l'écran d'état, chargez un autre fichier de texte dans la mémoire, par exemple 'ASSEMBLE.BS'.

Appelez alors l'écran d'état (par exemple à travers le menu 1,1).

L'écran suivant apparaît alors :



Sur l'écran d'état est affichée l'heure actuelle en haut à gauche, au-dessus de la boîte de Copyright. A droite apparaissent les boutons, que vous pouvez cliquer avec la souris.

Dans le bas de l'écran sont énumérés les textes en mémoire. Les lettres suivantes peuvent précéder les noms de ces textes : énumération des textes

A : Le texte est un source assembleur. Il figure en mémoire sous une forme comprimée.

B : Ce texte contient le bloc global (voyez la section 3.3.9.).

x : Des modifications ont été apportées à ce texte et n'ont pas encore été sauvegardées sur disquette.

Si vous voulez maintenant faire afficher un autre texte, il vous suffit d'effectuer un double clic sur le nom du texte voulu. L'écran d'état sera alors abandonné et le texte sera affiché.

Si vous voulez savoir combien de place mémoire occupe un texte, cliquez-le une fois avec le bouton droit de la souris. Une boîte d'informations apparaîtra alors. Pour en sortir, cliquez avec la souris sur le champ 'Ok' ou appuyez sur la touche Return. Les textes occupent, en moyenne, en mémoire 15 à 20% de place de plus que sur la disquette.

Pour savoir combien de mémoire est encore disponible, cliquez, dans la boîte de Copyright, le champ 'GFA Assembleur 1.X' (X représentant ici le numéro de version). Une boîte semblable à celle indiquant la place occupée par les textes apparaît alors.

Les boutons de l'écran d'état :

Effacer source

Sélectionnez un texte (cliquer une fois avec le bouton gauche de la souris) et cliquez alors le bouton 'Effacer source'. Le texte sélectionné sera alors supprimé de la mémoire, sans demande de confirmation préalable. Il convient donc de faire attention à ne cliquer 'Effacer source' que lorsque vous voulez vraiment supprimer un texte.

Sauver modifications

Tous les textes qui ont subi des modifications et n'ont pas encore été sauvegardés sur disque sous leur nouvelle forme seront sauvegardés l'un après l'autre. Ils seront toujours replacés à l'endroit d'où ils viennent. En conséquence, le programme vous demandera de changer de disquette si vous avez chargé des textes à partir de différentes disquettes placées dans le même lecteur. Pour tous les programmes chargés à partir d'une disquette, le GFA Assembleur retient le numéro d'identification et le nom de la disquette utilisée.

Il est donc vivement conseillé d'attribuer un nom à toutes les disquettes appelées à conserver des textes du GFA Assembleur et de noter ces noms sur les disquettes. Les textes chargés à par-

tir d'un disque RAM ou d'un disque dur sont également réécrits au même endroit.

Les textes nouvellement créés sont sauvegardés dans le répertoire affiché lorsque le texte a été créé (voyez aussi la section 1.).

Créer catalogue

Un écran de sélection de fichier apparaît, dans laquelle vous pouvez spécifier où doit être stockée le catalogue.

Dans le catalogue sont enregistrés tous les textes actuellement en mémoire. Pour les textes chargés à partir de la disquette figurent, ici aussi, le numéro d'identification et le nom de la disquette.

Pour pouvoir travailler avec ce catalogue, veuillez, dans le bureau GEM, déclarer l'extension '.TAB' comme application pour le GFA Assembleur (voyez aussi la section 1.1.).

De cette manière, il vous suffira ensuite de cliquer un catalogue pour que le GFA Assembleur soit chargé et qu'il recharge alors immédiatement tous les textes dans la mémoire, en vous invitant, le cas échéant, à effectuer les changements de disquette nécessaires.

Pour voir comment on travaille avec le catalogue, veuillez maintenant créer le catalogue 'DEMO.TAB', quittez le GFA Assembleur et lancez-le à nouveau à l'aide de cette table :

Pour créer le catalogue : cliquez avec la souris le bouton 'Créer catalogue'. Entrez : 'DEMO', <RETURN>.

Pour quitter l'écran d'état : <Esc> ou <RETURN>

Pour quitter le GFA Assembleur : Menu 10,

Pour lancer avec le catalogue : double clic sur 'DEMO.TAB'

Pour parvenir dans l'écran d'état : Menu 1,1

Sauver tout

Cette fonction sauvegarde tous les textes en mémoire. A cet effet apparaît un écran de sélection de fichier dans lequel vous pouvez sélectionner le répertoire dans lequel les textes doivent être sauvegardés.

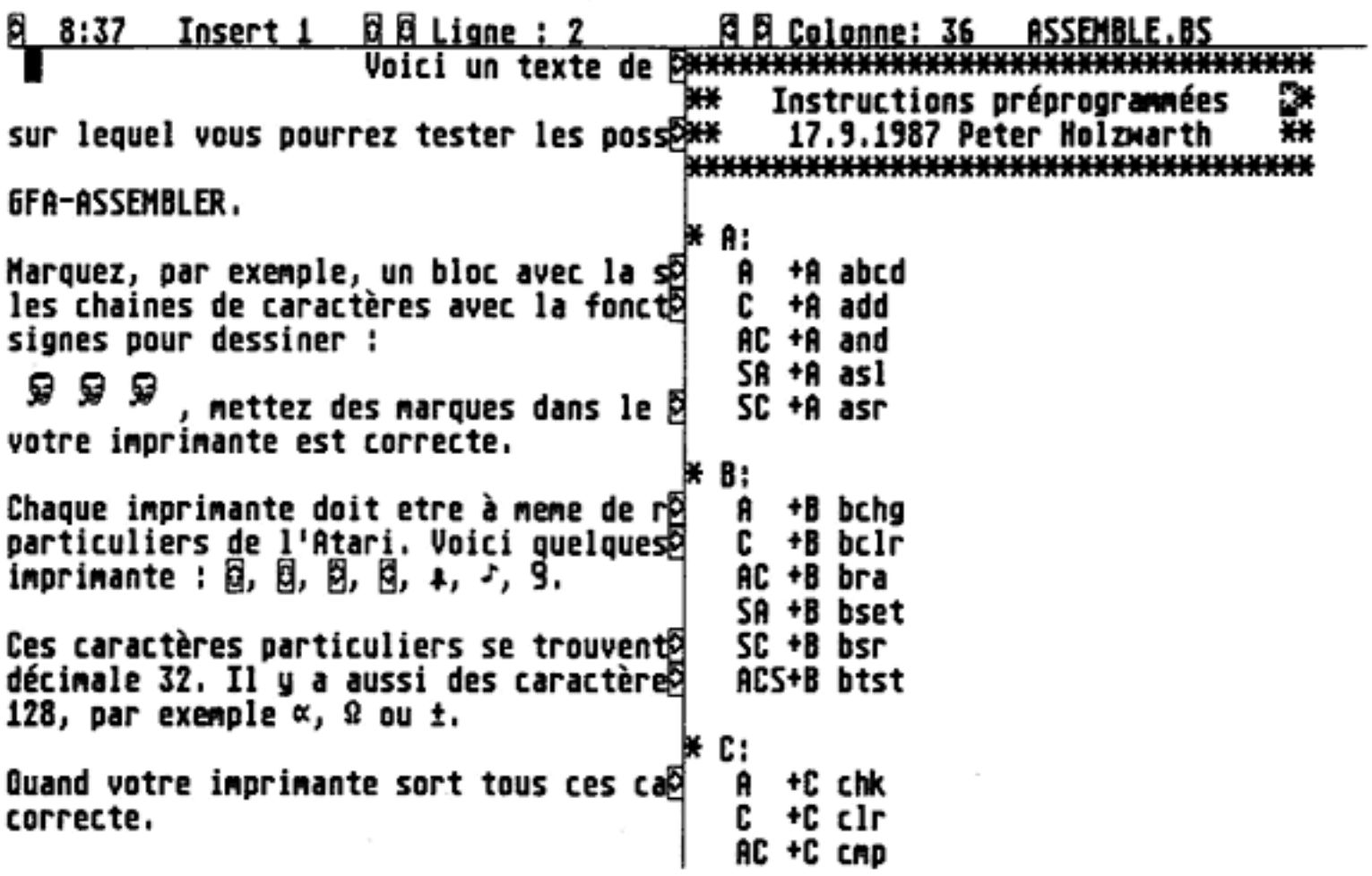
Texte 1, Texte 2

Pour faire afficher deux textes côte à côte, veuillez entrer les noms de ces textes dans les deux champs (cliquez chaque fois le texte voulu puis la boîte de texte). Pour revenir à l'écran d'édition, cliquez alors simplement sur le bouton 'Ok' ou appuyez sur la touche Return.

Le texte dont le nom figure dans la boîte de texte 1 sera affiché à gauche dans l'écran d'édition, le texte de la boîte de texte 2 à droite (mais si vous ne voulez afficher qu'un texte, il vous suffit d'effectuer un double clic sur le texte voulu).

Faites maintenant afficher le texte 'DEMO.TXT' sur le côté gauche et le texte 'ASSEMBLE.BS' sur le côté droit !

Apparaît alors un écran d'édition avec deux textes côte à côte, séparés par un trait :



Le curseur clignote dans le texte de gauche alors qu'il est fixe dans le texte de droite. Cela signifie que vous pouvez travailler sur le texte de gauche (c'est-à-dire que le texte de gauche est activé), alors que le texte de droite est simplement affiché pour mémoire. Dans la ligne d'état apparaît toujours, en haut à droite, le nom du texte actuellement activé. Vous pouvez maintenant commuter sur le texte de gauche avec Control + 'M' et sur le texte de droite avec Control + 'ù'. Ou bien vous cliquez avec la flèche de la souris dans le texte non activé.

Vous pouvez réduire la fenêtre pour le texte activé avec Control + '$' ou l'agrandir avec Control + '^' (voyez aussi la section 2.2.). La réduction ou l'agrandissement s'effectue toujours à raison de 8 caractères à la fois. La plus petite fenêtre possible pour le texte activé est d'une largeur de 24 caractères, la plus grande d'une largeur de 80 caractères.

Page x

S'il y a plus de 56 textes en mémoire, vous pouvez feuilleter le répertoire en cliquant les boutons '<<<<' et '>>>>'. Dans la boîte 'Page x' est toujours affichée la page du répertoire. Le nombre de textes pouvant être logés dans la mémoire n'est li-

mité que par la mémoire disponible. Vous pouvez donc parfaitement, dans de longs sources, les séparer en différents programmes partiels pour ne pas avoir à sauvegarder l'ensemble du texte lorsque vous n'avez effectué que de menues modifications. Ou bien, si vous créez des programmes composites, comportant des parties en assembleur et d'autres en langage évolué, vous pouvez malgré tout avoir en permanence en mémoire toutes les parties du source.

Ok

Lorsque vous cliquez le bouton 'Ok' ou que vous appuyez sur la touche Return, vous revenez à l'écran d'édition où sont affichés les textes indiqués dans les boîtes de texte. S'il n'y a de nom de texte inscrit que dans la boîte de texte 1, un seul texte est affiché.

3.1.2. Sauver un texte (F1 : Fichier F2 : Sauver)

Le texte actuel est sauvegardé dans le répertoire à partir duquel il avait été chargé. Il n'est cependant pas exigé que figure à nouveau la même disquette dans le lecteur.

3.1.3. Sauvegarder le texte sous un autre nom (F1 : Fichier F3 : Sauver sous...)

Un écran de sélection de fichier apparaît, dans lequel vous pouvez définir où le texte actuel doit être sauvegardé.

Vous pouvez évidemment conserver l'ancien.

3.1.4. Recharger texte (F1 : Fichier F4 : Restaurer)

Si vous avez apporté par erreur des modifications au texte actuel, vous pouvez faire recharger le même texte. Le texte modifié par erreur est alors effacé. Attention : si vous avez occupé tellement de mémoire que le texte de la disquette ne peut tenir dans la mémoire, le texte modifié par erreur sera effacé sans qu'un autre texte soit chargé.

3.1.5. Charger nouveau texte (F1 : Fichier F5 : Charger)

Vous pouvez ici charger un texte supplémentaire dans la mémoire ou créer un nouveau texte (voyez aussi la section 1). Après chargement, le nouveau texte est affiché dans l'écran d'édition.

3.1.6. Charger bloc (F1 : Fichier F6 : Charger Bloc...)

Cette fonction vous permet de charger un fichier, comme un bloc, dans l'emplacement actuel du curseur. Ce fichier sera ainsi reçu comme une nouvelle partie dans le texte actuel. La partie chargée est marquée comme bloc (voyez à ce propos la section 3.3.). Peu importe que le texte actuel ou le fichier à charger soient en format ASCII (en clair) ou en format assembleur (comprimé).

3.1.7. Sauvegarder bloc (F1 : Fichier F7 : Sauver Bloc)

Si vous avez marqué un bloc, vous pouvez le sauvegarder à l'aide de cette fonction. Un écran de sélection de fichier apparaît à nouveau à cet effet.

3.1.8. Supprimer fichier (F1 : Fichier F8 : Effacer Fichier)

Cette fonction vous permet de supprimer des fichiers sur disque, ce qui peut devenir nécessaire si vous manquez de place pour sauvegarder vos textes. L'écran de sélection de fichier apparaît aussi longtemps que vous ne revenez pas à l'écran d'édition avec la flèche Escape.

3.1.9. Formater une disquette (F1 : Fichier F9 : Formater disque)

Si vous n'avez plus que des disquettes pleines et que vous ne voulez supprimer aucun des fichiers qu'elles renferment, cette fonction vous permet de formater de nouvelles disquettes. Nous vous conseillons vivement d'attribuer des noms à ces disquettes dans la mesure où le GFA Assembleur se sert de ces noms (voyez la section 3.1.1.).

3.1.10. Quitter le GFA Assembleur (F1 : Fichier F0 : Quitter)

Cette fonction vous permet de mettre fin à votre travail sous le GFA Assembleur et de revenir au bureau. Si cependant, des textes qui n'ont pas été sauvegardés figurent encore dans la mémoire, une boîte de dialogue apparaît pour vous donner une possibilité de sauvegarder les fichiers modifiés avant qu'il ne soit trop tard (l'écran d'état vous indique quels fichiers n'ont pas encore été sauvegardés, voyez la section 3.1.1.).

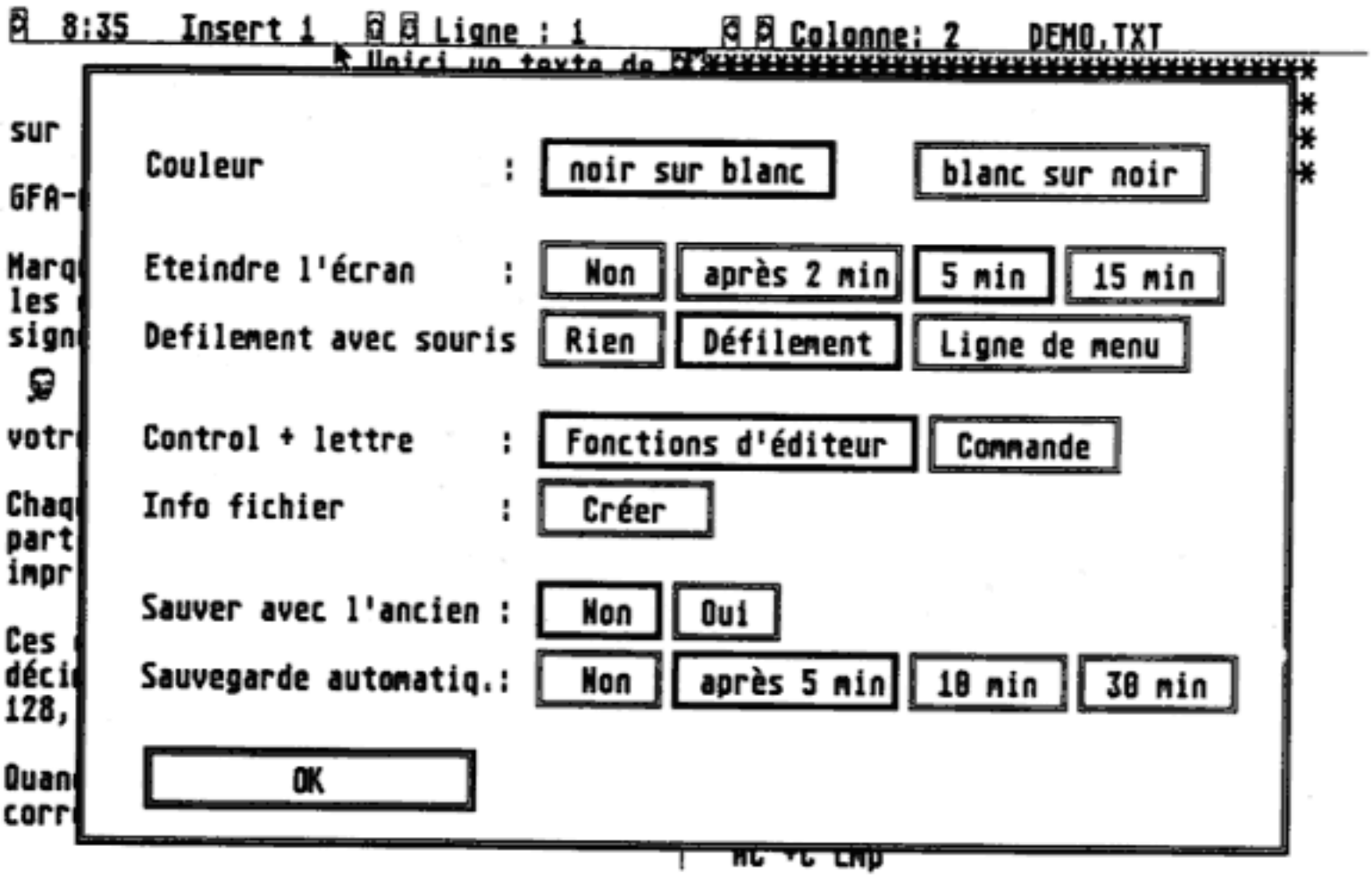
PS : Dans le cas où vous avez chargé GFA Assembleur par le GFA Basic 3.03, le retour se fait par le Basic.

3.2. Le menu Editeur

La boîte du menu 'Editeur' regroupe les fonctions en rapport direct avec l'édition des textes.

3.2.1. Paramètres d'édition (F2 : Editeur F1 : Paramètres)

Lorsque vous appelez cette fonction, l'écran suivant apparaît :



Si c'est un moniteur couleur qui est connecté à votre ordinateur, c'est la palette de couleurs actuellement sélectionnée qui apparaît en haut au lieu des deux boutons 'Noir sur blanc' et 'Blanc sur noir'.

Vous pouvez maintenant sélectionner pour un écran monochrome l'une des deux formes d'affichage indiquées ou bien sélectionner la palette de couleurs pour l'écran couleur. Il vous suffit pour cela de cliquer la couleur à modifier et de fixer alors à droite la nouvelle valeur RVB (RGB).

Vous pouvez modifier la valeur RVB (RGB) en cliquant les symboles de flèche. Vous pouvez cependant aussi amener la souris à l'aide des touches curseur sur la valeur R(R), V(G) ou B(B) voulue et modifier la valeur à l'aide des touches curseur Vers le haut et Vers le bas.

Dans la ligne située en dessous, vous pouvez choisir si vous souhaitez une extinction automatique de l'image, et si oui, après quel délai. Cette fonction sert à préserver le moniteur. Si c'est par exemple 'après 2 min' qui a été défini ici, l'écran sera éteint si aucune entrée n'est effectuée pendant 2 minutes. Dès qu'une entrée sera à nouveau effectuée (c'est-à-dire dès qu'une touche sera actionnée ou que la souris sera déplacée), l'écran sera immédiatement à nouveau allumé. Attention : si c'est un écran monochrome qui est connecté à votre ordinateur et s'il y a moins de 32 Ko de mémoire disponible, l'écran ne sera pas éteint.

Sur la ligne suivante, vous pouvez définir ce qui doit se passer lorsque le curseur de la souris est amené sur le bord supérieur ou inférieur de l'écran. Avec la sélection 'Défilement', le curseur sera alors déplacé rapidement dans la direction correspondante. Avec la sélection 'Ligne de menu', la ligne de menu apparaîtra avec la boîte de menu correspondante. Ainsi est obtenu un mode d'emploi très proche de celui des programmes GEM.

Sur la ligne 'Control + lettre', vous pouvez choisir, pour les touches actionnées en même temps que la touche 'Control', entre la définition standard (voyez la section 2.2.) et une table d'instructions (voyez la section 3.2.7.).

Si vous cliquez le bouton 'Créer fichier infos', le GFA Assembleur créera automatiquement un fichier appelé 'GFA-ASS.INF' dans lequel seront sauvegardés les réglages des boutons, le contenu des champs d'entrée et les tabulations fixées. Ce fichier sera automatiquement rechargé lors du lancement du programme, ce qui vous évite d'avoir à entrer chaque fois les mêmes sélections.

Si le bouton 'oui' est activé, à la ligne 'Sauver avec l'ancien', lorsqu'un texte est sauvegardé, un texte existant éventuellement déjà sous le même nom sera doté de l'extension '.OLD' pour ne pas être écrasé par la nouvelle version du texte.

Vous pouvez enfin, avec l'option 'Sauvegarde automatique', spécifier que, lorsque vous avez arrêté depuis un certain temps de travailler sur votre ordinateur alors que vous aviez modifié des textes au cours de la même séance de travail, ces textes soient automatiquement sauvegardés sur disquette. Seuls seront, dans ce cas, sauvegardés les textes auxquels l'ordinateur peut accéder sans changement de disquette. Par conséquent, si un texte modifié venait d'une disquette qui n'est pas celle actuellement placée dans le lecteur de disquette, ce texte ne sera pas sauvegardé.

3.2.2. Mode assembleur (F2 : Editeur F2 : Sc.en ASM...)

Lorsque 'Sc.en ASM...' est précédé d'un √, c'est que l'éditeur travaille actuellement en mode assembleur (voir plus bas). Si vous voulez convertir en format ASCII un texte qui a été réalisé en format assembleur, cliquez 'Source en ASCII'. Quelques instants plus tard, le texte figurera en mémoire sous forme d'un texte ASCII. De cette manière vous pouvez, par exemple, sauvegarder un source assembleur comme un fichier ASCII. Inversement, vous pouvez aussi convertir un texte ASCII (par exem-

ple un source assembleur que vous aviez réalisé auparavant sous un autre éditeur) en un source pour le GFA Assembleur, en le chargeant dans la mémoire (le GFA Assembleur reconnaîtra automatiquement qu'il s'agit d'un texte ASCII) puis en cliquant le mot 'Assembleur' (vous pouvez bien sûr également utiliser le clavier pour cela).

Entrée intelligente de programmes assembleur

Lors du développement d'un programme, le démon des erreurs ne sévit pas seulement avec des erreurs logiques ou des fautes de frappe. Il se manifeste parfois alors même que les différentes instructions du processeur ont été correctement utilisées individuellement. Les sources d'erreur les plus fréquentes sont les suivantes :

- Emploi de modes d'adressage interdits (par exemple move.w d0,4 (pc)).
- Dépassements de domaine de valeurs (par exemple moveq #300,d0).
- Emploi d'extensions interdites (par exemple adda.b symbol,a1).
- Oubli d'opérandes (par exemple dbra d0).
- Erreurs de syntaxe ou de frappe dans les instructions, les opérandes, les calculs etc.

Le GFA Assembleur détecte ces erreurs dès qu'une ligne est entrée. Dès que vous concluez une entrée avec 'Return', le texte transmis est converti en format tokenisé (c'est-à-dire codé et comprimé) et soumis à une détection d'erreurs éventuelles. Si une erreur est détectée, l'instruction erronée est dotée, sur la ligne, d'un message approprié et la ligne est à nouveau affichée.

L'utilisateur n'est cependant pas obligé de corriger cette erreur immédiatement. Il peut, en effet, aussi s'agir d'une erreur qui se réparera d'elle-même au fur et à mesure de l'entrée de la suite du programme, par exemple si vous avez utilisé un label dont l'orthographe précise devra être définie dans un autre endroit du texte source. Lorsqu'un programme est assemblé avec des erreurs d'entrée de ce type, ces erreurs sont à nouveau signalées par l'assembleur.

Si, par contre, une ligne entièrement correcte a été entrée, elle est automatiquement formatée en fonction des tabulations fixées.

Toutes les lignes en format tokenisé présentent fondamentalement la structure suivante (bien qu'une ligne puisse fort bien ne pas comporter tous les éléments indiqués ci-dessous !) :

Définition de symbole :	Mnémonique avec opérandes	; Commentaire
ou	ou	
Affectation de valeur (equ)	Appel de macro	

Avec le caractère ' ¡ ' (ASCII 173 ; accessible au clavier en appuyant sur Shift + Alternate et en tapant 173 sur le pavé numérique), plusieurs structures de ce type peuvent être rangées à la suite sur une même ligne, à condition que la longueur de ligne maximale de 255 caractères ne soit pas dépassée.

Les règles générales suivantes s'appliquent à l'entrée des lignes :

- Les mnémoniques assembleur sont identifiés sous quelque forme qu'ils soient écrits, majuscules, minuscules ou combinaison des deux (par exemple opérandes Move). Il n'est donc pas possible d'attribuer à une macro le nom d'une instruction assembleur.

- Les noms de symbole et de macro peuvent, si vous le souhaitez, être automatiquement convertis lors de l'entrée en majuscules ou en minuscules. Cette option peut être sélectionnée à l'aide du menu 6,1 (F6 : Assembleur F1 : Paramètres). (Voyez la partie C, section 1.1.).

➥ Les définitions de label (16 caractères au maximum !) doivent toujours se terminer par une double point (' : ', ASCII $3a). Un double double point (' :: ') aura, en outre, pour effet, lors de l'assemblage, de déclarer le symbole concerné comme global. Le double point sert de signe distinctif entre les définitions de label et les appels de macro. Certains assembleurs n'exigent pas cette distinction. C'est pourquoi un programme utilitaire fourni avec le GFA Assembleur vous permet de convertir les textes source étrangers pour les adapter à cette règle.

➥ Les mnémoniques (instructions du processeur) doivent être écrits in extenso lors de l'entrée, pour qu'ils puissent être distingués d'appels macro à consonance similaire. (Par exemple s'il avait été prévu que 'mo' soit automatiquement converti en 'move', il n'aurait plus été possible de permettre l'appel de macros appelés 'mo', 'Mo' etc.) Les conversions suivantes sont, toutefois, automatiquement effectuées lors de l'entrée :

add, sub, and, or, eor	-> addi, subi, andi, ori, eori
add, sub	-> adda, suba
move	-> movea

Si l'utilisateur souhaite réduire encore son travail de frappe, il a toujours la possibilité d'effectuer des définitions de touches personnelles, en attribuant des mnémoniques ou des instructions assembleur complètes à certaines touches, en fonction de ses besoins particuliers (voyez la section 3.2.1.).

➥ Les extensions devant être ajoutées à la plupart des mnémoniques pour spécifier la largeur de traitement peuvent être omises, auquel cas l'éditeur ajoutera des extensions par défaut, spécifiques à chaque instruction, lorsque cela peut faciliter la compréhension du texte. (par exemple 'move' deviendra 'move.w').

➥ La détection des dépassements de domaine de valeurs et des adressages incorrects est soumise à certaines limites, particulièrement lorsque la programmation macro entre en jeu (voyez les exemples).

Exemples :

```
(a)move.w symbol, #4          ; Peut être annoncé lors de l'entrée.
   move.w symbol,\parm        ; L'adresse définie par '\parm' ne
                              ; peut être déterminée qu'au moment de
                              ; l'assemblage et ne pourra donc être
                              ; contrôlée qu'à ce moment-là (voyez la
                              ; description des macros).

(b)add.w #100,d0              ; Corrigé lors de l'entrée en :
   addi.w #100,
   mais :
   add.w \parm,d0             ; Ne pourra, dans certains cas, être
                              ; adapté que lors de l'assemblage.

(c)addq.w #9,d0               ; Sera annoncée comme valeur illégale car
                              ; on dépasse la valeur 8 lors de l'entrée.
   addq.w #9-symbol,d0        ; Ne pourra être contrôlé qu'une fois que
                              ; la valeur de 'symbol' sera connue.

(d)move.w (a0)(a1),d0         ; Immédiatement détectable.
   move.w \parm(a1),d0        ; Vérifiable uniquement lors de
                              ; l'assemblage.
(e)move.b #1000-1,d0          ; Erreur de dépassement de capacité
                              ; annoncée qu'au moment de
                              ; l'assemblage car l'éditeur n'effectue
                              ; absolument aucun calcul.
```

Pour résumer, l'éditeur n'annonce que les erreurs qu'un utilisateur modèle pourrait lui-même détecter lors de l'entrée.

Il va de soi, par exemple, que les opérations complexes exécutées lors de l'assemblage ne peuvent être anticipées lors de l'entrée d'une ligne isolée de son contexte.

Les cas particuliers présentés ci-dessus correspondent cependant plus à l'exception qu'à la règle et il ne faudrait pas sous-estimer l'importance de l'aide que la correction automatique peut vous apporter lors d'une entrée rapide.

➥ Certaines erreurs posent, par contre, un problème beaucoup plus délicat, lorsque l'utilisateur n'a pas entré ce qu'il avait l'intention d'entrer (fautes de frappe) sans que l'ordinateur puisse détecter qu'il s'agit là d'erreurs.

Exemples :

➥ Symboles ou macros mal orthographiés.

➥ Les mnémoniques mal orthographiés, sans extension, peuvent être interprétés comme des appels de macro.

➥ Les définitions de label sans ':' sont interprétées comme un appel de macro

➥ Les parenthèses manquantes dans des expressions sont complétées sans avertissement lors de l'exécution d'un calcul.

➥ etc.

La ligne formatée, éventuellement complétée, telle qu'elle est affichée, après tokenisation, au lieu de la ligne entrée à l'origine, constitue un bon moyen de contrôle.

➥ Sont déclarés comme noms de registre (en majuscules ou en minuscules) :

```
d0 d1 d2 d3 d4 d5 d6 d7
a0 a1 a2 a3 a4 a5 a6 a7
r0 r1 r2 r3 r4 r5 r6 r7                (équivaut à d0-d7)
r8 r9 r10 r11 r12 r13 r14 r15          (équivaut à a0-a7)
sr ccr usp
```

➥ Le jeu d'instructions complet du Motorola 68000 est intégré.

➥ Les combinaisons de caractères incompréhensibles pour l'éditeur à la fin d'une ligne sont interprétées comme un

commentaire et séparées du début de la ligne avec un point-virgule.

Pour se familiariser avec le nouveau système d'entrée, il est conseillé, surtout aux utilisateurs inexpérimentés, de faire des essais avec des textes d'exercice.

Remarque :



Pour que les textes source assembleur du GFA Assembleur puissent plus aisément être distingués d'autres textes source, il est conseillé de toujours les sauvegarder sur la disquette avec l'extension '.IS'.

3.2.3. Mode de texte source général (F2 : Editeur F3 : Source en ASCII...)

Sous ce mode, vous pouvez utiliser l'éditeur du GFA Assembleur comme éditeur de texte source pour d'autres langages de programmation. L'éditeur du GFA Assembleur vous permettra alors de traiter les textes ASCII comme il est usuel sous d'autres éditeurs de texte source. Au sujet du travail sous d'autres langages de programmation, voyez aussi la partie D.

3.2.4. Caractères spéciaux (F2 : Editeur F4 : Caractères spéciaux)

Dans les textes que vous réalisez avec le GFA Assembleur, vous pouvez utiliser tous les caractères spéciaux de l'Atari (du code ASCII 1 au code ASCII 255). Vous obtenez ces caractères soit à l'aide du pavé numérique (voyez la section 2.2.), soit à l'aide de la boîte de caractères spéciaux. Cette boîte peut être appelée avec le menu 2,4 ou avec la touche 'F6' dans l'éditeur . Elle apparaît toujours dans la partie de l'écran où ne figure pas le curseur. Vous pouvez ainsi toujours garder un oeil sur la ligne dans laquelle vous insérez les caractères spéciaux. Vous sélectionnez les caractères spéciaux avec la souris en amenant la flèche de la souris sur le caractère voulu puis en appuyant sur le bouton gauche de la souris. Au clavier, vous pouvez déplacer la flèche de la souris graduellement dans la boîte de caractères spéciaux à l'aide des touches curseur 'Vers le haut', 'Vers le bas',

'A gauche' et 'A droite'. La touche 'Insert' permet alors d'insérer le caractère pointé par la souris.

La boîte de caractères spéciaux disparaît dès que vous entrez tout autre caractère.

3.2.5. Tabulations (F2 : Editeur F5 : Marquer tabs)

Les tabulations peuvent être fixées sur une largeur de 255 caractères, soit à intervalles fixes, soit librement. A cet effet apparaît dans le bas de l'écran une zone de 3 lignes 1/4. La première ligne représente ici les 80 premières colonnes, la deuxième les 80 colonnes suivantes etc. Les traits verticaux servent de points de repère. Vous pouvez alors, avec la souris, fixer des tabulations, en entrant des intervalles de grille fixes ou en cliquant directement la colonne voulue. Si vous préférez fixer les tabulations à l'aide du clavier, vous disposez pour cela des touches suivantes :

1 : Supprimer toutes les tabulations ou boîte "Epurer".

2-9,0 : Fixer une grille de tabulations. La touche '0' sert, dans ce cas, à fixer les tabulations de dix en dix.

Shift + pavé numérique :
Fixer / supprimer les différentes positions séparément. L'entrée s'effectue alors comme pour les caractères spéciaux en tenant la touche Shift enfoncée et en entrant la valeur voulue.

3.2.6. Déployer tabulations (F2 : Editeur F6 : Expanser tabs...)

Certains éditeurs de texte ou éditeurs d'icône etc... produisent des textes dans lesquels les espaces sont très largement remplacés par des caractères de tabulation. Cela permet de gagner de la place mais il faut en tenir compte lorsqu'on veut afficher des

textes de ce type. La fonction 'Déployer tabulations' convertit ces caractères de tabulation en espaces.

3.2.7. Sauver instructions (F2 : Editeur F7 : Sauver commandes)

Pour vous permettre d'entrer des programmes plus rapidement, vous pouvez réaliser, en 'mode source général', des listes d'instructions avec lesquelles vous attribuez aux touches du clavier jusqu'à 6 définitions supplémentaires, avec des instructions, des formules toutes faites etc... Examinez par exemple le texte 'ASSEMBLE.EC' :

```
 9:33   Insert 1   Ligne : 1        Colonne: 1    ASSEMBLE.EC
**************************************
**   Instructions préprogrammées   **
**    17.9.1987 Peter Holzwarth    **
**************************************

* A:
   A  +A abcd
   C  +A add
   AC +A and
   SA +A asl
   SC +A asr

* B:
   A  +B bchg
   C  +B bclr
   AC +B bra
   SA +B bset
   SC +B bsr
   ACS+B btst

* C:
   A  +C chk
   C  +C clr
   AC +C cmp
```

Au début de la ligne figurent les touches de commutation devant être tenues enfoncées, 'A' pour Alternate, 'C' pour Control et 'S' pour Shift. Toutes les combinaisons sont possibles à la seule exception de 'S' pour Shift (pour que les majuscules restent accessibles).

Après les touches de commutation et un signe '+' sont indiquées les touches de lettre voulues, puis la chaîne de caractères qui devra être sortie pour cette combinaison de touches. Cette chaîne peut se composer de tous les caractères du code ASCII 32 au code ASCII 255 du jeu de caractères de l'Atari. Cette liste doit être convertie en un format plus compact et sauvegardée

avec la fonction 'Save Instructions'. Si une erreur est détectée dans le format de la liste, rien n'est sauvegardé.

3.2.8. Charger commandes (F2 : Editeur F8 : Charger commandes)

Une table d'instructions sauvegardée avec 'Sauver instructions' peut être chargée dans la mémoire avec 'Charger instructions'. La nouvelle définition des touches est alors disponible. Pour que toutes ces définitions soient accessibles, le bouton 'commande' doit être activé dans la ligne 'Control + lettre' de l'écran des paramètres d'édition.

Cette table d'instructions reste en mémoire jusqu'à ce qu'elle soit remplacée par une autre.

Chargez la table d'instructions 'ASSEMBLE.KEY' et appelez par exemple l'instruction 'move'. La combinaison appropriée est indiquée dans le texte 'ASSEMBLE.EC'.

3.2.9. Comparaison de textes (F2 : Editeur F9 : Comparer Fichier)

Lorsque vous faites afficher deux textes côte à côte sur l'écran (voyez la section 3.1.1.), vous pouvez faire comparer ces textes automatiquement par le GFA Assembleur. La comparaison commence à partir de la position du curseur dans chacun des deux textes. Peu importe à cet égard que l'un des deux textes ou que les deux textes soient en mode assembleur. Si la comparaison fait apparaître des différences, le curseur est placé dans les deux textes sur la position appropriée.

3.3. Le menu 'Bloc'

Les fonctions de bloc vous permettent de déplacer des lignes de texte entières, de les copier, de les supprimer ou de les transférer d'un texte à un autre.

Un bloc particulier peut être marqué dans chaque texte en mémoire. Le point de départ et le point final du bloc peuvent être

situés en n'importe quel emplacement du texte. La zone du bloc est affichée soulignée.

3.3.1. Fixer le début du bloc (F3 : Bloc F1 : Marquer début)

Le début du bloc est fixé sur la position actuelle du curseur. Le bloc s'étend jusqu'à la fin du bloc, si une fin de bloc a été marquée, sinon jusqu'à la fin du texte. Pour contrôle, un crochet est affiché devant l'expression 'Début du bloc' et le bloc est affiché souligné.

3.3.2. Fixer la fin du bloc (F3 : Bloc F2 : Marquer Fin)

La fin du bloc est fixée sur la position actuelle du curseur. Si aucun début de bloc n'est marqué, le bloc commence au début du texte. Pour contrôle, un crochet est affiché devant l'expression 'Fin du bloc' et le bloc est affiché souligné.

Remarque : si vous voulez marquer un petit bloc apparaissant entièrement sur l'écran, vous pouvez le marquer plus rapidement avec la souris. Pour cela, cliquez au début du bloc, puis maintenez enfoncé le bouton gauche de la souris. Relâchez-le à la fin du bloc à marquer.

3.3.3. Copier bloc (F3 : Bloc F3 : Copier Bloc)

Lorsqu'un bloc est marqué, il est copié dans la position actuelle du curseur. Si le curseur figure à l'intérieur du bloc, l'instruction est ignorée.

Pour copier un bloc à l'aide de la souris, cliquez à l'endroit où doit apparaître le bloc de façon à valider l'emplacement du curseur. Puis, tout en maintenant le bouton droit enfoncé, cliquez le bouton gauche, le bloc est recopié à cet endroit.

3.3.4. Déplacer bloc (F3 : Bloc F4 : Déplacer)

Si un bloc est marqué, il est déplacé vers la position actuelle du curseur. Si le curseur figure à l'intérieur du bloc, l'instruction est ignorée.

3.3.5. Supprimer bloc (F3 : Bloc F5 : Effacer Bloc)

Si un bloc est marqué, il est effacé. L'effet de cette opération peut être annulé avec la touche 'Undo'.

3.3.6. Saut au début du bloc (F3 : Bloc F6 : Début)

Si le début du bloc a été fixé, le curseur y saute.

3.3.7. Saut à la fin du bloc (F3 : Bloc F7 : Fin)

Si la fin du bloc a été fixée, le curseur y saute.

3.3.8. Annuler marquage (F3 : Bloc F8 : Enlever Marques)

Le marquage du bloc est annulé.

3.3.9. Déclarer un bloc comme global (F3 : Bloc F9 : Bloc → Global)

Si un bloc est marqué, il est déclaré comme global, ce qui signifie que ce bloc pourra être transféré dans d'autres textes. Pour contrôle, un √ est affiché devant l'expression 'Bloc → global' et un 'B' apparaît devant le texte actuel, dans l'écran d'état (voyez la section 3.1.1.).

3.3.10. Insérer bloc global (F3 : Bloc F0 : Copier Global)

Si un bloc a été déclaré comme global dans un texte, il peut être copié à l'aide de cette fonction dans le texte actuel, à partir de la position du curseur. (Utilisez les touches Esc, F3 et F0 au lieu de la souris)

3.4. Le menu 'Recherche'

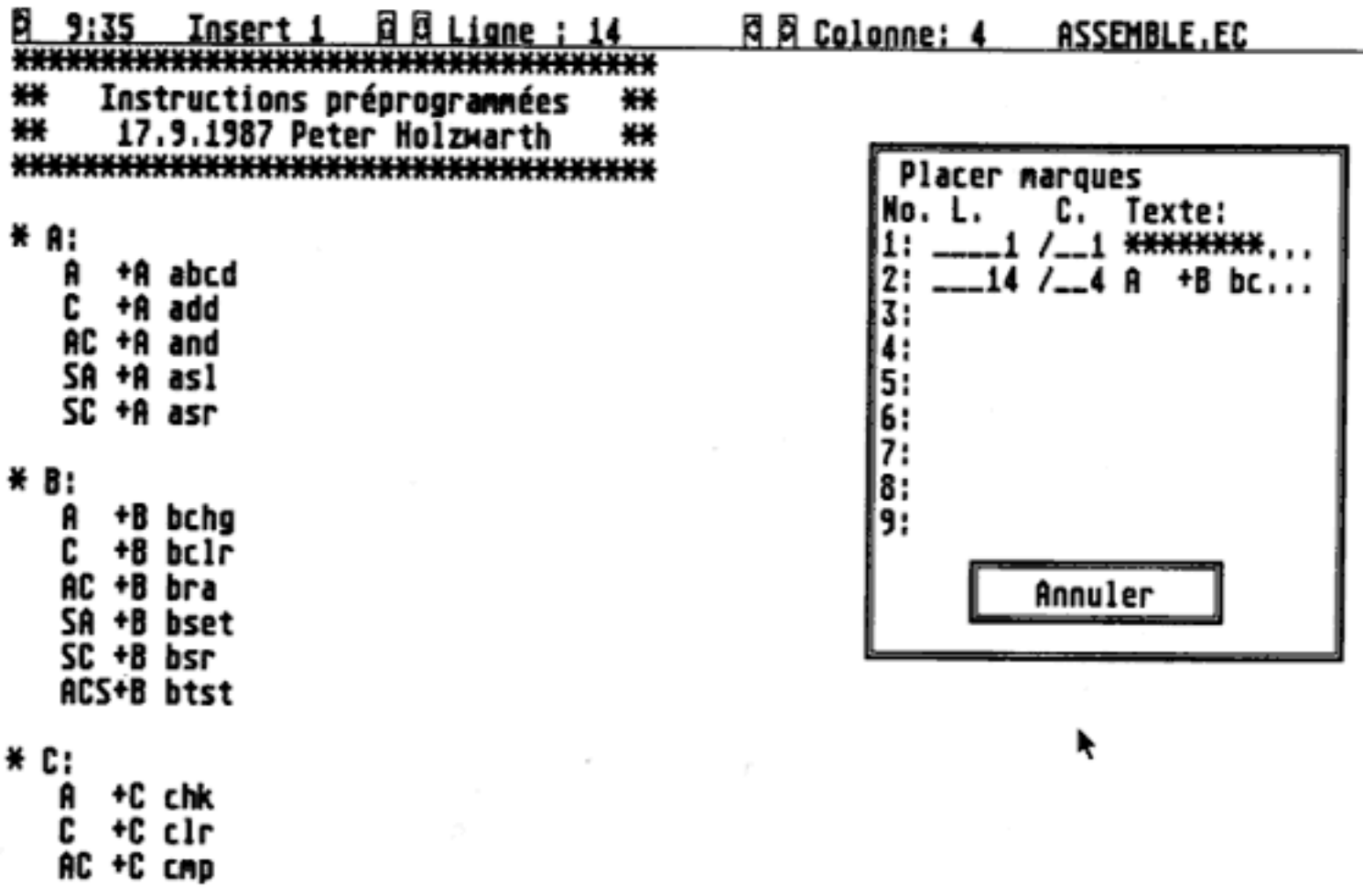
Dans la boîte du menu 'Recherche' sont regroupées les fonctions permettant de retrouver rapidement certains passages du texte.

3.4.1. Placer marque (F4 : Cherch. F1 : Placer Marques)

La possibilité de placer des marques et d'aller à ces marques permet de se déplacer rapidement à l'intérieur d'un texte, entre les passages les plus souvent utilisés.

Neuf marques peuvent être placées dans chaque texte indépendamment des autres textes.

Pour les placer, une boîte apparaît avec une liste des neuf marques disponibles.



Dans cette boîte sont inscrits, pour les marques déjà placées, la ligne, la colonne et les huit premiers caractères figurant à partir de cette position.

Pour placer une marque sur la position actuelle du curseur, cliquez la ligne correspondante de la boîte avec le bouton gauche de la souris ou bien entrez le numéro de la marque voulue directement au clavier. Vous pouvez utiliser pour cela une touche numérique ou une touche de fonction.

3.4.2. Aller à une marque (F4 : Cherch. F2 : Aller à une marque)

Les marques peuvent être appelées à partir de la boîte 'Aller à une marque' de la même façon qu'elles peuvent être placées.

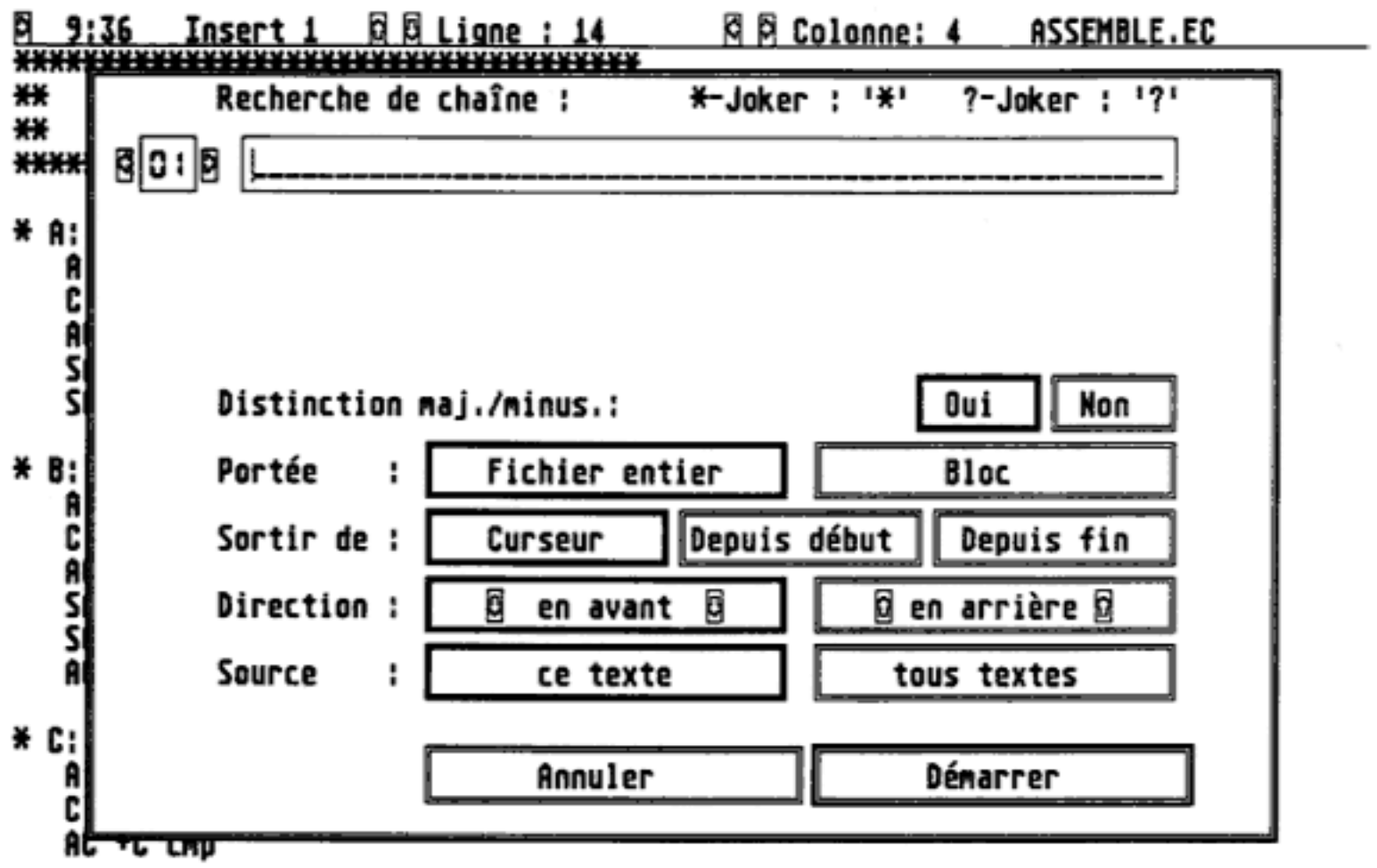
Remarque :



Les marques peuvent aussi être fixées au clavier, en appuyant sur les touches Alternate et un chiffre du pavé numérique et appelées, en appuyant sur la touche control et le chiffre voulu.

3.4.3. Chercher chaîne de caractères (F4 : Cherch. F3 : Chercher une chaîne)

Suivant un principe comparable au saut à des définitions de label, il est ici également possible de faire rechercher une chaîne de caractères figurant en n'importe quel endroit d'une ligne.



Dans la zone d'entrée, vous pouvez entrer la chaîne de caractères qui devra être recherchée dans le texte. 20 zones d'entrée de ce type sont disponibles, de sorte que vous n'êtes pas obligé d'entrer chaque fois l'expression à rechercher, même s'il y a plusieurs expressions dont vous avez souvent besoin. Vous pouvez passer d'une zone d'entrée à une autre en cliquant les symboles de flèche à gauche de la zone d'entrée. Le numéro de la zone d'entrée actuelle apparaît au fur et à mesure entre les flèches.

La chaîne de caractères peut comporter tous les caractères, du code ASCII 1 au code ASCII 255.

Les caractères '*' et '?' (ou d'autres caractères si vous le souhaitez, voyez plus bas) sont réservés en tant que "jokers". Dans les emplacements de la chaîne de caractères où figure le joker '?', il n'est pas tenu compte du caractère figurant effectivement dans le texte. Si vous entrez par exemple la chaîne 'vi?e', il ne sera pas tenu compte de la troisième lettre dans la comparaison, de sorte que les mots 'vite', 'vise', 'vire', 'vile', 'vi7e' etc. seront considérés comme étant la chaîne recherchée.

Si un joker '*' figure dans un emplacement de la chaîne de caractères à rechercher, il ne sera pas tenu compte dans la comparaison d'un nombre défini de caractères. Si vous entrez par exemple comme chaîne à rechercher 'La nourriture est*bonne', des chaînes telles que 'La nourriture est souvent bonne' seront retrouvées, mais aussi par exemple 'La nourriture est très rarement bonne'.

Si vous voulez utiliser les caractères '*' et '?' dans votre chaîne de recherche comme des caractères normaux, vous pouvez définir d'autres caractères comme remplaçants de ces deux jokers. Vous devez pour cela cliquer avec le bouton gauche de la souris l'expression '*' ou '?' et inscrire le nouveau caractère. Les deux jokers peuvent être utilisés autant de fois que vous le souhaitez à l'intérieur de la chaîne recherchée.

La signification des boutons pour distinguer entre les majuscules et les minuscules ainsi que pour la sélection du domaine de recherche est évidente. Il n'est pas possible d'effectuer une sélection illogique de ces boutons (vous ne pouvez pas par exemple effectuer une recherche en arrière à partir du début du texte).

Pour le bloc de texte, vous disposez des options 'Ce texte' et 'Tous textes'. Si le bouton 'Tous textes' est activé, la chaîne à trouver est recherchée dans le texte actuel à partir du curseur. Si elle n'est pas trouvée dans ce texte, elle est recherchée successivement dans tous les autres textes en mémoire, en commençant par le texte placé à la suite du texte actuel dans le répertoire de l'écran d'état. Le nom du texte dans lequel la recherche est en cours est affiché en permanence dans la ligne d'état.

Si l'expression recherchée est identifiée à l'intérieur du domaine spécifié, le curseur est amené dans l'emplacement correspondant. Si l'expression n'est pas trouvée, le curseur reste sur sa position.

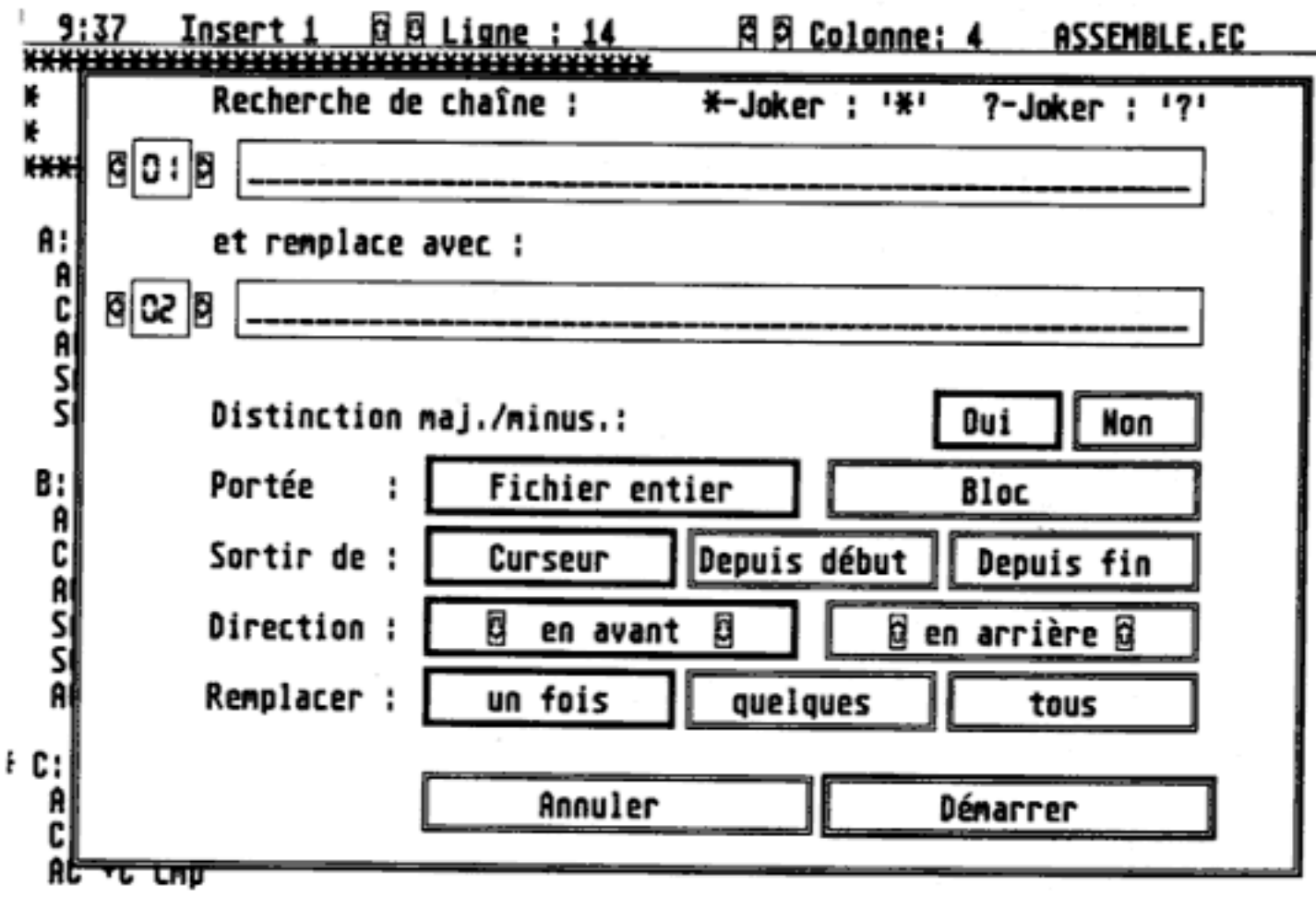
Pour vous permettre de contrôler que l'opération de recherche a bien été exécutée, un crochet est affiché devant l'expression 'Rechercher chaîne de caractères' dans la boîte du menu.

3.4.4. Continuer la recherche (F4 : Cherch. F4 : continuer Recherche)

'Continuer la recherche' offre une possibilité rapide de poursuivre la recherche de la chaîne recherchée à partir de la position du curseur. 'Continuer la recherche' ne peut être appelée que si une recherche ou un remplacement ont été effectués au préalable (ce qui est indiqué par un crochet devant la fonction concernée).

3.4.5. Chercher et remplacer (F4 : Cherch. F5 : Chercher & Remplacer)

Cette fonction offre une possibilité très commode pour faire remplacer une expression par une autre. La boîte de 'Chercher une chaîne', légèrement étendue, est mise en place à cet effet :



Cette boîte comprend un champ d'entrée supplémentaire, dans lequel peut être inscrite l'expression par laquelle la chaîne recherchée doit être remplacée. Aucun caractère joker ne peut être employé dans cette expression. La chaîne de remplacement sera donc utilisée telle qu'elle figure dans le champ d'entrée.

Cette boîte se distingue également par trois nouveaux boutons qui vous permettent de définir combien de fois doit s'effectuer le remplacement. Si vous cliquez ici 'confirmer', on vous demandera à chaque endroit du texte où la chaîne recherchée aura été trouvée d'indiquer s'il faut effectuer un remplacement à cet endroit.

3.4.6. Sauter au début du texte (F4 : Cherch. F6 : Début du texte)

Cette fonction (ou la combinaison des touches Shift + Clr/Home) fixe le curseur au début du texte.

3.4.7. Sauter à la fin du texte (F4 : Cherch. F7 : Fin du texte)

Cette fonction (ou la combinaison des touches Control + Clr/Home) fixe le curseur à la fin du texte.

3.4.8. Sauter à la dernière position (F4 : Cherch. F8 : Dernière Position)

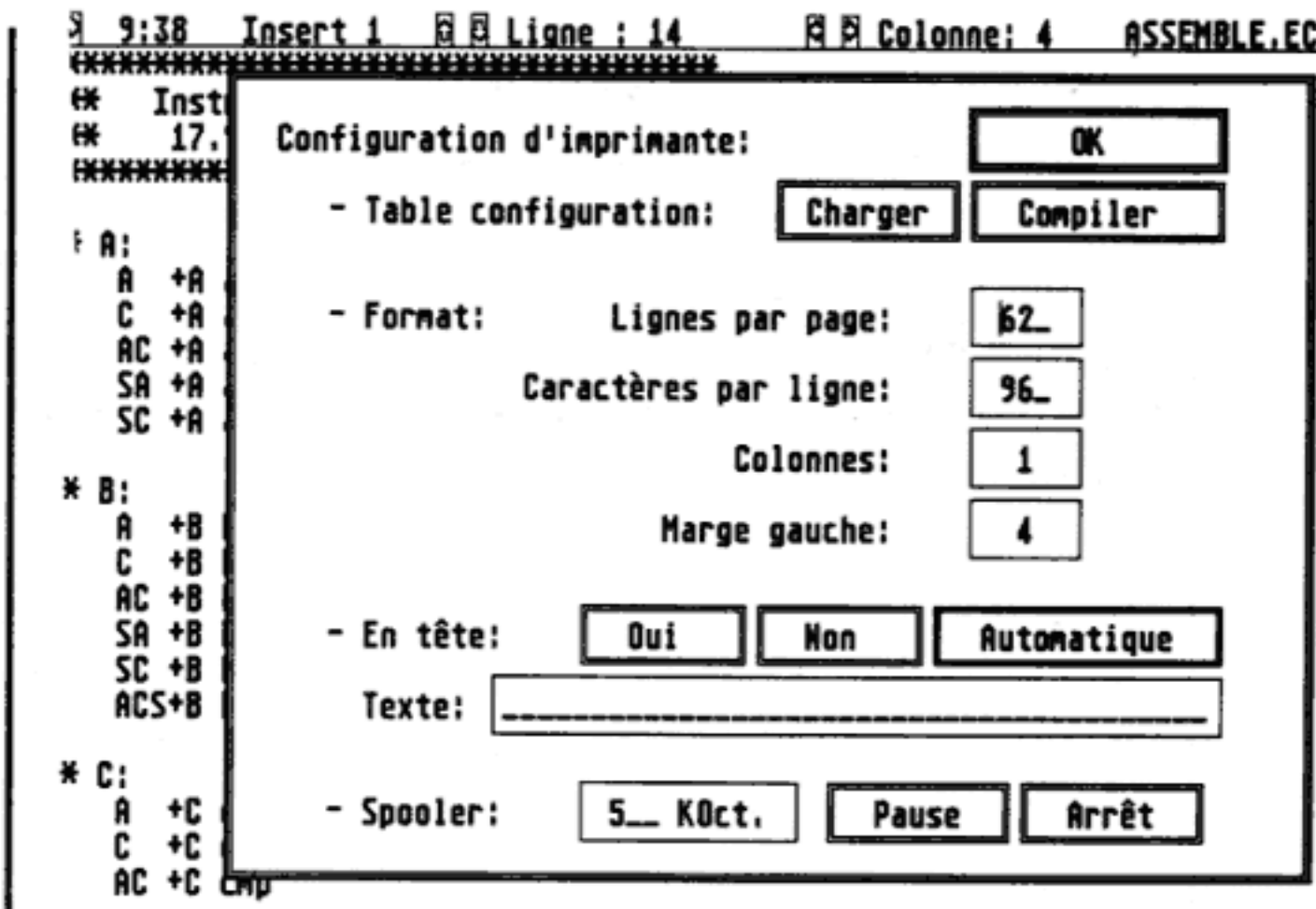
Cette fonction fixe le curseur sur la position du texte sur laquelle il se trouvait avant un saut à travers le texte (dû par exemple à une recherche de label, à une recherche de chaîne de caractères, à un saut à une marque etc.).

3.5. Le menu 'Imprimante'

Cette boîte de menu regroupe les fonctions contrôlant la commande de l'imprimante ainsi que l'opération d'impression elle-même. L'impression peut être interrompue en appuyant sur la touche 'Esc' pendant que les caractères sont écrits dans le buffer du Spooler (voyez plus bas) ou si le Spooler est désactivé. Une fois que tous les caractères sont dans le buffer du Spooler, l'impression peut être interrompue (Pause) ou arrêtée définitivement (Stop) dans la boîte de paramètres d'impression (voyez plus bas).Interruption de l'impression avec la touche 'Esc'

3.5.1. Paramètres d'impression (F5 : Imprim. F1 : Paramètres...)

Une boîte apparaît pour vous permettre de fixer le format de page, le Spooler ainsi que les tables de conversion des caractères et de configuration : Créer table de configuration.



La table de configuration que vous pouvez créer dans la première ligne contient les informations qui permettent au GFA Assembleur de communiquer avec différentes imprimantes.

La table de configuration doit être créée en format ASCII, sous le mode de texte source général, puis être traduite à l'aide du bouton 'Compiler table de configuration' en un fichier '.CFG' beaucoup plus compact avant d'être sauvegardée. Concernant le format du fichier ASCII, voyez l'annexe A4 dans la partie E.

Lors du lancement du GFA Assembleur, une table 'PRINTER.CFG' est recherchée et automatiquement chargée en mémoire. Vous avez donc tout intérêt à baptiser ainsi la table de votre imprimante standard. Si aucun fichier de ce nom n'est trouvé, le GFA Assembleur imprime avec les réglages standard (voyez l'annexe A3 dans la partie E).

Lorsque vous travaillez avec plusieurs imprimantes différentes, vous pouvez remplacer la table standard par un autre fichier '.CFG' à l'aide de la fonction 'Charger' sur la ligne 'Table configuration'.

Dans les champs d'entrée du format de page, vous pouvez fixer la taille du papier, la marge à ne pas imprimer sur la gauche et le nombre de colonnes à imprimer à la suite. L'impression de deux ou trois colonnes juxtaposées peut, en effet, se révéler fort

pratique pour de très longs listings assembleur car elle permet d'obtenir un résultat imprimé plus court et plus lisible. Pour imprimer sur plusieurs colonnes, vous devez fixer votre imprimante sur le nombre maximum de caractères qu'elle puisse imprimer par ligne. L'expérience révèle, par exemple, qu'avec 136 caractères par ligne deux colonnes d'un listing assembleur tiennent parfaitement côte à côte.

Pour doter vos textes imprimés d'un en-tête personnalisé, activez le bouton 'Oui' à la suite de 'En-tête'. Entrez ensuite, dans le champ d'entrée en dessous, le texte qui devra apparaître en tête de chaque page.

Vous pouvez cependant aussi faire créer des en-têtes automatiquement par le GFA Assembleur, en sélectionnant le bouton 'automatique'. Avec ce mode, figurera en tête de chaque page le nom du texte imprimé, l'indication qu'il s'agit d'un texte complet ou seulement d'un bloc, la date, l'heure et le numéro de la page.

Pour envoyer des chaînes de caractères directement sur l'imprimante (par exemple pour fixer le style, le nombre de caractères par ligne etc.), inscrivez la chaîne de caractères voulue dans le champ d'entrée pour l'en-tête et exécutez alors un double clic sur le champ d'entrée.

Cette boîte vous permet aussi, en dernier lieu, de contrôler le spooler d'imprimante intégré dans le programme. Ce spooler vous permet d'imprimer des textes en arrière-plan pendant que vous continuez à éditer d'autres textes au premier plan, etc. Vous pouvez fixer dans le champ d'entrée la taille du buffer du spooler en Koctets. Un grand buffer réduit naturellement la mémoire disponible pour l'entrée de textes, pour l'assemblage etc. L'expérience montre, toutefois, qu'une capacité de 5 à 30 Ko suffira pour la plupart des applications.

Si vous voulez écrire à la suite plusieurs longs textes dans le buffer pour les faire imprimer en arrière-plan, il faut naturellement que la taille du buffer soit prévue en conséquence. Ce

spooler est conçu de manière à pouvoir, sans problème, travailler aussi avec les imprimantes laser et avec un disque dur, car il travaille pratiquement en mode multi-tâches. Cela signifie qu'il imprime lorsque l'éditeur attend les entrées de l'utilisateur. Il n'y a donc pas d'impression pendant l'assemblage ou pendant d'autres opérations réclamant également beaucoup de temps de calcul.

Le bouton 'Pause' vous permet de suspendre l'impression provisoirement, le bouton 'Stop' de l'interrompre définitivement. Attention : lorsque vous voulez un texte source assembleur, il ne suffit pas de prévoir une taille de buffer spooler identique à la longueur du texte telle qu'indiquée dans l'écran d'état. C'est en effet en format ASCII que le texte doit être placé dans la buffer spooler, avec tous les caractères de formatage nécessaires. Il est donc conseillé de prévoir, pour le buffer spooler, une taille environ double de celle du source assembleur.

3.5.2. Imprimer le texte complet (F5 : Imprim. F2 : Imprimer tout)

Le texte complet sera imprimé avec les réglages effectués dans la boîte de paramètres d'impression.

3.5.3. Imprimer bloc (F5 : Imprim. F3 : Imprimer bloc)

Le bloc marqué sera imprimé avec les réglages effectués dans la boîte de paramètres d'impression.

3.5.4. Imprimer avec numéros de ligne (F5 : Imprim. F4 : Avec numéros)

La zone voulue sera imprimée avec des numéros de ligne.

3.5.5. Imprimer sans numéros de ligne (F5 : Imprim. F5 : Sans numéros)

La zone voulue sera imprimée sans numéros de ligne.

3.5.6. Imprimer toutes les pages (F5 : Imprim. F6 : Toutes pages)

Toutes les pages de la zone voulue seront imprimées.

3.5.7. Imprimer uniquement les pages impaires (F5 : Imprim. F7 : Pages impaires).

Seules les pages impaires de la zone voulue seront imprimées.

3.5.8. Imprimer uniquement les pages paires (F5 : Imprim. F8 : Pages paires)

Seules les pages paires de la zone voulue seront imprimées

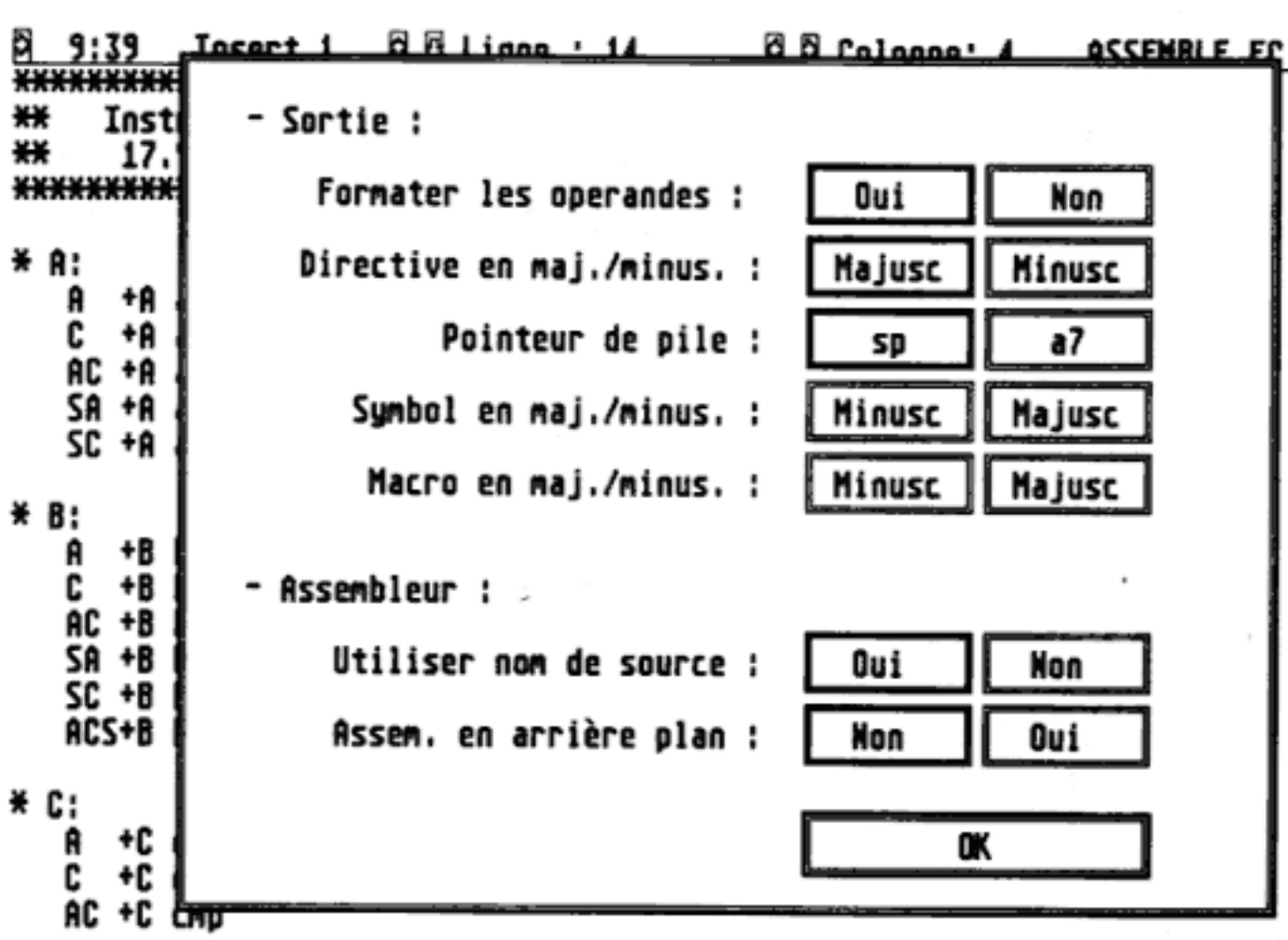
Les deux dernières options permettent d'imprimer recto verso en faisant tout d'abord imprimer toutes les pages impaires puis en retournant le paquet de feuilles ou le bloc de papier continu pour faire imprimer les pages paires sur le verso.

Partie C
Emploi de l'assembleur

Chapitre 1

Mode d'emploi avec le menu

1.1. Paramètres (F6 : Assembleur F1 : Paramètres...)



Les réglages dans le menu de paramètres agissent sur la sortie des textes sources tokenisés. Ils permettent aussi de définir si le nom des fichiers que l'assembleur devra sauvegarder devra être repris du texte source ou être entré spécialement dans le menu de sélection de fichier.

Placer les opérandes

Sortie ou interdiction d'un saut de tabulation entre les mnémoniques assembleur ou les appels macro et leurs opérandes.

Pseudo-codes opération (directive en maj./minus.)

Les pseudo-codes opération (voyez le chapitre 2) servent à commander l'assembleur et n'ont rien à voir avec les mnémoniques ni avec le jeu d'instructions du langage machine du processeur. Il est plus facile de les distinguer, on les localise plus rapidement lorsqu'elles sont sorties en majuscules.

Pointeur de pile (sp/a7)

Définit si le pointeur de pile ('**S**'tack '**P**'ointer) devra être désigné par l'abréviation 'sp' ou par la désignation du registre 'a7'.

Ecriture des symboles (minuscules/majuscules)

Pour les noms de symbole, les majuscules sont distinguées des minuscules. Vous pouvez choisir de spécifier que les symboles soient uniformément convertis en majuscules ou minuscules dès l'entrée du texte source.

Fonction des boutons Maj./Minusc.:

Bouton :	'minuscules'	'majuscules'	Fonction :
	non	non	Pas de conversion
	non	oui	Conversion en majuscules
	oui	non	Conversion en minuscules
	oui	oui	Pas de conversion

Ecriture des macros (minuscules/majuscules)

Même principe que pour l'écriture des symboles.

Assemblage en arrière-plan (oui/non)

Si ce bouton est activé, l'assemblage du texte source actuellement traité sera automatiquement entrepris pendant les temps d'attente entre les entrées de l'utilisateur. L'utilisateur ne s'en rendra pas compte dans un premier temps.

Si cet assemblage réussit (pas d'erreur), le résultat peut être sauvegardé en appuyant sur F10.

➥ Les caractères suivants peuvent, à cette occasion, apparaître devant le nom de fichier actuel pour votre information :

»	Assemblage multi-tâches
√	Prêt à être sauvegardé
?	Arrêt à cause d'erreur(s)

Remarques



Cet assemblage pseudo-multi-tâches présente l'avantage de faciliter votre travail (assemblage sans qu'il soit nécessaire d'en faire la demande) et d'utiliser les temps morts dans le travail de l'utilisateur dus aux pauses de réflexion, à la sieste etc. La vitesse d'assemblage élevée permet de réduire pratiquement à néant les pauses forcées qui sont normalement imposées à l'utilisateur lors du codage du texte source.

Si le texte concerné par l'assemblage automatique contient des instructions Include, celles-ci ne sont exécutées que si les textes source appropriés figurent déjà en mémoire. Dans le cas contraire, l'assemblage est interrompu. (L'utilisateur risquerait sinon d'être dérangé par des opérations de chargement au cours de son travail sous l'éditeur).

Si l'utilisateur modifie le texte source actuel pendant l'assemblage en arrière-plan, l'opération est immédiatement interrompue. L'assemblage recommence alors au début dès la prochaine pause dans le travail de l'utilisateur.

Utiliser le nom du source (oui)

L'assembleur reprend généralement le nom du texte source pour les fichiers qu'il produit. On peut cependant définir soi-même le chemin voulu à l'aide du menu de sélection de fichier (voyez le point suivant).

Utiliser le nom du source (non)

Une fois l'assemblage achevé, le menu de sélection de fichier s'affiche pour vous permettre de spécifier le nom de fichier voulu.

Comme, dans ce cas, l'assembleur ne connaît pas encore le chemin et le nom du fichier nécessaires, tous les fichiers produits (programme, code objet, liste, références croisées) reçoivent des noms temporaires :

```
GFA_TEMP.PRG pour les programmes
GFA_TEMP.O pour les modules objet
GFA_TEMP.LST pour les protocoles d'assemblage
GFA_TEMP.REF pour les listes de références croisées
```

Les fichiers correspondants sont placés dans le répertoire racine du lecteur dont vient le texte source.

Les fichiers qui existent déjà sous les mêmes noms sont détruits !

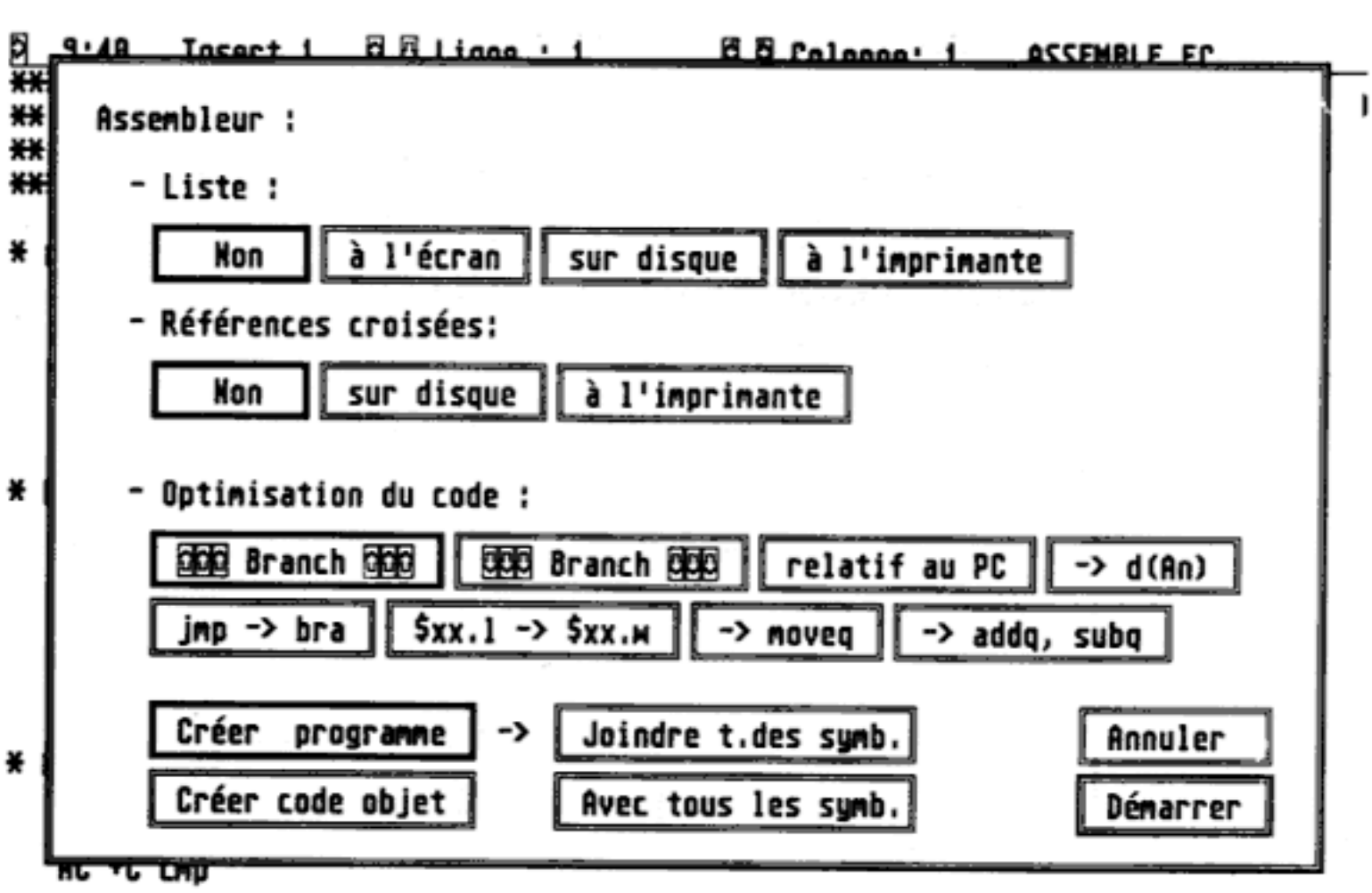
Quand le nom de fichier définitif a été fixé, les fichiers sont renommés et placés dans le chemin voulu.

Tant que le fichier n'a pas été sauvé, le lecteur ou chemin ne doit pas changer.

1.2. Assembleur (F6 : Assembleur F2 : Assembleur)

L'assembleur est appelé avec le menu 6,2.

Il ne peut être lancé que si un texte tokenisé est activé sous l'éditeur (Menu F2 : Editeur F2 : √ Sc. en Asm...). Il est commandé à l'aide du menu suivant :



Explication des différents boutons :

Liste

Fixe le périphérique de sortie pour les protocoles d'assemblage.

La sélection par défaut est 'non', c'est-à-dire que toutes les directives de protocole figurant dans le texte source seront ignorées (pour plus de détails, veuillez consulter le chapitre 2.1., pseudo-codes opération).

Références croisées

Les listes de références croisées sont très utiles lors du débogage et aident à garder la maîtrise des longs programmes. Elles énumèrent tous les symboles définis, tous les appels de macro et fichiers Include d'un texte source et indiquent quels emplacements du programme (à quels numéros de ligne) accèdent à ces éléments. C'est ainsi qu'on peut, par exemple, déterminer les routines fréquemment utilisées pour les optimiser du point de vue de leur vitesse d'exécution.

La sortie peut être effectuée dans des fichiers (lecteur de disquette, disque dur ; extension .REF) ou sur imprimante. La longueur du format de page est celle fixée dans les réglages de l'écran de paramètres d'impression (menu 5,1), alors que la largeur est toujours de 80 caractères par ligne.

Deux modes de travail sont prévus pour l'établissement de références croisées :

a) au cours de l'assemblage
b) appel séparé à travers le menu 6,3

Avec le mode **a)**, la valeur ainsi que le mot d'attribut est indiqué en plus pour les symboles. D'autre part, les symboles et macros apparaissant dans les fichiers Include sont intégrés dans la liste.

Une entrée de la liste de références croisées présente le format suivant :

Définition du symbole
Nom du symbole N de ligne/N Include {valeur et attribut}
{Lignes avec référence}

Variable/valeur absolue
* Nom du symbole N de ligne/N Include {valeur et attribut}
{Lignes avec référence}

(Si la valeur de la variable est modifiée avec 'set', c'est sa valeur à la fin de l'assemblage qui est sortie).

Définition de macro <Nom de la macro> N de ligne/N Include {valeur et attribut}
{Lignes avec appel}

(Tous les numéros de ligne sont suivis d'un trait de fraction puis du numéro du texte source concerné : 0 signifie texte principal, tout numéro plus élevé désigne le fichier Include dans lequel figure cette ligne).

Optimisation

1) Le processeur traite deux types fondamentaux de branchements conditionnels (Branch Mnemonics) :

   La version courte (⇑⇑⇑Branch⇑⇑⇑) occupe un mot seulement et contient dans l'octet faible la destination du saut sous forme d'une distance (offset) relative, signée, sur 8 bits. Elle permet donc uniquement des sauts sur -128 ou +127 octets par rapport à la position actuelle du compteur de programme.

   Dans la version far (⇓⇓⇓Branch⇓⇓⇓), la distance indiquée à la suite du code opération de l'instruction occupe un mot distinct. La distance de saut autorisée est, par conséquent, de - 32768 à +32767 octets. Cette variante est naturellement plus puissante mais elle requiert plus de temps d'exécution, à cause de l'accès supplémentaire au bus, et plus de place mémoire.

Il est donc préférable d'utiliser le premier type, le plus rapide, chaque fois que possible. L'ordinateur peut constater de lui-même si cela est possible, à condition que l'adresse de la destination du saut soit connue. Si

cette adresse est inférieure à l'adresse de l'instruction de branchement, il est facile de déterminer si elle est à la portée de sauts courts ('Optimisation en arrière').

2) Si cependant l'adresse de destination est supérieure à l'adresse de l'instruction de branchement, l'assembleur doit considérer, a priori, que la destination du saut est hors d'atteinte, de façon à ce que le branchement puisse être correctement réalisé quoi qu'il en soit. S'il apparaît par la suite, lors de la définition de l'adresse, qu'un saut court aurait suffi, ce saut peut encore être optimisé. Cela revient cependant à réduire l'instruction de branchement de 2 octets et il faut donc décaler l'ensemble du texte suivant pour le ramener de 2 octets en arrière. Le programme et les symboles doivent être adaptés à cette nouvelle disposition des adresses (optimisation vers l'avant).

A la suite de ces optimisations, il peut aussi arriver que des instructions considérées jusqu'ici comme ne pouvant être optimisées, puissent maintenant également être optimisées (parce que la destination du saut a maintenant été ramenée à la portée de sauts courts.)

3) Les sauts absolus (jsr, jmp) peuvent être convertis en leurs variantes relatives avec distance de 16 bits (bsr, bra). Les même principes que pour l'optimisation des branchements s'appliquent ici pour l'optimisation en arrière ou vers l'avant (JMP --> BRA).

4) Une autre forme d'optimisation est constituée par la conversion d'adressages longs absolus en adressages relatifs au pc. Certaines conditions doivent être remplies à cet effet :

a) L'adresse appelée doit être à la portée de l'adressage relatif (distance sur 16 bits, voyez les branchements !).

b) L'adressage d(pc) doit être autorisé pour l'instruction à optimiser.

c) Si un code objet doit être produit pour le linker, l'adresse concernée doit en outre figurer dans le même segment (texte) car ce n'est que lors du linkage que l'adresse définitive pourra être déterminée et ce n'est donc, aussi, qu'à ce moment-là qu'il sera possible de savoir si l'adresse est à la portée de l'instruction.

Pour l'optimisation d(pc) des instructions jsr ou jmp, cette option est en concurrence avec l'optimisation jmp -> bra, sur laquelle elle a cependant priorité.

Absolu->Indirect de registre d'adresse

Lorsqu'on appelle des constantes et des variables en programmation assembleur, on a souvent à travailler avec des adressages longs absolus. Ceux-ci présentent cependant 2 inconvénients :

Une longueur de 4 octets par adressage
Une durée d'exécution plus longue

Pour éviter ces inconvénients, on intègre souvent certaines zones de données dans le segment de programme et on transforme les accès en adressages relatifs au PC. Les types d'adressage relatifs au PC ne sont malheureusement autorisés que pour les opérandes source.

Une autre possibilité consiste à employer le mode d'adressage "indirect de registre d'adresse avec offset" (-->d(An)). Cet adressage est autorisés partout où l'adressage long absolu est également possible. Il consiste à charger dans un registre d'adresse la base de la zone de données à adresser et à accéder ensuite aux différentes données à l'aide d'offsets de mot. En convertissant entièrement un programme d'après cette méthode, on peut économiser jusqu'à 40% de temps d'exécution et de

place mémoire. Les zones de données très éloignées les unes des autres sont alors, dans certains cas, appelées à l'aide de registres d'adresse différents. Si cependant il n'est pas possible de se priver en permanence de certains registres dans ce but, on aura souvent recours à cette optimisation uniquement dans certaines parties du programme global.

La conversion après coup en adressages indirects de registre d'adresse risque toutefois de dénaturer un texte source jusqu'à le rendre illisible. D'autre part, le calcul des offset par rapport à l'adresse placée dans le registre d'adresse de base n'est pas toujours simple.

Le GFA Assembleur offre à l'utilisateur une possibilité plus commode pour effectuer une optimisation de ce type :

⇨ Des parties du texte source peuvent être encadrées par les directives :

```
BASE NumeroDeRegistre,AdresseDeBase
        ...
        : Texte source
        ...
         et    .ENDB NumeroDeRegistre
```

Dans ce cas, tous les adressages absolus de cellules de mémoire situés dans le même segment (voyez l'instruction SECTION !) que "AdresseDeBase" et qui n'en sont pas éloignées de plus de +- 32 Ko seront convertis en adressages indirects de registre d'adresse avec offset. L'offset est ici calculé automatiquement. C'est le registre désigné par NumeroDeRegistre qui est utilisé comme registre d'adresse. L'utilisateur doit veiller LUI-MEME à ce que la valeur "AdresseDeBase" spécifiée figure effectivement dans le registre d'adresse spécifié au moment de l'exécution du programme.

- Plusieurs optimisations concernant des registres d'adresse différents (et des SECTIONS différentes !) peuvent être exécutées en même temps. (En imbriquant des structures .BASE/.ENDB !)

- Pour activer l'optimisation, il faut encore que le bouton approprié (d(An)) soit cliqué dans le menu 6,2 d'appel de l'assembleur. L'utilisateur a ainsi la possibilité de produire des versions optimisées ou non optimisées (par exemple pour le débogage symbolique) de son programme, sans modifier le texte source. Si des instructions .BASE et .ENDB figurent dans le texte source mais que l'assemblage est lancé sans optimisation, l'assembleur avertit l'utilisateur de ce fait par un message approprié.

- La nouvelle optimisation ne peut être utilisée que combinée à l'optimisation de branchement en avant.

- Programme d'exemple :

```
Optimisation du code avec          |Optimisation ma
                                   |nuelle :
les instructions .BASE             |(résultat identi
                                   |que)
(optimisations activées :          |
branchement en avant, relatif      |
au PC (d(An) )                     |
                                   |
    .TEXT                          |   .TEXT
    lea      BASE_1,a5             |   lea     BASE_1,a5
    movea.l  #BASE_2,a6            |   move.l  #BASE_2,a6
    .BASE    5,BASE_1              |   ;
    .BASE    6,BASE_2              |   ;
    move.w   Symbole_1,Symbole_2   |   move.w Symbole_1
                                   |BASE_1(a5),Symbole
                                   |_2-BASE_2(a6)
    tst.l    Symbole_3             |   tst.l
                                   |Symbole_3-BASE_1(a5)
    beq      Label                 |   beq.s   Label
    bclr     #2,Symbole_4          |   bclr    #2,Symbole
                                   |_4-BASE_2(a6)
Label:                             |Label:
    jsr      Symbole_5             |   jsr     Symbole
                                   |_5(pc)
    nop                            |   nop
    nop                            |   nop
Symbole_5:                         |Symbole_5:
```

```
   rts                                  |  rts
   .ENDB 5                              |  ;
.ENDB 6                                 |  ;

                        .SECTION 6,BSS
                BASE_1:
                Symbole_1: .DS.w   1
                Symbole_3: .DS.l   1
                        .SECTION 7,DATA
                BASE_2:
                Symbole_2: .DS.w   1
                Symbole_4: .DS.b   1
                        .END
```

5) Les adresses figurant dans les premier et dernier blocs de 32 Ko de la zone d'adressage du processeur sont accessibles par adressage absolu court ($xx.L-->$xx.w).

L'adresse n'est donc plus un long mot comme pour l'adressage absolu long mais un mot. Le processeur étend ce mot à 32 bits, en en conservant le signe, avant l'accès au bus. Cet adressage permet donc d'accéder plus rapidement à toutes les adresses système de l'ATARI ST. Si vous le souhaitez, tous les adressages de long mot non relogeables dans ce domaine de valeurs peuvent être convertis automatiquement par l'assembleur en leur équivalent absolu-court.

6) D'autre part, les instructions d'addition et de soustraction immédiates (addi, adda bzw. subi, suba) peuvent être converties en instructions addq et subq, à condition que le premier opérande ne soit pas inférieur à 1 ni supérieur à 8 (addq, subq). De même, les instructions move.l #xx,Dn peuvent être converties en instructions moveq.l #xx, Dn si on a : -128 <= xx <= 127 (--> moveq).

Toutes ces optimisations sont intégrées dans le GFA Assembleur. La sélection par défaut est : optimisation 1 activée, optimisations 2 à 6 désactivées. Cela correspond à ce que nous appelons le mode de développement car cela permet un assemblage extrêmement rapide mais ne produit pas le code programme le plus compact possible. Cette sélection convient donc parfaitement à la phase de

test d'un programme car elle permet de travailler avec des délais d'attente négligeables. Une fois, cependant, que le programmeur a achevé son "oeuvre", il peut sélectionner des optimisations plus strictes. La durée de l'assemblage s'accroît à raison du degré d'optimisation souhaité.

Remarques



- En mode de développement, l'assembleur examine le texte source en deux parcours (passes). Pour les branchements vers l'avant, la forme courte peut être imposée à l'aide de l'extension .s (pour short). Le programmeur doit, dans ce cas, vérifier lui-même que la destination n'est pas hors de portée (faute de quoi un message d'erreur en résultera). Il n'est pas tenu compte de l'extension pour l'optimisation vers l'avant.

- Les modes d'optimisation peuvent dans certains cas entraîner plus de deux parcours, l'assembleur ne laissant rien passer.

- Seule l'optimisation des branchements vers l'avant (2) requiert vraiment beaucoup de temps. (à cause des décalages d'adresses après coup).

- Toutes les autres optimisations ne requièrent pas, normalement, de délai d'assemblage supplémentaire. Ce n'est qu'en liaison avec (2) qu'elles peuvent contribuer à augmenter le temps d'assemblage puisqu'il faut, dans ce cas, examiner également les références en avant.

- Les branchements vers d'autres sections (voyez les instructions de structuration du programme !) ne sont absolument pas optimisées.

- Si une référence (étiquette) contient une importation (Include), elle n'est pas optimisée.

➥ L'optimisation pc peut être imposée en intégrant, par exemple, délibérément, lors de la programmation, des zones de données dans le segment de texte. (Les accès aux données relèvent de l'adressage absolu long).

Code programme - code objet

Si aucune référence n'a posé de problème (du fait, par exemple, de la programmation modulaire avec des fichiers Include), des programmes directement exécutables (sans passer par le linker) peuvent être produits.

Dans le cas contraire ('fichier objet'), le code objet est produit en format DRI (Digital Research) (pour plus de détails sur le code objet, nous vous invitons à vous reporter à la documentation du linker).

Lorsqu'on travaille avec un débogueur symbolique, il est souvent intéressant de lier des tables de symboles à la suite des programmes à tester. Au lieu d'adresses hexadécimales inintelligibles, ce sont les symboles qui apparaîtront alors lors du désassemblage (pour plus de détails, veuillez vous reporter à la documentation du débogueur !).

Le code objet contient toujours une table de symboles pour le linker. Si nécessaire, il est possible de spécifier que ne soient pas seulement intégrées dans cette table les exportations et les importations mais aussi les autres symboles, pour qu'on puisse éventuellement encore y accéder, par la suite, dans le programme exécutable.

Le choix entre symboles globaux uniquement et tous les symboles est également permis pour les tables de symboles des programmes exécutables.

Au cours de l'assemblage, certaines informations sur les opérations d'assemblage sont communiquées à l'utilisateur à travers des fenêtres en incrustation. Sont notamment affichés les

accès de fichier avec les désignations de chemin employées. La fenêtre du haut ("MESSAGES :") contient ici simplement des messages pour l'utilisateur alors que la fenêtre du bas ("AVERTISSEMENTS :") est chargée de signaler les erreurs survenant lors de l'assemblage.

L'assembleur distingue deux catégories d'erreurs :

- Les "erreurs fatales" qui entraînent l'arrêt immédiat de l'assemblage. Une tentative Include d'un texte source non disponible entraînerait, par exemple, une erreur de ce type.

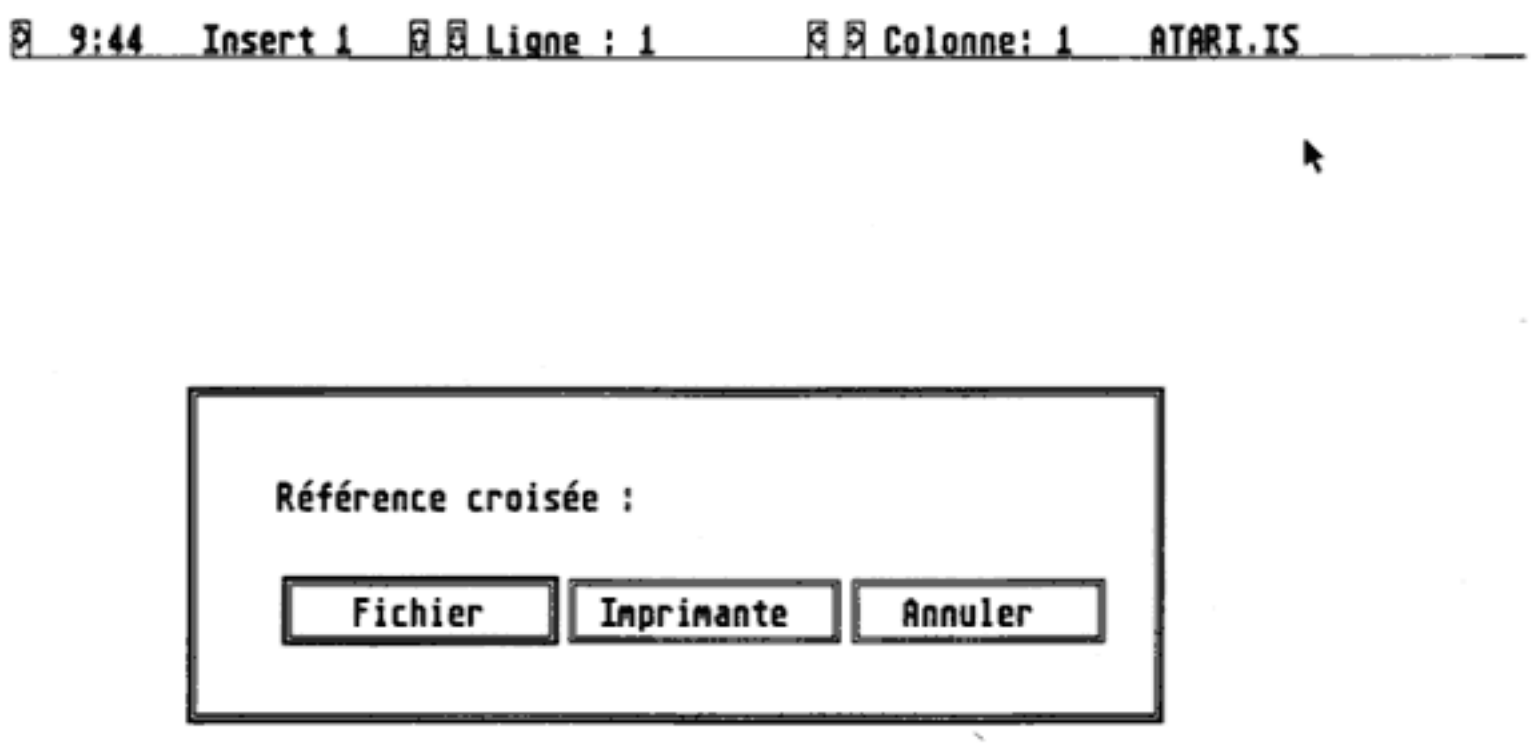
- Les "avertissements" qui n'empêchent pas l'assembleur de poursuivre le codage du texte source et ne sont donc communiqués à l'utilisateur qu'à titre d'information. Lorsque des erreurs de ce type apparaissent, le périphérique de sortie (écran, imprimante, fichier) peut être fixé en frappant une touche. Les annexes du guide de l'assembleur indiquent à l'utilisateur la signification des différents messages d'erreur et donnent des indications sur les causes possibles de ces erreurs.

Une fois un assemblage terminé, l'assembleur attend qu'une touche soit frappée pour vous permettre de lire en toute quiétude jusqu'à la fin tous les messages affichés sur l'écran. Dès que la touche Return est actionnée, on revient au mode d'édition.

1.3. Références croisées

Une liste de références croisées peut aussi être obtenue indépendamment de l'assemblage. Comme nous l'indiquions déjà dans le chapitre consacré à l'appel de l'assembleur, les valeurs et attributs des symboles ainsi que les fichiers Include ne peuvent, toutefois, être soutenus sous ce mode.

La sortie peut être, au choix, envoyée sur l'imprimante ou dans un fichier (même chemin que le texte source, extension .REF).



;

Chapitre 2

Directives du texte source

2.1. Pseudo-codes opération

2.1.1. Instructions de structuration du programme

Un programme assembleur se compose de trois segments logiques :

- Le segment de texte (qui contient le code programme)
- Le segment de données initialisé
- Le segment de données non initialisé

Lors de la réalisation du texte source, l'utilisateur définit, à l'aide de directives appropriées, auquel de ces types de données les diverses parties du programme devront être attribuées lors de l'assemblage. Par ailleurs, il est également possible de subdiviser chacun de ces trois segments en d'autres segments. Les programmes sous leur forme codée peuvent ainsi être disposés autrement que dans le texte source. Cela favorise la programmation structurée et accroît le confort d'utilisation. De plus, les structures en tableau, telles qu'elles existent dans les langages de programmation évolués peuvent être simulées à l'aide d'un quatrième segment virtuel, grâce auquel des offsets pourront être définis pour accéder aux données.

Pour définir la structure du programme, le GFA Assembleur dispose des instructions suivantes :

.TEXT

Le code programme suivant appartiendra au segment de texte.

.DATA

Introduit un segment de données initialisé.

.BSS

Début d'une zone de données non initialisée. La zone .BSS d'un programme n'est pas sauvegardée avec le programme mais elle est notée dans la tête du programme. Lorsqu'un programme est chargé, le système d'exploitation réserve une zone de mémoire appropriée à la suite du segment .DATA.

Attention :



Dans le segment .BSS, seules les instructions .DS sont autorisées.

setclu N°, Valeur

Fixer un cluster sur une nouvelle valeur. L'ancienne valeur du cluster est affichée et une question de sécurité apparaît. Cette instruction permet par exemple de marquer un cluster détecté comme étant défectueux ou bien protéger un cluster 'à la main'.

Attention :



Il convient de ne recourir à cette instruction qu'avec une prudence extrême car son emploi inconsidéré peut entraîner la perte de données précieuses sur la disquette ou le disque dur.

Si les anciennes valeurs du cluster ne coïncident pas dans les deux FATs, un message d'erreur est sorti et rien n'est modifié.

.ABS {Offset}

Le segment .ABS est une aide pour le programmeur et ne revêt pas de signification au-delà de l'opération d'assemblage. Son utilisation ressemble à celle du segment .BSS si ce n'est qu'il sert uniquement à définir les symboles absolus et qu'il n'occupe aucune place en mémoire.

Le paramètre optionnel 'Offset' désigne ici un offset de base auquel se réfèreront toutes les autres définitions dans le même segment .ABS (Définition par défaut : 0).

La directive .DS permet de définir des structures comme dans l'exemple ci-dessous. Le compteur de programme dans ce segment augmente alors automatiquement de définition de symbole en définition de symbole à raison de la zone réservée.

Lorsque le segment .ABS est abandonné, ce compteur n'est pas perdu. Sa valeur sera, en effet, reprise comme offset de base lors de l'ouverture du prochain segment .ABS, à moins qu'une valeur différente ne soit explicitement spécifiée.

Exemple 1.1

```
                    .BSS
control:            .DS.w 14        ; Zone de travail pour VDI
ptsin:              .DS.w 12

                    .ABS 0
Opcode:             .DS.w 1         ; Offsets dans le tableau control
Points_ds_ptsin:    .DS.w 1         ; vaut 2
Points_ds_ptsout:   .DS.w 1         ; vaut 4
Long_intin:         .DS.w 1         ; etc.
Long_intout:        .DS.w 1
ID_fonction:        .DS.w 1
Peripherique:       .DS.w 1

Coordonnee_X:       .DS.w 1         ; Offsets dans le tableau ptsin
Coordonnee_Y:       .DS.w 1
                    .DS.w 2
```

```
Rayon:              .DS.w 1
                    .DS.w 1

                    .TEXT
                    ; Faire dessiner sous VDI cercle
                    ; de centre (200,100) et de rayon 75

Dessiner_cercle:    lea.l       control,a0
                    move.w      #3,Points_ds_ptsin(a0)
                    clr.w       Points_ds_ptsout(a0)
                    move.w      #10,ID_fonction(a0)
                    move.w      grhandle,Peripherique(a0)
                    move.w      #200,Coordonnee_X(a0)
                    move.w      #100,Coordonnee_Y(a0)
                    clr.w       Coordonnee_Y+2(a0)
                    clr.w       Coordonnee_Y+4(a0)
                    move.w      #75,Rayon(a0)
                    clr.w       Rayon+2(a0)

                    jmp         VDI
```

(Dans le programme ci-dessus, le tableau à transmettre est initialisé avec des adressages rapides indirects de registre d'adresse, sans qu'il vous faille pour autant renoncer à utiliser des symboles suffisamment parlants. Cela facilite nettement la compréhension des textes source assembleur.)

.CARGS {#Offset_1},Symbole_1.w{,Symbole_2.l}
{,#Offset_2} {,Symbole_3.w}

L'instruction .CARGS est une variante plus compacte du segment .ABS. Elle agira donc également sur le compteur de programme d'un éventuel segment .ABS antérieur. Si elle est utilisée dans un segment .ABS, celui-ci est terminé, l'instruction .CARGS est exécutée et le segment .ABS reprend, mais avec le niveau du compteur de programme issu de l'instruction .CARGS. Les offsets et les symboles peuvent être combinés dans n'importe quel ordre.

Si plusieurs offsets sont transmis directement à la suite les uns des autres, l'assembleur additionne leur somme au compteur de programme. Seuls '.w' pour les mots et '.l' pour les longs mots peuvent être utilisées comme extensions de symbole, en fonction de quoi le niveau du compteur de programme sera respec-

tivement augmenté de 2 ou 4 octets. Seules des expressions paires peuvent être utilisées comme offsets. Si la première expression après '.CARGS' n'est pas un offset, 4 sera utilisé comme valeur par défaut. La raison tient à la possibilité d'employer l'instruction .CARGS pour une définition nominale d'offsets de paramètres transmis sur la pile, un peu comme cela se fait en langage C.

Exemple 2.1 : Fonction de définition des longueurs de chaînes

(L'adresse de la chaîne de caractères est transmise à travers la pile. Lors de cet adressage, l'adresse de retour de la routine doit être sautée.)

```
Longueur_chaine: .CARGS   .string
                 move.l   .  string(sp),a0    ; .string=4
                 moveq       #-1,d0           ; Longueur par défaut
.1:                  addq.l  #1,d0            ; lit jusqu'à
                 tst.b       (a0)+            ; '0'=Fin
          bne        .1
                 rts
```

Exemple 3.1

La définition de structure de l'exemple 1 pourrait aussi être formulée à l'aide d'instructions CARGS :

```
            .BSS
control:    .DS.w 14
            .CARGS   #0,Opcode.w,Points_ds_ptsin.w,Points_ds_ptsout.w
            .CARGS   #6,Long_intin.w,Long_intout.w,ID_fonction.w
            .CARGS   #12,Peripherique.w

pt_sin:     .DS.w 12
            .CARGS   #14,Coordonnee_X.w,Coordonnee_Y.w,#8,Rayon.w

            .TEXT
            ; etc.
```

.SECTION numero,type_segment

L'instruction SECTION est la variante de définition de segment la plus commode. Elle permet au programmeur de mettre en place 64 segments de programme avec des compteurs de programme indépendants (numéros 0 à 63). A chaque section est attribué l'un des quatre types de segment indiqués plus haut (.TEXT, .DATA,.BSS,.ABS etc.). Lors de la production du code programme, l'assembleur classe tous les types de segment utilisés par types et numéros. Toutes les sections de texte sont donc d'abord assemblées (classées par numéros croissants), puis toutes les sections Data et enfin toutes les sections Bss. L'utilisateur n'est cependant pas obligé d'employer des numéros de segment qui se suivent. Il peut même, par exemple, spécifier le numéro de segment comme paramètre à transmettre à une macro.

Un seul type peut être attribué à chaque segment. Lorsqu'un segment est abandonné puis repris ultérieurement, l'assembleur reprend l'ancien niveau du compteur de programme. (Exception : le segment .ABS, voyez plus haut). Pour le reste, une 'section' se comporte comme les segments "standard".

Les branchements (Bcc) et les adressages relatifs au PC sont autorisés entre les segments mais ils ne peuvent être optimisés, pour des raisons structurelles.

Remarques :



- Comme le système d'exploitation ne distingue que trois segments, la division en sections n'a pas d'effet au-delà de l'opération d'assemblage. Elle représente simplement un élément de la programmation structurée.

- Les sections 0 à 3 sont prédéfinies par les instructions .TEXT, .DATA, .BSS et .ABS et ne peuvent être appelées qu'avec les types correspondants.

```
.TEXT équivaut donc à.SECTION 0,TEXT
.DATA équivaut à.SECTION 1,DATA
.BSS  équivaut à.SECTION 2,BSS
.ABS {Offset} à.SECTION 3,ABS {Offset}
```

.END

L'instruction .END, à la fin du texte source, est facultative. Au milieu du texte, elle a pour effet de faire irrévocablement ignorer toutes les instructions suivantes.

Elle marque également la fin des fichiers .INCLUDE (voyez plus bas).

.LOCAL

Les symboles locaux ne valent qu'entre deux symboles non locaux. Cette instruction permet de délimiter des zones de symboles locaux sans définir un label non local. (Le chapitre consacré aux symboles vous en apprendra plus à ce propos.)

.GLOBL, .EXTERN, .PUBLIC, .XDEF, .XREF

sont des synonymes pour la déclaration globale de symboles. Si l'assembleur est censé produire non pas un programme exécutable mais un code objet pour un linker, tous les symboles devant être importés ou exportés doivent être déclarés à l'aide de l'une des instructions indiquées ci-dessus. (La déclaration globale de labels peut également se faire avec un double double point ('::') placé à la suite du label ! Voyez le chapitre sur les symboles.)

COMM Symbole,Expression

Cette instruction s'applique uniquement au linker. Si un code objet est produit, le symbole spécifié avec .COMM sera interprété comme une importation et recevra le statut de symbole

'common'. Le linker recherchera les symboles 'common' du même nom dans tous les modules objet à traiter. Une valeur a été attribuée, avec 'Expression', à chacun de ces symboles lors de l'assemblage. Lors du linkage, le symbole 'common' sera repris dans le segment .BSS et doté d'une adresse appropriée. La zone de données non initialisée subséquente comportera autant d'octets que défini par la plus grande des valeurs chaque fois spécifiées par 'Expression'.

Cette option permet de faire aisément employer des zones BSS par plusieurs modules objet lorsque ces routines n'exigent pas que ces zones de données demeurent intactes.

2.1.2. Macros et Include

Les macros constituent le principal outil de programmation structurée en assembleur. Elles permettent de regrouper de nombreux mnémoniques assembleur et pseudo-codes opération sous un nom de fonction. Une macro ainsi définie peut être utilisée à tout moment avec des paramètres de communication, comme lors d'un appel de fonction dans un langage évolué (comme le C par exemple). Lorsque le programmeur, au cours de son activité, a accumulé une collection suffisamment complète de définitions de macros, qui peuvent alors être par exemple chargées avec un fichier Include, il peut alors réaliser en très peu de temps des programmes assembleur très lisibles.

La directive Include est la grande soeur de l'appel de macro. Elle permet d'intégrer des fichiers de texte source complets dans l'assemblage en cours. Elle permet non seulement, comme nous venons d'y faire allusion, d'intégrer des fichiers Include contenant des définitions de macros et de variables mais aussi d'organiser le programme en unités plus maniables, ce qui rend le programme d'ensemble beaucoup plus facile à maîtriser.

Pour démontrer la puissance de la programmation des macros, nous voudrions d'emblée vous proposer un premier exemple sans explication :

Exemple 2.0 : L'instruction Print sous forme d'une macro

Définition de la macro :

```
.MACRO    BCONOUT dev,char     ; Fonction BIOS
    clr.w       d0                  ; Bconout comme macro
    move.b      \char,d0            ; Caractère à sortir
    move.w      d0,-(a7)            ; (étendu à un mot)
    move.w      #\dev,-(a7)         ; Périphérique de sortie
    move.w      #3,-(a7)            ; Numéro de fonction
    trap        #13                 ; Appel BIOS
    addq.l      #6,sp               ; Correction de la pile

                .ENDM

        .   MACRO Print Chaine,Device   ; Macro Print
            .DATA
            .IF         \?Chaine        ; Si chaîne transmise, la
\~a:    .DC.b           \Chaine         ; sauvegarder dans le segment
\~b:                                    ; de données pour la sortir
            .ELSE
\~a:    .DC.b           13,10           ; sinon : sortir carriage
\~b:                                    ; return
            .ENDIF
            .TEXT
            lea.l       \~a,a3          ; Adresse de la chaîne
            moveq.l     #\~b-\~a-1,d3   ; Longueur de chaîne moins 1
.\~:    BCOUNOUT    \Device,(a3)+       : Sortir 1 caractère
            dbra        d3,.\~          ; Caractère suivant
            .ENDM
```

Appel de la macro :

```
            Print "Salut !",2           ; Ecrit Salut ! et une
            Print ,2                    ; ligne vide sur l'écran
```

2.1.2.1. Définition de macro

.MACRO Nom_macro {Nom_1}{,}{Nom_2}
ou Nom_Macro . Macro {Nom_1}{,{{Nom_2}

Une définition de macro commence par le pseudo-code opération .MACRO suivi d'un nom, pouvant comporter jusqu'à 16 caractères, et de jusqu'à 16 noms de paramètres optionnels. Ces paramètres de communication ne doivent pas être confondus

avec des labels. Ils n'entraînent pas de définition de symbole mais permettent le test symbolique des données transmises lors de l'appel de la macro. C'est pourquoi les opérateurs de label ne peuvent d'ailleurs pas être utilisés. Le texte suivant l'instruction .MACRO représente le corps de la macro. Les instructions utilisées dans le corps du macro ne sont soumises à aucune limitation. Une zone de macro se termine par la directive .ENDM. En liaison avec l'assemblage conditionnel, une macro peut aussi être abandonnée prématurément avec .EXITM. (Plus de détails à ce propos dans le chapitre consacré à l'assemblage conditionnel !)

Exemple 2.1. :

```
.MACRO Nom_macro Parametre
;
; Ici figure ce que l'assembleur doit
; exécuter lorsque la macro est appelé.
;
.ENDM
```

L'imbrication de définitions de macros constitue un cas particulier. Le corps de la macro encadrée par les instructions .MACRO-.ENDM le plus à l'intérieur n'est attribué qu'à cette macro. Les imbrications extérieures respectives "ignoreront" donc les définitions de macros effectuées en leur sein. Nous vous déconseillons de recourir trop fréquemment à cette variante qui peut contribuer à créer une confusion dont l'utilisateur sera à la fois la cause et la victime.

Exemple 2.2. :

a) Définitions imbriquées :

```
.MACRO Definition_1 Parametre_1
nop;                    appartient à Definition_1
nop             ;       appartient à Definition_1
    .MACRO Definition_2 Parametre_2
    move.w      d0,d1   ; appartient à Definition_2
    move.w      d0,d1   ; appartient à Definition_2
    .ENDM
```

```
    nop                    ; appartient à Definition_1
.ENDM
```

b) Définitions équivalentes sans imbrication :

```
.MACRO Definition_1 Parametre_1
nop
nop
nop
.ENDM

.MACRO Definition_2 Parametre_2
move.w      d0,d1
move.w      d0,d1
.ENDM
```

2.1.2.2. Appel de macro

Nom_macro {Parametre_1}{,}{Parametre_2}...

Une macro ne peut être appelée avant d'avoir été définie. Un appel de macro a pour effet de faire insérer par l'assembleur dans la position actuelle du texte source les instructions spécifiées dans le corps de la macro lors de la définition de la macro. Il est possible d'appeler un nombre illimité d'autres macros à partir d'une macro donnée (Exemple 2.0).

Les paramètres suivant éventuellement le nom d'appel peuvent être toutes les expressions pouvant être identifiées comme correctes sur le plan de la syntaxe par l'éditeur, y compris des modes d'adressage.

Pour vous aider à mieux comprendre les possibilités d'application de la programmation des macros, nous vous renvoyons également aux définitions du système d'exploitation fournies avec le produit !

2.1.2.3. Fonctions spéciales dans le corps de la macro

Outre les directives possibles pour l'ensemble du texte source, il existe diverses fonctions spéciales qui peuvent être employées dans le corps d'une macro.

Test des paramètres transmis :

\Parametre

Le nom suivant le 'Backslash' (=trait transversal vers l'arrière).

\\Parametre

Coïncide avec un nom de paramètre indiqué dans l'instruction .MACRO lors de la définition de la macro correspondante.

L'expression spécifiée lors de l'appel de la macro sera évaluée dans le texte source au lieu du test du paramètre.

Remarques :



a) Plusieurs paramètres de communication de ce type peuvent être employés dans des expressions à calculer. Cependant, comme la syntaxe '\Parametre', lors de la définition de la macro, ne comporte encore aucune indication sur l'expression qui devra effectivement être traitée ultérieurement, lors de l'appel de la macro, le programmeur doit veiller à ne pas imbriquer ou combiner des modes d'adressage différents.

Exemple 2.3.1.1. : correct

```
.MACRO Exemple_1 Parametre_1,Parametre_2
move.w    \Parametre_1+2(a0,d0.l),\Parametre_2
.ENDM
Exemple_1 4,(a1)+                ; Appel
```

produira : move.w 6(a0,d0.l),(a1)+

Exemple 2.3.1.2 : incorrect !

```
.MACRO Exemple_2 Parametre
move.w      #\Parametre,d0
move.w      d1,\Parametre
.ENDM
Exemple_2   #8              ; Appel
```

produira :

```
move  ##8,d0        ; incorrect car "double" adressage
move  d1,#8         ; incorrect car adressage illégal
```

Lors de la transmission des paramètres de macro, il convient donc de tenir compte de l'emploi qui en sera fait dans la macro appelée. Les erreurs seront interceptées par des messages appropriés.

b) Le programmeur n'est pas obligé d'utiliser effectivement lors de l'appel toutes les variables de communication prévues lors de la définition de la macro.

Il ne faut, par contre, pas transmettre plus d'expressions qu'il n'avait été prévu lors de la définition car cela risquerait d'entraîner des pertes de données indésirables.

Exemple 2.3.1.3. :

```
.MACRO Exemple_3 Parm_1,Parm_2,Parm_3,Parm_4
; Corps du macro
.ENDM

; Appels
Exemple_3 ABC,DEF,GHI,JKL     ; Tous les paramètres sont définis
Exemple_3 ,,GHI               ; Seul Parm_3 est défini
Exemple_3 ABC,,JKL            ; Seuls Parm_1 et Parm_4 sont définis
Exemple_3 ABC,,GHI,JKL,MNO    ; Trop de paramètres
```

c) En ce qui concerne l'imbrication des appels de macros (exemple 2.0), il convient de noter que le seul critère de recherche des paramètres est leur nom.

➪ Si par exemple une Macro_A est appelée à partir d'une Macro_B et qu'un paramètre Parm_1 est transmis à la Macro_B, ce paramètre sera accessible aussi bien à la Macro_A qu'à la Macro_B sous le nom spécifié lors de la définition de Macro_B.

Exemple 2.3.1.4 :

Soient les définitions suivantes :

```
        .MACRO Macro_A Parm_2
        move.w      \Parm_1,\Parm_2
        .ENDM
et :    .MACRO Macro_B Parm_1,Parm_3
        ; ...
        Macro_A Parm_3
        ; ...
        .ENDM
```

```
        L'appel de Macro_B : Makro_B d0,d1
        donnera :   ; ...
        move.w      d0,d1
        ; ...
```

Macro_A tire 'd0' directement des paramètres transmis à Macro_B alors que 'd1', par contre, est d'abord transmis par Macro_B à Parm_2 de Makro_A.

➥ Si un paramètre de communication portant le même nom de Parm_1 est défini aussi bien dans Macro_A que dans Macro_B, seul le paramètre de la macro elle-même est accessible de l'intérieur de la macro (de Macro_A en l'occurrence).

➥ *En résumé :* lorsqu'un paramètre de communication est réclamé, l'assembleur le compare d'abord aux noms spécifiés dans la macro appelée le plus à l'intérieur d'une imbrication (ici la Macro_A). S'il ne trouve pas ce nom dans cette macro, il cherche chaque fois dans le niveau immédiatement au-dessus (ici la Macro_B) jusqu'à ce

qu'il trouve le nom voulu ou atteigne à nouveau le niveau normal du texte source.

\n

- n peut revêtir des valeurs de 0 à 16 et doit suivre immédiatement le trait transversal vers l'arrière. Les valeurs 0 et 10 ont le même effet.

- On peut se référer aux paramètres transmis aux macros non seulement à l'aide de leurs noms mais aussi à l'aide de leur position dans la liste des paramètres définis. Dans ce cas, la recherche automatique de paramètres homonymes y compris dans des niveaux d'imbrication plus élevés n'a pas lieu.

Exemple 2.3.2. :

```
.MACRO Comparaison Source,Destination
; ...
cmp.b       \1,\2
seq         Flag
; ...
.ENDM
Comparaison (a0)+,d0

donnera :  ; ...
cmp.b (a0)+,d0
; ...
```

\#

Fournit comme valeur de réponse le nombre de paramètres effectivement transmis à une macro lors de l'appel des paramètres transmis. Peut être utilisé comme une expression numérique dans tous les calculs.

\?Parametre

Détermine si un paramètre déterminé a été transmis lors de l'appel.

Réponse :	true (=-1) si le paramètre a été défini false (=0) si le paramètre n'a pas été défini

\?n

➥ n peut revêtir des valeurs de 0 à 16 et doit suivre immédiatement le trait transversal vers l'arrière. Les valeurs 0 et 10 ont le même effet.

➥ Semblable à : \?Parametre, si ce n'est qu'il faut utiliser le numéro de paramètre au lieu du nom de paramètre. Ainsi peuvent être testées même des valeurs transmises sans être dotées d'un nom.

\!Parametre

Détermine le mode d'adressage d'une expression transmise. Les nombres renvoyés comme valeurs de réponse représentent les types suivants :

Modes d'adressage :

1 :	Dn	6 :	d(An)	12 :	#immédiat
2 :	An	7 :	d(An,Xm)	13 :	SR
3 :	(An)	9 :	absolu	14 :	CCR
4 :	(An)+	10 :	d(PC)	15 :	USP
5 :	-(An)	11 :	d(PC,Xm)		

1 :	Paramètre non transmis
8 :	Non encore défini (peut-être court absolu)

\!n

➥ n peut revêtir des valeurs de 0 à 16 et doit suivre immédiatement le trait transversal vers l'arrière. Les valeurs 0 et 10 ont le même effet.

➥ voyez \!Parametre.

\'Parametre

Test du type de données.

Valeurs de réponse :	1 :	Paramètre non transmis
	0 :	Le paramètre est une valeur numérique
	>0 :	Le paramètre est une chaîne de caractères, la valeur renvoyée est la longueur de la chaîne en octets

\'n

➥ n peut revêtir des valeurs de 0 à 16 et doit suivre immédiatement le trait transversal vers l'arrière. Les valeurs 0 et 10 ont le même effet.

➥ voyez \'Parametre

.\~{Nom} - Définition de label spécifique aux macros.

Lorsque l'utilisateur veut appeler une macro déterminée plusieurs fois dans un programme, des problèmes de redéfinition peuvent se poser si des symboles sont définis dans cette macro. Ces problèmes sont résolus grâce à une légère modification automatique du nom du symbole lors de chaque nouvel appel de la macro.

L'expression à côté de l'instruction se comporte comme un symbole (la seconde version est locale).

Lors de l'assemblage, '\~' sera converti en ."M"+Str$ (numéro d'appel actuel de la macro){+"Nom"}.

Peu importe à cet égard qu'on se réfère au symbole à l'aide de l'opérateur placé devant, ou en désignant le nom explicitement. (Il faut, dans ce cas, déterminer le numéro d'appel en comptant les appels de la macro. Ce mode d'accès est, toutefois, déconseillé.)

Pour plus de détails sur l'emploi des symboles, nous vous invitons à vous reporter au chapitre approprié !

Remarques :



La saisie tokenisée avec contrôle de la syntaxe par l'éditeur contraint à ne définir que des expressions complètes et de syntaxe correcte. Il en résulte cependant une trop grande limitation des possibilités de la programmations avec macros.

Exemple :

(a) La construction suivante ne serait pas possible :

```
            .MACRO Loop Registre_1,Registre_2,Registre_3
.\~Loop:    move.b     (\Registre_1)+,(Registre_2)+
            dbra       \Registre_3,.\~Loop
            .EMDM
```

L'appel " Loop a0,a1,d0 " devrait produire :

```
.M1Loop:    move.b     (a0)+,(a1)+
            dbra       d0,.M1Loop
```

mais la 2ème ligne de la définition de macro serait rejetée dès la saisie.

(b) Il ne serait pas possible de reconstituer des instructions :

```
        .MACRO Transfert Extension,Source,Dest
        move.\Extension  \Source
        .ENDM
```

L'éditeur rejetterait l'entrée de " move.\Extension ".

Pour que l'utilisateur n'ait pas à supporter les restrictions indiquées ci-dessus tout en disposant des avantages du contrôle de la syntaxe, la règle suivante a été adoptée sous GFA Assembleur.

Toutes les expressions, tous les modes d'adressage ou lignes d'instructions devant être composés à partir de différents frag-

ments et présentant une syntaxe encore incorrecte au moment de la saisie doivent être placés entre accolades { }.

A l'intérieur des accolades peut figurer un nombre illimité d'expressions (y compris donc des chaînes de caractères et des variables alphanumériques) ou de modes d'adressage, séparés par des virgules ou des espaces. Chacune de ces expressions et chacun de ces modes d'adressage sera soumis à un contrôle de syntaxe mais séparément et non en groupe.

Au cours de l'assemblage, les différents fragments seront juxtaposés et évalués. Les chaînes seront traitées sans les guillemets. C'est le contenu des variables alphanumériques qui sera sorti. Les paramètres de macro seront remplacés par les valeurs qui leur ont été affectées.

Quelques exemples et remarques pour illustrer ce principe :

(1) L'exemple (a) doit être formulé de la façon suivante sous GFA Assembleur :

```
      .MACRO Loop Registre_1,Registre_2,Registre_3
      .\~Loop:  move.b    {"(",\Regis-
tre_1,")+"},{"(",\Registre_2,")+"}
                dbra      \Registre_3,.\~Loop
                .ENDM
```

(2) Voici comment devrait être écrit l'exemple (b) :

```
      .MACRO Transfert Extension Source,Dest
      {"move.",\Extension,\Source,",",\Dest}
                .ENDM
```

(3) A l'intérieur des accolades, les expressions ne doivent obligatoirement être spécifiées comme chaînes de caractères que lorsqu'elles ne peuvent satisfaire au contrôle de syntaxe séparé :

```
{a,b,c,d,e} et {"a","b","c","de"} produiront toutes deux
abcde
```

```
move.w   {a,b,c,d,e},d0     donnera
move.w   abcde,d0
```

(4) Lorsqu'on a besoin des guillemets qui servent à délimiter les chaînes de caractères dans le corps des expressions qui seront produites lors de l'assemblage en clair, il faut les indiquer deux fois puisque les chaînes sont traitées sans les guillemets :

Exemple :

Pour composer l'instruction cmp.b #"A",d0 :

```
{"cmp.b ",'#"A",',d0}
```

(5) Les accolades peuvent être imbriquées, c'est-à-dire que les expressions apparaissant à l'intérieur des accolades peuvent elles-mêmes comporter des accolades. Il convient dans ce cas d'être prudent dans l'emploi des chaînes de caractères :

```
{'"d0"'}    donnera "d0"
{{'"d0"'}} donnera tout d'abord {"d0"}, mais d0 lors de
la deuxième étape !
```

(6) Les accolades peuvent aussi être placées dans des calculs et des adressages :

```
        .MACRO Magique Chiffre          |  produira :
        move.w #1+{1,\Chiffre},d0       |     move.w #1+15,d0
        .ENDM                           |
        Magique 5                       |
```

(7) Il convient de bien distinguer 2 types d'application fondamentaux :

```
{} à l'intérieur d'expressions (voyez l'exemple (1))
{} comme lignes d'instructions complètes (codes d'opéra-
tion avec paramètres, voyez l'exemple (2))
```

(8) Les accolades ne peuvent être employées qu'à l'intérieur de macros.

(9) Attention : l'usage intensif de l'assemblage de texte en clair ralentit nettement l'assemblage.

Définition de labels avec paramètres de macros :

A l'intérieur des macros peuvent être définis des symboles dont les noms peuvent être transmis comme paramètres de macros :

Exemple :

```
.MACRO DefinitionLabel Nom_1,Nom_2
\Nom_1:  nop                    ; \ écritures
\\Nom_2: nop                    ; / équivalentes
            .ENDM
           DefinitionLabel Symbole_1,Symbole_2
```

donnera :

```
Symbole_1: nop
Symbole_2: nop
```

Seuls peuvent cependant être transmis comme paramètres ceux auxquels un nom a été affecté lors de la définition de la macro.

Les définitions telles que \1: ne sont pas possibles.

Avertissement :



Certains assembleurs permettent de transmettre aux macros des fragments d'expression, d'instruction ou de mode d'adressage. Comme ces assembleurs ne travaillent pas avec un format de texte source tokenisé, il est ainsi possible, par simple copie de texte de composer à partir de tels fragments de nouvelles instructions, de nouveaux symboles, de nouveaux modes d'adressage etc.

C'est délibérément que le GFA Assembleur n'offre pas cette possibilité pour ne pas saper le principe d'un développement structuré des programmes.

2.1.2.4. Insertion de texte source supplémentaire

INCLUDE "NOM.EXT"
PATH "Chemin du fichier" ; spécification d'un chemin de recherche

Lorsqu'un texte source est chargé à la suite du texte principal à l'aide de l'instructions Include, ce texte supplémentaire est traité par l'assembleur exactement comme s'il figurait directement dans le texte source en cours de traitement au lieu de l'instruction .INCLUDE. (.INCLUDE peut être utilisée en n'importe quel endroit du texte source.)

Si un texte de ce nom figure déjà en mémoire (puisqu'un nombre illimité de textes peuvent être stockés simultanément), c'est ce texte qui sera utilisé. Si ce n'est pas le cas, le texte voulu sera recherché dans le chemin actuel.

L'utilisateur peut en outre indiquer jusqu'à 16 chemins de fichier dans lequel le texte voulu devra être recherché. Les chemins sont traités dans l'ordre dans lequel ils ont été spécifiés. C'est le premier fichier portant le nom voulu qui sera utilisé.

2.1.3. Assemblage conditionnel

Le principal champ d'application de l'assemblage conditionnel réside dans sa combinaison avec les macros. Cette technique permet, en effet, de produire des réactions différentes en fonction des paramètres transmis, voire de commander l'assemblage de façon très flexible à l'aide d'un petit nombre de valeurs initiales.

L'assemblage conditionnel est très simple à utiliser car il reprend la structure If-Else-Endif bien connue dans de nombreux langages évolués. 'Else' a un effet alternatif, c'est-à-dire que le texte source à la suite d'un double 'Else' sera à nouveau traité si 'true'(=-1).

Il existe aussi une variante à une ligne : **.IIF**

L'assemblage conditionnel peut être imbriqué sans restriction.

IF...ELSE...ENDIF

```
...
.IF Condition
    ; ...
    ; assembler si Cond.true(=-1)
    ; ...
.ELSE
    ; ...
    ; assembler si Cond.false(=0)
    ;...
.ELSE
    ; ...
    ; assembler si Cond.true(=-1)
    ; ...
...etc.
.ENDIF
```

.IIF Condition, Instructions...

Les instructions suivant la condition ne seront assemblées que si la valeur de vérité est vraie, true(=-1). Cette structure se termine à la fin de la ligne.

A titre d'exemple, voici deux exemples venant s'ajouter à 2.0 :

Exemple 3.1 :

Appel Gem-Dos avec correction de pile

```
MACRO GEMDOS Numero_Trap,Zone_Pile
move.w    #\Numero_Trap,-(sp)          ; N de fonction Gemdos
```

```
trap        #1                      ; Appel Trap
.IF \Zone_Pile<=8                   ; Si la pile ne peut pas
addq.l      #\Zone_Pile,sp          ; être corrigée de plus de
.ELSE                               ; 8 octets, une instruction
adda.l      #\Zone_Pile,sp          ; d'addition rapide peut
.ENDIF                              ; être employée
.END
```

Exemple 3.2 :

Déposer une chaîne et son adresse

```
   .MACRO Depot Chaine,Adresse,Segment
        .TEXT
        .IF \?Adresse               ; Si dépôt possible, indiquer
        move.l      #\~,\Adresse    ; comme adresse de chaîne,
        .ELSE                       ; sinon utiliser adresse
        pea     \~                  ; sur la pile
        .ENDIF
        .SECTION Segment,DATA       ; Déposer chaîne dans segment
\~:     DC.b        \Chaine,0       ; Data spécifié et marquer avec
        .TEXT                       ; symbole
        .ENDM
```

Exemple 3.3 :

Abandon conditionnel d'une macro

```
MACRO Increment Valeur,Dest     ; Macro à incrémenter
IIF !\?Dest,.EXITM              ; Sortie si aucune destination
addi.l   #\Valeur,\Dest         ; spécifiée
ENDM
```

En dehors de cette forme universelle de définition d'une condition, il existe aussi différentes variantes d'instruction dans lesquelles une condition donnée est automatiquement testée, le texte suivant étant traité si cette condition est vraie.

.IFEQ Expression ^**...ELSE ...ENDIF** ou
.IIFEQ Expression, ...

Expression=0.

.IFNE Expression **...ELSE ...ENDIF** ou
.IIFNE Expression, ...

Expression<>0.

.IFGT Expression **...ELSE ...ENDIF** ou
.IIFGT Expression, ...

Expression>0.

.IFGE Expression **...ELSE ...ENDIF** ou
.IIFGE Expression, ...

Expression>=0.

.IFLT Expression **...ELSE ...ENDIF** ou
.IIFLT Expression, ...

Expression<0.

.IFLE Expression **...ELSE ...ENDIF** ou
.IIFLE Expression, ...

Expression<=0.

Exemple 3.4 :

Transfert de bloc

```
.MACRO Transfert Debut,Dest,Longueur
.IFNE Longueur          ; Utiliser routine de transfert
lea.l    \Debut,a0      ; d'un bloc de mémoire
lea.l    \Dest,a1       ; Si le bloc à transférer a une
adda.l   \Longueur,a0   ; longueur de 0 octets, interrompre
adda.l   \Longueur,a1   ; l'assemblage avec un message
move.w\Longueur,d0      ; d'erreur. (.FAIL)
subqw.  #1,d0
```

```
.\~:    move.b (a0)+,(a1)+      ; (Boucle de copie)
        dbra   d0,.\~
    .   ELSE
        .FAIL
        .ENDIF
        .ENDM
```

.ASSERT Condition

➥ Interrompt l'assemblage avec un message d'erreur si 'Condition' est fausse, c'est-à-dire si elle vaut 0.

.FAIL

➥ Interrompt systématiquement l'assemblage (exemple 3.4).

Remarque :



Des programmeurs particulièrement "astucieux" pourraient essayer de distinguer les parcours d'assemblage (par exemple en calculant les différences de symbole dans les modes d'optimisation). Ces méthodes de programmation destructrices ne peuvent mener à rien sinon à perturber l'assembleur. Elles sont donc automatiquement détectées et interdites.

2.1.4. Assemblage répété

L'assemblage répété consiste à faire répéter plusieurs fois le traitement de parties du texte marquées. Cela réduit le travail d'écriture et améliore la clarté des programmes. Cette technique est intéressante pour les applications où une vitesse très élevée est exigée.

On accepte alors de consommer beaucoup plus de mémoire en répétant des sections du code programme identique pour éviter toute perte de temps d'exécution due à une boucle (dans le programme).

L'utilisateur peut en outre mettre en place une variable de comptage à l'aide de définitions de symbole et faire exécuter plusieurs fois des sections du texte source avec des valeurs chaque fois modifiées.

```
.Syntax:    .  REPT Expression
               ; ...
               ; Ici figure le texte à traiter plusieurs fois
               ; ...
               .ENDR
```

Le texte encadré par les directives .REPT et .ENDR sera assemblé autant de fois que spécifié par 'Expression'.

Exemple 4.1 :

Effacer rapidement un bloc dans la programmation graphique avec motif de bits (16 longs mots).

Forme répétée :	Variante : (avec variable de comptage : INDEX)	Forme habituelle :
lea.l Basis,a0	lea.l Basis,a0	lea.l Basis,a0
.REPT 16	INDEX set 0	clr.l (a0)+
clr.l (a0)+	.REPT 16	clr.l (a0)+
.ENDR	clr.l INDEX*4(a0)	.. en tout 16x
	INDEX = INDEX+1	clr.l (a0)+
	.ENDR	clr.l (a0)+

Exemple 4.2 :

Initialisation d'un bloc de mémoire avec les nombres 0 à 255

Boucle :		Habituel :	
	.DATA		.DATA
	Nombre = 0		.DC.b 0,1,2,3,...
	.REPT 256		...
	.DC.b Nombre		.DC.b ...,253,254,255
	Nombre = Nombre+1		
	.ENDR		

Remarque :



Lorsque des symboles sont définis dans une boucle REPT, cela peut entraîner des problèmes de redéfinition. La solution consiste à utiliser de façon rigoureuse les symboles locaux et l'instruction .LOCAL au début ou à la fin de chaque parcours de boucle. (Pour plus de détails, veuillez vous reporter au chapitre 2.2.1. sur les symboles !)

2.1.5. Instructions d'initialisation de la mémoire

2.1.5.1. Dans TEXT et le segment de données initialisé (.DATA)

Il existe, en programmation assembleur, des applications nombreuses et variées pour les initialisations de la mémoire, par exemple l'insertion de textes, tables de saut, motifs de bits etc. dans le segment de données ou encore le stockage de données spéciales dans le segment de texte pour des programmes se modifiant eux-mêmes ou les codes opération LINEA.

.DC{.x} Expression{,Expression}{,Expression}...

.x indique la largeur de traitement : .b : octet, .w : mot, .l : long-mot. 'Expression' est une valeur numérique quelconque qui sera insérée une fois, avec la largeur spécifiée, dans la position actuelle du code objet (compteur de programme). Si la taille des données est 'octet', les chaînes de caractères sont également autorisées. Elles seront stockées lettre par lettre.

Exemple 5.1 :

Stocker une chaîne de caractères dans la mémoire.

```
.DC.b "Salut !"          ;équivaut à
.DC.b $53,$61,$6c,$75,$74,$20,$21
```

.DCB{.x} Nombre, Modele de remplissage

L'instruction .DCB exécute l'instruction '.DC{.x} Modele de remplissage' autant de fois que spécifié par 'Nombre'.

Exemple 5.2 :

Initialisation d'une zone de mémoire avec '-1'

```
.DCB.w 5,-1              ;équivaut à
.DC.w -1,-1,-1,-1,-1
```

.INIT{.W} {Nmb_1,} Modele_1{.l} {,Modele_2{.w}} {,#Nmb_2,Modele_3{.b}}...

L'instruction .INIT représente la forme la plus souple d'initialisation de la mémoire. #Nmb_1 ou #Nmb_2 indique, chaque fois, combien de fois les valeurs inscrites jusqu'à la prochaine spécification doivent être placées dans la position actuelle, avec la largeur de données chaque fois spécifiée.

(La largeur de données par défaut est .w, la largeur de données indiquée directement à la suite de l'instruction .INIT devient la largeur par défaut pour tous les autres modèles de remplissage.) Cette instruction est particulièrement utile lorsqu'il s'agit de définir de longues structures avec différents types et largeurs de données.

Exemple 5.3 :

```
.INIT.w -1.l,0.w,#8,"Chaîne".b,#3,"A".b,"B".b
; équivaut à :
.DC.l -1
.DC.w 0
.DCB.b 8,"Chaîne"
.DC.b "A","B","A","B","A","B"
```

2.1.5.2. Dans le segment de données non initialisé (.BSS) et les structures .ABS

(Pour plus de détails sur ces segments, veuillez vous reporter au chapitre 2.1.1. sur la structure du programme !)

.DS{.x} Nombre

Le compteur de programme est incrémenté de façon à réserver, à partir de l'adresse actuelle, la place nécessaire pour un nombre égal à 'Nombre' de données de la largeur spécifiée par l'extension (x).

Exemple 5.4 :

```
.DS.w 30        ; réserve 60 octets
.DS.l 2         ; réserve 2 octets
```

2.1.6. Instructions de gestion de listing (Protocoles d'assemblage)

Pour rechercher les erreurs dans les programmes, il est souvent très utile d'avoir une trace écrite de tout ou partie des opérations d'assemblage. On peut ainsi suivre en détail le déroulement des sections du programme les plus délicates mais c'est aussi le moyen, lorsqu'on recourt systématiquement à la programmations de macros, d'obtenir un listing complet du code effectivement produit, pour le débogage.

Un protocole d'assemblage peut être envoyé sur trois périphériques de sortie : l'écran, l'imprimante et un fichier (lecteur de disquette, disque dur ; extension .LST). Des directives spéciales placées dans le texte source permettent à l'utilisateur de sélectionner les passages qu'il sera le plus intéressant pour lui de pouvoir suivre à la loupe :

.LIST

Active le protocole, toutes les opérations suivantes de l'assembleur seront retracées comme dans l'exemple 6.1.

.NOLIST

Désactive le protocole.

.TTL "En-tête"
.SUBTTL "Bas de page"

Chaque page de protocole présente la structure suivante :

```
Ligne titre système (Action,Fichier,Date,Page)
{En-tête} (pouvant être librement défini par l'utilisateur)
_____Ligne vide_____
...
Listing
...
_____Ligne vide_____
{Bas de page} (pouvant être librement défini par l'utilisateur)

La définition d'un en-tête ou d'un bas de page s'applique pour toutes les
pages suivantes. Après une première définition d'en-tête et de bas de
page, toute autre définition entraîne en outre un saut de page.
```

.PLEN {Expression}

Fixe la longueur d'une page de protocole.

.LLEN {Expression}

Fixe la longueur de ligne.

Les longueurs de page et de ligne ne peuvent être modifiées que lorsque le protocole n'est pas en cours ! (par exemple avant .LIST ou après .NOLIST)

Remarques :



Spécification du format de page

- Sur chaque page de protocole, 5 lignes sont réservées pour l'en-tête et le bas de page. Toutes les lignes restantes sont employées pour la sortie du listing. Si aucun en-tête ni bas de page n'est spécifié, une ligne vide supplémentaire les remplace (voyez TTL, SUBTTL).

- Si l'utilisateur ne spécifie aucun format de page dans le texte source, l'assembleur reprend automatiquement les valeurs correspondantes de l'écran de paramètres d'impression (Menu 5,1) (pour la sortie dans un fichier ou sur l'imprimante). Si un en-tête est activé dans ce menu, le nombre de lignes s'en trouve réduit de 3. Le nombre effectif de caractères (colonnes) par ligne est obtenu en soustrayant la marge gauche ainsi que l'éventuelle numérotation de ligne (voyez le mode d'emploi de l'éditeur !). Si ces valeurs sont inférieures aux valeurs minimales de 80 caractères par ligne et de 6 lignes par page, un format par défaut de 80 caractères sur 58 lignes est employé.

- Si la sortie s'effectue sur l'écran, le format de page est toujours de 80 caractères sur 25 lignes (une page écran). Une touche doit être frappée à la suite de chaque page sortie, pour permettre à l'utilisateur de lire à son aise.

➥ Pour l'impression, chaque page est construite dans un buffer de page avant d'être imprimée. La taille de ce buffer de page découle du réglage dans le menu d'impression de l'éditeur. Elle se calcule ainsi :

Colonnes * (Lignes-En-tête) * (Caractères+Marge)		
avec :	Colonnes :	Nombre de colonnes
	Lignes :	Lignes par page
	En-tête :	0 si pas d'en-tête
		3 si en-tête automatique ou auto-défini
	Caractères :	Caractères par colonne

Au cours de l'assemblage, cette taille de buffer ne doit pas être dépassée à la suite d'une modification du format de page. Si, toutefois, c'est tout de même le cas, chaque page sera automatiquement réduite du nombre de lignes nécessaire pour que le buffer suffise.

➥ Lorsqu'on dispose de suffisamment de place mémoire pour cela, il est recommandé, pour la sortie d'un protocole imprimé, d'installer un spooler suffisamment grand, pour ne pas freiner l'assemblage inutilement.

➥ Pour la sortie sur imprimante, il est théoriquement possible d'obtenir une sortie du listing d'assemblage sur plusieurs colonnes. Comme chaque colonne doit avoir une largeur d'au moins 80 caractères, peu d'imprimantes permettent d'obtenir une largeur suffisante pour au moins deux colonnes.

➥ L'impression obéit, pour le reste, aux règles (adaptation de l'imprimante, etc...) indiquées dans la partie de ce manuel consacrée à l'éditeur.

➥ Ne confondez pas l'en-tête du protocole avec celui du menu d'impression. L'en-tête du menu d'impression n'est pas imprimé sur les protocoles !

.LMODE Expression 1 <= Expression <= 3 !

Comme dans l'exemple 6.1, un protocole d'assemblage est formaté orienté colonnes. Si le contenu d'une colonne dépasse la longueur réservée à cet effet, le reste peut être traité de différentes manières :

1 : Sortie sur la ligne suivante sur la position de colonne correspondante.

2 : Tronquer.

3 : Continuer à écrire dans la colonne suivante. Toutes les colonnes suivantes sur cette ligne se trouveront alors décalées sur la droite.

.PRINTER Expression {,Expression} ...

➥ Des chaînes de caractères peuvent être spécifiées ici, comme avec l'instruction DC.b. Si cette instruction est donnée alors même qu'un protocole est en cours d'impression, les caractères spécifiés sont envoyés à l'imprimante et terminés par carriage return, line feed. Il est donc possible, de cette façon, d'envoyer des instructions à l'imprimante, par exemple pour obtenir une impression comprimée, d'autres styles etc.

Exemple :

```
.PRINTER 27,15 ; Ecriture comprimée
```

.PAGE {Expression}

Sans paramètre : saut de page.

Avec paramètre : saut de page s'il reste moins de 'Expression' lignes libres sur la page actuelle (sans compter le bas de page).

.SPACE {Expression}

Insère 'Expression' lignes vides.

Dans la version sans paramètre, 1 fait office de valeur défaut.

.MLIST
.NOMLIST

.MLIST : Les appels de macro sont listés in extenso.

NOMLIST : Pour les macros, seul l'appel sera affiché, pas le contenu.

.CLIST
.NOCLIST

.CLIST : Pour l'assemblage conditionnel, les sections sautées à cause de conditions non remplies sont malgré tout affichées.

.NOCLIST : Seules les instructions effectivement exécutées sont affichées dans les structures .IF (true).

Exemple 6.1 :

L'impression de protocole suivante a été réalisée lors de l'assemblage du programme présenté dans l'exemple 2.0. Un format de 80 caractères sur 20 lignes avait été fixé.

Colonne 1

contient le numéro de ligne de l'instruction dans le texte source. Les caractères de statut suivants peuvent figurer immédiatement à la suite :

.	Ligne venant d'un appel de macro
:	Ligne venant d'un fichier Include
+	Pour l'assemblage conditionnel : true
-	Pour l'assemblage conditionnel : false

Colonne 2

indique l'adresse du code objet (par rapport à l'adresse de base 0). Tenez compte des changements de type de segment. Dans les programmes, le segment de données commence à la fin du segment de texte, le segment BSS à la fin du segment de données. Dans les modules objet, tous les segments ont 0 comme adresse de base.

Colonne 3

reproduit, en format hexadécimal, le code assembleur généré.

Colonne 4

est réservée aux définitions de symboles.

Colonne 5

affiche en clair l'instruction exécutée.

Colonne 6

contient le commentaire correspondant.

```
{Ligne d'état système : Explication                                    }
{Produit avec :  Fonction :  Chemin :  Date :   Heure : Page}

GFA-Ass V1.0 Ass-Prot.:D:\PRINT.LST Jour :6.2.1986 Heure :2:9:0
S:1
Ceci est l'en-tête.

6     000000                        .TEXT
8     000000                        .MACRO bconout dev,char   ;BIOS
18    000000                        .MACRO Print Chaine,Device;Macro

34    000000                        .LMODE 2
35    000000                       Print "Salut !",2          ;Ecrit
19    000000                        .DATA
20  . 00003c                        .IF \?Chaine              ;
21  .+00003c 48616c6c6f21  \~a      .DC.b \Chaine             ;
23  .+000042               \~b      .ELSE
24  .-000042               \~a      .DC.b 13,10               ;
26  . 000042               \~b      .ENDIF

Ici figure le bas de page.
{form feed}
GFA-Ass V1.0 Ass-Prot.:D:\PRINT.LST Jour :6.2.1986 Heure :2:9:0
S:2
Ceci est l'en-tête.

28  . 000042                        .TEXT
29  . 000000 47f9 0000003c          lea.l    \~a,a3           ;
30  . 000006 7605                   moveq.l #\~b-\~a-1,d3     ;
31  . 000008               .\~      bconout \Device,(a3)+     ;
9   . 000008 4240                   clr.w    d0               ;
10  . 00000a 101b                   move.b   \char,d0         ;
11  . 00000c 3f00                   move.w   d0,-(a7)         ;
12  . 00000e 3f3c 0002              move.w   #\dev,-(a7)      ;
13  . 000012 3f3c 0003              move.w   #3,-(a7)         ;
14  . 000016 4e4d                   trap     #13              ;
15  . 000018 5c8f                   addq.l   #6,a7            ;
16  . 00001a                        .ENDM
32  . 00001a 51cb ffec              dbra     d3,.\~           ;
33  . 00001e                        .ENDM
36    00001e                        .SPC 3

Ici figure le bas de page.
{form feed}
```

2.2. Calculs, symboles, Operateurs

2.2.1. Définition de symboles

L'assembleur distingue entre les symboles relogeables et les symboles absolus.

Sont dits relogeables les symboles qui se réfèrent à une position par rapport au début du programme, par exemple les adresses de saut dans le texte du programme. Les symboles ab-

solus peuvent revêtir n'importe quelles valeurs et sont comparables aux variables entières sur 32 bits du BASIC. C'est pourquoi sont spécifiés lors de la définition d'un symbole sa valeur, mais aussi son type, son appartenance à un segment etc... Les symboles ont 16 caractères de longueur au maximum et peuvent être définis fondamentalement de deux façons différentes :

➥ En marquant des labels dans le texte du programme. Ces symboles sont toujours relogeables. C'est pourquoi leurs valeurs correspondent à des offsets par rapport au début du programme ou du segment (en cas de réalisation d'un module-O).

➥ Par affectation directe avec la syntaxe suivante :

```
Nom du symbole {Instruction d'affectation} Valeur

Les instructions d'affectation sont :
                equ, .equ , ==        (définition unique)
ainsi que :
                set, .set , .=        (définition répétée)
et :
                reg, .reg             (définition de liste de registres)
```

Les symboles ainsi définis portent comme attribut, l'attribut de l'expression affectée.

Exemple :

```
Symbole1 equ Symbole2
Si Symbole2 est relogeable, Symbole1 le sera aussi.
```

Les symboles définis comme des labels peuvent avoir deux "portées" différentes.

➥ Les symboles locaux sont précédés d'un point. Ils ne seront reconnus qu'entre deux symboles non locaux. Les programmeurs peu imaginatifs n'ont donc pas à se fatiguer l'esprit et les nerfs pour inventer chaque fois de nouveaux noms de symbole, ce qui est surtout difficile lorsqu'on développe de longs textes de programme (Des

zones locales peuvent être délimitées à l'aide de la directive .LOCAL même en l'absence d'une définition de label non local !). Dans la version de l'assembleur qui vous est proposée, même les symboles peuvent pourtant être exportés !

➥ Les symboles normaux sont reconnus dans l'ensemble du texte du programme. La méthode suivante constitue par exemple une combinaison judicieuse des deux types de symboles :

➥ Les points d'entrée dans les sous-routines sont désignés par des labels normaux alors que les symboles à l'intérieur de la sous-routine sont systématiquement déclarés comme locaux et, si nécessaire, encore séparés avec .LOCAL.

➥ Les symboles globaux sont exportés. Les déclarations globales ne sont par conséquent indispensables que pour le travail avec les fichiers O, bien qu'elles puissent être utilement employées dans d'autres situations.

➥ On peut décider, lors de l'appel de l'assembleur, si une table de symboles doit être liée au programme et si tous les symboles ou seulement les symboles globaux doivent y être placés. Lorsqu'on veut produire une table de symboles à des fins de débogage, on peut déclarer comme globaux tous les symboles qu'on souhaite y voir figurer et sélectionner ainsi les symboles admis dans la table.

Les symboles globaux peuvent être définis à l'aide des pseudo-codes opération appropriés ou avec '::'.

```
Exemple : Label1:     ; Label1 est défini comme normal
          Label2::    ; Label2 est défini comme global
```

2.2.2. Appel de symboles

Les symboles peuvent apparaître en n'importe quel endroit du texte du programme où des valeurs numériques seraient autorisées.

2.2.3. Les calculs

Les calculs en GFA Assembleur respectent la hiérarchie des opérateurs et admettent autant de niveaux de parenthèses que vous le souhaitez. Toute expression numérique peut être le résultat d'un calcul. Les opérateurs suivants sont admis :niveau de parenthèses

Les comparaisons logiques à résultat booléen (0 ou -1) (rang de priorité le plus faible dans les calculs) :

```
=     : égale (logiquement)
<>    : différent de
<     : inférieur à
>     : supérieur à
<=    : inférieur ou égal
>=    : supérieur ou égal
```

Opérateurs de calcul (classés par ordre de priorité croissant) :

```
P      |  +     :  Addition de valeurs 32 bits
r      |  -     :  Soustraction de valeurs 32 bits
i      |
o  c   |  *     :  Multiplication de deux valeurs 16 bits \
r  r   |  /     :  Division d'une valeur 32 bits par une valeur 16 bits
i  o   |
t  i   |        :  /    avec signe !
é  s   |  &     :  Combinaison AND sur 32 bits
   s   |  |     :  Combinaison OR sur 32 bits
   a   |  ^     :  Combinaison EOR sur 32 bits
   n   |
   t   |  %     :  Modulo
   e   |  <<    :  Décalage bit par bit sur la gauche
       V  >>    :  Décalage bit par bit sur la droite
```

Autres opérateurs : (précèdent les expressions concernées)

-	:	signe négatif (complément à 2)
~	:	NOT bit par bit (complément à 1)
!	:	NOT logique (le résultat est 0 ou -1)

Fonctions spéciales :

*	:	PC actuel
^^defined Symbol	:	Symbole défini ? *)
^^referenced Symbol	:	Symbole appelé ? *)
^^streq chaine1,chaine2	:	Comparaison de chaînes
^^date	:	Date système actuelle en format GEMDOS
^^time	:	Heure système actuelle en format GEMDOS
^^macdef NomMacro	:	Macro défini ?

Remarque :



Dans un assembleur à plusieurs parcours, il faut absolument que le pseudo-code opération .IF débouche toujours sur les mêmes décisions (voyez le chapitre correspondant. C'est pourquoi ne sont considérés comme définis, au sens de '^^defined', que les symboles fixés avant un test de ce type. Pour des raisons similaires, le test '^^referenced' ne fonctionne qu'avec les symboles déjà définis avant d'être appelés. La fonction '^^referenced' ne peut donc être utilisée que dans des limites assez étroites. Elle a été intégrée dans le jeu d'instructions essentiellement par souci de compatibilité.

Il nous faut d'ailleurs ici vous rappeler l'existence des opérateurs de macros qui peuvent être utilisés dans tous les calculs à l'intérieur des macros.

Pour reloger des résultats de calculs :

Les calculs de symboles relogeables sont soumis à la règle suivante dès lors que des opérations de calcul autres que '+' et '-' sont effectuées, la valeur du symbole devient absolue !

Si seuls '+' et '-' sont utilisés, le résultat sera également relogé s'il y a un nombre impair de symboles relogeables.

23. Variables alphanumériques

Les variables alphanumériques existent également dans la plupart des langages de programmation évolués. Dans la programmation en assembleur, elles permettent, surtout combinées avec les macros et les nouvelles fonctions { }, de paramétrer très largement les textes source.

Les 5 principes suivants suffisent en fait à en décrire le mode d'emploi :

- ➥ Définition et utilisation comme pour les symboles absolus définis avec .SET.

- ➥ Les variables alphanumériques se comportent à tout point de vue comme les chaînes de caractères spécifiées explicitement avec des guillemets. Elles peuvent être employées partout où des chaînes de caractères explicites pourraient également figurer.

- ➥ Les variables alphanumériques ne sont identifiées que durant l'opération d'assemblage. Elles ne peuvent donc être ni importées ni exportées lors du linkage.

- ➥ Pour être distinguées des symboles numériques habituels, elles commencent par un double point (:).

- ➥ Les chaînes de caractères peuvent comporter au maximum 255 caractères.

Syntaxe de la définition :

```
:Nom ExpressionChaine_1,ExpressionChaine_2 ...
```

Sous :Nom seront sauvegardées toutes les chaînes de caractères suivantes (reliées pour former une chaîne de caractères unique).

Exemples :

(a)

```
:Chaine_1 "est"
:Chaine_2 "Ceci ",:Chaine_1," un exemple de chaîne de
caractères"
```

DC.b :Chaine_2 équivaudra alors à

```
DC.b "Ceci est un exemple de chaîne de caractères"
```

(b)

```
:Character "A"
```

=> cmp.b #:Character,d0 équivaudra à

```
cmp.b #"A",d0
```

Les variables alphanumériques peuvent également être employées à l'intérieur des accolades. Remarquez bien, toutefois, que dans ce cas, ce n'est pas la variable elle-même mais son contenu qui sera appelé !

2.4. Fonctions de chaînes de caractères

Les fonctions suivantes, qui vous rappelleront certainement celles du BASIC, vous facilitent la manipulation des variables alphanumériques :

SYNTAXE :		Description :	Exemple :
mid{Chaine,x,y}	(1)	Extraire y caractères à partir du x-ième caractère de Chaine	mid{"123456",2,3} donnera "234"
left {"Chaine,x}	(1)	Extraire de chaine les x caractères les plus à gauche	left{"123456",4} donnera "1234"
right{Chaine,x}	(1)	Extraire de Chaine les x caractères les plus à droite	right{"123456",4} donnera "3456"
dup{Chaine,n}	(1)	Dupliquer n fois Chaine pour	dup{"123",4} donnera

```
                          |former une nouvelle chaine    |"123123123123"
                          |                              |
len{Chaine_1,..,Chaine_n}|Donnera la somme des longueurs|len{"1234","abc"}
                       (2)|des chaines indiquées         |donnera 7
                          |                              |
asc{Chaine}            (2)|Donnera le code ASCII du 1er  |asc{"ABC"}
                          |caractère de Chaine           |donnera 65
                          |                              |
chr{Expression}        (1)|Donnera une chaine composée   |chr{65}
                          |du caractère ayant Expression |donnera "A"
                          |comme code ASCII              |
                          |                              |
instr{Chaine_1,Chaine_2} |Recherche Chaine_1 dans       |instr{"123456","345"}
                       (2)|Chaine_2                      |donnera 3
                          |Réponse :- 0_pas trouvé       |
                          |          - n,n > 0           |
                          |            Chaine_2 commence |
                          |            en n-ième position|
                          |            de la Chaine_1    |
-------------------------------------------------------------------------
```

(1) Le résultat est une chaîne de caractères
(2) Le résultat est une valeur numérique

Si le résultat d'une fonction de chaîne de caractères est à son tour une chaîne de caractères, la fonction correspondante peut être utilisée comme une variable alphanumérique, partout où les chaînes de caractères sont autorisées.

Attention



Les accolades apparaissant dans les fonctions de chaînes de caractères ne doivent pas être confondues avec celles de l'assemblage du texte en clair.

Instruction I L L E G A L :

L'assembleur traite l'instruction "illegal" comme une macro. Ce n'est pas une erreur mais un choix délibéré pour permettre à l'utilisateur de choisir librement le code d'opération illégal qu'il souhaite. Si vous avez besoin d'une instruction "illegal", vous pouvez donc aisément la définir vous-même :

```
                          .MACRO illegal
                          .DC.w $4afc   ; ou tout autre
code illégal
                          .ENDM
```

Chapitre 3

Le linker

Le linker joue un rôle capital pour le travail en liaison avec des langages évolués ainsi que pour le développement de programmes de grande envergure. Des modules d'objet de même format peuvent être combinés, indépendamment de leurs origines respectives, pour former de nouveaux programmes. Des fonctions complexes peuvent être reprises de bibliothèques de routines (par exemple du paquet de développement). Le programmeur n'est pas obligé de "réinventer le fil à couper le beurre" lors du développement de chaque programme.

Par souci de compatibilité, c'est le format de linker du C de Digital Research (paquet de développement) qui a été utilisé. Ce format impose certaines restrictions dont le programme doit tenir compte :

- Aucun calcul ne peut être effectué lors du linkage, c'est-à-dire que lors de la réalisation du module objet, il ne doit pas y avoir plus d'un symbole importé dans une expression donnée.

- Les valeurs des symboles ne peuvent plus être modifiées pendant le linkage.

- La distinction entre 64 segments opérée par l'assembleur ne peut plus être supportée lors du linkage. On revient donc aux seuls segments TEXT, DATA et BSS habituels (voyez plus haut).

Pendant une opération de linkage sont affichées les fenêtres de message et d'avertissement déjà connues sous l'assembleur. L'utilisateur peut ainsi suivre le travail du linker et il est, notamment, informé de l'intégration des divers modules tirés des bibliothèques. Si l'assemblage est opéré au cours du linkage,

tous les messages de l'assembleur apparaissent également dans les fenêtres du linker.

Pour distinguer les informations de l'assembleur de celles du linker, des caractères spéciaux sont placés devant les textes de message :

, >	avant des messages de l'assembleur
= , =	avant des messages du linker
!	avant des instructions pour l'utilisateur
->	avant l'affichage de la durée de l'assemblage ou du linkage

Pour automatiser le plus largement possible les opérations d'assemblage et de linkage et pour éviter à l'utilisateur la corvée d'avoir à taper les mêmes instructions lors de chaque appel du linker, l'opération est commandée indirectement, à travers un fichier de contrôle du linker. Il s'agit d'un fichier ASCII ordinaire, qui doit donc être créé sous le mode de texte source de l'éditeur. Sur chaque ligne peut figurer une des instructions décrites ci-dessous.

Création d'un fichier de contrôle

➥ Création comme fichier de contrôle de linkage d'un texte ASCII sous le mode 'Source général' de l'éditeur, sous un nom quelconque. Ce texte doit comporter les instructions du linker décrites dans le manuel.

➥ Toutes les fonctions de l'éditeur, y compris donc la sauvegarde, le chargement, etc. peuvent être appliquées aux fichiers de contrôle du linkage.

➥ Le menu 6,4 doit être appelé pour lancer le linker. Par sécurité, le menu de linkage n'est affiché que si un texte ASCII (c'est-à-dire un texte non tokenisé) se trouve en mémoire. Les commutateurs que comporte ce menu ont alors les fonctions suivantes :

"créer :" - programme	le produit du linkage sera un programme.
- code objet	le produit du linkage sera à son tour un module objet, équivaut à l'instruction de fichier de contrôle "-p" (linkage partiel)
- fichier IMG	surveillance des relogements (voyez le manuel)
- absolu	commutateur d'activation pour l'instruction "-a".

Attention :



L'instruction "-a" n'est exécutée que si ce bouton est également activé. Il est ainsi possible, sans modifier le fichier de contrôle, de produire aussi bien un programme exécutable qu'un code absolu.

"Table de symboles :"	voyez les instructions "-l" et "-s" dans le fichier de contrôle du linkage.
"Load Map :"	voyez l'instruction "-m" dans le fichier de contrôle du linkage.
"Linker"	lance l'opération de linkage.

Instruction :

(Chemin est à remplacer par un chemin de fichier quelconque !)

Chemin

➥ Intégration d'un module objet

?Chemin

➥ Intégration d'une bibliothèque (Archive). Lorsqu'une bibliothèque (composée de différents modules objet) est linkée avec le programme, le linker détermine quelles références ne sont pas encore résolues pour le moment (c'est-à-dire quels symboles sont importés sans être exportés à partir d'un autre module du programme et ne sont, par conséquent, pas encore définis).

Tous les modules de la bibliothèque sont ensuite examinés pour y rechercher l'exportation de ces symboles manquants. Chaque fois que le linker trouve un de ces symboles, le module qui l'exporte est intégré.

Remarques :



➥ L'analyse de la structure interne d'une bibliothèque très complète prendrait beaucoup trop de temps au cours du linkage. C'est pourquoi une bibliothèque est traitée en liaison avec son 'fichier d'index'. Le fichier d'index d'une bibliothèque contient toutes les informations nécessaires au linker pour traiter cette bibliothèque. Ce fichier d'index (extension .NDX) est créé la première fois qu'une bibliothèque est linkée. Il devient ainsi immédiatement disponible pour tout autre travail ultérieur (Le fichier d'index peut être consulté à travers la gestion de bibliothèque et il peut aussi être produit séparément). L'annexe vous fournit plus de détails sur la structure et

le fonctionnement des fichiers d'index (compatibilité avec le standard de développement ATARI !).

➥ Lorsqu'un symbole est exporté par plusieurs modules d'une bibliothèque, c'est le dernier de ces modules qui est repris.

➥ Si, par contre, plusieurs modules (ne faisant pas partie d'une bibliothèque) exportent un symbole de même nom, c'est la première de ces exportations qui est transmise à d'autres modules.

@Chemin

➥ Intégration de tous les modules d'une bibliothèque (dans l'ordre alphabétique de leurs noms).

#Chemin

➥ Intégration d'un fichier de données binaires. Le fichier spécifié est repris tel quel dans le segment DATA. Pour pouvoir accéder à ces données à l'aide de symboles, il est donc conseillé de définir des symboles dans le module objet censé précéder ce bloc de données, à la fin du segment de données.

➥ Exemple d'application : Intégration de textes, de dessins avec motifs de bits, de Resources etc.

^Chemin

➥ Le fichier spécifié sera assemblé pendant le linkage. Aucun fichier objet ne sera envoyé sur le lecteur de disquette ou le disque dur car le code objet produit sera directement linké dans la mémoire avec les autres modules. Cette possibilité est surtout intéressante pour ceux qui ne possèdent que des lecteurs de disquette puisqu'elle ne nécessite aucun accès disque. Cette option ne peut être

déclenchée qu'une seule fois au cours d'une opération de linkage. Le fichier à assembler doit pour cela figurer dans la mémoire.

➥ Cette option est pratique lorsqu'on n'a modifié qu'un seul module objet, parmi un grand nombre d'autres, et qu'on veut recomposer un programme actualisé.

Attention :



Pour des raisons techniques, l'option '^' n'est pas compatible avec la gestion des bibliothèques. Il en résulte que les symboles nécessités par un module objet produit avec '^' ne peuvent être fournis par une bibliothèque. (A moins qu'ils ne soient déjà utilisés par ailleurs et qu'ils soient donc, de toute façon, assurés d'être importés.) S'il faut, cependant, absolument que des symboles soient, malgré tout, importés d'une bibliothèque, l'utilisateur devra avoir recours à l'option '=' décrite ci-dessous, à laquelle cette restriction ne s'applique pas.

= Chemin

➥ Assemblage au cours du linkage. Si un fichier portant le nom spécifié dans le chemin figure déjà en mémoire, il sera assemblé. Dans le cas contraire, il sera chargé à partir du chemin spécifié. L'assemblage terminé, un fichier objet sera écrit sur disque où il sera alors à la disposition des opérations de linkage ultérieures. Il est, d'autre part, intégré dans l'opération de linkage en cours.

➥ Un nombre illimité d'assemblages de ce type peuvent être effectués.

➥ Ce sont les réglages de paramètres effectués dans le menu d'appel de l'assembleur (Menu 6,2) qui s'appliquent pour l'assemblage.

Instruction

➥ Réglage des paramètres d'exploitation. Le trait d'union est suivi d'une ou plusieurs instructions :

s ou S -	Ajouter une table de symboles, uniquement exportations et importations
l ou L -	Table de symboles avec tous les symboles
m ou M -	'load map' - liste alphabétique de tous les symboles traités
p ou P -	Linkage partiel : produire un module objet au lieu d'un programme
u ou U -	'unresolved' - si des références non résolues demeurent après intégration de tous les modules objet spécifiés, cela entraîne normalement l'arrêt du linkage. Si cependant l'instruction U est donnée, l'opération se poursuit, le linker prenant alors la valeur 0 pour les symboles manquant.
a ou A -	Linkage absolu - production de codes déjà relogés à une adresse fixe pouvant être réglée de façon séparée suivant les segments (par exemple pour brûler une Eprom)
Syntaxe : A	Parametre_1 (text) Parametre_2 (data) Parametre_3 (bss) Les paramètres peuvent ici revêtir les formes suivantes : r le segment correspondant doit être relogeable x exactement comme le segment précédent (en tant que paramètre de Text, identique à 'r') $xx (nombre hexadécimal), le segment correspondant doit être relogé à l'adresse spécifiée par le nombre hexadécimal.

Remarques :



➥ Les fichiers produits par linkage absolu ont une tête de fichier (header) différente du format habituel. Sa structure est, en effet :

Offset :	Contenu :	
$00	Longueur du segment de texte	a)
$04	Longueur du segment de données	
$08	Longueur du segment Bss	
$0C	Longueur de la table de symboles	
$10	Adresse de base du segment de texte	b)
$14	Adresse de base du segment de données	b)
$18	Adresse de base du segment Bss	b)
$1c	Fin du Header	

a) Si des informations de relogement sont disponibles, le bit n ° 31 est mis à 1.

b) = 0, si à reloger

Le système d'exploitation ne soutient pas le format d'un fichier linké de façon absolue. Le développeur doit donc veiller lui-même au bon positionnement des segments dans la mémoire.

➥ Toutes instructions autres suivant l'instruction 'A' sur la même ligne seront ignorées !

+	Ligne de commentaire
;	Commentaire en fin de ligne

Un commentaire peut être écrit à la suite des caractères '+' et ';'.

Chapitre 4

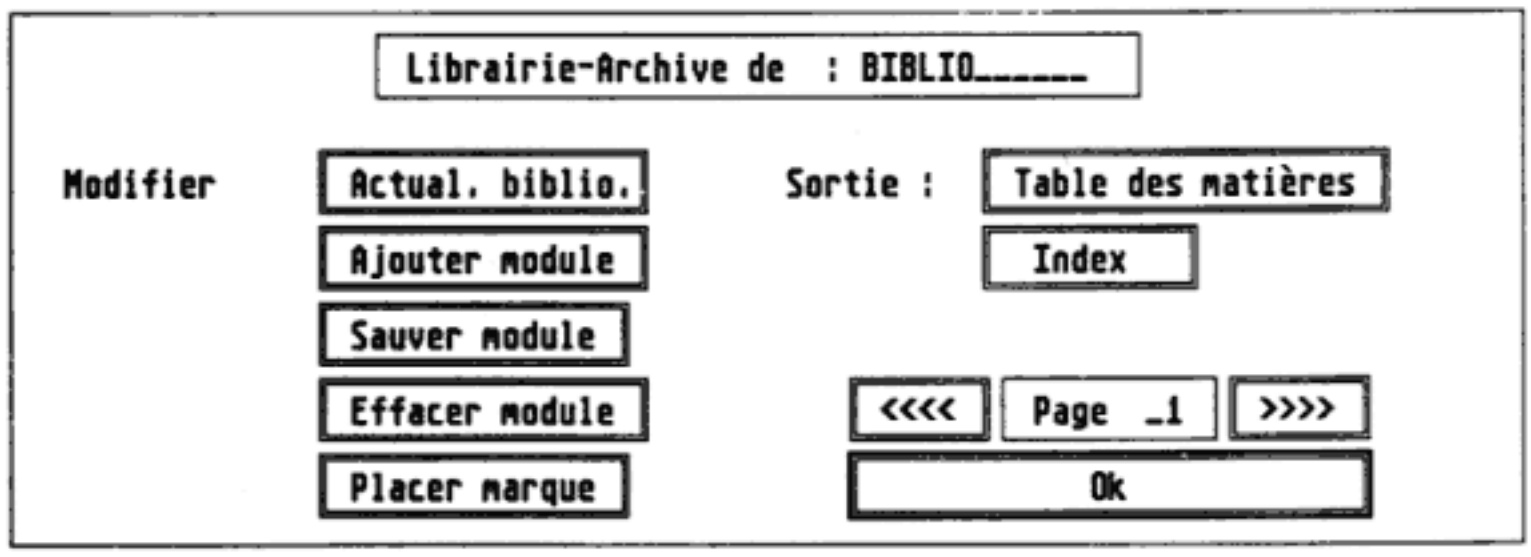
Gestion de bibliothèques

Lorsque l'utilisateur ne veut pas se contenter d'utiliser des bibliothèques étrangères et qu'il veut pouvoir en créer lui-même pour son travail ultérieur, il lui faut un utilitaire pour gérer ces bibliothèques. Le menu Archive (Menu 6,5) du GFA Assembleur constitue un utilitaire de ce type intégré.

Son mode d'emploi se base sur la gestion des fichiers sous GEM. En sélectionnant le point approprié du menu (Menu 6,5), l'utilisateur parvient tout d'abord dans l'écran de sélection de fichier pour sélectionner la bibliothèque à traiter. Lorsqu'est spécifié ici un nom qui n'existe pas encore ou qui n'appartient pas à une bibliothèque, une nouvelle bibliothèque est créée sous ce nom.

Dans le cas contraire, la bibliothèque voulue est chargée.

Une fois le nom de la bibliothèque défini, l'écran de travail suivant apparaît :



MAKE_HEX 0

Dans le bas de l'écran sont affichés les modules contenus dans la bibliothèque, alors que des fonctions de traitement peuvent être déclenchées avec la souris dans le haut de l'écran :

'<<<<' et '>>>>'

➥ Feuilleter le répertoire des modules. Le numéro de la page du répertoire actuellement affichée s'inscrit entre les boutons.

'Actualiser bibliothèque'

➥ Comme dans un répertoire sous GEM, la date de création est notée, dans une bibliothèque, pour chaque module qu'elle contient. Lorsque l'utilisateur retire des modules de la bibliothèque pour les modifier, la bibliothèque est périmée. Dans de telles situations, l'option 'Actualiser bibliothèque' soutient une actualisation automatique.

Dans le dossier à partir duquel la bibliothèque a été chargée seront recherchés les divers modules objet dont les noms coïncident avec des noms figurant dans la bibliothèque. Tous les modules de la bibliothèque dont la date et l'heure sont plus anciennes seront remplacés par les versions plus récentes trouvées dans le dossier.

Si l'utilisateur veut pouvoir utiliser cette fonction, il doit veiller à ce que les date et heure système fixées soient toujours exactes, faute de quoi des informations précieuses risqueraient d'être détruites à cause d'indications de date erronées.

'Ajouter un module'

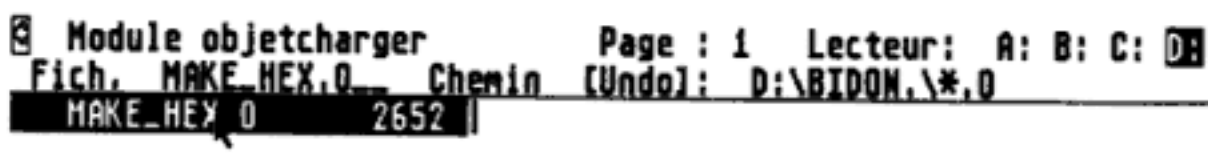
➥ Extension de la bibliothèque. Des modules objet isolés ou bien des bibliothèques entières peuvent être ajoutés à la bibliothèque. Les fichiers correspondants sont définis à l'aide de l'écran de sélection de fichier (qui apparaît si nécessaire).

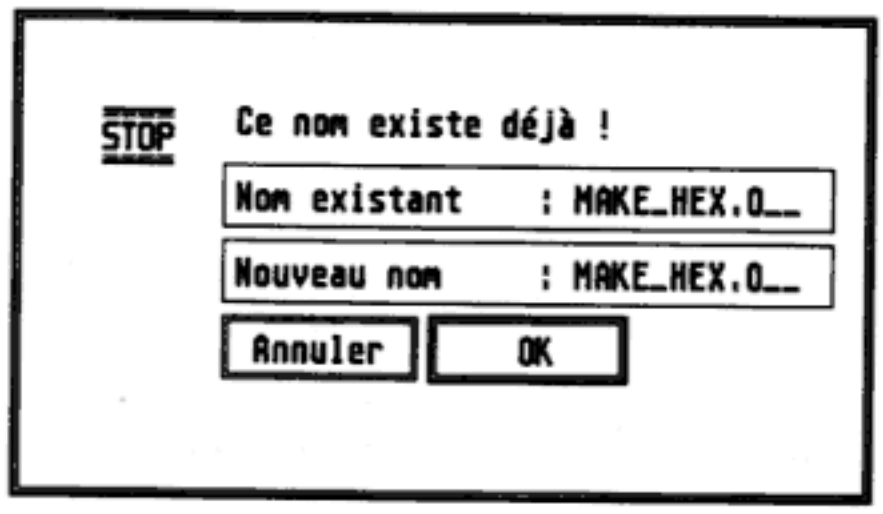
Remarques :



➥ Comme il est précisé en annexe, une bibliothèque n'est, au fond, pour l'essentiel, rien d'autre qu'une succession de modules objet séparés regroupés dans un fichier. C'est pourquoi les sous-répertoires ne sont pas soutenus comme pour la gestion de fichiers (ne serait-ce que par souci de compatibilité).

➥ Une bibliothèque ne doit pas comporter plusieurs modules du même nom car il ne serait pas possible de les distinguer. Le cas échéant apparaît une boîte dans laquelle vous pouvez, au choix, modifier les doubles noms, écraser le module concerné ou interrompre l'opération.





➥ Une bibliothèque peut comporter un nombre de modules illimité.

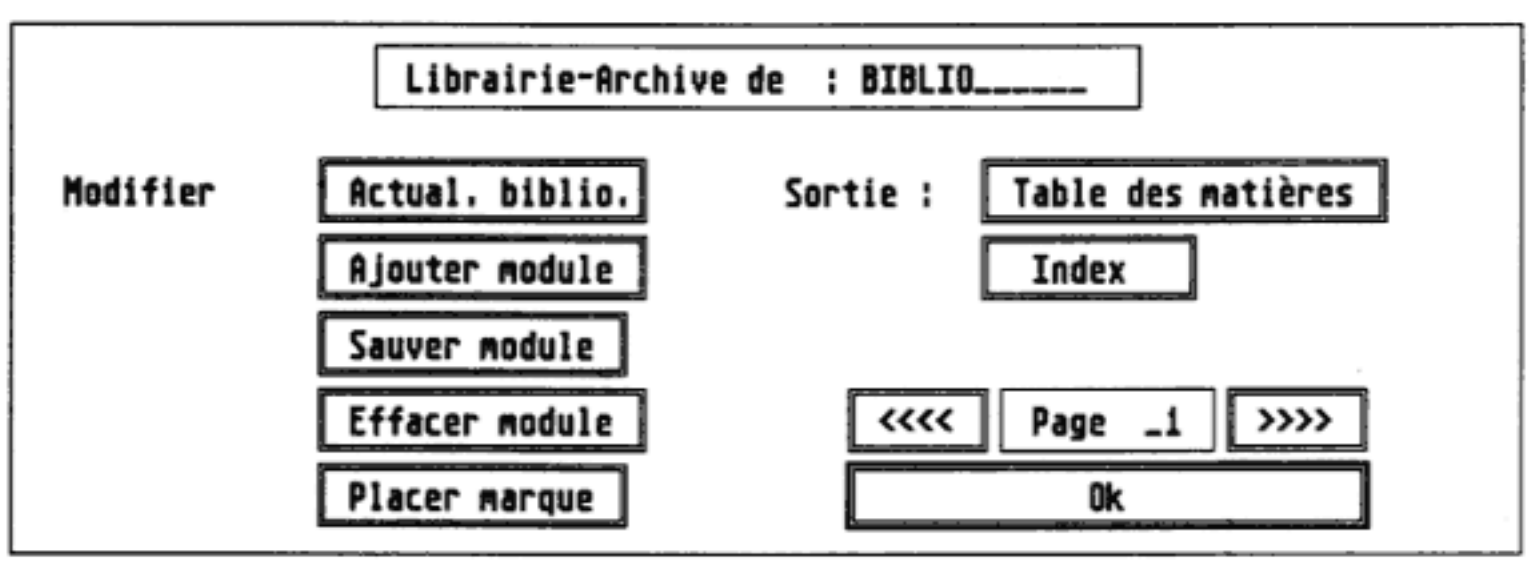
'Effacer module'

➥ Suppression d'un module. Il doit pour cela être placé en inversion vidéo à l'aide du bouton gauche de la souris. Si la touche Shift est tenue enfoncée pendant la sélection, plusieurs modules peuvent être éliminés à la fois. L'opération de suppression elle-même (Cliquer 'Supprimer module' avec le bouton gauche de la souris) est irrévocable !

'Sauver module'

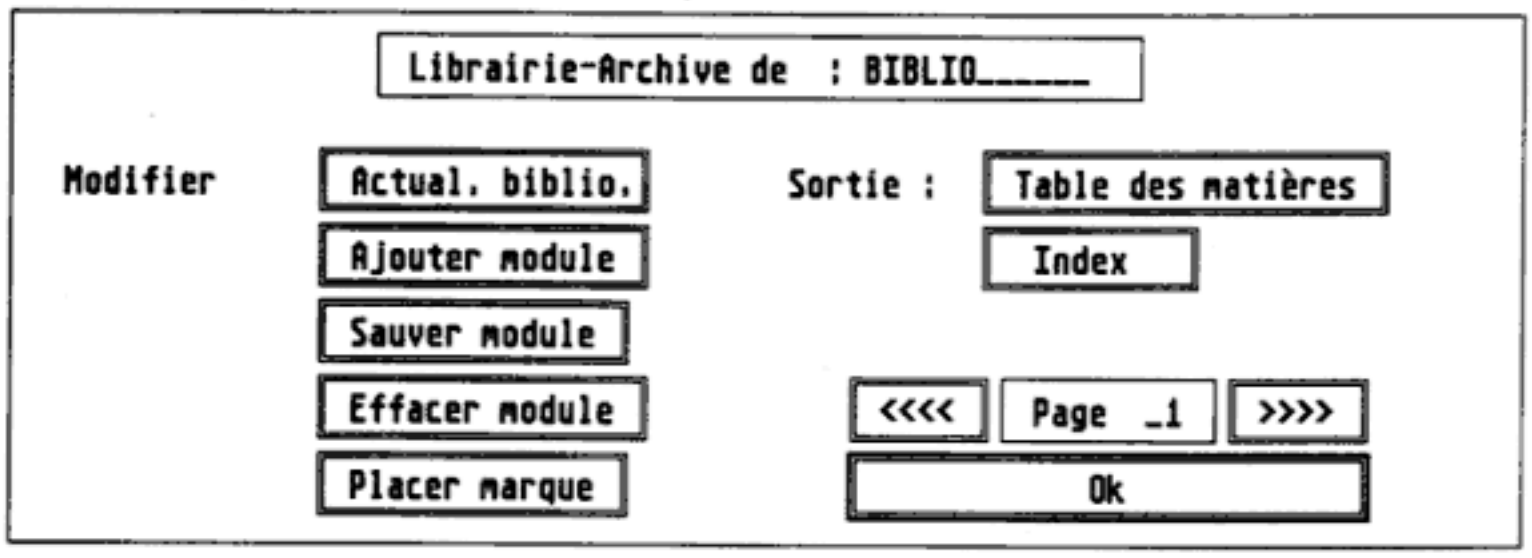
➥ Après sélection d'un ou plusieurs modules dans le répertoire (voyez 'Effacer module'), ce ou ces modules sont sauvegardés en tant que fichier objet indépendant.

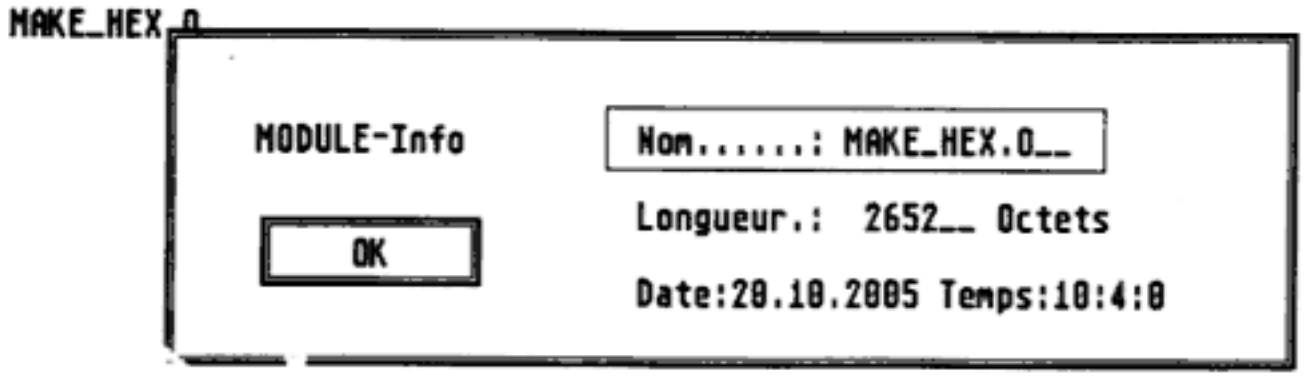
On a le choix comme désignation du fichier entre, d'une part, le chemin de la bibliothèque avec le nom du module et, d'autre part, la définition du nom du fichier à l'aide du menu de sélection de fichier.



Cliquer des modules avec le bouton droit de la souris.

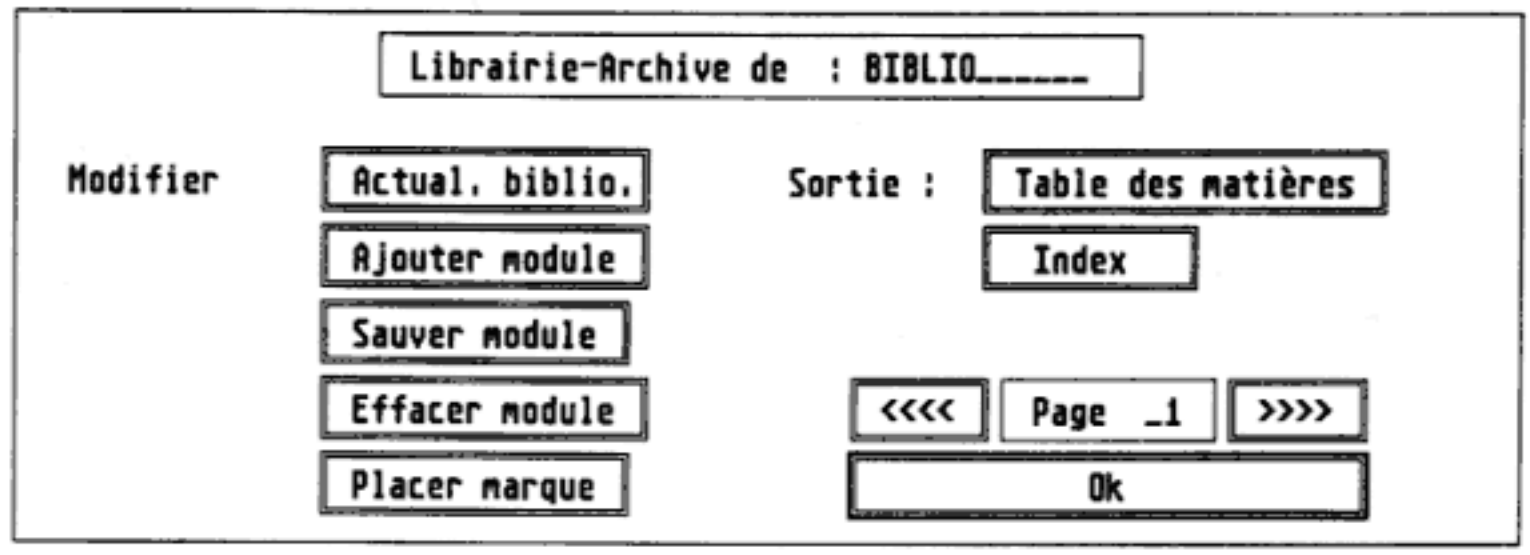
➥ Affichage d'informations complémentaires dans une boîte. (Nom, modifiable ; taille en octets ; date, heure).

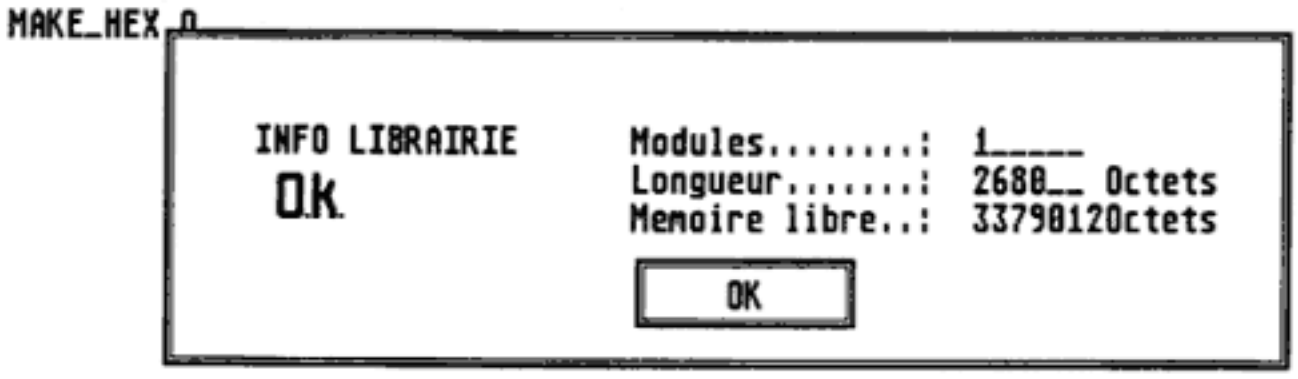




Cliquer le nom de la bibliothèque avec le bouton gauche de la souris

➥ Informations sur la bibliothèque. Affiche le nombre de modules gérés, la taille de la bibliothèque, la place mémoire disponible (la longueur d'une bibliothèque ne doit pas excéder la place mémoire libre disponible !).





'OK'

➥ La bibliothèque est sauvegardée sur disque sous sa forme modifiée (si vous le souhaitez, sous un nouveau nom, voyez Extraire module). On revient ensuite au menu principal de l'éditeur.

'Index'

➥ Comme nous l'indiquions dans l'introduction de ce chapitre consacré au linker, le linker accède aux bibliothèques à travers des fichiers d'index.

Si une bibliothèque ne possède pas de fichier d'index, il doit tout d'abord être créé. Pour ne pas ralentir l'opération de linkage, l'utilisateur peut faire sauvegarder la bibliothèque modifiée et faire produire en même temps le fichier d'index correspondant, en sortant de la gestion de bibliothèque avec 'Index' (vous trouverez de plus amples explications sur le fichier d'index en annexe).

Attention :



➥ Lorsque des modifications sont apportées à une bibliothèque sans que son fichier d'index soit également actualisé, cela risque de perturber le linker et entraîner des erreurs lors du linkage !

'Contenu'

➥ Dans le même dossier que la bibliothèque sera mis en place un fichier ASCII contenant le fichier d'index sous une forme lisible par un être humain normalement constitué. L'utilisateur reçoit ainsi des indications sur la structure interne d'une bibliothèque, telle qu'elle est décrite en annexe. (Noms et dates des modules contenus, exportations et importations, dépendance mutuelle).

Exemple :

Extrait d'un répertoire de bibliothèque (Extension : .DIR) :

```
GFA-Ass V1.0 Fichier d'index :A:\COPY.\GEMLIB.DIR Jour :6.2.1986 Heure :1:17:0
S:1

D E P E N D E N C Y   T A B L E :
Le module.....dépend de :
0      :  83   93   96
1      :  59   60   61   62   63   64   65   66   67   76   77   81   82   83   84   85   91   93
          94   96   98   103 105 106 120 121 122
2      :  33   60   61   62   63   64   65   67   70   71   72   73   74   75   76   77   78
79   80   81   82   83   84   85   93   96   104   116   120 121 122
3      :  31   83   93   96   98   103
4      :  31   83   93   96   98   103
...

L I S T E   D E S   M O D U L E S :

0    : channel5.o...000002  1   : xmainnw.o....00056a  2   : mallocdb.o...000fce
3    : noascii.o....00137a  4   : nobinary.o...0014ea  5   : nodisk.o.....00165e
6    : nofilesz.o...001866  7   : nofloat.o....00191a  8   : nolong.o.....001b26
9    : nottyin.o....001c46  10  : access.o.....001d52  11  : atoi.o.......001fae
12   : atol.o.......0021be  13  : calloc.o.....0023fe  14  : exec.o.......0025f6
15   : fdopen.o.....00285e  16  : fgets.o......002b0e  17  : fopen.o......002cb2
...
L I S T E   D E S   E X P O R T A T I O N S :

____atab......105     ___atab......105     ___chini.....83      ___fdecl.....64
___main.......58      ___noasc.....3       ___nobin.....4       ___nodis.....5
___noflo......7       ___nolon.....8       ___open......91      ___prtin.....78
___prtld......79      ___prtsh.....80      __aflist.....76      __afreeb.....76
__allocc......83      __arith......118     __atof.......109     __blkio......86
...
```

```
Variables G L O B A L E S  et  E X T E R N E S  des modules :

channel5.o.(Offset: $000002)
           EXPORTE:  __fds     _errno    ___cpmrv  __errcpm  _maxfile  __chvec
                     __allocc  __freec   __chinit  ___chini  __chkc
           IMPORTE:  __fds     _errno    _blkfill  ___cpmrv  __errcpm

xmainnw.o..(Offset: $00056a)
           EXPORTE:  ___pname  ___atab   __fds     ___main   _nowildc
           IMPORTE:  _main     _strcpy   _sbrk     _exit     _brk      ___pname
                     ___atab   ___BDOS   _creata   _strcat   _opena    _lseek
                     _strchr   _close    __fds
...
```

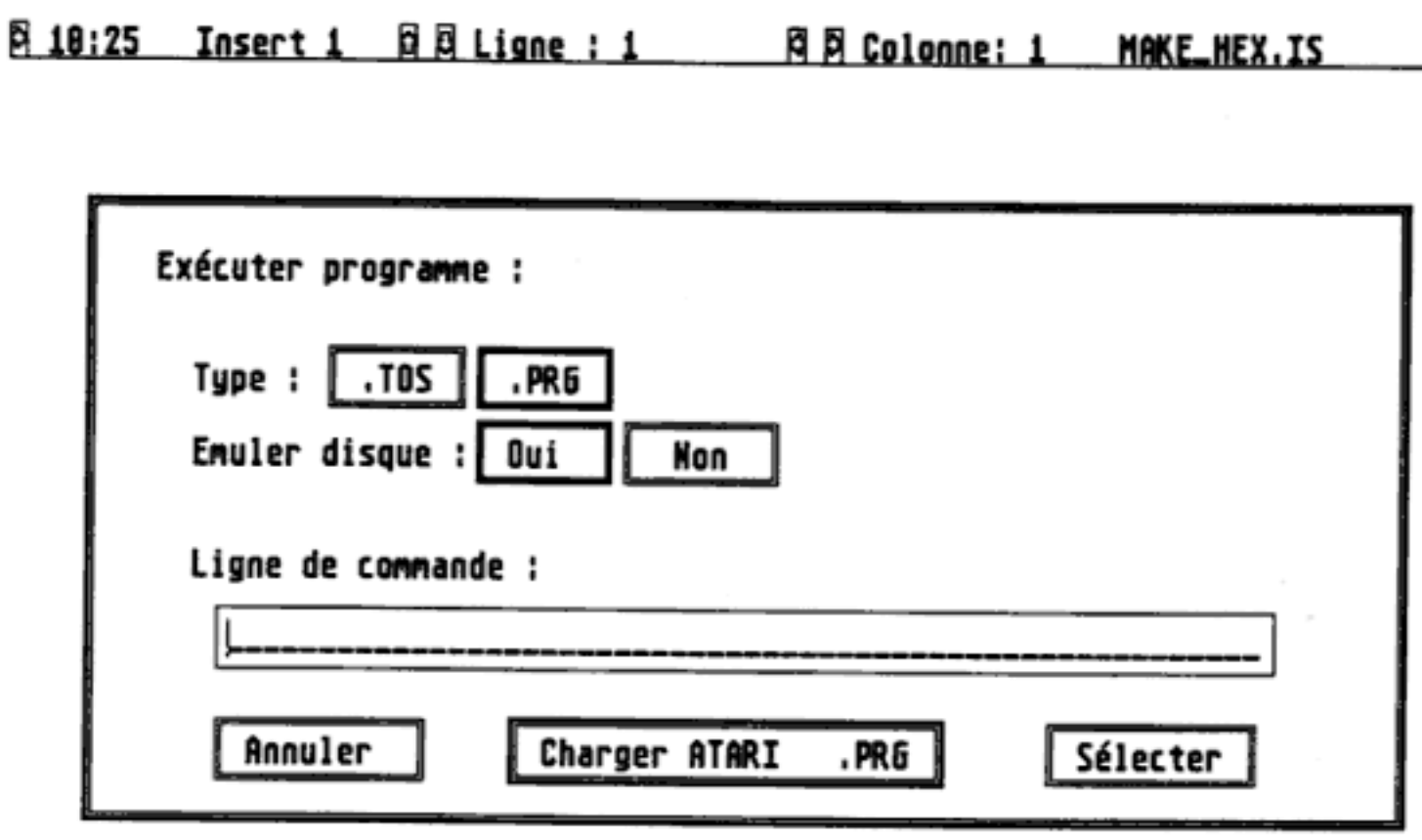
Partie D
Charger les programmes
Appeler les débogueurs

Chapitre 1

Exécuter programme

La fonction 'Exécuter programme' (Menu 6,6) vous permet de charger à la suite n'importe quel programme directement à partir du GFA Assembleur. La seule condition est que la mémoire de travail encore disponible suffise pour charger le programme voulu. Si le GEMDOS annonce une erreur de mémoire en chargeant ou en abandonnant le programme à charger, cette erreur est affichée et le chargement interrompu.

Si vous chargez des programmes GEM qui ne sont pas conçus proprement, l'ordinateur risque de se planter ou le programme chargé risque de ne pas tourner correctement.



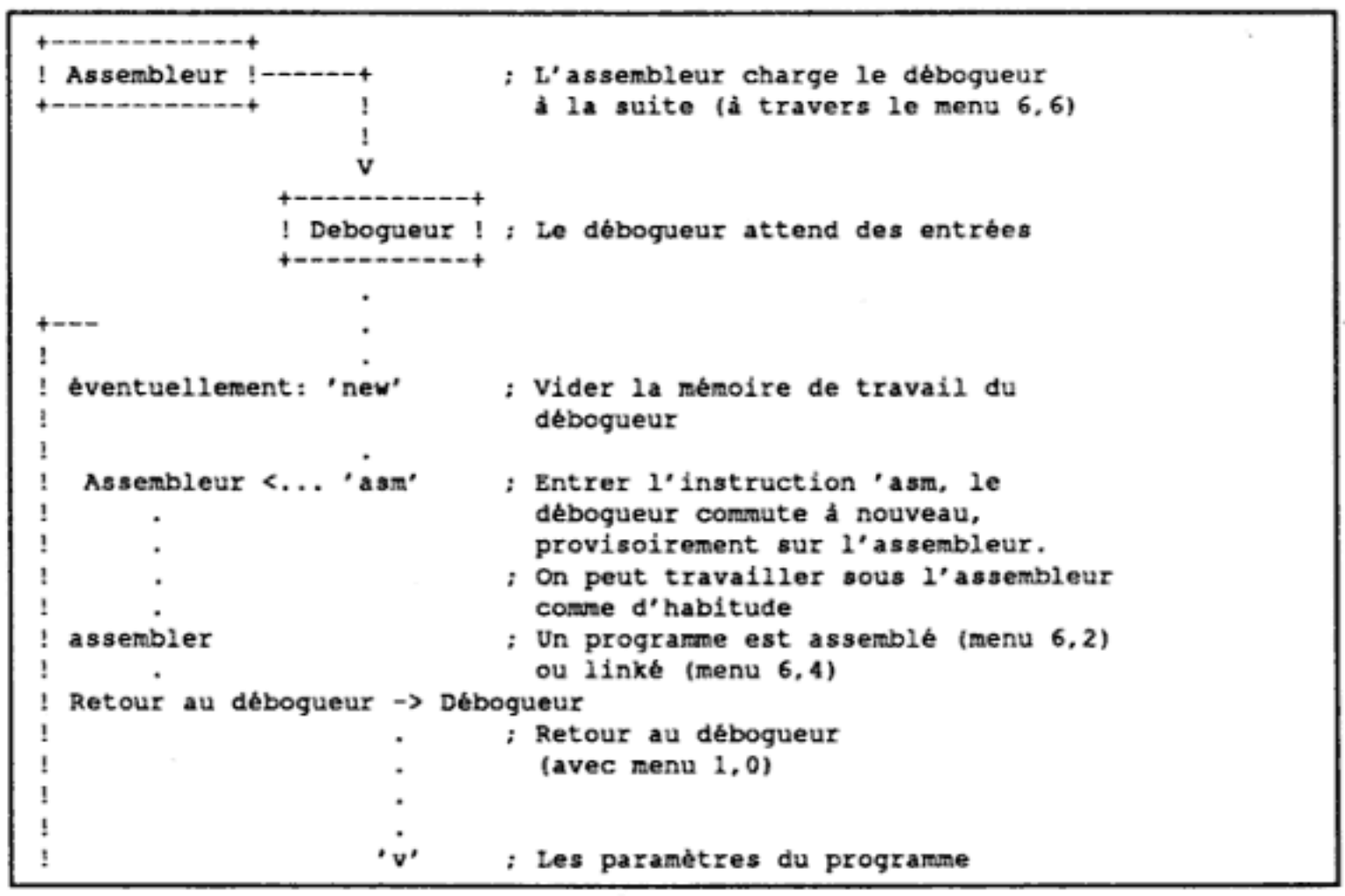
La fonction 'Exécuter programme' répond essentiellement à quatre domaines d'application :

- Chargement à la suite du programme produit : Le programme peut être testé directement. Si une erreur apparaît dans le programme (erreur bus, erreur d'adressage,

etc.), le GFA Assembleur sortira un message d'erreur approprié et mettra fin au programme chargé à la suite. Si, toutefois, les vecteurs pour ces exceptions sont dirigés vers la Ram dès le chargement du GFA Assembleur, ce dernier supposera qu'un débogueur figure déjà en mémoire et renoncera, par conséquent, à intercepter les erreurs de ce type.

➥ Chargement à la suite du débogueur GFA : Si vous voulez tester un programme de façon plus précise, vous pouvez aussi faire charger tout d'abord le débogueur, puis, sous le débogueur, charger le programme à tester. Vous pouvez travailler sous le débogueur comme si vous l'aviez directement lancé à partir du bureau. Lorsque vous quittez le débogueur (avec quit), vous revenez directement sous l'assembleur.

Sous le GFA Debugger, vous pouvez, cependant, également commuter par intervalle le GFA Assembleur. Cela peut être très utile, par exemple, pour consulter brièvement le source du programme en cours de débogage ou pour transmettre un programme directement de l'assembleur au débogueur. Le schéma suivant illustre ce procédé :



```
!                                   transféré peuvent être affichés
!                        .
!                        .          ; Le programme transféré peut être
!                                     examiné
+---                     .
                       'quit'       ; Une fois achevée la recherche des
                          !           erreurs, le débogueur est abandonné
                          !           avec 'quit'
+------------+            !
! Assembleur ! <-------- +          ; La mémoire réservée pour le
+------------+                       débogueur est libérée, l'ensemble de
                                     la mémoire est à nouveau disponible
                                     pour l'assembleur.
```

La zone encadrée peut être répétée autant de fois que vous le souhaitez. Il est ainsi possible d'éliminer une erreur dans le texte source du programme, puis de transférer à nouveau le programme dans la mémoire du débogueur pour le tester.

➥ Chargement à la suite d'un compilateur : Si vous utilisez le GFA Assembleur comme éditeur de texte source pour des langages de programmation évolués, ou si, par exemple, vous n'avez écrit que certaines parties de votre programme en assembleur et que vous avez écrit d'autres parties sous un autre langage de programmation, vous pouvez faire charger n'importe quel compilateur compatible avec les programmes batch, autrement dit n'importe quel compilateur qui puisse être appelé à partir d'un interpréteur de commandes ou d'autres programmes. Vous pouvez transmettre à ce compilateur, dans la ligne de commande, le nom du texte source qu'il doit compiler.

Pour pouvoir travailler encore plus vite avec les compilateurs, assurez-vous que le bouton 'Oui' est bien activé après 'Emuler disque' dans la boîte de dialogue de l'option 'Exécuter programme'. L'éditeur du GFA Assembleur sera, en effet, alors à même de transmettre directement au compilateur un texte source figurant dans la mémoire de texte. Dans ce cas, le GFA-Assembler simule les appels GEMDOS correspondants. Il ne simule, cependant, pas un lecteur spécial mais transmet simplement ses textes au lieu de ceux du lecteur de disquette. Il faut donc que tout fichier que le GFA Assembleur est censé transmettre figure également, sous le même nom,

sur la disquette. Pour plus de détails sur le fonctionnement de cette émulation, veuillez vous reporter à l'annexe B.

Nous allons essayer d'illustrer, à l'aide d'un exemple, la marche à suivre lors du chargement d'un compilateur : Vous avez par exemple, sur disquette, un texte source en Pascal, appelé 'TEST.PAS'. Vous chargez ce texte dans la mémoire de texte du GFA Assembleur. Vous pouvez alors modifier / compléter / corriger ce texte. Vous appelez ensuite avec le menu 6,6 la fonction 'Exécuter programme'. Dans la boîte, vous entrez 'TEST.PAS' comme ligne de commande et vous activez le bouton 'Oui' sur la ligne 'Emuler disque'. Vous cliquez alors sur le bouton 'Sélecter' et vous sélectionnez votre compilateur sur l'écran de sélection de fichier.

Le compilateur est alors chargé et il charge à son tour, à la suite, le texte 'TEST.PAS.

Ce texte lui est transmis directement par le GFA Assembleur. Des accès disque ne sont donc plus effectués que pour lire le répertoire et pour la sauvegarde.

Si le compilateur interrompt l'opération de traduction avec un message d'erreur, vous vous retrouvez immédiatement sous l'éditeur où vous pouvez alors éliminer l'erreur. Si vous voulez à nouveau appeler le compilateur, vous n'avez plus besoin de le sélectionner à nouveau car le GFA Assembleur retient le nom et le chemin du dernier programme chargé. Pour tout nouvel appel du même programme, il suffit donc désormais, après Menu 6,6, d'actionner la touche <Return> ou de cliquer avec la souris sur le bouton Défaut.

➥ Chargement à la suite du GFA-BASIC 3.0 : Le nouveau GFA-BASIC 3.0 peut, avec une instruction, commuter provisoirement le GFA Assembleur. Voici comment procéder à cet effet :

- Chargez le GFA-BASIC.

- Placez une instruction INLINE (dont la syntaxe est décrite dans le manuel du BASIC) en n'importe quel endroit du texte du programme BASIC.

- Placez le curseur sur l'instruction INLINE.

- Appuyez sur la touche Help.

  o Le point de sélection ' ASM ' apparaît dans la ligne d'état du haut de l'écran.

- Cliquez ce point de sélection ou appuyez sur la touche A.

  o Le BASIC transmet au GFA Assembleur l'adresse et la longueur de la zone réservée à la suite de l'instruction INLINE.

Si vous assemblez ou linkez maintenant un code source sous le GFA Assembleur, il sera automatiquement copié dans le texte du programme. Le programme ainsi produit ne doit pas comporter d'adresses relogeables, faute de quoi un message d'erreur approprié sera affiché.

Il convient, sous l'assembleur, de tenir compte des remarques suivantes : Si la place mémoire disponible ne suffit pas pour travailler correctement avec les deux programmes simultanément (ce qui exige que 200 Ko environ soient encore libres avant de charger un autre programme), la commutation des deux programmes n'est pas réalisée. Lorsque c'est le GFA Basic 3.0 qui est chargé à la suite, il est reconnu par le GFA Assembleur. Avant que l'assembleur ne charge le Basic, il affiche une boîte dans laquelle vous pouvez spécifier quelle proportion de la mémoire encore libre du GFA Assembleur vous voulez conserver par devers vous. L'assemblage requiert un minimum de 60 Ko de mémoire disponible.

Cette fonction vous permet, par exemple, d'intégrer des programmes machine directement dans des lignes Basic pour les y faire exécuter. Ces programmes machine doivent pouvoir être librement relogés dans la mémoire. Si l'assembleur rencontre, lors de la traduction d'un programme de ce type, une instruction non relogeable, ou bien si le programme atteint une taille excédant la place réservée à cet effet dans la ligne Basic, un message d'erreur approprié est affiché.

Pour revenir au Basic, il vous suffit de quitter le GFA Assembleur avec le Menu 1,10. Si vous n'avez plus besoin de travailler sous GFA Basic, vous pouvez le quitter avec 'Quit' ou 'System'. Vous vous retrouverez alors immédiatement sous le GFA Assembleur et l'opération de chargement à la suite sera achevée.

Chapitre 2

Appel du débogueur résident

Lorsque le débogueur est résident (c'est-à-dire si la version résidente a été bootée avec le système à partir du dossier Auto ou bien si vous avez auparavant installé, de façon inamovible, le débogueur dans la mémoire avec l'instruction 'resident'), un crochet apparaît devant le point 6,7 du menu. Dans ce cas, cette fonction vous permet d'accéder directement au débogueur résident (sans, par conséquent, que vous soyez obligé de le charger à partir de la disquette). Le travail avec le débogueur résident ne se distingue par du travail avec le débogueur chargé par la suite.

Cette fonction a donc le même effet que la fonction 'Exécuter programme' lorsque cette dernière est utilisée pour charger le débogueur à la suite. L'avantage est, cependant, qu'il n'est pas nécessaire de charger le débogueur à partir de la disquette, ce qui permet de gagner du temps.

Partie E
Annexes

Annexe A

Tables ou formats d'entrée de l'éditeur

A1 : Emploi des champs d'entrée

Les champs d'entrée apparaissant dans les boîtes du GFA Assembleur se répartissent en deux catégories :

- Les boutons doivent être cliqués avec la souris. Les boutons activés sont affichés remplis. Sous les boutons figure toujours un bouton principal déclenchant une fonction centrale. Il se distingue des autres boutons par un cadre plus épais. Le bouton principal peut être sélectionné avec <Return>. La boîte peut être abandonnée avec <Esc>.

- Les champs d'entrée au clavier. Dans ces champs, seuls sont admis les caractères utiles pour l'entrée concernée (seuls des chiffres peuvent donc être entrés dans un champ d'entrée attendant un nombre). Le curseur de trait clignote dans le champ d'entrée dans lequel les entrées peuvent être effectuées. Tous les caractères entrés sont insérés à droite de ce curseur de trait. Les touches suivantes permettent d'effectuer des corrections dans l'entrée :

Curseur gauche :
Curseur vers la gauche.

Curseur droit :
Curseur vers la droite.

Shift + Curseur gauche :
Curseur au début du champ d'entrée.

Shift + Curseur droit :
Curseur à la fin du champ d'entrée.

Home : Curseur au début du champ d'entrée.

Clr : Effacer chaîne de caractères entrée.

Delete : Supprimer caractère à droite du curseur de trait. Tous les caractères suivants sont ramenés d'une position sur la gauche.

Backspace :
Supprimer caractère à gauche du curseur de trait. Tous les caractères suivants sont ramenés d'une position sur la gauche.

Lorsque plusieurs champs d'entrée figurent dans une même boîte, vous pouvez déplacer le curseur d'un champ à l'autre avec les touches 'Curseur haut' et 'Curseur bas'.

A2 : Le format des tables d'instructions

Chaque ligne d'une table d'instructions se compose des zones suivantes :

```
FFF   +B Séquence de caractères
!      !        !
!      !        +-----> Séquence de caractères est la séquence de
!      !                caractères devant apparaître lorsqu'on
!      !                appuie sur la touche voulue. Cette séquence
!      !                de caractères peut comporter tous les
!      !                caractères ASCII de 1 à 255 (décimal).
!      !
!      +--------------> Lettre à laquelle cette séquence de
!                           caractères doit être attribuée.
!
+----------------------->Touches de commutation devant être enfoncées
                         pour appeler la séquence de caractères. Toute
                         combinaison autre que <Shift> est autorisée.
```

Les lettres suivantes doivent être utilisées pour désigner les touches de commutation voulues :

- A pour Alternate
- C pour Control
- S pour Shift

La table d'instructions peut aussi contenir des lignes vides ou des lignes de commentaires, non exécutables, commençant par un '*'.

A3 : Réglage standard de l'imprimante

Lorsque le GFA Assembleur ne trouve pas, lors de son lancement, de fichier dénommé 'PRINTER.CFG' et qu'aucun fichier '.CFG' n'est chargé à la suite (dans la boîte des paramètres d'impression, avec le bouton Table de configuration : 'charger'), le GFA Assembleur imprimera avec les réglages standard suivants :

- $0D,$0A	: Carriage Return
- $0C	: Form Feed
- $1B,'-',$01	: Souligné activé
- $1B,'-',$00	: Souligné désactivé
- $07	: Fin de l'impression

Tous les caractères seront sortis sur l'imprimante exactement tels qu'ils figurent dans le texte.

A4 : Format des tables de configuration de l'imprimante

Les tables de configuration portent, avant d'être converties en fichiers '.CFG' avec le bouton Compiler table de configuration, l'extension '.HEX' et se présentent en format ASCII. Leur texte est organisé en quatre sections :

- ➥ Nom d'imprimante
- ➥ Type d'imprimante
- ➥ Séquences de commande pour le formatage

➥ Table de conversion de caractères

Le contenu des différentes sections est le suivant :

➥ *Nom d'imprimante :*
Une ligne avec le nom de l'imprimante (cette section n'est pas encore utilisée dans la version actuelle ; elle doit néammoins être spécifiée par souci de compatibilité).

➥ *Type d'imprimante :*
6 octets hexadécimaux caractérisant le type de l'imprimante (cette section n'est pas encore utilisée dans la version actuelle ; elle doit néammoins être spécifiée par souci de compatibilité).

➥ *Séquences de commande pour le formatage :*
Il s'agit d'une table présentant la structure suivante :

```
- 01   xx,xx,...    :   Caractères de commande devant être transmis
                        pour Carriage Return.
- 1a   xx,xx,...    :   Caractères de commande devant être transmis
                        pour 'Souligné activé'.
- 1b   xx,xx,...    :   Caractères de commande devant être transmis
                        pour 'Souligné désactivé'.
- 1e   xx,xx,...    :   Caractères de commande devant être transmis
                        pour Form Feed.
- 21   xx,xx,...    :   Caractères de commande devant être transmis
                        après l'impression.
!      !
!      !
!      +----------> :   Nombre quelconque d'octets hexadécimaux.
!
+-----------------> :   Numéro de la fonction. Ces entrées doivent se
                        présenter en ordre croissant. Les numéros de
                        fonction autres sont également traduits mais
                        ne sont pas utilisés par la présente version
                        du GFA Assembleur.
```

➥ *Table de conversion de caractères :*
Il s'agit d'une table dans laquelle sont énumérés, l'un après l'autre, tous les caractères devant être remplacés par un ou plusieurs autres caractères. Chaque entrée se compose du code hexadécimal du caractère et de la séquence de caractères devant remplacer ce caractère.

Vous pouvez trouver sur la disquette le fichier 'NL-10.HEX' qui constitue un exemple de fichier de ce type. Cette table peut être modifiée pour être adaptée à d'autres imprimantes.

A5 : Particularités survenant lorsque la mémoire est pleine

Le GFA Assembleur utilise la mémoire disponible de l'Atari ST pour exécuter ses fonctions le plus rapidement possible. Cela signifie que lorsque la mémoire est largement remplie, certaines fonctions risquent de tourner légèrement moins vite. Dans ce cas, il se peut que les fonctions 'Sauver Commandes', menu 2,7 et 'Compiler table de conversion' menu 5,1, ainsi que l'extinction automatique de l'image sur un moniteur noir et blanc soient bloquées.

Annexe B

Formats de fichier de l'assembleur et du linker

B1 : Format des programmes exécutables

Un programme exécutable, c'est-à-dire un programme que vous pouvez faire exécuter en le cliquant à partir du bureau, se compose des éléments suivants :

Header du programme	(28 octets)
Segment de texte	(Code programme)
Segment de données	(Données du programme)
Segment BSS	(ne fait pas partie du fichier et n'est mis en place que lors du chargement, par le système d'exploitation)
Table de symboles	(Optionnelle, contient, au choix, tous les symboles utilisés lors de l'assemblage ou bien seulement les symboles globaux ; très utile pour le débogage)
Informations de relogement	(Suivant la mémoire déjà occupée, le système d'exploitation chargera un programme à exécuter à des adresses différentes. S'il utilise des adressages absolus, le programmeur ne peut donc pas connaître la position du programme dans la mémoire lors de l'exécution. C'est pourquoi tous les programmes sont codés par l'assembleur comme s'ils étaient censés être exécutés à partir de l'adresse 0 de la mémoire. Avant de commencer l'exécution d'un programme, le système d'exploitation convertit ensuite automatiquement les adressages absolus en positions physiques, à l'aide des informations de relogement.)

Format du header :	Position actuelle du programme	Contenu :
	$00 = 0	$601a "Magic" Code d'identification d'un programme pour le GEMDOS
	$02 = 2	Long. du segment de texte
	$06 = 6	Long. du segm. de données

$0a = 10	| Longueur du segment BSS
$0e = 14	| Longueur de la table de symboles
$12 = 18	| 10 octets nuls
$1c = 28 ...	| Début du programme

Informations de relogement

Les informations de relogement commencent, à une adresse paire, par un long mot désignant la distance relative entre le premier adressage absolu d'un programme et le début de ce programme. Les coordonnées de tous les autres adressages absolus sont codées chacune sous forme de distances en octets par rapport à l'adressage absolu précédent. Si, toutefois, la distance entre deux adressages absolus est supérieure à 256 octets, on placera 'n' fois un 1, de sorte qu'on aura : Distance = n*256 + Reste, 'n' étant le nombre de 1 et 'Reste' un nombre inférieur à 256. 'n'+1 octets seront donc nécessaires.

Comme les adresses ne peuvent naturellement se situer qu'à des adresses paires, les nombres constituant les informations de relogement seront donc tous pairs à l'exception du 1. Les informations de relogement se terminent par un octet nul. Le programmeur ne doit pas oublier que le segment de données peut lui aussi comporter des adresses absolues, par exemple sous forme de tables de saut !

B2 : Format des modules objet (Format Digital Research Inc)

Les modules objet ne peuvent être exécutés directement et doivent tout d'abord (éventuellement en même temps que d'autres modules) être reliés par un linker pour former un programme exécutable. Leurs structure ressemble, cependant, beaucoup à celle des programmes :

Header du programme	(Comme dans un programme exécutable !)
Segment de texte	(Les adresses absolues toujours
Segment de données	relatives à la base du segment !)
Segment BSS	
Table de symboles	(Obligatoire, contrairement à la table de symboles d'un programme !)
Informations de relogement	(Ne sont pas identiques aux informations de relogement d'un programme !)

Un linker regroupe les segments de texte des modules traités dans un segment de texte commun, de même pour les segments de données et les segments BSS. C'est pourquoi la position relative d'un segment de texte par rapport à son segment de données et à son segment BSS ne peut être connue au moment de la création du module. Tous les adressages absolus se réfèrent donc au DEBUT DU SEGMENT et ne sont convertis que par le linker par rapport au DEBUT DU PROGRAMME. Il en va de même pour les valeurs des symboles de la table de symboles.

Les informations de relogement d'un programme, avec leurs distances relatives, ne conviennent pas, pour la même raison, pour décrire des modules objet. C'est pourquoi le format utilisé est le suivant :

Les informations de relogement ont exactement la même taille que les segments de texte et de données réunis du module. Elles se composent de mots et longs mots ayant la signification suivante :

$0000	ne contient pas d'information
$0007	ne contient pas d'information, marque le début d'un code opération (tous les codes opération sont, en effet, au départ des mots auxquels s'adjoignent ensuite les opérandes.) Le linker du GFA Assembleur peut traiter ces informations mais il ne les génère pas dans ses modules objet !
$0005	indique que le long mot figurant à l'adresse relative correspondante dans le segment de texte ou le segment de données doit être relogé. Ce code est suivi d'un autre mot avec l'un des contenus suivants : $0002 Adresse absolue relevant du segment de texte $0001 Adresse absolue relevant du segment de données $0003 Adresse absolue relevant du segment BSS

$xxx4	Référence non résolue. Le nom du symbole correspondant peut être ainsi obtenu : ($xxx4 - 4)/8 = Numéro du nom de la table de symboles correspondant au module (en partant de 0) L'adresse relativement au début de la table de symboles sera donc : Adresse = (($xxx4 - 4)/8)*14
$yyy6	Indique qu'est effectué, dans l'emplacement correspondant dans le segment de texte, l'adressage de mot relativement au PC d'une référence non résolue ou bien d'une référence en direction du segment de données. Lorsqu'on a recours à l'adressage relatif au PC au-delà des limites de segment, ou avec des adresses encore inconnues, la prudence est, de toute façon, de rigueur car la portée de ces modes d'adressage (environ +- 32 Ko) peut se trouver dépassée à la suite de la fusion des modules. Le linker avertit l'utilisateur le cas échéant. L'adresse de symbole correspondante devient donc, comme pour les adressages de long mot non résolus : Adresse de symbole relative = (($yyy6 - 6)/8)*14

Structure de la table de symboles :

Chaque entrée de la table de symboles a une longueur de $e (=14) octets et est ainsi composée :

Adresse par rapport au début de l'entrée :	Contenu :
$0...$7 = 0...7	Nom du symbole (comblé avec 0)
$8 = 8	Statut du symbole *)
$9 = 9	Non défini
$a...$d = 10...13	Valeur du symbole

*) Octet de statut de symbole :

N de bit :	Signification (si bit=1)
0	Symbole du segment BSS
1	Symbole du segment de texte
2	Symbole du segment de données
3	Symbole non défini
4	Registre **)
5	Symbole connu globalement
6	Symbole = constante
7	Symbole est défini

**)	N'est pas soutenu par le linker GFA ! Si le bit 4 est à 1, la valeur de symbole correspondante est un masque de registre pour l'instruction 'movem'. (Peut le cas échéant être émulé en fixant le bit 6.)

Remarque :



On a affaire au type de symbole COMMON lorsque les bits 3 et 7 sont mis dans l'octet de statut de symbole et que la valeur du symbole est différente de 0.

Attention :



Comme le format de la table de symboles vous l'indique, tous les symboles ne comportent plus que 8 caractères lors du linkage. Si des symboles plus longs ont été utilisés lors de l'assemblage, ils sont tronqués et ramenés à 8 caractères SANS AVERTISSEMENT !

B3 : Format de fichier ABS

Les fichiers ABS sont créés à l'aide de l'option 'A' du linker. Ils présentent le même format que les programmes exécutables, avec les restrictions indiquées dans le guide d'utilisation du linker. Il faut aussi tenir compte, comme indiqué également dans le guide du linker (Chapitre C/3), de la différence de header.

B4 : Format de fichier IMG

Il arrive qu'il ne soit pas possible ou pas souhaitable de faire tout d'abord reloger des programmes (surtout des programmes courts) avant leur exécution. Lorsque l'utilisateur crée des programmes censés pouvoir être décalés sans restriction dans la mémoire, il ne peut utiliser aucun adressage absolu (sauf, bien sûr, pour accéder à des adresses de l'électronique). En outre, le header et les informations de relogement sont superflus. L'op-

tion IMG du linker sert à créer ce type de fichiers. Le linker contrôle si le programme comporte des adresses absolues à reloger et il avertit l'utilisateur lorsqu'il en trouve.

Le header et les informations de relogement ne sont jamais produits, une table de symboles peut être adjointe au programme si vous le souhaitez.

Remarque :



Ce format est aussi celui des programmes transmis à GFA-BASIC 3.0 avec l'instruction INLINE !

B5 : Structure des bibliothèques

Les bibliothèques traitées par le linker contiennent des modules objet et présentent une structure très simple : elles commencent par un mot d'identification ("Magic", $ff65), suivi, de façon séquentielle, des modules objet sous le format décrit à l'annexe B1.

Chaque module objet est, cependant, précédé d'un header supplémentaire, d'une taille de 28 octets :

Adresse par rapport au début du header	Contenu :
$00...$0d = 0...13	Nom du module
$0e...$0f = 14...15	Date de création en format GEMDOS
$10...$11 = 16...17	Heure de création en format GEMDOS
$12...$13 = 18...19	Octets nuls
$14...$15 = 20...21	$01b6
$16...$19 = 22...25	Longueur du module objet
$1a...$1b = 26...27	Octets nuls
$1c...$37 = 28...55	Header de module objet
$38... = 56...	Module objet

La fin d'une bibliothèque est désignée par un mot nul.

B6 : Fichiers d'index (pour les bibliothèques)

Le fichier d'index d'une bibliothèque fournit au linker toutes les informations dont il a besoin sur la structure de cette bibliothèque et sur les symboles qu'elle exporte. Le fichier d'index peut être converti en une version texte compréhensible par l'utilisateur, contenant les mêmes informations, à l'aide du menu de gestion de bibliothèque de l'éditeur/assembleur. Ces deux fichiers (dont les extensions sont respectivement ".NDX" ou ".DIR") ne doivent pas être confondus.

Structure du fichier d'index :

$ff75	Mot d'identification ("Magic")
$xxxx	Nombre de symboles exportés par la bibliothèque (mot)
$yyyy	Nombre de symboles contenus dans la bibliothèque (mot)
Table des modules	1)
Table des exportations	2)
Table de dépendance ("dependence table")	3)

Sur le point 1)

Une entrée de module a une longueur de 20 octets et contient le nom du module (16 octets) ainsi que l'offset du module relativement au début de la bibliothèque (les 4 autres octets). Les noms des modules sont triés par ordre alphabétique.

Sur le point 2)

Une entrée d'exportation contient le nom de l'exportation (8 octets) et le numéro du module exportant sous forme d'un long mot (4 octets).

Sur le point 3)

Lorsqu'un module est intégré, il comporte souvent lui-même des références encore non résolues qui doivent être alimentées par les exportations d'autres modules. C'est pourquoi il existe

pour chaque module un masque dans lequel est indiqué quels modules complémentaires doivent être également intégrés lorsqu'on utilise ce module. A chaque module correspond un bit de ce masque.

Exemple :

Supposons qu'il s'agisse de réaliser un fichier d'index pour une bibliothèque de 19 modules :

```
=>    Chaque masque a une longueur de 3 octets (c'est-à-dire 24 bits
      dont 19 seulement sont utilisés).

=>    Chaque masque présente la structure suivante :

      0000 0000|0011 1111|111      Numéro du   x,y,z = 0 :
      0123 4567|8901 2345|678. .... module     module non nécessaire

      xxxx xxxx|yyyy yyyy|zzzz zzzz  Bit corresp. x,y,z = 1:
      Octet 1  |Octet 2  |Octet 3    du masque module nécessaire

=>    Il y a 19 masques de ce type.
```

Le format du fichier d'index en clair est décrit dans le guide d'utilisation de la gestion de bibliothèque.

B7 : Format tokenisé du texte source

Le format tokenisé du texte source constitue l'une des particularités de l'assembleur GFA. La connaissance de ce format est, en principe, sans intérêt pour l'utilisateur. Les auteurs fourniront, cependant, volontiers des informations détaillées sur ce format aux développeurs intéressés qui les contacteraient par l'intermédiaire de la société GFA-Systemtechnik.

Annexe C

Messages d'erreur de l'assembleur et du linker

Comme nous l'avons déjà indiqué dans les guides de l'assembleur et du linker, la moindre difficulté lors de l'assemblage n'entraîne pas d'emblée une interruption de l'opération. Il est ainsi possible de tolérer des erreurs mineures, sans conséquence pour le développement du programme. Toutes ces erreurs sont cependant signalées à l'utilisateur.

Voici une liste de tous les messages d'erreur avec leurs causes possibles. En cas d'hésitation, veuillez vous reporter au chapitre approprié de ce guide d'utilisation.

Message d'erreur		Explication :
Erreur de syntaxe	1)	Une instruction ou un pseudo-code opération n'a pas été entré correctement
Mode d'adressage illégal	1)	Adressage non autorisé pour cette instruction
Extension illégale	1)	Extension d'instruction incorrecte (.b, .w, .l), certaines instructions ne fonctionnent pas avec toutes les tailles de données
Instruction incomplète	1)	Il manque un opérande
Expression illégale	1)	Dépassement du domaine de valeurs, type de données non autorisé, calculs de syntaxe incorrecte etc.
Pseudo-code opération non interprétable	1)	Un pseudo-code opération inconnu a été entré
Nom de symbole trop long	1)	Les noms de symboles (y compris ceux produits avec '\') ne doivent pas comporter plus de 16 caractères.
Branchement hors portée		Branchement relatif au PC au-delà de la portée de l'adressage utilisé
Symbole redéfini		----
Label non défini		Lorsqu'un programme exécutable est produit avec l'assembleur, tous les symboles utilisés

	doivent être définis. (Cette erreur peut aussi être occasionnée par une référence à un label local figurant dans une zone locale séparée, par mégarde, par un label non local
Nom de fichier non spécifié . . . . 1)	Il manque le nom de fichier (pour Include ou pour le linkage)
Tentative de relogement	Aucun adressage absolu ne doit être entrepris sous le mode IMG du linker ou lors de la transmission d'un programme au GFA-BASIC.
Manque nom de symbole . . . . . 1)	Le symbole (en tant qu'opérande de fonction) n'a pas été spécifié
Nombre de paramètres incorrect, éventuellement . . . . . *)	-----
Include impossible . . . . . *)	Tentative d'inclure un fichier de texte source inexistant
Mémoire pleine . . . . . *)	Voyez plus loin
Division par zéro	-----
Manque instruction EVEN . . . . . *)	Lorsque des mots (des instructions du processeur, des données) doivent être écrits à des adresses impaires, .EVEN doit être exécutée auparavant.
Seul assemblage 1 est possible.. . . . . *)	L'instruction '^' a été utilisée plus d'une fois dans le fichier de contrôle du linker.
Discrimination des parcours . . . . *)	Voyez les explications sur .IFBranchement vers l'instruction suivante Uniquement sous les modes d'optimisation des branchements ! Si l'optimisation n'a pas été réclamée, ce sera une instruction de branchement avec un offset de mot qui sera codée.
.ENDIF sans .IF . . . . . *)	-----
.IF sans .ENDIF . . . . . *)	-----
.ELSE sans .IF . . . . . *)	-----
.ENDR sans .REPT . . . . . *)	-----
.REPT sans .ENDR . . . . . *)	.ENDR a été oubliée
Macro redéfinie . . . . . *)	-----
Macro non définie . . . . . *)	Les macros doivent avoir été définies AVANT d'être appelées.
.ENDM/EXITM sans .MACRO . . *)	-----
.MACRO sans .ENDM . . . . . *)	-----
.ENDM/.EXITM superflu . . . . . *)	-----
Type de segment inconnu . . . . . *)	Voyez les instructions SECTION
Erreur lors du linkage . . . . . *)	Erreur fatale du linker (causes diverses)

Respecter le type de segment	*)	Instruction .DS.x dans un segment de texte ou de données ou bien .DC.x ou une instruction de 68k dans un segment BSS ou ABS
Définition de segment incorrecte	*)	Redéfinition d'un segment ou définition d'un segment de n > 63
Erreur de protocole	*)	Erreur d'accès de fichier lors de l'établissement du protocole d'assemblage
Adressage relatif hors de portée	2)	Le linker ne peut résoudre une référence de 16 bits à cause de la trop grande distance (supérieure à +/- 32 Ko environ)
Combinaison de symboles de types de segment différents		Voyez les guides de l'assembleur et du linker
ASSERT/FAIL	*)	Arrêt de l'assemblage provoqué par une instruction de l'utilisateur
Trop de chemins		Un maximum de 16 chemins de recherche peuvent être spécifiés. (La recherche s'effectue dans l'ordre de spécification)
Format de page incorrect	*)	Format de page incorrect ou buffer de page trop restreint (-> sélectionner un format de page plus petit ou augmenter le buffer de page dans le menu d'impression !)

1) également signalé dès l'entrée
2) uniquement message d'erreur du linker
*) erreur fatale (-> arrêt de l'opération !)

Mesures à prendre lorsque la mémoire est pleine :

Le GFA Assembleur travaille, à tous égards, orienté mémoire. Lorsque les masses de données traitées sont très importantes, il peut donc arriver, surtout avec les types d'ordinateurs les moins bien dotés en mémoire, que la mémoire de travail ne suffise pas. C'est pourquoi nous souhaitons vous indiquer un certain nombre de mesures d'urgence à prendre qui permettent, dans ces situations, d'économiser de la mémoire. Nous vous conseillons de les appliquer l'une après l'autre, au fur et à mesure que la mémoire disponible se réduit :

- ➥ Annulez la déclaration de tous les accessoires, disques RAM, extensions du système d'exploitation et autres "mangeurs de mémoire" ! (Reset si nécessaire !)

- ➥ Ne travaillez pas avec ASM à partir du débogueur ou du BASIC !

- ➥ Eliminez de la mémoire de travail tous les textes dont vous n'avez pas besoin !

- ➥ Réduisez la taille du buffer de spooler et du buffer de page dans le menu d'impression !

- ➥ Utilisez INCLUDE avec la disquette ou le disque dur ! (L'organisation de la mémoire de l'éditeur réclame 7 octets de moins pour les fichiers de texte source chargés à la suite lors de l'assemblage/linkage.)

- ➥ Renoncez à assembler lors du linkage ! (Réalisez tout d'abord les modules objet séparément, puis linkez-les !)

- ➥ Décomposez votre programme en modules objet pour le linker !

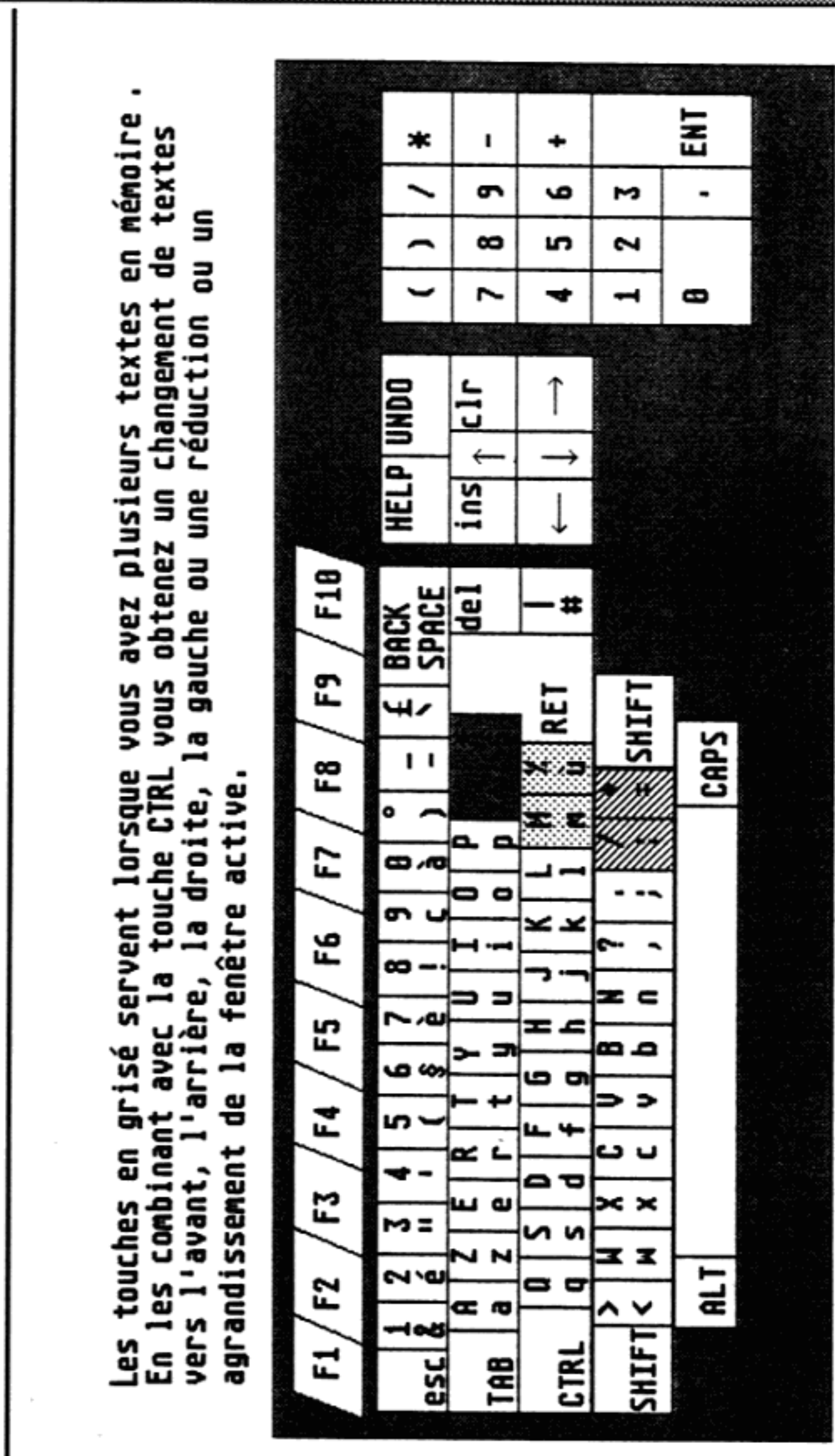
- ➥ Eliminez (avec "Recherche et remplacement") les commentaires de votre texte source !

- ➥ Achetez une extension de mémoire !

Annexe D

Manipulations des textes en mémoire

Les touches en grisé servent lorsque vous avez plusieurs textes en mémoire.
En les combinant avec la touche CTRL vous obtenez un changement de texte
vers l'avant, l'arrière, la droite, la gauche ou une réduction ou un
agrandissement de la fenêtre active.



Index

!

#Chemin 3-133
#Nmb_1 2-114
#Nmb_2 2-114
* 1-15
+ 2-126
- 2-126
.ABS {Offset} 1-87
.ASSERT 2-110
.BSS 1-86
.CARGS 1-88
.CLIST 2-120
.DATA 1-86
.DC{.x} 2-113
.DCB 2-113
.DCB{.x} 2-113
.DS{.x} 2-115
.END 1-91
.EXTERN 1-91
.FAIL 2-110
.GLOBL 1-91
.IFEQ 2-109
.IFGE 2-109
.IFGT 2-109
.IFLE 2-109
.IFLT 2-109
.IFNE 2-109
.IIF 2-107
.IIFEQ 2-109
.IIFGE 2-109
.IIFGT 2-109
.IIFL 2-109
.IIFLT 2-109
.IIFNE 2-109
.INIT 2-114
.INIT{.w} 2-114
.LIST 2-115
.LLEN 2-116
.LMODE 2-118
.LOCAL 1-91, 2-112
.MACRO 2-93
.MLIST 2-119
.NOCLIST 2-120
.NOLIST ..... 2-115 - 2-116
.NOMLIST 2-119
.PAGE 2-119
.PLEN 2-116
.PRINTER 2-119
.PUBLIC 1-91
.SECTION 1-90
.SPACE 2-119
.SUBTTL 2-116
.TEXT 1-86
.TTL 2-116
.XDEF 1-91
.XREF 1-91
1 3-52
2-9,0 3-52
= Chemin 3-134
? 1-15
?Chemin 3-132
@@Chemin 3-133
\!n 2-100
\!Parametre 2-100
\# 2-99
\'n 2-101
\'Parametre 2-101
\?n 2-100
\?Parametre 2-100
\\Parametre 2-96
\~ 2-101
\~{Nom} 2-101
\n 2-99
\Parametre 2-96
^Chemin 3-133

A

A 3-36
A droite 2-20
A gauche 2-20
Actualiser bibliothèque 4-138
Adda.b symbol,a1 3-46
Addi 1-81
Adressages rapides indirects 1-88
Ajouter un module 4-139
Alternate 3-53
Appels de macro 1-74
Appels GEMDOS 1-149
Appels macro 1-69
Archive 4-137
ASCII 3-42, 4-143
Assemblage 1-9
Assemblage automatique 1-71
Assemblage conditionnel 2-107
Assemblage en arrière-plan (oui/non) . 1-71
Assembleur 1-8
Avertissements 1-84

B

B 3-36
Backslash 2-96
Backspace A-158, 2-21
Barre des menus 2-18
Basic 1-152
Bibliothèques 3-129
Bits 1-75
Bloc 3-34
Boucle REPT 2-112
Buffer 3-62
Bureau GEM 3-38

C

Calculs 2-124, 3-46
Caractères 3-59
Chaîne de caractères .....3-58 - 3-59
Champ d'application 2-107
Champ d'entrée 3-61, 3-64
Champs d'entrée A-157
Chemin 1-14, 3-132
Chemin actuel 1-15, 2-106
Chiffres 1 à 9 2-24
Clavier 2-19
Cliquer des modules 4-141
Clr A-158
Code objet 1-72, 1-83
Code programme - code objet 1-82
Compilateurs 1-149
Contenu 4-143
Control 1-14, 2-19, 3-53
Control + '$' 2-23
Control + '0' 2-19
Control + ':' 2-23
Control + '=' 2-23
Control + '^' 2-23
Control + 'B' 2-25
Control + 'C' 2-24
Control + 'D' 2-25
Control + 'E' 2-25
Control + 'F' 2-24
Control + 'H' 2-25
Control + 'I' 2-24
Control + 'K' 2-25
Control + 'L' 2-25
Control + 'M' 2-23
Control + 'P' 2-25
Control + 'Q' 2-24
Control + 'R' 2-25
Control + 'S' 2-25
Control + 'T' 2-24
Control + 'ù' 2-23, 2-25
Control + 'V' 2-25
Control + 'W' 2-25
Control + 'X' 2-25
Control + 'Y' 2-25
Control + 'Z' 2-25
Control + A droite 2-21
Control + A gauche 2-21
Control + Delete 2-21
Control + Help 2-23
Control + Home 2-20
Control + Vers le bas 2-21
Control + Vers le haut 2-21
Copie de bloc 2-27

Créer catalogue 3-38
Crochet 3-55
Curseur 2-26, 3-40
Curseur droit A-157
Curseur gauche A-157
Cursor down 1-14
Cursor left 1-15
Cursor right 1-15
Cursor up 1-14

D

dbra d0 3-46
Débogueur 1-10, 2-153
Débogueur GFA 1-148
Défilement 2-27
Définition de macro 1-75
Définition du symbole 1-74
Delete A-158, 2-21
Désassemblage 1-83
Directive .DS 1-87
Directives de protocole 1-73
Disques RAM 3-35
Double clic 1-15

E

Ecran d'édition 2-17
Ecran d'état 1-15
Ecran de sélection de fichier 3-38
Ecriture des macros 1-70
Ecriture des symboles 1-70
Editeur 1-8, 3-43
Editeur de texte source 1-149
Effacer module 4-140
Effacer source 3-37
Enter 1-15, 2-22
Entrée d'exportation B-169
Entrée de module B-169
Erreurs de syntaxe 3-46
Erreurs fatales 1-84
Esc A-157, 3-38
Escape 1-13, 2-18, 2-24, 2-26
Etablissement de références croisées ..1-74
Exportations 1-83

F

F1 2-22
F10 2-23
F2 2-22
F3 2-22
F4 2-22
F5 2-22
F6 2-22
F7 2-23
F8 2-23
F9 2-23
Feuilleter le texte 2-19
Fichier ASCII 3-45
Fichier d'index B-169
Fichier IMG (Format GFA Basic) .. B-167
Fichier objet 1-83
Fichiers ABS B-167
Fixer 3-52
Fonction des boutons 1-70
Fonctions 3-62
Format de page 2-116
Format DRI 1-83
Format tokenisé B-170, 3-46
Formatage 1-9
Forme courte 1-82

G

GEMDOS 1-147
Global 3-48, 3-56

H

Haute résolution B-12
Header B-168, 3-135
Home A-158, 2-20

I

Icône 3-52
If-Else-Endif 2-107
Imbrication 2-98
Importations 1-83
Imprimer en arrière-plan 3-64
Include 1-71, 1-74, 1-84, 1-92, 2-106
Index 4-142
Informations de relogement B-164
INLINE 1-151
Insert 2-22
Insert 1 2-18
Insert 2 2-18
Instructions 3-46
Instructions préprogrammées 3-53
Interdiction 1-69
Interface utilisateur B-12
Interruption 3-62

J

Jeu d'instructions 1-70
Joker '?' 3-59
Joker'*' 3-59

K

Koctets 3-64

L

Label 2-26, 2-122, 3-48
Langage machine 1-70
Langages de programmation 3-51
Lecteur actuel 1-14
Ligne 2-26
Ligne d'état 2-18
Ligne de menu 2-27, 3-33
LINEA 2-112
Linkage 3-129
Linker 1-83, 1-91, B-165, 3-129
Liste ..... 1-72 - 1-73
Listes de références croisées 1-74
Longmot 2-113

M

Macro 2-19, 2-26, 2-93
Majuscules 1-70
Marques 3-57
Masque d'extension 1-14
Mémoire pleine (Mesures à prendre) C-173
Menu 1,0 3-38
Menu 1,1 3-39
Menus 3-33
Minuscules 1-70
Mnémoniques ..... 3-47 - 3-48
Mnémoniques assembleur 1-69
Mode assembleur 2-19
Mode d'écriture actuel 2-18
Modules objet B-164
Moniteur 3-44
Mot 2-113
Mot d'attribut 1-74
Mot d'identification B-168
Move 3-47
Move.w d0,4 (pc) 3-46
Multi-tâches 3-65

N

Nom d'imprimante A-160
Nom_macro 2-95
Noms de symbole 1-70

O

Obtention d'un code objet 1-82
Octet 1-75, 2-113
Offset 1-75, 1-85
Ok 3-41, 4-142
Opérandes 1-69, 3-46
Opérateurs de calcul 2-125
Opérateurs de label 2-94

Opérations de chargement 1-71
Optimisation 1-75
Overwrite 2-19

P

Page x 3-40
Paramètre 1-90
Paramètre de communication 2-98
Paramètres homonymes 2-99
PATH 2-106
Pavé numérique 2-24
Périphérique de sortie 1-85, 2-115
Placer les opérandes 1-69
Pointeur de pile 1-70
Points d'entrée 2-123
Processeur 1-70
Programme 1-72
Programme exécutable B-163
Programme GEM 2-27
Programmes batch 1-149
Programmes GEM 1-147
Pseudo-codes opération 1-70

R

Ram 1-148
Recherche de chaîne de caractères ... 3-62
Recherche de label 3-62
Recouvre 2-19
Références croisées 1-72, 1-74
Registre d'adresse 1-88
Répertoire 3-38
Répertoire actuel 1-14
Répertoire racine 1-15
Résultat booléen 2-125
Return 1-15, 2-22, 3-38
RVB 3-44

S

Sauver 1-15
Sauver modifications 1-15, 3-37
Sauver module 4-140
Sauver tout 3-39
Segment .ABS 1-87
Segment .BSS 1-87
Segment de données initialisé 1-85
Segment de données non initialisé 1-85
Segment de texte 1-85
Sélection de fichier GEM 3-34
Séquences de commande
pour le formatage A-160
Shift 1-14, 3-53, 4-140
Shift + A droite 2-20
Shift + A gauche 2-20
Shift + Backspace 2-22
Shift + Control + → 2-21
Shift + Control + ← 2-21
Shift + Curseur droit A-158
Shift + Curseur gauche A-158
Shift + Delete 2-21
Shift + Home 2-20
Shift + Insert 2-19, 2-22
Shift + pavé numérique 3-52
Shift + Vers le bas 2-20
Shift + Vers le haut 2-20
Sortie 1-69, 1-74
Sortie sur imprimante 2-118
Source assembleur 1-13
Spooler 2-118, 3-62
Structure de la table de symboles .. B-166
SUBTTL 2-117
Supprimer 3-52
Symboles 3-50
Symboles de flèche 3-59
Symboles globaux 1-84, 2-124
Symboles locaux 2-123
Symboles normaux 2-123
Symboles relogeables 2-122
Système d'exploitation B-12

T

TAB 2-22
Table d'instructions 3-54
Table de configuration 3-63
Table de conversion de caractères .. A-160
Tables de conversion 3-62

Tabulations 3-52
Taille de buffer 2-117, 3-64
Texte 1, Texte 2 3-39
Textes sources tokenisés 1-69
Tokenisation 1-9
Touches 2-19
Touches curseur 2-17
Touches du clavier 3-53
Transfert de bloc 2-110
TTL 2-117
Type d'imprimante A-160

U

Undo 1-14, 2-23
Utiliser le nom du source 1-72

V

Variable/valeur absolue 1-74
Vers le bas 2-20
Vers le haut 2-20
Version courte 1-75
Version far 1-75
Vitesse d'assemblage 1-71

X

X 3-36

GFA Debugger

Sommaire

1. Mode d'emploi du débogueur 189
1.1. L'éditeur 189
1.2. Lancement du GFA Debugger 189
1.3. Définition du clavier 191
1.3.1. Les touches d'arrêt du programme 194
1.4. Rôle de la souris 195

2. Instructions du débogueur 197
2.1. Débogage 198
e {'Nom de fichier'}{,nmb}; 198
ek 'Chaîne' 199
ee 'Chaîne' 200
v 201
new 202
ls {'de'}{,'à'} 202
ln {de,à} 203
diffscr 203
samescr 203
g {ad}{,breakpoint} 204
c {ad}{,breakpoint} 205
bssc 205
initx 206
b0= ... b9= ad{,Compteur de parcours} 206
b 207
bc 207
x 208
x liste 208
t {nmb} 209
ta 209
regon 210
memon1 Expression1
memon2 Expression2 210
mv 212
regoff 212
u {nmb} 212
qon 212
qoff 213
ql {de{,à}} 213
usm Expression, us0 Expression ... us9 Expression 214
uv 218
ul 219
! Instruction assembleur 219

o Liste220
ob Liste 222
co 222
oboff 223
ooff 223

2.2. Manipulation de la mémoire 223
d {de{,à}}
da {de{,à}}
dw {de{,à}}
dl {de{,à}} 223
$ octets,chaînes,mots,longs mots
$a octets ou 'Chaîne'
$w mots
$l longs mots 225
L {de{,à}} 226
dlon 227
dloff 228
cson / csoff 228
hex, dec, oct, bin 228
[Code hexa] {Symbole:} Mnémonique
{Opérande{,Opérande}} 229
fl {'a',}'Nom de fichier',de,à 230
is 'Nom de fichier' 231
hn de,à,ad ou hn de,à,'Chaîne' 234
hf de,à,'Partie d'instruction' 234
hm de,à,'Instruction' 236
mc zone1,zone2 236
mmc zone1,zone2 237
m de,à,vers 238
mcl de,à 239
fill Nombre,Liste 239

2.3. Opérations disque 240
dir {'L'} ou dir {'Chemin'} 240
setdrv 'L' ou setdrv n 241
delete 'Chemin :\nomdefichier.extension' 241
r 'Fichier'{,ad} 242
w {'Fichier'{,ad,len}} 242
rv 243
rsec {'L',}de,nmb{,ad} 244
wsec {{'n',}de,nmb{,ad}} 244
secno ad 245
getbpb {'L'} 245
sclu {'L',}clu ou sclu {'L',}-sec 246
fclu {'L',}déb{,mod}
lclu {'L',}déb{,mod} 246

eclu {'L',}déb,nmb
leclu {'L',}déb,nmb 247

2.4. Autres instructions 250
scrollon, scrolloff 250
pon, poff 250
" 250
h Expression 251
k 251
asm {nmb} 251
resident {nmb} 252
quit 253
doreset 253

3. Evaluation de fonction 255

Annexe A : Structure de la disquette 261

Annexe B : Liste alphabétique des commandes du débogueur 265

Chapitre 1

Mode d'emploi du débogueur

1.1. L'éditeur

Contrairement à beaucoup de débogueurs antérieurs, l'éditeur du GFA Debugger travaille orienté écran, c'est-à-dire que vous n'êtes plus contraint d'effectuer toutes vos entrées dans la ligne unique que vous attribue le débogueur et que vous pouvez positionner le curseur librement, où bon vous semble sur l'écran, et corriger, par exemple, les erreurs d'entrée directement sur la ligne fautive.

L'éditeur du débogueur offre, en outre, des fonctions précieuses pour permettre une entrée plus commode.

C'est ainsi que vous pouvez travailler sur quatre écrans différents du débogueur et que vous disposez en plus d'un écran spécial pour le programme à déboguer. Que ce soit dans les listings du désassembleur, dans les Dumps, dans le répertoire disque ou dans la mémoire Trace back, vous pouvez faire défiler le texte aussi bien en avant qu'en arrière. Des lignes de texte peuvent être très rapidement copiées, avec la souris, à l'emplacement actuel du curseur, etc...

1.2. Lancement du GFA Debugger

Le GFA Debugger figure sur la disquette programme en deux versions

- La version résidente en mémoire (GFA-DBGA), qui peut être copiée dans le dossier Auto du lecteur de lancement du système. Dans ce cas, le débogueur sera automatiquement installé en mémoire lorsque le système sera

booté. Il pourra ainsi intercepter des erreurs éventuelles du système et donner à l'utilisateur la possibilité de rechercher dans les programmes les causes de dysfonctionnement. Cette version résidente peut aussi être appelée après installation, à l'aide du programme 'CALL_DEB.PRG' ou sous le GFA Assembleur (Menu 6,7).

➥ La version "normale" (GFA-DBG) : Le débogueur peut être appelé à partir du bureau ou à partir du GFA Assembleur (à l'aide du point 6,6 du menu, Charger programme). Le nom d'un fichier de programme ou d'un fichier objet peut être communiqué au GFA Debugger à cette occasion (sous le bureau : renommer le débogueur avec .TTP ou APP suivant version de TOS ou le déclarer comme .TTP, sous l'assembleur : inscrire le nom voulu dans la ligne de commande). On peut aussi charger un programme à déboguer à l'aide de la commande "e nom fichier", "g" lance le déboguage, "Quit" l'arrête.

Lors de son lancement, le débogueur détecte si une exception est survenue auparavant. Si c'est le cas, il charge les registres que le système d'exploitation avait sauvegardé lors de l'exception. Il est ainsi encore possible, après coup, de déterminer la cause d'une exception.

Remarque :



Le GFA Debugger utilise le Trap #6 pour des accès internes à la mémoire.

1.3. Définition du clavier

La définition du clavier recoupe largement celle du GFA Assembleur.

Les touches suivantes servent à commander le curseur :

Curseur haut :
Curseur vers le haut.

Curseur bas :
Curseur vers le bas.

Curseur gauche :
Curseur vers la gauche.

Curseur droit :
Curseur vers la droite.

Control + Curseur haut :
Défilement du curseur vers le haut.

Control + Curseur bas :
Défilement du curseur vers le bas.

Lors d'un défilement du curseur, le curseur doit se situer sur une ligne où figure une ligne de Dumping de la mémoire, de listing assembleur, de répertoire disquette, ou d'affichage Trace back. Dans ce cas, le curseur peut être déplacé vers le haut ou le bas, à l'intérieur de la fonction concernée, comme le curseur de texte sous un éditeur ordinaire.

Shift + Curseur haut :
Curseur dans le coin supérieur gauche.

Shift + Curseur bas :
Curseur dans le coin inférieur gauche.

Shift + Curseur gauche :
Curseur au début de la ligne.

Shift + Curseur droit :
Curseur à la fin de la ligne.

Home :
Curseur dans le coin supérieur gauche.

Shift + Home :
Effacer l'écran.

Delete :
Supprimer caractère sous le curseur. Tout ce qui se trouve à droite du curseur est ramené d'une position sur la gauche.

Backspace :
Supprimer caractère à gauche du curseur. Tout ce qui se trouve sous le curseur et à droite du curseur est ramené d'une position sur la gauche.

Control + Delete :
Supprimer une ligne.

Ctrl + Curseur droit :
Supprimer ligne à partir de la position du curseur.

Ctrl + Curseur gauche :
Supprimer ligne jusqu'à la position du curseur.

Shift + Ctrl + Curseur droit :
Vider écran à partir de ligne du curseur.

Shift + Ctrl + Curseur gauche :
Vider écran jusqu'à ligne du curseur.

Insert :
Insérer un espace à l'emplacement actuel du curseur.

Shift + Insert :

Commutation des modes 'Insert' et 'Recouvre' (effacer-remplacer).

TAB :

Le curseur est placé sur la prochaine position de tabulation. Les positions de tabulation sont disposées tous les 10 caractères. Dans les lignes contenant, par exemple, un listing désassemblé, un Dumping ou un répertoire disquette, le curseur est placé sur la prochaine position "significative".

Return :

Entrée de la ligne actuelle pour exécution.

Enter :

Comme Return.

F1 à F4 :

Commutation des quatre écrans du débogueur.

Escape :

Alterner l'écran entre l'écran programme à déboguer et le débogueur. Une touche quelconque permet de revenir au débogueur. Cette touche permet de consulter ce que le programme à déboguer a affiché sur l'écran.

Remarque :



Cette commutation ne focntionne que s'il n'a pas été spécifié auparavant, avec l'instruction 'samescr', que le débogueur doit utiliser le même écran que le programme chargé à la suite.

Control + Help :

Impression du contenu de l'écran.

Pavé numérique :

En tenant la touche Shift enfoncée, vous pouvez entrer sur le pavé numérique le code ASCII (en décimal) d'un

caractère qui sera affiché à l'emplacement du curseur dès que vous relâcherez la touche Shift. Vous pouvez ainsi utiliser sous le débogueur tous les caractères ASCII de 1 à 255 (par exemple pour les insérer dans des chaînes de recherche ou pour les écrire directement dans la mémoire avec l'instruction '$', Entrée mémoire.

Chiffres 0 à 9 :

En tenant la touche Alternate enfoncée et en appuyant sur l'une des touches de chiffre 0 à 9, vous pouvez stocker provisoirement jusqu'à 10 lignes avec l'emplacement du curseur. Vous pouvez ensuite rappeler ces lignes en tenant la touche Control enfoncée et en appuyant sur la touche de chiffre voulue. Cette fonction permet aussi de transférer des lignes d'un des quatre écrans du débogueur à l'autre.

1.3.1. Les touches d'arrêt du programme

Ces touches permettent d'arrêter le programme en cours en fonction de différentes conditions.

Ctrl+Alt+Shift droite :

Arrêt du programme. Cette fonction vous permet d'arrêter le programme lancé sous le débogueur (ou le programme exécuté si le débogueur est résident). Même un programme qui s'est planté et qui a modifié des paramètres peu importants peut être arrêté de cette manière.

Shift gauche +Shift droite :

Arrêt du programme lors du prochain appel Trap à observer (voyez le chapitre 2.1).

Shift gauche +Shift droite+Ctrl :

Arrêt du programme lors du prochain Trap ou lors du prochain Breakpoint rencontré (quel que soit le contenu du compteur de parcours).

Les 2 dernières combinaisons de touches indiquées permettent de produire une situation voulue dans le programme pour arrêter ensuite le programme à un endroit déterminé.

1.4. Rôle de la souris

➥ **Clic simple à gauche :**

Le curseur est placé sur la position de la souris.

➥ **Double clic à gauche :**

Si la flèche de la souris se trouve sur un label, c'est le Dump de la mémoire ou le listing désassemblé correspondant à la valeur du label qui sera sorti à partir de la ligne suivante. Pour les labels dont l'adresse pointe sur le segment de texte, il y aura désassemblage alors qu'un Dumping de la mémoire sera sorti pour d'autres labels.

➥ **Bouton droit tenu enfoncé :**

Tout ou partie d'une ligne peut être marqué avec la flèche de la souris. La zone marquée est affichée en inversion vidéo, pour contrôle. Dès que le bouton droit de la souris est relâché, la zone marquée est copiée sur la position du curseur. En mode Insert, cette zone s'insère dans le texte en l'écartant, alors qu'en mode Recouvre, elle recouvre et efface le texte situé dans l'emplacement du curseur. Cette fonction permet, par exemple, d'ajouter facilement à une instruction des adresses ou des chaînes figurant déjà autre part sur l'écran.

➥ Si le défilement souris a été activé avec l'instruction 'scrollon', le fait que la flèche de la souris bute sur le bord supérieur ou inférieur de l'écran entraînera le même défilement, dans une fonction, que les combinaisons de touches Control + 'Curseur haut' ou Control + 'Curseur bas'. Il est ainsi possible de parcourir la mémoire ou un répertoire disque, etc. très rapidement et très facilement.

Chapitre 2

Instructions du débogueur

Ce chapitre énumère toutes les instructions du débogueur, réparties par groupes logiques. Pour chaque instruction est tout d'abord indiquée sa syntaxe, avec les paramètres possibles, suivie d'une rapide définition de sa fonction, puis d'une explication détaillée et enfin, si nécessaire, d'un ou plusieurs exemples.

Les paramètres figurant {entre accolades} peuvent être omis. Dans ce cas, des valeurs par défaut remplacent automatiquement les paramètres omis.

Des symboles (par exemple 'init') ou des expressions complexes (calculs avec symboles et constantes, voyez la section 2.4) peuvent être transmis comme paramètres aux instructions du débogueur. C'est pourquoi il est recommandé, pour tester un programme, de faire doter ce programme d'une table de symboles lors de l'assemblage ou du linkage (voyez à cet effet le guide de l'assembleur).

Remarque :



Les instructions 'qon' et 'dir' utilisent en commun un même buffer de données. Lorsque l'une de ces instructions est exécutée, le contenu du buffer est vidé. Il en résulte donc qu'un répertoire disquette lu, par exemple, avec 'dir' restera disponible jusqu'à ce qu'un nouveau répertoire soit chargé ou jusqu'à ce que l'instruction 'qon' soit entrée.

2.1. Débogage

e {'Nom de fichier'}{,nmb};

Charge un programme ou code objet à déboguer dans la mémoire (Le code objet est identifié par l'extension '.O' dans le nom de fichier). 'Nom de fichier' est le nom du programme qu'il s'agit de charger, 'nomb' le nombre maximum de symboles que l'utilisateur veut redéfinir.

Lorsqu'un programme est chargé, le système d'exploitation (GEMDOS) réserve pour cela la totalité de la mémoire disponible. Celle-ci reste occupée jusqu'à ce que le programme chargé en libère volontairement certaines sections. C'est pourquoi il n'est jamais possible de traiter plus d'un programme à la fois.

Lorsque le débogueur charge un programme exécutable, il ne reste donc plus de mémoire disponible. Cela n'affecte, cependant, pas le fonctionnement du débogueur lui-même.

Il en va autrement lorsqu'un code objet est chargé : Les codes objet ne sont pas directement exécutables car ils comportent souvent des références non résolues (c'est-à-dire des symboles à définition externe) et car leurs informations de relogement présentent un format différent de celles des programmes exécutables. Toutefois, lorsque le débogueur charge un fichier O, il le prépare de telle façon qu'il puisse être traité comme un programme exécutable. Aucune mémoire complémentaire n'est cependant réclamée au-delà de la longueur du fichier. Par ailleurs, les symboles non définis pour les raisons que nous venons d'indiquer, peuvent causer des mauvais fonctionnements en cas d'exécution séparée de certains modules objet !

Si le programme ou le module objet voulu contient une table de symboles, les listings produits par le désassemblage (voyez l'instruction 'l') seront beaucoup plus lisibles. Lorsqu'on déve-

loppe des programmes, il est vivement recommandé de ne pas se priver de cette facilité et donc de toujours faire ajouter, à la suite d'un programme, par l'assembleur/linker des symboles pour faciliter le débogage.

D'autre part, ces symboles ainsi éventuellement que d'autres symboles définis par l'utilisateur, peuvent être utilisés dans tous les calculs.

Les annexes du guide d'utilisation de l'assembleur vous fourniront de plus amples détails sur les formats de données des programmes et des modules objet !

Si le nom de fichier est omis, la boîte de sélection de fichier de GEM apparaît pour vous permettre de sélectionner le fichier voulu de la façon la plus commode. Il en va de même pour les instructions 'is', 'fl', 'r' et 'w'.

Exemple :

```
e ,50
```

chargera un fichier de programme ou un fichier objet à travers la boîte de sélection de fichier et réservera la place mémoire nécessaire pour 50 symboles définis par l'utilisateur.

ek 'Chaîne'

inscrit une chaîne de caractères comme ligne de commande dans la Base page d'un programme chargé. 'Chaîne' est la chaîne de caractères à inscrire dans la ligne de commande.

Les programmes évaluant la ligne de commande portent normalement l'extension '.TTP' ou 'APP'. Cela entraîne, lors du lancement du programme à partir du bureau, l'apparition d'une boîte dans laquelle la ligne de commande peut être entrée. Ou

bien les programmes sont déclarés comme application pour des fichiers dotés de certaines extensions et l'utilisateur effectue un double clic sur ces fichiers. Le nom de fichier est alors automatiquement inscrit par le GEMDOS dans la ligne de commande du programme.

Ces possibilités n'existent généralement pas lors du débogage. C'est pourquoi l'instruction 'ek' permet, sous le débogueur, d'entrer une chaîne dans la ligne de commande.

La ligne de commande, dans la page de base du programme, présente la structure suivante :

longueur.b :	Longueur de la ligne de commande, y compris le 'cr' terminal
contenu.b :	Chaîne de caractères
13.b :	'cr' comme terminaison de la ligne

Avec l'instruction 'ek', seule la chaîne elle-même doit être spécifiée. La longueur et le 'cr' terminal sont produits automatiquement.

Exemple :

```
e 'editor.prg'      -> charger le programme 'editor.prg'
ek 'test.doc'       -> lui transmettre la ligne de commande 'test.doc'
g                   -> lancer le programme
```

ee 'Chaîne'

Inscrit l'adresse de chaînes de caractères dans la page de base d'un programme chargé, comme environnement. 'Chaîne' représente les chaînes de caractères placées dans un buffer et dont l'adresse doit être inscrite comme environnement dans la page de base du programme. Les chaînes sont séparées entre elles par un octet nul. La fin de la liste des chaînes est marquée par deux octets nuls consécutifs.

Les chaînes d'environnement sont utilisées sous MS-DOS pour transmettre au programme des informations sur l'environnement du système (par exemple le chemin de recherche par défaut). Cette possibilité n'est, toutefois, pas utilisée par le système d'exploitation de l'Atari ST. Elle peut donc être employée par l'utilisateur pour échanger des données entre programmes.

Exemple :

Un programme charge un autre programme à la suite et transmet son nom à ce dernier ou bien l'adresse d'une zone de données que le programme chargé à la suite pourra/devra exploiter.

v

Affiche les paramètres d'un programme ou code objet chargé à la suite.

Après un chargement, l'éditeur dispose d'un certain nombre d'informations importantes sur le programme chargé, telles, par exemple, que les adresses et longueurs des trois segments TEXT, DATA et BSS. Comme ces informations sont souvent nécessaires lors du débogage, elles peuvent être obtenues aussi bien à l'aide de l'instruction 'v' qu'à l'aide des fonctions :

^^data	pour l'adresse de la zone des données
^^text	pour l'adresse de la zone texte
^^bss	pour l'adresse de la zone BSS

new

Libérer la mémoire de travail.

Lorsqu'un programme a été chargé avec l'instruction 'e', toute la mémoire a été réservée pour cela. Cela signifie qu'aucun autre programme ou code objet ne peut plus être chargé. L'instruction 'new' permet ensuite de libérer la mémoire de travail pour pouvoir charger un nouveau programme.

Du fait de plusieurs erreurs dans le GEMDOS, cette instruction ne fonctionne pas toujours correctement.

ls {'de'}{,'à'}

Sortir une liste alphabétique des symboles. 'de' est la séquence de caractères à partir de laquelle les symboles doivent être affichés, 'à' la séquence de caractères jusqu'à laquelle ils doivent l'être.

La section de table voulue doit être spécifiée un peu comme dans l'annuaire téléphonique. La sortie se fait sur trois colonnes. Chaque symbole est suivi de sa valeur et de son type de données :

T signifie	:	appartenant au segment de données
D signifie	:	appartenant au segment de texte
B signifie	:	appartenant au segment de BSS
X signifie	:	défini à l'extérieur
A signifie	:	définition absolue
C signifie	:	type de données COMMON

Voyez aussi, à ce propos, les explications correspondantes dans la partie assembleur/linker de ce manuel. L'affichage peut être suspendu avec la barre 'espace' ou définitivement interrompu avec la touche 'Esc'.

In {de,à}

Afficher une liste des symboles classés numériquement. 'de' est la valeur à partir de laquelle les symboles doivent être affichés, 'à' la valeur jusqu'à laquelle ils doivent l'être.

Même fonction que 'ls', la seule différence résidant dans le critère de tri.

diffscr

Deux écrans différents seront utilisés pour le débogueur et le programme.

Le GFA Debugger étant à même de distinguer son écran propre de celui du programme à déboguer, le programme peut véritablement travailler dans son écran sans être "dérangé" et les sorties du programme ne sont pas effacées ou décalées par les messages du débogueur. Le GFA Debugger peut même déboguer en résolution moyenne un programme travaillant en faible résolution avec une autre palette de couleurs. La touche 'Esc' commute l'écran du programme pour observation. On peut alors revenir à l'écran du débogueur avec une touche quelconque.

samescr

Un seul et même écran sera utilisé pour le débogueur et le programme.

Dans certains cas particuliers, il peut être souhaitable que le débogueur et le programme travaillent dans le même écran, par exemple lorsque le programme effectue peut de sorties et qu'on veut le suivre à la trace. Avec un écran unique, on évite, dans

ce cas, d'avoir à commuter en permanence les deux écrans et on évite aussi le tremblotement qui en résulte lors du traitement pas à pas.

g {ad}{,breakpoint}

Lancer le programme. 'ad' est l'adresse à laquelle doit commencer l'exécution du programme. Si 'ad' n'est pas spécifiée, le PC actuel sera pris comme valeur par défaut. 'breakpoint' est l'endroit du programme où l'exécution devra être interrompue. Si 'breakpoint' est entré, il est sauvegardé comme breakpoint 0 (voyez à ce propos l'instruction 'b0...b9='), sinon aucun breakpoint n'est modifié.

L'instruction 'g' lance le programme directement. Si une adresse est spécifiée à cette occasion, le programme sera lancé à partir de cette instruction, sinon il sera lancé à partir du PC actuel (qui peut être appelé à l'aide de x). Lors du chargement d'un programme ainsi que lors de l'affichage des paramètres du programme (instruction 'v'), le PC est fixé sur le début du segment TEXT.

Comme second paramètre peut être spécifiée l'adresse d'un breakpoint, c'est-à-dire d'un emplacement où l'exécution du programme devra être interrompue. Un programme lancé avec 'g' peut être terminé par un appel Term de GEMDOS ou être interrompu par un breakpoint (voyez l'instruction b0...9=), un Trap (voyez l'instruction 'ob') ou par un arrêt du programme (voyez le chapitre 1.3.1).

Exemple :

```
e 'test.prg'     ->  Charger le programme
g ,point1        ->  Lancer le programme, l'interrompre au 'point1'.
                     (Remarque : le PC a été fixé par l'instruction 'e' sur
                     le début du segment TEXT.)
```

C {ad}{,breakpoint}

Appeler un sous-programme, puis retour au débogueur.

L'instruction 'c' a pratiquement le même effet que l'instruction 'g', si ce n'est qu'elle place, en plus, sur la pile une adresse de retour. Ainsi peut être exécutée une sous-routine terminée par un RTS.

Exemple :

```
r 'test.img'      ->  Charger un sous-programme pouvant être décalé
                      sans restriction
c ^^memad         ->  L'exécuter (à propos de ^^... voyez la section 2.4)
```

bssc

Effacer le segment BSS du programme.

Cette instruction efface le segment BSS du programme chargé à la suite. Si le programme a été construit correctement, c'est-à-dire notamment s'il ne se modifie pas lui-même, ou plus exactement s'il ne modifie pas son segment TEXT ni son segment DATA, il peut être relancé du début après 'bssc'. Si le programme a déjà effectué une réservation de mémoire avec la fonction GEMDOS 'Mshrink', il est déconseillé que cette fonction soit à nouveau exécutée. Il faudrait donc, dans ce cas, que le PC soit fixé à la suite de l'appel de fonction correspondant.

Exemple :

```
e 'test.prg'      ->  Charger le programme
g                 ->  Lancer le programme

bssc              ->  Effacer la zone BSS
Init x
g ^^text          ->  Relancer le programme au début
```

initx

Effacer le contenu des registres.

Cette instruction vide tous les registres de données et d'adresse. PC, USP, SSP et SR sont fixés sur leurs valeurs défaut.

b0= ... b9= ad{,Compteur de parcours}

Fixer un breakpoint. 'ad' est l'adresse à laquelle le programme doit être interrompu.

'Compteur de parcours' définit combien de fois le breakpoint devra être parcouru avant que le programme ne soit interrompu. Si 'Compteur de parcours' n'est pas spécifié, le programme sera interrompu la première fois que cet endroit sera atteint. (Ne pas oublier l'espace entre'=' et l'adresse !).

Un breakpoint est un point du programme où l'exécution du programme doit être interrompue. Cela peut, par exemple, être un endroit où l'on souhaite examiner un certain nombre de paramètres pour retrouver une erreur dans le programme.

On peut faire exécuter, par exemple, le programme jusqu'à un endroit situé juste avant la section du programme supposée renfermer une erreur. On peut alors poursuivre l'exécution du programme, à partir de cet endroit, en mode pas à pas, pour localiser l'erreur plus précisément.

Exemple :

```
e 'test.prg'            ->  Charger programme défectueux
b1= point2,3            ->  Interrompre le programme à cet endroit lors du
ou B1= $Adresse{,1}         troisième parcours
g ,point1               ->  Exécuter le programme, 'point1' étant pris comme
                            breakpoint 0 et le programme étant interrompu à
                            l'endroit 'point1' dès le premier parcours.
```

Dans cet exemple, le programme sera interrompu en deux endroits, au 'point1' lors du premier parcours ou au 'point2' lors du troisième parcours. L'emploi de plusieurs breakpoints indépendants permet de retrouver une erreur pouvant figurer dans plusieurs routines.

b

Afficher la liste des breakpoints.

La liste des 10 breakpoints possibles est affichée. L'adresse '$0' sera annoncée pour tous les breakpoints non fixés. Si un breakpoint est doté d'un compteur de parcours différent de 1, ce dernier est sorti à la suite de l'adresse correspondante. Le compteur de parcours est diminué d'une unité lors de chaque parcours de l'adresse correspondante. C'est pourquoi le compteur de parcours actuel (qui peut être différent de la valeur de départ) est toujours affiché dans cette liste si cet emplacement du programme a déjà été parcouru une ou plusieurs fois.

bc

Supprimer tous les breakpoints.

Tous les breakpoints fixés sont supprimés. De ce fait, le programme ne sera plus interrompu par les breakpoints.

X

Afficher le contenu des registres.

Le débogueur note les valeurs de tous les registres du processeur pour le programme à déboguer. Avant qu'un programme ne soit lancé ou qu'une instruction ne soit exécutée, les registres ainsi sauvegardés sont transmis au processeur. Après que le programme ait été interrompu ou qu'une instruction ait été exécutée, les registres sont à nouveau sauvegardés. Ces registres peuvent être affichés avec l'instruction 'x' et être modifiés avec l'instruction 'x liste' (voyez ci-dessous).

x liste

Spécifier le contenu des registres. 'liste' est la liste des registres du processeur auxquels doivent être affectées de nouvelles valeurs.

Pour lancer, par exemple, une sous-routine avec des valeurs de départ bien précises ou pour pouvoir corriger directement des erreurs mineures (valeurs de registres incorrectes), cette instruction permet de modifier la valeur d'un ou plusieurs registres du processeur.

Exemple :

```
x pc= $12345                                          ->  PC sur $12345
x pc= label1, usp= ^^memad+^^memfre, sr= $0300->  PC sur la
                                                      valeur de 'label1', usp
                                                      sur la fin de la mémoire
                                                       et sr sur $0300
x d0= 12                                              ->  d0 sur 12 décimal
x a3= $50000, $12345, ^^resvector                     ->  a3 sur $50000, a4 sur
                                                      $12345 et a5 sur $42a
```

Remarque :



Les registres PC, USP, SSP et SR ne peuvent être fixés que dans cet ordre. Les registres de données et les registres d'adresse peuvent être fixés un par un ou rangés par ordre croissant dans une instruction. Seul le premier registre à fixer doit être spécifié, dans ce cas, ensuite seront fixés autant de registres qu'il y a eu d'entrées effectuées. Si, cependant, vous tentez d'accéder à un registre au-delà de d7 ou a7, un message d'erreur apparaîtra.

t {nmb}

Trace. 'nmb' est le nombre d'instructions devant être suivies à la trace. Si 'nmb' n'est pas spécifié, une seule instruction sera exécutée.

L'instruction Trace vous permet de tester un programme de fond en comble. Cette instruction sort la liste des registres (comme l'instruction 'x') après chaque instruction exécutée. Si vous entrez seulement 't', c'est exactement une instruction, celle figurant sur le PC actuel, qui sera exécutée. Si vous spécifiez également un nombre, un nombre correspondant d'instructions seront exécutées.

ta

Auto-Trace.

Le Trace automatique est une version étendue de l'instruction Trace, qui exécute une instruction chaque fois que vous appuyez sur la barre espace. Lorsque le programme se trouve dans une sous-routine, celle-ci peut être exécutée automatiquement avec la touche Return. Le programme ne sera à nouveau suivi à la trace qu'après retour à la routine d'appel. La touche 'Esc' met fin au mode Trace automatique.

regon

Incruster de façon permanente la liste des contenus de registres dans le haut de l'écran. L'instruction 'regon' représente une importante extension des deux instructions Trace. Cette instruction fait afficher en permanence, en haut de l'écran, la liste des registres avec leurs valeurs actuelles respectives. Cette instruction facilite considérablement le suivi de l'exécution pas à pas d'un programme car la liste des contenus des registres n'apparaît qu'une seule fois sur l'écran et les instructions exécutées sont alors affichées directement à la suite les unes des autres.

Remarques :



Les contenus des registres peuvent être modifiés directement dans la zone incrustée. Lorsque les registres sont affichés de façon permanente, le défilement dans une fonction (voyez la section 1.3) ne se déroule plus que dans la zone située en dessous. L'incrustation permanente des registres peut être fixée séparément pour chacun des quatre écrans du débogueur.

Exemple :

```
e 'test.prg'                                -> Charger le programme
regon                                       -> Incruster les registres
ta                                          -> Auto-Trace activé
<Espace> <Espace> <Espace> <Espace>         -> Exécuter quatre
                                               instructions
<Esc>                                       -> Mettre fin à Auto-Trace
```

memon1 Expression1
memon2 Expression2

Les instructions memon constituent des extensions de l'instruction regon : chaque fois que les registres sont incrustés dans le cadre supérieur de l'écran, un ou deux extraits de dumping

hexadécimal de la mémoire sont affichés en dessous pour les adresses spécifiées par 'Expression1' ou 'Expression2'. Les différentes zones de l'écran sont séparées à cet effet par des lignes horizontales. Les accès à la mémoire d'un programme peuvent ainsi aisément être surveillés.

Expression1 et Expression2 autorisent les mêmes entrées que l'instruction 'usm'. Les sous-conditions définies par us0 à us9 peuvent être intégrées dans les calculs à l'aide de '\n' (0<=n<=9). Toujours comme pour l'instruction usm, les entrées ainsi effectuées sont compilées en petits programmes machine pour augmenter la vitesse d'exécution. Pour détecter les erreurs éventuelles (adresses impaires, etc., voyez 'usm' !), ces programmes sont exécutés une fois dès leur saisie. Il peut donc arriver qu'ils signalent une erreur système !

'memon2' ne doit être utilisé que si 'memon1' a déjà été activé ! L'incrustation d'une zone de mémoire est désactivée par

(a) l'instruction regoff
(b) 'memon1' (désactive les deux zones !) ou bien 'memon2' non suivie d'une expression.

'usm' et 'us0'à 'us9' peuvent également être désactivées en étant entrées sans être suivies d'une expression !

Exemple :

```
memon1 l;a1          -> affichera 32 octets de dumping hexadécimal
                        à partir du contenu du registre 'a1' du
                        processeur
  ta                 -> activer Auto-Trace
  <Espace> <Espace>  -> exécuter 2 instructions à partir du  niveau
                        du PC, la zone de mémoire incrustée
                        étant actualisée en permanence.
```

mv

Les adresses spécifiées avec 'memon1' et 'memon2' peuvent être considérées comme des formules de texte en clair (voyez l'instruction 'uv' !)

regoff

Mettre fin à l'affichage de la liste des contenus des registres.

La zone occupée par la liste des registres est à nouveau libérée.

u {nmb}

Untrace (Trace invisible). 'nmb' est le nombre d'instructions qui doivent être suivie par untrace. Si 'nmb' n'est pas spécifié, une seule instruction est exécutée.

L'instruction 'untrace' ressemble à l'instruction 'trace'. La différence est que les registres du processeur ne sont pas ici affichés après chaque instruction mais seulement après que le nombre d'instructions spécifié aient été exécutées ou après que l'opération ait été interrompue.

qon

Activer Trace back.

'Trace back' signifie que le débogueur note les instructions exécutées et peut les afficher par groupes ultérieurement. Ces instructions sont sauvegardées au cours du travail avec 'untrace' (voyez plus haut). L'instruction 'qon' vide le buffer Trace back

dans lequel le débogueur écrira à nouveau à partir de la prochaine instruction 'u'.

Le GFA Debugger peut stocker jusqu'à 4096 instructions. Il est ainsi possible de faire tourner automatiquement même des programmes très longs et d'examiner par la suite la succession d'instructions exécutées. L'instruction 'ql' affiche la liste des instructions sauvegardées (voyez plus bas).

qoff

Désactiver le mode Trace back.

Désactive le Trace back. Les instructions que le débogueur avait enregistrées peuvent néammoins toujours être consultées.

ql {de{,à}}

Affichage de la liste des instructions sauvegardées dans le buffer Trace back. 'de' est le numéro de l'instruction à partir de laquelle la liste doit être affichée, la valeur défaut étant 0. 'à' est le numéro de l'instruction jusqu'à laquelle la liste doit être affichée. Si 'à' n'est pas spécifié, 20 instructions sont affichées.

Cette instruction permet d'afficher une section quelconque du buffer Trace back. Une ligne vide est sortie après chaque instruction dans laquelle figure un saut du PC (par exemple du fait d'une boucle, d'un appel de sous-programme, d'un retour de sous-programme, etc.). On peut parcourir le listing Trace back avec le défilement de fonction (à l'aide du clavier ou de la souris, voyez les sections 1.3 et 1.4).

Exemple :

```
e 'test.prg'     ->  Charger le programme
qon              ->  Activer Trace back
u 1000           ->  Untrace pour 1000 instructions
ql 200           ->  Liste à partir de la 200ème
                     instruction exécutée
```

usm Expression, **us0** Expression **...** **us9** Expression

Fixer des conditions pour l'interruption sous Untrace.

En fixant des conditions d'interruption, il est possible d'accéder de façon sélective aux endroits défectueux d'un programme. Lorsque certaines conditions définies comme critère d'interruption sont remplies, le débogueur intervient au niveau approprié du PC. Pour permettre une souplesse et une vitesse maximales, une méthode ouverte est appliquée :

Les conditions d'interruption sont constituées par le résultat logique (false, true) de calculs complexes (avec diverses fonctions spéciales). L'entrée s'effectue en clair, comme habituellement sous l'éditeur/assembleur, puis elle est automatiquement compilée en un petit programme machine qui est aussitôt sauvegardé. L'utilisateur peut faire afficher ce programme et le modifier comme il le souhaite. Lors de l'exécution du programme en mode Untrace, ce sous-programme sera appelé après chaque instruction exécutée. S'il renvoie la valeur de vérité true (=-1) en réponse, le programme sera interrompu. Le degré de ralentissement de l'exécution du programme occasionné par ce supplément de calculs est directement fonction de la complexité des conditions d'interruption.

Sousconditions : us0 Expression ... us9 Expression

Spécification d'une sous-condition. Une sous-condition n'est jamais exécutée isolément mais seulement lorsqu'elle est appe-

lée à partir de la condition principale. Les sous-conditions renvoient une valeur booléenne (-1/0).

'Expression' est ici une formule quelconque. Les fonctions spéciales suivantes peuvent être employées outre les fonctions admises pour l'assemblage Inline :

1. Adressage Direct

b;Mode d'adressage - équivaut à la séquence assembleur suivante :

```
move.b    Mode d'adressage,d0   ; Aller chercher valeur
ext.w     d0                    ; Etendre à 'long'
ext.l     d0                    ; Résultat dans 'd0'
```

Ex :

```
! Move.b #$-34,d0      ; ce qui donne D0 = $DE.
                       ; On trouve alors $FFDE dans D0
```

Ainsi peuvent être employés aussi bien des adressages absolus de la mémoire que des adressages de registres du processeur. Les registres reçoivent ici les valeurs fournies par le programme surveillé. Ils ne sont pas modifiés par le test d'interruption.

w;Mode d'adressage - fonction analogue à 'b;Mode d'adressage' avec la largeur de données 'word'. Séquence assembleur équivalente :

```
move.w    Mode d'adressage,d0
ext.l     d0
```

Pour les adressages de la mémoire, l'utilisateur doit veiller à n'appeler que des adresses paires, faute de quoi une erreur bus surviendra, qui sera, cependant, interceptée et annoncée par le débogueur. Il en va de même pour l'accès aux adresses de mémoire non autorisées.

```
l;Mode d'adressage - voyez 'b;', 'w;'. Séquence assembleur :

move.l Mode d'adressage,d0
```

2. Adressage Indirect

b:Mode d'adressage - Les fonctions 'b:'-, 'w:'- et 'l:' représentent des extensions des variantes 'b;', 'w;' et 'l;' correspondantes. On ne s'intéresse souvent pas à la valeur elle-même mais au contenu de la cellule de mémoire désignée par cette valeur, ce qui correspond, en langage machine, à l'opération suivante :

```
move.l #Valeur,a0       ; Valeur=N de la cellule de mémoire
move.x (a0),d0          ; Résultat=Contenu de cette cellule de mémoire

=> 'Valeur' est l'argument d'un adressage indirect.

L'équivalent assembleur de la fonction 'b:' sera donc :

movea.l   ModeDAdressage,a0     ; Aller chercher la valeur
                                ; (toujours .l!)
move.b    (a0),d0               ; Contenu
ext.w     d0                    ; Etendre à 'long'
ext.l     d0                    ; Résultat dans 'd0'
```

Il convient de veiller à ce que 'Mode d'adressage' n'appelle pas des adresses de mémoire impaires ou illégales, faute de quoi il y aurait interruption à la suite d'une erreur de bus !

w:Mode d'adressage - Fonction analogue à 'b:Mode d'adressage' avec une largeur de données 'word'. Séquence assembleur équivalente :

```
movea.l   Mode d'adressage,a0   ; Aller chercher la valeur
                                ; (toujours .l!)
move.w    (a0),d0               ; Contenu
ext.l     d0                    ; Résultat dans 'd0'
```

l:Mode d'adressage - voyez 'b:', 'w:'. Séquence assembleur :

```
movea.l   Mode d'adressage,a0   ; Aller chercher la valeur
                                ; (toujours .l!)
move.l    (a0),d0               ; Résultat dans 'd0'
```

Conditionprincipale : usm Expression

Pour le mode d'emploi, voyez us0 ... us9. Il existe, toutefois, une autre fonction spéciale pour l'intégration des résultats de sous-conditions :

3. Appel de Sous-conditions :

\0 ... \9 -	Appel des sous-conditions us0 ... us9 (Valeurs de réponse -1/0, voyez plus bas)

L'utilisateur intéressé par la question peut comparer les routines compilées avec les critères d'interruption entrés chaque fois, de façon à évaluer le délai d'exécution nécessaire ou pour effectuer des optimisations à l'aide de l'assemblage Inline. Pour vous aider à analyser les produits compilés, indiquons que certains des modes d'adressage possibles sous les fonctions spéciales évoquées sous 1. et 2. ne sont pas stockés tels quels mais émulés par d'autres, pour des raisons techniques.

Exemples :

(a) Entrée de

```
us0 w;d0=100
us1 l;a0=^^physbase
us2 b:(sp)+=%1101
usm (\0&\1)|\2
```

entraînera l'interruption du programme lorsque les sous-conditions us0 et us1 seront remplies ou lorsque la condition us2 sera remplie. Les opérations logiques travaillent bit par bit ! Cela ne fait, cependant, aucune différence car les valeurs de réponse de us0 ... us9 sont, de toute façon, uniquement -1 et 0.

us0 est remplie lorsque le mot faible du registre 'd0' contient la valeur 100.

us1 est remplie quand A0 pointe sur le début de l'adresse écran.

us2 est remplie quand l'adresse contenue dans la pile contient la valeur binaire 1101 (il faut bien entendu que le pointeur de pile soit positionné sur cet emplacement de la pile).

(b) Entrée de

```
usm (w;(a3)+&w;(a3))=%101010
```

entraîne une interruption si la combinaison AND (instruction du processeur) des deux mots placés successivement à l'adresse désignée par 'a3' produit le résultat %101010. Voici quelle serait la teneur d'un programme assembleur équivalent :

```
move.w    (a3)+,d0          ; 1er mot
and.w     (a3),d0           ; 2nd mot, combinaison AND
cmp.w     #%101010,d0       ; Comparer le résultat avec %101010
beq       .Condition_vraie  ; Identité, condition remplie
bne       .Condition_fausse ; Condition non remplie
```

(c) Entrée de

```
us5 w;d5=$10
```

aucun effet car aucune condition principale n'a été définie.

uv

Afficher les conditions.

Les critères d'interruption pour le mode Untrace définis avec les instructions us0 ... us9 et usm sont affichés en clair (formatés). (Ils peuvent être modifiés si vous le souhaitez.)

Note : Lorsque vous entrez vos conditions us0...us9, usm, vous devez rentrer usm en dernier, mais lorsque vous les affichez avec uv, c'est usm qui est affiché en premier.

Exemple :

Si vous aviez, par exemple, entré les conditions indiquées, dans l'exemple (a), pour les définitions us0...us9 et usm, 'uv' vous renverrait :

```
usm (\0&\1)|\2
us0 w;d0=100
us1 l;a0=^^physbase
us2 b:(sp)+=%1101
us3
...
us9
```

ul

Afficher le listing du programme en langage machine produit par les conditions.

Les programmes compilés à partir des conditions d'interruption Untrace sont placés dans un buffer de 4 Ko dans le segment BSS du débogueur. Cette zone peut être désassemblée avec 'ul', à des fins de contrôle et de manipulation. Le début et la fin de la sous-routine de contrôle sont également indiqués. Une routine de contrôle d'interruption se termine par 'rts' et sauvegarde tous les registres utilisés. Si son résultat est 'true', elle fixe le flag Zero dans le registre CCR du processeur.

! Instruction assembleur

Exécuter une instruction assembleur.

Cette instruction de débogage assemble une instruction assembleur dans un buffer et l'y exécute. Ce mode est comparable au 'mode direct' d'un interpréteur de langage évolué. Il est ainsi

possible d'étudier l'effet d'une instruction isolée, voire même de faire exécuter directement un petit programme. L'instruction utilisera tous les registres du processeur pouvant être appelés avec 'x' hormis le PC, qui reste inchangé.

Exemple :

```
$F0000: $ 'Salut, ô utilisateur !',0   -> Stocker la chaîne
$F0012: ! move.l #$f0000,-(sp)         -> Adresse de la chaîne sur la pile
$F0012: ! move #9,-(sp)                -> Appel de Cconws GEMDOS
$F0012: ! trap #1                      -> exécuter
$F0012: ! addq.l #6,sp                 -> Nettoyer à nouveau la pile
```

Remarque :



Nous supposons que $f0000 est une adresse libre dans la mémoire de travail. Toutes les autres adresses sont proposées par le débogueur lui-même.

O Liste

Observer, surveiller les appels Trap. 'Liste' est une liste de tous les appels Trap devant être interceptés. Chaque entrée se compose de deux éléments : le numéro de Trap et le numéro de fonction correspondant à ce Trap.

Les numéros de Trap autorisés sont :

```
1 pour GEMDOS,
115 pour GEM-VDI
200 pour GEM-AES
13 pour BIOS
14 pour XBIOS
```

Les numéros de fonction correspondants sont ceux indiqués par Atari. Si un -1 est transmis comme numéro de fonction, toutes les fonctions de ce Trap seront interceptées.

L'instruction 'o' a été mise au point pour suivre le fonctionnement d'un programme ou pour retrouver rapidement les appels erronés du système d'exploitation dans un programme. Une liste d'appels du système d'exploitation peut être transmise à l'instruction 'o'. Ensuite, lorsqu'un appel du système d'exploitation se produit, le système testera s'il s'agit d'un appel à intercepter. Si c'est le cas, l'appel est affiché avec le numéro de fonction, le nom et les paramètres. Pour les Traps 1, 13 et 14 sont affichés les paramètres figurant sur la pile, avec la longueur correspondante (mot ou long mot) et, s'il s'agit de paramètres contenant un caractère ASCII, avec en plus ce caractère ASCII.

Pour les appels GEM-VDI et GEM-AES sont affichés non seulement le nom et le numéro de la fonction mais aussi les adresses des tableaux de communication. Les fonctions GEMDOS 'Pterm' ($0), 'Ptermres' ($31) et 'Pterm' ($4c) et la fonction GEM 0 (term), qui terminent un programme, sont systématiquement interceptées. Si la fonction du système d'exploitation renvoie une valeur de réponse, celle-ci est sortie après exécution du Trap.

La surveillance des appels Trap s'applique à chaque exécution du programme commandée par le débogueur (c'est-à-dire avec 'g', 'c', 't', 'ta' et 'u').

En ce qui concerne l'interruption conditionnelle lors des appels du système d'exploitation, voyez la section 1.3.1.

Remarque : Pour que vous puissiez suivre les appels du système d'exploitation dans un programme, même lorsque ces appels sont nombreux, nous vous conseillons d'activer également le protocole imprimé avec 'pon' (pon = affichages imprimés ; poff = affichages non imprimés

Exemple :

```
e 'test.prg'        -> Charger le programme
pon                 -> Activer protocole
                       imprimé
```

o 1,-1, 200,10, 200,19, 13,3, 14,0, 14,10	->	Observer tous les Traps GEMDOS, appl_init et appl_exit GEM-AES, Bconout BIOS, Initmous et Flopmt XBIOS
g	->	Lancer le programme

ob Liste

Observer avec interruption après chaque affichage. (Observer + Break).

L'instruction 'ob' ne se distingue de 'o' que par le fait que l'exécution du programme est interrompu après tout affichage de paramètres et après tout affichage de réponse. Les paramètres transmis peuvent ainsi être modifiés à volonté (Cela permet, par exemple, de renvoyer artificiellement un message d'erreur au programme à la suite d'un accès disquette ou, au contraire, d'éliminer artificiellement un message d'erreur, pour tester le comportement du programme dans chaque situation).

Remarque :



Lorsque ce n'est qu'une fois à l'intérieur d'un programme qu'une interruption lors des appels du système d'exploitation apparaît souhaitable, la combinaison de touches <Shift gauche> + <Shift droite> permet de passer du mode sans interruption au mode avec interruption (voyez aussi la section 1.3.1).

co

Continuer après observer. Reprendre l'exécution du programme après une interruption.

Lorsque le programme est interrompu par un appel Trap qui figurait dans la liste de l'instruction 'ob', l'exécution du programme est interrompue et le débogueur se présente à nouveau. Pour faire reprendre l'exécution du programme, il faut entrer l'instruction 'co'.

oboff

Désactiver break avec Observe.

Le mode Observe demeure activé pour les fonctions entrées. Cette instruction a simplement pour effet de ne plus faire exécuter d'interruption après chaque Trap. Cela peut être utile, par exemple, lorsqu'on a interrompu l'exécution du programme avec les deux touches Shift pour examiner les paramètres d'un Trap déterminé et que l'on souhaite maintenant faire reprendre l'exécution du programme avec 'observe', mais sans interruption cette fois.

ooff

Désactiver Observe.

Les réglages de l'instruction 'o' s'appliquent jusqu'à ce qu'ils soient désactivés avec l'instruction 'ooff'.

2.2. Manipulation de la mémoire

d {de{,à}}
da {de{,à}}
dw {de{,à}}
dl {de{,à}}

Sortir les contenus de la mémoire. 'de' est l'adresse à partir de laquelle le contenu de la mémoire doit être affiché. 'à' est l'adresse jusqu'à laquelle le contenu de la mémoire doit être affiché. Si 'à' est omis, 11 lignes de contenu de la mémoire seront affichées.

Si les deux paramètres sont omis, 11 lignes seront affichées à partir de l'adresse à laquelle l'affichage avait été effectué par la dernière instruction 'dx'.

Les quatre instructions 'd', 'da', 'dw' et 'dl' affichent le contenu de cellules de mémoire. Elles ne se distinguent que par le format d'affichage :

'd'	affiche 16 octets sous forme de chiffres hexadécimaux et d'une chaîne ASCII.
'da'	affiche 64 octets sous forme d'une chaîne ASCII.
'dw'	affiche 8 mots sous forme de chiffres hexadécimaux et d'une chaîne ASCII.
'dl'	affiche 4 longs mots sous forme de chiffres hexadécimaux et d'une chaîne ASCII.

L'affichage de la mémoire peut être suspendu avec la barre <espace> (appuyez à nouveau sur <espace> pour relancer l'affichage) et être interrompu définitivement avec la touche <Esc>.

Vous pouvez parcourir le listing du contenu de la mémoire à l'aide du défilement de fonction (à l'aide du clavier ou de la souris, voyez les sections 1.3 et 1.4).

Exemple :

dl 0	->	11 lignes d'affichage de longs mots à partir de l'adresse 0
dw ^^text	->	11 lignes d'affichage de mots à partir du début du segment TEXT
da chaine,chaine+100	->	Affichage ASCII de 100 octets à partir du symbole 'chaine'
d ^^data,^^bss	->	Affichage d'octets du segment DATA

$ octets,chaînes,mots,longs mots
$a octets ou 'Chaîne'
$w mots
$l longs mots

Entrée de valeurs dans des cellules de mémoire. Seuls nombres hexadécimaux et les chaînes de caractères sont admis comme valeurs d'entrée.

Cette instruction permet d'attribuer un nouveau contenu à une ou plusieurs cellules de mémoire à partir de l'adresse actuelle du prompt (message d'attente du débogueur). Les quatre instructions servent à placer des valeurs différentes dans la mémoire :

'$' place n'importe quelles valeurs dans la mémoire. Pour les entrées hexadécimales, c'est en fonction de la longueur de l'entrée que la valeur sera stockée comme un octet, un mot ou un long mot :

Longueur de la valeur = 1 à 2 caractères	: Un octet sera stocké
Longueur de la valeur = 3 à 4 caractères	: Un mot sera stocké
Longueur de la valeur = 5 à 8 caractères	: Un long mot sera stocké

Un message d'erreur apparaît lorsqu'on tente de placer un mot ou un long mot à une adresse impaire.

'$a' stocke des octets.
'$w' stocke des mots.
'$l' stocke des longs mots.

Avec '$a', '$w' et '$l', peu importe la taille des valeurs entrées.

Une fois qu'une ligne de ce type a été placée dans la mémoire, l'adresse suivant la dernière valeur stockée apparaît sur la ligne suivante comme prompt.

Exemple :

```
$50000: $ 3c,'salut',3c2,fc0000,404      -> stocke à partir de
                                            l'adresse $50000 :
                                            $3c.b
                                            'salut'.b
                                            $03c2.w
                                            $fc0000.l
                                            $0404.w

$5000E: $a 'Salut, ici le GFA DEBUGGER !',fc0000,404
                                         -> stocke à partir de
                                            l'adresse $5000E les octets
                                            'Salut...',00 et 04.

$50032: $w 'GFA',0,f8000,10              -> stocke à partir de $50032
                                            les mots
                                            0047,0046,0041,0000,8000
                                            et 0010.

$5003E: $l f8000,0,'a',ffc04             -> stocke à partir de $5003E
                                            les longs mots
                                            000f8000,00000000,00000061
                                            et 000ffc04.
```

Remarque :



Nous supposons que la première adresse de prompt ($50000) est une adresse située dans la mémoire libre. Toutes les autres adresses seront automatiquement proposées par le débogueur.

L {de{,à}}

Désassembler.

Lors du débogage de programmes, il est pratiquement indispensable, pour y comprendre quelque chose, de faire reconvertir le texte codé du programme en un texte en clair. Un listing de désassembleur est donc censé ressembler le plus possible au texte source du programme listé. Cela n'est cependant possible que sous certaines limites car il n'est, par exemple, plus possible, au vu du code programme, de savoir si ce code a été produit à l'aide de macros ou de l'assemblage conditionnel etc. C'est

pourquoi la retraduction ne s'effectue généralement que ligne par ligne. Si le programme désassemblé dispose d'une table de symboles, toutes les valeurs numériques sont remplacées par leurs symboles. Les définitions de labels à certaines adresses sont affichées dans une colonne spéciale devant les mnémoniques sous forme du symbole suivi d'un double point.

Remarque importante :



La forme du listing de désassemblage peut être définie sous certaines limites à l'aide des instructions 'dlon', 'dloff', 'cson', 'csoff', 'dec', 'hex', 'oct' et 'bin'.

dlon

Activer sortie de dump hexadécimal lors du désassemblage.

On peut, au choix, faire également afficher le dumping hexadécimal des instructions assembleur à côté de leurs mnémoniques. Cette option est activée dès le lancement du débogueur.

Les codes hexadécimaux sont placés entre crochets '[' ']'. Ils se composent d'un mot hexadécimal pour le code opération et d'autres mots ou longs mots pour les opérandes. L'affichage de codes hexadécimaux est très utile, notamment, lorsqu'on travaille avec des symboles et que l'on veut, par exemple, connaître non seulement le nom des adresses appelées mais aussi leur valeur explicite.

Pour l'assemblage Inline, les textes placés entre crochets au début, quelle que soit leur nature, ne gênent pas l'évaluation de l'instruction.

dloff

Désactiver 'dlon'.

Interdiction de l'affichage du dumping hexadécimal correspondant devant les instructions désassemblées.

cson / csoff

Activer ou désactiver l'affichage de petites valeurs absolues sous forme de symboles.

Lorsqu'un programme ou un module objet est chargé avec une table de symboles, le débogueur attribue, dans la mesure du possible, lors du désassemblage, le symbole approprié à chaque valeur numérique. Lorsqu'une table de symboles comporte beaucoup de définitions absolues, celles-ci remplacent aussi n'importe quelle valeur numérique correspondante.

Or ces définitions absolues n'ont généralement été effectuées que pour un seul emplacement déterminé du programme. Elles peuvent donc, dans certains cas, rendre d'autres endroits du programme inintelligibles. C'est ainsi qu'un 'addq.l #6,sp' sera souvent plus parlant que 'addq.l #Symbole_XY,sp'.

hex, dec, oct, bin

Sélectionner le système numérique à utiliser pour l'affichage des valeurs numériques lors du désassemblage.

Explication :

hex	: Affichage en système hexadécimal (de base 16)
dec	: Affichage en système décimal (de base 10)
oct	: Affichage en système octal (de base 8)
bin	: Affichage en système binaire (de base 2)

[Code hexa] {Symbole:} Mnémonique {Opérande{,Opérande}}

ou simplement {Symbole:} Mnémonique {Opérande{,Opérande}}

Assemblage Inline d'instructions isolées.

Assemblage direct de la mémoire à l'adresse du prompt. Entrées des instructions assembleur comme sous l'éditeur/assembleur. Aucune des instructions de commande du programme et de traitement des macros (pseudo-codes opération) n'est disponible ! Par contre, toutes les variables système documentées et beaucoup d'autres adresses importantes peuvent être appelées comme fonctions à l'aide de noms

Syntaxe pour affichage de la fonction système :

```
h=    ^^Variable
```

'Variable' peut ici être l'un des noms de la liste du chapitre 3 :

(La valeur après '=' indique chaque fois la valeur de réponse.)

Définition de symbole utilisateur : (uniquement si la mémoire de symboles a été réclamée avec l'instruction 'e' !)

à l'adresse de prompt :	Symbole: {instruction assembleur 68 Ko} Le double point peut être omis si le symbole voulu ne peut être confondu avec une instruction du débogueur.)			
directement	:	Symbole	=	Expression
ou	:	Symbole	.equ	Expression
		Symbole	equ	Expression
		Symbole	.set	Expression

```
Symbole   set     Expression
Symbole   = =     Expression
Symbole   .=      Expression
```

Contrôle de syntaxe :

Comme sous l'assembleur/éditeur, un contrôle complet de plausibilité des instructions assembleur entrées est également effectué lors de l'assemblage Inline. Les entrées incorrectes ne sont pas assemblées mais sanctionnées par un message d'erreur approprié.

Après une entrée admise, l'instruction assembleur est à nouveau affichée, formatée, pour contrôle. L'adresse de prompt désigne chaque fois la position à la suite de la dernière instruction assemblée, de sorte que de petits programmes assembleur peuvent ainsi être entrés en continu.

Offsets relatifs au PC :

Les offsets par rapport au niveau du PC sont à formuler ainsi :

```
(Adresse de destination =) *+Offset
```

Cela est particulièrement important avec les mnémoniques Bcc et DBcc lorsqu'on ne travaille pas avec des symboles.

fl {'a',}'Nom de fichier',de,à

Réassemblage de sections de la mémoire ou de sections de programme.

La directive 'fl' combine le désassemblage sur l'écran et le réassemblage dans un fichier. La zone s'étendant de l'adresse 'de' à l'adresse 'à' est interprétée comme code assembleur et placée, tokenisée, dans le fichier 'Nom de fichier' spécifié. Les opérandes numériques (destinations de saut, adressages immé-

diats etc.) ne sont remplacés par des symboles (tirés de la table de symboles éventuellement chargée ou bien produits automatiquement) que si leur valeur se situe dans l'intervalle entre 'de' et 'à'.

L'utilisateur peut également choisir si le texte source doit être lié à un fichier déjà existant ou bien être envoyé dans un nouveau fichier à créer. Dans le premier cas, le paramètre 'a' doit être ajouté à l'instruction. Si un nouveau fichier est créé, tout fichier déjà existant sous le même nom sera supprimé !

Exemple :

Voyez l'exemple pour l'instruction 'is'. C'est, toutefois, uniquement un segment de texte qui sera produit.

is 'Nom de fichier'

Réassemblage symbolique d'un programme / code objet.

Lorsque l'utilisateur fait analyser des programmes d'autres auteurs, il est souvent souhaitable de pouvoir reconvertir un programme machine exécutable en un texte source pour pouvoir le modifier, le réutiliser etc. Cette opération peut être exécutée entièrement automatiquement. Une fois qu'un programme a été chargé dans la mémoire avec l'instruction 'e', vous pouvez spécifier un nom de chemin pour le faire réassembler dans un fichier sous forme d'un code source TOKENISE, donc compréhensible par le GFA Assembleur.

Si le code assembleur traité dispose d'une table de symboles propre, ces symboles seront utilisés pour le réassemblage. Sinon, des symboles artificiels, de format 'L_Adresse', seront produits.

'Adresse' désigne ici, sous forme d'un nombre hexadécimal, l'ancienne valeur du symbole.

Le code source produit se compose d'un segment de texte, d'un segment de données et d'un segment Bss. Dans le segment de données, les tables de saut sont identifiées et organisées avec des symboles, à l'aide des pseudo-codes opération DC.l. Toutes les autres données sont fixées systématiquement avec des instructions DC.b.

La subdivision du segment BSS se base sur les définitions de symboles dans le segment BSS. Elle présente la forme suivante :

```
Symbole1: .DS.b n_1
Symbole2: .DS.b n_2
...
Symbolen: .DS.b n_n

avec 'n_i' : nombre d'octets jusqu'au prochain symbole
```

Le texte source reçoit un en-tête contenant le nom du fichier réassemblé et ses informations de header. Toutes les mnémoniques (segment de texte) sont suivies, sous forme de commentaire, de l'adresse de la mémoire dont elles sont tirées ainsi que du code hexadécimal correspondant. Toutes les séquences .DC.b (segment de données) sont suivies, sous forme de commentaire, d'une chaîne composée des caractères ASCII correspondant aux données.

Remarque :



Les utilisateurs ayant recours à cette directive ne doivent pas oublier que les programmes d'autres auteurs sont souvent protégés par la législation sur le droit d'auteur (C'est notamment le cas du GFA Debugger, par exemple...).

Exemple :

Listing de réassemblage d'un programme de démonstration sans table de symboles propre (extraits) :

```
;FILE:      D:\DOSSIER\DEMO.PRG
;          TEXT      DATA      BSS
;BASE:     00041570  00044b12  00045d42
;LEN :     000035a2  00001230  00009446

.TEXT
bsr        L_0417b4          ;00041570:   6100   0242
bsr        L_041f64          ;00041574:   6100   09ee
bsr        L_041928          ;00041578:   6100   03ae
clr.w      -(sp)             ;0004157c:   4267
trap   #$1                   ;0004157e:   4e41
movea.l#L_045fae,a3          ;00041580:   267c   00045fae
L_041586: clr.l d5           ;00041586:   4285
clr.l d4                     ;00041588:   4284
L_04158a: move.b   $0(a3,d4.w),d3 ;0004158a:   1633   4000
bsr        L_0415a4          ;0004158e:   6100   0014
asl.l      #$4,d5            ;00041592:   e985
or.b       d3,d5             ;00041594:   8a03
cmpi.b     #$7,d4            ;00041596:   0c04   0007
beq        L_0415a2          ;0004159a:   6700   0006
addq.b     #$1,d4            ;0004159e:   5204
bra.s      L_04158a          ;000415a0:   60e8
L_0415a2: rts                ;000415a2:   4e75
...
           .DATA
...
L_044c20: .DC.l   L_044c44,L_044c5c                    ;00044c20: "........"
          .DC.l   L_044c7a,L_044c92                    ;00044c28: "........"
          .DC.l   L_044cc1,L_044cee                    ;00044c30: "........"
          .DC.l   L_044d17,L_044d42                    ;00044c38: "........"
          .DC.l   L_044d6c                             ;00044c40: "...."
L_044c44: .DC.b   $2e,$22,$20,$49,$6e,$73,$74,$72      ;00044c44: "." Instr"
          .DC.b   $75,$63,$74,$69,$6f,$6e,$20,$69      ;00044c4c: "uctioni"
          .DC.b   $6e,$63,$6f,$6e,$6e,$75,$65,$20      ;00044c54: "nconnue"
L_044c5c: .DC.b   $21,$30,$2e,$22,$20,$4e,$6f,$6d      ;00044c5c: "!0."Nom"
          .DC.b   $62,$72,$65,$20,$64,$65,$20,$70      ;00044c64: "bredep"
          .DC.b   $61,$72,$61,$6d,$8a,$74,$72,$65      ;00044c6c: "aramètre"
...
           .BSS
...
L_045d42: .DS.b   $4
L_045d46: .DS.b   $4
L_045d4a: .DS.b   $2
L_045d4c: .DS.b   $2
L_045d4e: .DS.b   $2
L_045d50: .DS.b   $2
...
           .END
```

hn de,à,ad ou **hn** de,à,'Chaîne'

Rechercher une adresse ou une chaîne de caractères. 'de' est l'adresse à partir de laquelle doit commencer la recherche. 'à' est l'adresse jusqu'à laquelle doit s'effectuer la recherche. 'ad' est l'adresse (long mot) qu'il s'agit de rechercher. 'Chaîne' est la chaîne de caractères à rechercher.

Suivant qu'une adresse ou une chaîne de caractères aura été entrée comme troisième paramètre, le débogueur recherchera l'expression voulue dans la zone spécifiée. La chaîne de caractères peut comporter tous les caractères ASCII de 1 à 255 (décimal). Si l'expression figure effectivement dans la zone spécifiée, une liste de toutes les adresses où elle a été trouvée sera affichée.

Ensuite, la première zone où apparait l'expression sera affichée sous forme de Dump de la mémoire. Le curseur apparaîtra sur la ligne du milieu, sur l'octet correspondant.

Exemple :

hn ^^data,^^bss,'GFA'	->	Rechercher la chaîne 'GFA' dans le segment de données
hn 0,$f8000,^^basepage	->	Rechercher l'adresse de la page de base du programme charge dans la mémoire de travail (1 Mo)

hf de,à,'Partie d'instruction'

Rechercher une partie d'une instruction assembleur.

Il arrive souvent, lors du débogage, qu'on ait à rechercher toutes les instructions appelant une cellule de mémoire ou un registre du processeur déterminé, ou bien qu'on s'intéresse à une instruction dont on ne se rappelle précisément qu'un opérande.

Pour trouver, dans ces situations, des parties d'instruction, on utilise l'instruction 'hf' qui fonctionne exactement comme la recherche de chaîne de caractères sous l'assembleur/éditeur (menu 4,3).

C'est '\' qui fait office de joker *

et '~' qui fait office de joker ?.

Le critère de recherche ainsi spécifié est comparé avec le listing de désassemblage de l'adresse 'de' à l'adresse 'à'. Les espaces résultant du formatage par le désassembleur sont à prendre en compte lors de l'entrée du critère de recherche ou au moins à sauter avec le joker \.

Toutes les lignes qui coïncident sont affichées avec l'adresse de leur position. L'opération de recherche peut être suspendue provisoirement avec la barre 'espace' ou être interrompue définitivement avec la touche 'Esc'.

Exemple :

Si on avait entré, lors du débogage d'un programme :

```
hf ^^text,^^text+$200,"movea.l\,a"
```

(C'est-à-dire : rechercher dans les 512 premiers octets du segment de texte toutes les instructions 'move' qui écrivent dans un registre d'adresse en format de long mot !)
on obtiendrait l'affichage suivant :

```
$BB024:[  2668 0008          ]            movea.l   B_P_L(a0),a3
$BB046:[  2059               ]            movea.l   (a1)+,a0
$BB062:[  2068 004c          ]            movea.l   $4c(a0),a0
$BB0B6:[  2279 000d3d70      ]            movea.l   adrecra,a1
$BB12A:[  2079 000cbf6c      ]asciiout:   movea.l   postext,a0
$BB158:[  2279 000d3d74      ]            movea.l   posecra,a1
```

hm de,à,'Instruction'

Rechercher une instruction assembleur complète.

Contrairement à la recherche avec 'hf', l'instruction 'hm' n'autorise pas les jokers ni les fragments d'instructions assembleur. L'instruction attend une expression de syntaxe correcte telle qu'elle pourrait être traitée par l'assembleur Inline.

Le critère de recherche ainsi spécifié est automatiquement transformé en code assembleur par l'assembleur Inline et est recherché sous forme de code hexadécimal.

On obtient ainsi des vitesses de recherche nettement plus élevées que pour la recherche d'un texte non codé. Cette différence de vitesse est surtout sensible lorsqu'il s'agit d'examiner des zones de mémoire étendues.

Exemple :

```
hm ^^text,^^data,"trap #14"
```

recherchera les 14 appels Trap du segment de texte et les affichera comme avec l'instruction 'hf'.

mc zone1,zone2

Comparer des zones de la mémoire. 'zone1' et 'zone2' sont les adresses respectives des deux zones à comparer.

Cette instruction permet, par exemple, de comparer deux fichiers quelconques qui auront été chargés au préalable dans la mémoire. Dès que le débogueur trouve une différence, il affiche deux lignes de Dump de la mémoire dans lesquelles figure le premier octet de la différence constatée.

Exemple :

```
r 'fichier1.xxx',$50000      -> Charger le fichier 1 en $50000
r 'fichier2.xxx',$60000      -> Charger le fichier 2 en $60000
mc $50000,$60000             -> Comparer les fichiers jusqu'à la première
                                différence
```

mmc zone1,zone2

Rechercher s'il y a des différences de mnémoniques entre deux zones de mémoire. 'zone1' et 'zone2' sont les adresses des deux zones qu'il s'agit de comparer.

Cette instruction constitue une variante plus complète de l'instruction 'mc'. L'instruction 'mmc' compare les deux zones mot d'instruction par mot d'instruction. Elle compare donc le premier mot de la zone 1 avec le premier mot de la zone 2.

Si une différence est alors constatée, les deux instructions assembleur figurant à ces adresses sont affichées. Sinon, la longueur de l'instruction est calculée chaque fois pour comparer ensuite les mots d'instruction des deux instructions suivantes.

Cette instruction permet, par exemple, de comparer deux programmes ou deux modules objet pour repérer les différences entre une version plus ancienne et une version nouvelle.

Exemple :

```
r 'prog1.prg',$50000         -> Charger le programme 1
r 'prog2.prg',$60000         -> Charger le programme 2
mmc $5001c,$6001c            -> Comparer les programmes à partir du
                                segment de texte et jusqu'à la première
                                différence
```

Remarque :



L'instruction 'r' (voyez la section 2.3) est employée ici parce qu'elle n'effectue pas de réservation de la mémoire. Elle permet donc de charger autant de programmes que vous le souhaitez à des adresses quelconques.

Mais vous pouvez aussi comparer directement des zones de mémoire sans utiliser 'r'.

m de,à,vers

Transférer une zone de mémoire. 'de' est l'adresse à partir de laquelle doit s'effectuer le transfert (début de la source). 'à' est l'adresse jusqu'à laquelle doit s'effectuer le transfert (fin de la source). 'vers' est l'adresse vers laquelle doit s'effectuer le transfert (début de la destination).

Cette instruction transfère une zone de mémoire "intelligemment", c'est-à-dire qu'une zone de mémoire peut être décalée en arrière ou en avant sur une distance inférieure à sa longueur sans que la zone de destination ne soit endommagée. Lorsqu'une zone est décalée sur une distance supérieure à sa longueur, la zone source est également conservée et cette zone figure alors en deux exemplaires entiers dans la mémoire.

Exemple :

```
m $50000,$60000, $80000     -> Sans chevauchement des zones
m $50000,$60000, $55000     -> Avec chevauchement des zones
m $50000,$60000, $45000     -> Avec chevauchement des zones
```

mcl de,à

Vider une zone de mémoire. 'de' est l'adresse à partir de laquelle la zone doit être effacée, 'à' l'adresse jusqu'à laquelle elle doit l'être.

Exemple :

```
mcl ^^MEMad, ^^MEMend      -> Vider la mémoire de travail
```

fill Nombre,Liste

Remplir une zone de mémoire.

L'instruction Fill constitue une méthode très pratique pour initialiser des zones de mémoire avec des motifs.

'Nombre' définit combien de fois le motif de remplissage 'Liste' devra être stocké à partir de l'adresse de prompt.

'Liste' est une séquence d'expressions (que vous pouvez, si vous le souhaitez, séparer par des virgules), chacune complétée par une extension ('.b','.w','.l') pour indiquer le format de données (respectivement octet, mot, long mot), qui représente le motif de remplissage. S'il résulte de cette liste que des mots ou des longs mots devraient être placés à des adresses impaires, le débogueur les placera à la prochaine adresse paire et laissera intact l'octet ainsi sauté. L'expression est affichée pour contrôle avec toutes les extensions.

Exemple :

```
$40000: fill 100,"Stocker un texte d'exemple".b,0.b,^^physbase
```

A partir de l'adresse de prompt $40000 sera déposée 100 fois la chaîne 'Stocker un texte d'exemple' avec un octet nul termi-

nal. L'octet suivant restera inchangé pour parvenir à une adresse paire. La base de l'écran physique sera enfin ajoutée à la suite sous forme d'un long mot.

2.3. Opérations disque

dir {'L'} ou **dir** {'Chemin'}

Afficher le répertoire. 'L:' est la désignation du lecteur, pour un répertoire "complet". 'Chemin' est le chemin de fichier pour lequel un répertoire devra être affiché. Si vous entrez uniquement 'dir', donc sans chemin ni désignation de lecteur, le répertoire "complet" du lecteur par défaut sera affiché.

Cette fonction permet de faire afficher le répertoire "complet" de la racine du lecteur spécifié ou bien le répertoire "normal" d'un chemin quelconque.

Contrairement au répertoire "normal", un répertoire "complet" indique, en plus, pour chaque fichier le cluster et le secteur de départ.



Ceci n'est cependant pas toujours valable pour tous les RAM Disk.

Exemple :

```
dir                    -> Répertoire "complet" du lecteur actuel
dir 'c:'               -> Répertoire "complet" du lecteur C
dir 'a:\auto\*.*'      -> Répertoire "normal" du chemin 'a:\auto\*.*'
```

setdrv 'L' ou **setdrv** n

Fixer le lecteur par défaut. 'L' est la désignation du nouveau lecteur défaut. 'n' est le numéro du nouveau lecteur défaut (A=0, B=1, C=2, ...).

Le lecteur défaut est le lecteur à partir duquel le débogueur, ou le programme l'ayant appelé, a été lancé. Toutes les entrées de fichier sans spécification de chemin sont appliquées à ce lecteur. Lorsqu'on a souvent à travailler, sous le débogueur, avec un autre lecteur, on peut fixer un nouveau lecteur par défaut avec cette instruction. Ou bien, lorsqu'un programme est censé charger automatiquement des fichiers, par exemple des Ressources ou des fichiers d'infos, il ne pourra les trouver que si le lecteur approprié a été commuté avec 'setdrv' avant lancement de ce programme.

Exemple :

```
setdrv 'd' ou
setdrv 3          ->  Ces deux instructions fixent le lecteur D comme
                      nouveau lecteur défaut
```

delete 'Chemin :\nomdefichier.extension'

Supprimer un ou plusieurs fichiers. 'Chemin' est le chemin où se trouvent les fichiers censés être supprimés. nomdefichier.extension désigne les fichiers à effacer. Les jokers '*' et '?' peuvent être utilisés.

Cette instruction supprime tous les fichiers correspondant à la désignation de chemin spécifiée.

Exemple :

delete 'salut.rie'	->	Supprime le fichier 'salut.rie' dans le répertoire actuel
delete '\hugo\a*.ttp'	->	Supprime dans le répertoire '\hugo\', sur le lecteur actuel, tous les fichiers commençant par la lettre 'a' et portant la terminaison '.TTP'.

r 'Fichier'{,ad}

Charger un fichier. 'Fichier' est le nom ou chemin du fichier qu'il s'agit de charger. 'ad' est l'adresse à laquelle le fichier doit être chargé. Si 'ad' n'est pas spécifié, le fichier sera chargé au début de la mémoire libre.

Cette instruction permet de charger un fichier quelconque dans la mémoire de travail de l'ordinateur. La place nécessaire n'est pas réservée comme avec 'e' par exemple.

Exemple :

r 'test.img'	->	Le fichier 'test.img' sera chargé au début de la mémoire libre.

W {'Fichier'{,ad,len}}

Ecrire un fichier sur disque. 'Fichier' est le nom ou chemin de fichier sous lequel la zone de mémoire voulue doit être sauvegardée. 'ad' est l'adresse à partir de laquelle doit commencer la sauvegarde. 'len' est la longueur de la zone à sauvegarder. Si un fichier a été chargé auparavant avec 'r', il n'est pas nécessaire de spécifier 'Fichier', 'ad' et 'len' car le fichier sera alors sauvegardé avec les paramètres qui avaient servi à le charger précédemment avec 'r'.

Cette instruction permet d'écrire une zone de mémoire sur disque.

Exemple :

```
r 'test.prg'        -> Charger le programme
...                 -> Le modifier
w                   -> Le sauvegarder à nouveau
                       après modification
```

ou

```
w 'salut.img',^^physbase,32000   -> Sauvegarder le contenu de
                                    l'écran du programme à
                                    déboguer.
```

ou

```
r 'a:fichier.ext'
; Changer de disquette
w                   -> Copie rapide d'un fichier avec
                       un seul lecteur de disquette
```

rv

Afficher les paramètres du fichier chargé ou des secteurs chargés.

Lorsqu'un fichier a été chargé avec 'r' (voyez plus haut) ou qu'une séquence de secteurs a été chargée avec rsec (voyez plus bas), cette instruction permet de faire afficher l'adresse de base dans la mémoire ainsi que la longueur ou le nombre de secteurs.

rsec {'L',}de,nmb{,ad}

Charger un ou plusieurs secteurs consécutifs. 'L' est le lecteur à partir duquel les secteurs doivent être chargés. 'de' est le secteur à partir duquel doit commencer le chargement. 'nmb' est le nombre de secteurs à charger. 'ad' est l'adresse à laquelle les secteurs lus devront être placés. Si 'L' n'est pas spécifié, les secteurs seront chargés à partir du lecteur défaut. Si 'ad' n'est pas spécifié, les secteurs seront écrits au début de la mémoire libre.

Exemple :

```
rsec 'a',0,1        ->  Charge le secteur boot du lecteur A
d                   ->  Affiche les 176 premiers octets du secteur
```

wsec {{'n',}de,nmb{,ad}}

Ecrire sur disque un ou plusieurs secteurs consécutifs. Les paramètres ont la même signification que pour 'rsec'.

Si 'rsec' a été exécuté auparavant, tous les paramètres peuvent être omis, auquel cas seront sauvegardés les mêmes secteurs que ceux lus auparavant avec 'rsec'.

Exemple :

```
rsec 18,100         ->  Lire 100 secteurs à partir du 18ème
   ...              ->  Modifier les secteurs
wsec                ->  Réécrire sur disque les secteurs modifiés
```

ou

```
rsec 'a',0,1440
; Changer de disquette
wsec                ->  Copier une disquette double face (s'il y
                        a suffisamment de mémoire de travail
                        disponible)
```

secno ad

Calcule le numéro de secteur et le numéro d'octet dans le secteur correspondant à l'adresse spécifiée. 'ad' est l'adresse dont le numéro de secteur doit être calculé.

Après qu'un certain nombre de secteurs aient été lus avec rsec, cette instruction permet de calculer le secteur et la position correspondant à une adresse donnée.

Exemple :

```
rsec 18,10                    -> Charger 10 secteurs à partir du 18ème
* A partir du secteur.....à partir de l'adresse $50000
                              -> Message du débogueur :
                                 Les secteurs figurent en mémoire à
                                 partir de $50000
secno $5020e                  -> A quel secteur correspond cette
                                 adresse ?
* Correspond au secteur 19, octet 14
                              -> Message du débogueur
```

getbpb {'L'}

Charger BPB du Bios. 'L' est le lecteur dont le BPB doit être affiché. Si L n'est pas spécifié, le BPB du lecteur défaut sera affiché (voyez 'setdrv').

Cette instruction charge le bloc de paramètres de démarrage (Boot Parameters' Block) du BIOS d'un lecteur, à l'aide de la fonction 7 du BIOS. Les données contenues dans ce bloc sont alors affichées, avec leur signification.

Exemple :

```
getbpb 'a'   -> Lire le BPB du lecteur A:
```

sclu {'L',}clu ou **sclu** {'L',}-sec

Afficher le statut de cluster. 'L' est le lecteur dont le statut de cluster doit être lu. 'clu' est le cluster dont le statut doit être affiché. 'sec' désigne un secteur qui, pour être distingué d'un cluster, doit être marqué par un signe moins. Ce secteur est converti en un cluster correspondant dont le statut est alors affiché.

Cette instruction permet de connaître le statut d'un cluster déterminé. Pour éviter d'avoir à convertir le numéro de secteur en numéro de cluster correspondant, on peut aussi spécifier directement le numéro de secteur, en le faisant précéder d'un signe moins. Les valeurs possibles, comme statut de cluster, sont les suivantes :

0	:	Le cluster est libre (mais pas nécessairement vide).
$ff7	:	(ou $fff7 pour un disque dur) Le cluster est défectueux.
$fff	:	(ou $ffff pour un disque dur) Le cluster est occupé, c'est le dernier cluster d'un fichier
n	:	(tout nombre autre) Le cluster est occupé et le nombre indique le numéro du cluster suivant du fichier

fclu {'L',}déb{,mod}
lclu {'L',}déb{,mod}

Suivre (fclu) et charger (lclu) un fichier. 'L' est le lecteur dont les clusters doivent être lus. 'déb' est le cluster à partir duquel le fichier doit être suivi. 'mod' fixe le sens dans lequel le fichier doit être suivi. Si 'mod' n'est pas spécifié, c'est un nombre positif qui sera supposé comme valeur défaut.

L'instruction 'fclu' permet de suivre un fichier sur disquette. On spécifie à cet effet un cluster à partir duquel la recherche devra commencer. Si un nombre positif est entré comme mot de mode, le fichier sera suivi du cluster spécifié jusqu'à la fin, alors

qu'avec un nombre négatif, il sera suivi jusqu'au début. Il est ainsi possible, à partir d'un secteur quelconque, de déterminer à quel fichier il appartient (Comme pour l'instruction 'sclu', vous pouvez aussi spécifier des secteurs, en les faisant précéder d'un signe moins).

L'instruction 'lclu' charge, en outre, les clusters trouvés dans la mémoire s'il s'agit d'une recherche vers l'arrière.

Exemple :

Vous voulez charger le fichier dont fait partie le cluster 100 :

```
fclu 100,-1     -> Suivre le fichier jusqu'à son début
lclu 70         -> Si 70 est le premier cluster du fichier, cette instruction
                   chargera le fichier dans la mémoire
```

Remarque :



Lorsque des clusters ont été chargés en mémoire, l'instruction 'rv' permet d'appeler le nombre de secteurs et l'adresse de base dans la mémoire.

eclu {'L',}déb,nmb

leclu {'L',}déb,nmb

Rechercher des clusters libres (eclu) et les charger (leclu). 'L' est le lecteur sur lequel les clusters doivent être recherchés. 'déb' est le cluster à partir duquel des clusters libres doivent être recherchés. 'nmb' est le nombre de clusters libres devant être recherchés.

L'instruction 'eclu' permet de trouver des clusters libres sur disque. Vous devez entrer le cluster de départ et le nombre de clus-

ters à rechercher. Le débogueur examine alors la FAT à partir du cluster spécifié et affiche les clusters libres trouvés.

L'instruction 'leclu' charge en outre les clusters trouvés.

Exemple :

Supposons que nous voulions restaurer un fichier supprimé. Cela est possible si aucune autre opération d'écriture ou de suppression n'a été effectuée sur le lecteur correspondant après la suppression du fichier concerné. Si d'autres fichiers avaient déjà été supprimés auparavant, les secteurs de ces fichiers risquent de se trouver intercalés entre les secteurs du fichier qu'il s'agit de restaurer.

Supposons qu'un programme a été supprimé sur le lecteur C:.

```
setdrv 'c'      ->  Toutes les opérations suivantes se réfèreront au
                    lecteur C:
dir             ->  Afficher un répertoire complet du lecteur C:
```

L'entrée du fichier figure encore dans le répertoire mais elle est marquée comme supprimée par le caractère σ au début du nom de fichier. Lisez maintenant le cluster de départ et la longueur du fichier et spécifiez les informations obtenues comme paramètres de l'instruction 'leclu' :

```
leclu ClusterDeBase,Longueur/1024+1
```

Remarque :



1024 est la longueur d'un cluster (par exemple leclu 1298,9675/1024+1).

Attention :



Si les clusters de fichiers supprimés auparavant sont intercalés entre les clusters du fichier à restaurer, il faudra lire un plus grand nombre de clusters, jusqu'à ce que tous les clusters du programme soient chargés dans la mémoire.

Recherchez ensuite, dans la zone chargée, quels clusters n'appartiennent pas au programme. Ce n'est pas difficile s'il n'y a pas de cluster étranger intercalé ou, s'il y en a, s'ils sont d'un autre type (par exemple des textes ASCII). Cela peut, par contre, devenir très difficile si on a affaire, par exemple, à deux programmes mélangés.

Les secteurs étrangers peuvent être éliminés en décalant vers le début du fichier, avec l'instruction 'm', tous les secteurs suivants (l'instruction 'secno' permet de calculer le numéro de secteur correspondant à une adresse). Supposons, par exemple, que les secteurs 60 et 61 ne fassent pas partie du programme, l'instruction

```
m ^(60+2), ^^MEMend, ^60
```

décalera vers le début du programme tous les secteurs à partir du secteur 62.

Une fois que tous les clusters figurent à la suite dans la mémoire, vous pouvez les sauvegarder comme un programme prêt avec l'instruction 'w'.

2.4. Autres instructions

scrollon, scrolloff

Active ou désactive le défilement de fonction avec la souris (voyez la section 1.4). Remarque : Après lancement du programme, le défilement de fonction est activé.

pon, poff

Tous les affichages sont, en plus, imprimés ou, au contraire, ne seront plus imprimés.

Ces instructions commandent le protocole d'impression. Si le protocole est activé, toutes les sorties continuent d'apparaître sur l'écran mais sont, en plus, imprimées.

"

Marque de commentaire.

Cette instruction n'a pas d'autre effet que d'empêcher que l'interpréteur du GFA Debugger tente d'évaluer les instructions suivantes.

h Expression

Calcul.

Cette fonction calcule une expression quelconque et affiche le résultat en décimal, hexadécimal, binaire et octal.

Exemple :

```
h ^^physbase
h 1+2*3
h 3*(7+4)
```

k

Afficher l'adresse de base et la fin du débogueur.

asm {nmb}

Appeler l'assembleur. 'nmb' est le nombre maximum de symboles que l'utilisateur souhaite pouvoir redéfinir. L'assembleur aura besoin de ce paramètre si on lui demande d'écrire un programme dans la mémoire du débogueur.

Si le débogueur a été appelé à partir de l'assembleur (menu 6,6 ou menu 6,7), cette instruction permet de passer provisoirement sous l'assembleur pour y travailler comme d'habitude. La mémoire disponible est, toutefois, limitée à la quantité réservée lors de l'appel du débogueur. Si un programme est assemblé ou linké sous l'assembleur, il sera automatiquement transféré dans la mémoire du débogueur. Pour ne pas oublier que l'assembleur a été rappelé à partir du débogueur, le texte "retourner au débugger" apparaît dans le point 1,0 du menu. Pour mettre fin à

ce retour provisoire à l'assembleur, vous devez donc sélectionner le menu 1,0.

L'instruction 'asm' a donc deux fonctions fondamentales :

- Lors du travail sous le débogueur, on peut consulter le texte source du programme qui, grâce aux commentaires et au découpage en paragraphes, est nettement plus compréhensible que le listing désassemblé du débogueur.

- L'assembleur peut transmettre directement au débogueur des programmes ou modules objet qui peuvent alors être testés sous le débogueur.

Exemple :

```
asm     ->  Retour à l'assembleur pour y faire assembler un
...         programme. Lors de l'assemblage, il est conseillé de
...         produire systématiquement une table de symboles. Quitter
...         ensuite l'assembleur avec <Esc> +1 +0 et le débogueur
...         sera à nouveau activé.
v       ->  Vous pouvez alors faire afficher sous le débogueur les
            paramètres du programme transféré puis tester le
            programme.
```

resident {nmb}

Le débogueur est abandonné mais reste résident dans la mémoire. 'nmb' est la taille de la zone qui doit être réservée, comme mémoire de travail, à la suite du débogueur.

Cette instruction permet d'obtenir le même résultat que si la version résidente du débogueur figurait dans le dossier Auto.

quit

Abandonner le débogueur.

doreset

Effectuer une réinitialisation complète du système (Reset).

Cette instruction est utile lorsqu'un programme a été testé partiellement et que l'adresse de la routine Exit (qui devrait, par exemple, refermer les canaux de données pour GEM ou restaurer d'autres vecteurs importants) du programme n'est pas connue ou qu'elle ne peut être appelée. Dans ce cas, en effet, un retour du débogueur au bureau ou un lancement ultérieur d'un autre programme pourraient provoquer un plantage du système.

Chapitre 3

Evaluation de fonction

L'évaluation de fonction sous le débogueur est pratiquement identique à celle effectuée sous l'assembleur. Sous le débogueur également, l'évaluation de fonction se déroule de façon hiérarchique et avec autant de niveaux de parenthèses que vous le souhaitez. Les différences suivantes, par rapport à l'assembleur, sont, toutefois, à noter :

- Comme le débogueur ne dispose pas de gestion de macros, les fonctions et opérateurs relevant de la gestion des macros ne sont pas soutenus.

- Des modes d'adressage peuvent être employés dans les conditions d'interruption pour les instructions 'usm' et 'us0' à 'us9'. Pour plus de détails, nous vous invitons à vous reporter à la description de ces instructions (Chapitre 2.1).

- Sous le débogueur, les variables suivantes sont prédéfinies une fois pour toutes :

Mémoire et Shifter :

```
memconf   =  $ffff8001   ;  Configurations de la mémoire
dbaseh    =  $ffff8201   ;  Octet fort de l'adresse de l'écran
dbasel    =  $ffff8203   ;  Octet médian de l'adresse de l'écran
vcounthi  =  $ffff8205   ;  Octet fort du compteur d'adresse vidéo
vcountmid =  $ffff8207   ;  Octet médian compteur d'adresse vidéo
vcountlow =  $ffff8209   ;  Octet faible compteur d'adresse vidéo
syncmode  =  $ffff820a   ;  Mode de synchronisation
color0    =  $ffff8240   ;  Registres de couleur Shifter 0 à 15
color1    =  $ffff8242
color2    =  $ffff8244
color3    =  $ffff8246
color4    =  $ffff8248
color5    =  $ffff824a
color6    =  $ffff824c
color7    =  $ffff824e
color8    =  $ffff8250
color9    =  $ffff8252
```

```
color10    =  $ffff8254
color11    =  $ffff8256
color12    =  $ffff8258
color13    =  $ffff825a
color14    =  $ffff825c
color15    =  $ffff825e
shiftmd    =  $ffff8260    ;   Résolution de l'image
```

DMA et Disk :

```
diskctrl=     $ffff8604    ;   Sélection de registres du contrôleur disque
DMAmode=      $ffff8606    ;   Etat/mode DMA
DMAhigh   =   $ffff8609    ;   Base et compteur DMA : high
DMAmid    =   $ffff860b    ;   Base et compteur DMA : mid
DMAlow    =   $ffff860d    ;   Base et compteur DMA : low
```

Registres 1770 :

```
cmdreg    =   $80;    1770/FIFO Sélection du registre de commande
trackreg  =   $82;    1770/FIFO Sélection du registre de piste
sectorreg =   $84;    1770/FIFO Sélection du registre de secteur
datareg   =   $86;    1770/FIFO Sélection du registre de données
```

Sound Chip :

```
PSGselect    =  $ffff8800    ;   (W) Sélection de registre
PSGread      =  $ffff8800    ;   (R) Lire données
PSGwrite     =  $ffff8802    ;   (W) Ecrire données
PSGtoneAf    =           0   ;   Canal A : réglage fin
PSGtoneAc    =           1   ;   Canal A : réglage grossier
PSGtoneBf    =           2   ;   Canal B
PSGtoneBc    =           3
PSGtoneCf    =           4   ;   Canal C
PSGtoneCc    =           5
PSGnoise     =           6   ;   Générateur de bruit
PSGmixer     =           7   ;   Commande I/O, Mixer
PSGampA      =           8   ;   Canaux A, B, C : Amplitude
PSGampB      =           9
PSGampC      =          $a
PSGenvlpf    =          $b   ;   Période de courbe d'enveloppe :
                                 réglage fin
PSGenvlpc    =          $c   ;   Période de courbe d'enveloppe :
                                 réglage grossier
PSGportA     =          $e   ;   PORT A (Seulement sortie)
PSGportB     =          $f   ;   PORT B (Sortie Centronics)
```

Bits de "PSGportA" :

```
RTSout   =  8        ;  RTS output
DTRout   =  $10      ;  DTR output
STROBE   =  $20      ;  Centronics strobe output
OUT      =  $40      ;  "general purpose" output
```

68901 (MFP) :

```
MFP      =  $fffffa00      ;  Base des registres
MFPgpio  =  $fffffa00+1    ;  I/O
MFPaer   =  $fffffa00+3    ;  active edge
MFPddr   =  $fffffa00+5    ;  Sens des données
MFPiera  =  $fffffa00+7    ;  interrupt enable A
MFPierb  =  $fffffa00+9    ;  interrupt enable B
MFPipra  =  $fffffa00+$b   ;  interrupt pending A
MFPiprb  =  $fffffa00+$d   ;  interrupt pending B
MFPisra  =  $fffffa00+$f   ;  interrupt in service A
MFPisrb  =  $fffffa00+$11  ;  interrupt in service B
MFPimra  =  $fffffa00+$13  ;  interrupt mask A
MFPimrb  =  $fffffa00+$15  ;  interrupt mask B
MFPvr    =  $fffffa00+$17  ;  Registre de vecteur
MFPtacr  =  $fffffa00+$19  ;  timer A control
MFPtbcr  =  $fffffa00+$1b  ;  timer B control
MFPtcdcr =  $fffffa00+$1d  ;  timer C & D control
MFPtadr  =  $fffffa00+$1f  ;  timer A data
MFPtbdr  =  $fffffa00+$21  ;  timer B data
MFPtcdr  =  $fffffa00+$23  ;  timer C data
MFPtddr  =  $fffffa00+$25  ;  timer D data
MFPscr   =  $fffffa00+$27  ;  sync char
MFPucr   =  $fffffa00+$29  ;  USART control reg
MFPrsr   =  $fffffa00+$2b  ;  receiver status
MFPtsr   =  $fffffa00+$2d  ;  transmit status
MFPudr   =  $fffffa00+$2f  ;  USART data
```

ACIA's 6850 :

```
ACIAkeyctrl  =  $fffffc00  ;  keyboard ACIA control
ACIAkeydr    =  $fffffc02  ;  keyboard data
ACIAmidicr   =  $fffffc04  ;  MIDI ACIA control
ACIAmididr   =  $fffffc06  ;  MIDI data
```

Variables du BIOS documentées et Variables Linea :

etv_timer	=	$400	VPLANES	=	0
etv_critic	=	$404	VWRAP	=	2
etv_term	=	$408	CONTRL	=	4
etv_xtra	=	$40c	INTIN	=	8
memvalid	=	$420	PTSIN	=	12
memcntlr	=	$424	INTOUT	=	16
resvalid	=	$426	PTSOUT	=	20
resvector	=	$42a	COLBIT0	=	24
phystop	=	$42e	COLBIT1	=	26
_membot	=	$432	COLBIT2	=	28
_memtop	=	$436	COLBIT3	=	30
memval2	=	$43a	LSTLIN	=	32
flock	=	$43e	LNMASK	=	34
seekrate	=	$440	WMODE	=	36
_timr_ms	=	$442	X1	=	38
_fverify	=	$444	Y1	=	40
_bootdev	=	$446	X2	=	42
palmode	=	$448	Y2	=	44
defshiftmd	=	$44a	PATPTR	=	46
sshiftmd	=	$44c	PATMSK	=	50
_v_bas_ad	=	$44e	MFILL	=	52
vblsem	=	$452	CLIP	=	54
nvbls	=	$454	XMINCL	=	56
_vblqueue	=	$456	YMINCL	=	58
colorptr	=	$45a	XMAXCL	=	60
screenpt	=	$45e	YMAXCL	=	62
_vbclock	=	$462	XDDA	=	64
_frclock	=	$466	DDAINC	=	66
hdv_init	=	$46a	SCALDIR	=	68
swv_vec	=	$46e	MONO	=	70
hdv_bpb	=	$472	SRCX	=	72
hdv_rw	=	$476	SRCY	=	74
hdv_boot	=	$47a	DSTX	=	76
hdv_mediach	=	$47e	DSTY	=	78
_cmdload	=	$482	DELX	=	80
conterm	=	$484	DELY	=	82
trp14ret	=	$486	FBASE	=	84
criticret	=	$48a	FWIDTH	=	88
themd	=	$48e	STYLE	=	90
_____md	=	$49e	LITEMSK	=	92
savptr	=	$4a2	SKEWMSK	=	94
_nflops	=	$4a6	WEIGHT	=	96
con_state	=	$4a8	ROFF	=	98
save_row	=	$4ac	LOFF	=	100
sav_context	=	$4ae	SCALE	=	102
_bufl	=	$4b2	CHUP	=	104
_hz_200	=	$4ba	TEXTFG	=	106
_drvbits	=	$4c2	SCRTCHP	=	108
_dskbufp	=	$4c6	SCRPT2	=	112
_autopath	=	$4ca	TEXTBG	=	114
_vbl_list	=	$4ce	COPYTRAN	=	116
_dumpflg	=	$4ee	SEEDABORT	=	118
_prtabt	=	$4f0			

```
_sysbase  =  $4f2  |
_shell_p  =  $4f6  |
end_os    =  $4fa  |
exec_os   =  $4fe  |
scr_dump  =  $502  |
prv_lsto  =  $506  |
prv_lst   =  $50a  |
prv_auxo  =  $50e  |
prv_aux   =  $512  |
```

Numéros de Périphériques :

```
PRT     =  0  ;  Imprimante
AUX     =  1  ;  RS-232
CON     =  2  ;  Ecran (Emulateur vt-52)
MIDI    =  3  ;  MIDI
IKBD    =  4  ;  Clavier
RAWCON  =  5  ;  Ecran (raw characters)
```

Variables de Programme :

```
basepage  ;  Adresse de la page de base d'un programme chargé
text      ;  Segment de texte d'un programme chargé
data      ;  Segment de données d'un programme chargé
bss       ;  Segment Bss d'un programme chargé
symbols   ;  Tables de symboles d'un programme chargé
MEMfree   ;  Début de la mémoire libre (y compris PRG chargé)
MEMbase   ;  Longueur de la mémoire libre (y compris PRG chargé)
MEMend    ;  Fin de la mémoire libre (y compris PRG chargé)

physbase  ;  Adresse physique de l'écran
logbase   ;  Adresse logique de l'écran
```

Annexe A

Structure de la disquette

Une disquette (ou une partition du disque dur) se compose des parties suivantes :

1) Secteur Boot
2) FAT
3) Répertoire
4) Fichiers

Les nombres sont stockés sur disque en format Intel (c'est-à-dire à l'envers, en partant de l'octet faible) pour faciliter les échanges avec des disquettes en format MS-DOS.

Toutes les informations figurant sur la disquette sont placées dans des secteurs. Chaque secteur a normalement une longueur de 512 octets. Dans la zone où sont stockés les fichiers, les secteurs sont toujours groupés par deux pour former des clusters, ce qui permet de réduire le travail de gestion de fichier nécessaire.

1) Le secteur Boot est le premier secteur de chaque disquette. Il contient des informations importantes sur la structure du disque et éventuellement un programme Boot servant, par exemple, à faire exécuter un autre programme ou à charger le système d'exploitation à partir du disque.

Si le système est censé être lancé à partir d'une partition du disque dur, celle-ci doit également comporter un petit programme Boot.

Le secteur Boot présente la structure d'ensemble suivante :

Offset		Contenu	Signification
$00	0	BRA.S	Saut au programme Boot
$02	2	Oems	Réservé pour l'utilisateur
$08	8		Numéro de série sur 24 bits
$0B	11	BPS	Nombre d'octets par secteur
$0D	13	SPC	Nombre de secteurs par cluster (2)
$0E	14	RES	Nombre de secteurs réservés
$10	16	NFATS	Nombre de FATs sur la disquette
$11	17	NDIRS	Nombre d'entrées du répertoire
$13	19	NSECTS	Nombre de secteurs sur la disquette
$15	21	MEDIA	Descripteur de support (inutilisé)
$16	22	SPF	Nombre de secteurs dans la FAT
$18	24	SPT	Nombre de secteurs par piste
$1A	26	NSIDES	Nombre de faces de la disquette
$1C	28	NHID	Nombre de secteurs cachés (inutilisé)
$1E	30	Execflg	Mot copié après cmdload
$20	32	Ldmode	Si Ldmode=0 Charger fichier, sinon lire nombre de secteurs dans Sectcnt à partir de Ssect
$22	34	Ssect	Si Ldmode<>0, charger à partir d'ici
$24	36	Sectcnt	Si Ldmode<>0, lire Sectcnt secteurs
$26	38	Ldaddr	Adresse de chargement pour fichier ou secteurs
$2A	42	Fatbuf	Adresse pour buffer FAT et DIR
$2E	46	Fname	Nom de fichier : 8 caractères pour le nom, 3 caractères pour l'extension
$39	57		Réservé
$3A	58	Bootcode	Programme Boot
$01FE	510	CHECKSUM	Utilisé pour la somme de contrôle
$0200	512		

La fonction 7 du BIOS, 'Getbpb', et donc aussi l'instruction 'getbpb' du débogueur, fournissent les données suivantes, parmi celles indiquées ci-dessus :

BPB.w	:	Octets par secteur (512)
SPC.w	:	Nombre de secteurs par cluster (2)
BPC.w	:	Octets par cluster (1024)
SID.w	:	Nombre de secteurs dans le répertoire
SPF.w	:	Nombre de secteurs par FAT
SF2.w	:	Numéro de secteur de la seconde FAT
SCL.w	:	Numéro de secteur du premier cluster de données
NCL.w	:	Nombre de clusters de données sur le disque

2) Les fichiers sur la disquette ou dans la partition sont répartis entre les secteurs de la disquette. Il n'est, cependant, pas garanti que tous les secteurs d'un fichier figurent nécessairement sur des secteurs physiques consécutifs. Si un fichier a, par exemple, été prolongé, il peut parfaitement arriver que les secteurs placés à la suite de ce fichier contiennent déjà des données appartenant à un autre fichier. C'est pourquoi il existe une liste (La FAT : File Allocation Table) contenant pour chaque cluster de données le numéro du prochain cluster contenant la suite des données du fichier. Cette FAT existe en deux exemplaires placés à la suite, après le secteur Boot de la disquette. Le système d'exploitation travaille toujours avec la FAT 2 mais actualise toujours les deux FATs à la fois.

Chaque entrée de la FAT occupe 12 bits sur la disquette (pour des raisons d'économie de place).

Comme 12 bits ne permettent de définir que 4096 clusters, soit "seulement" 4 Mo, un format de 16 bits doit être employé pour les partitions du disque dur. Une entrée de ce type permet encore d'inscrire trois informations supplémentaires :

0	:	Le cluster est vide
$fff (ou $ffff sur disque dur)	:	Le cluster est le dernier cluster d'un fichier
$ff7 (ou $fff7 sur disque dur)	:	Le cluster est défectueux

Les entrées de 16 bits du disque dur sont très simples à décoder : chaque mot de 16 bits est stocké en format Intel (voyez plus haut) et chaque mot de la FAT correspond à un cluster. Les deux premiers numéros de cluster (0 et 1), et donc, du même coup, les deux premières entrées de la FAT, ne sont pas attribués. C'est normalement un 0 qui figure dans ces entrées sur le disque dur.

Les entrées de 12 bits sont déjà plus compliquées à lire. Du fait de cette longueur de 12 bits, deux entrées sont toujours réunies sur trois octets. Il faut donc décoder les entrées deux par deux. Voici comment lire une de ces imbrications de deux entrées :

23 c1 ab, ce qui correspond aux bits suivants :

1	:	Bits 8 à 11	de l'entrée 1
2	:	Bits 4 à 7	de l'entrée 1
3	:	Bits 0 à 3	de l'entrée 1
a	:	Bits 8 à 11	de l'entrée 2
b	:	Bits 4 à 7	de l'entrée 2
c	:	Bits 0 à 3	de l'entrée 2

Ces entrées peuvent être lues et affichées automatiquement, pour chaque cluster, à l'aide de l'instruction 'sclu'.

3) Le répertoire du disque contient pour chaque fichier stocké le nom, la longueur, l'heure et la date de création, le cluster de départ et un octet d'attribut indiquant, par exemple, que le fichier est protégé contre l'écriture ou qu'il ne doit pas apparaître dans le répertoire du disque. Chaque entrée du répertoire a une longueur de 32 octets et présente la structure suivante :

Nom du fichier	(8 octets)
Extension de fichier	(3 octets)
Attribut	(1 octet)
Zone libre	(10 octets)
Heure	(2 octets codés bit par bit)
Date	(2 octets codés bit par bit)
Premier cluster	(2 octets)
Longueur	(4 octets)

Ces informations sont affichées par l'instruction 'dir' du débogueur pour chaque entrée.

4) Comme nous l'avons déjà indiqué, les fichiers sont répartis parmi les clusters de la disquette. Le cluster de départ de chaque fichier est indiqué dans le répertoire alors que les clusters suivants doivent être déduits de la FAT.

Annexe B

Liste alphabétique des commandes du débogueur

!

! Instruction assembleur 2-219
" 2-250
$ octets,chaînes,mots,longs mots ... 2-225
$a octets ou 'Chaîne' 2-225
$l longs mots 2-225
$w mots 2-225
[Code hexa] {Symbole:} Mnémonique {Opérande{,Opérande}} 2-229

A

asm {nmb} 2-251

B

b 2-207
b0= ... b9= ad 2-206
bc 2-207
bssc 2-205

C

c {ad}{,breackpoint} 2-205
co 2-222
cson / csoff 2-228

D

d {de{,à}} 2-223
da {de{,à}} 2-223
Delete 2-241
diffscr 2-203
Dir {'chemin'} 2-240
Dir {'L'} 2-240
dl {de{,à}} 2-223
dloff 2-228
dlon 2-227
doreset 2-253
dw {de{,à}} 2-223

E

e {'Nom de fichier'}{,nmb} 2-198
eclu {'L',}déb,nmb 2-247
ee 'Chaîne' 2-200
ek 'Chaîne' 2-199

F

fclu {'L',}déb{,mod} 2-246
fill Nombre,Liste 2-239
fl {'a',}'Nom de fichier',de,à 2-230

G

g {ad}{,breakpoint} 2-204
getbpb {'L'} 2-245

H

hex, dec, oct, bin 2-228
h Expression 2-251
hf de,à,'Partie d'instruction' 2-234
hm de,à,'Instruction' 2-236
hn de,à,ad ou hn de,à,'Chaîne' 2-234

I

initx 2-206
is 'Nom de fichier' 2-231

K

k 2-251

L

l {de{,à}} 2-226
lclu {'L',}déb{,mod} 2-246
leclu {'L',}déb,nmb 2-247
ln {de,à} 2-203
ls {'de'}{,'à'} 2-202

M

m de,à,vers 2-238
mc zone1,zone2 2-236
mcl de,à 2-239
memon1 Expression1 2-210
memon2 Expression2 2-210
mmc zone1,zone2 2-237
mv 2-212

N

new 2-202

O

o Liste 2-220
ob Liste 2-222
oboff 2-223
ooff 2-223

P

pon, poff 2-250

Q

ql {de{,à}} 2-213
qoff 2-213
qon 2-212
quit 2-253

R

r 'Fichier'{,ad} 2-242
regoff 2-212
regon 2-210
resident {nmb} 2-252
rsec {'L',}de,nmb{,ad} 2-244
rv 2-243

S

samescr 2-203
sclu {'L',}-sec 2-246
sclu {'L',}clu 2-246
scrollon, scrolloff 2-250
secno ad 2-245
setdrv 'L' 2-241
setdrv n 2-241

T

t {nmb} 2-209
ta 2-209

U

u {nmb} 2-212
ul 2-219

usm Expression, us0 Expression ...
us9 Expression 2-214
uv 2-218

V

v 2-201

W

w {'Fichier'{,ad,len}} 2-242
wsec {{'n',}de,nmb{,ad}} 2-244

X

x 2-208
x liste 2-208

Achevé d'imprimer
sur les presses de l'imprimerie IBP
à Rungis (Val-de-Marne 94) (1) 46.86.73.54
Dépôt légal - Décembre 1988
N° d'impression: 5070

ATARI ST

GFA
ASSEMBLEUR

Vous voulez programmer des applications hyper-rapides, seul un langage extrêmement puissant, faisant directement appel aux instructions de base du microprocesseur pourra répondre à votre attente. GFA ASSEMBLEUR est le nouvel assembleur de choc de votre ATARI ST.

Conçu pour satisfaire tous les développeurs, même débutants, il tire le maximum du 68000 et de son vaste jeu d'instructions. Pour une programmation confortable, GFA ASSEMBLEUR propose un compilateur rapide, et un éditeur universel corrigeant automatiquement les erreurs lors de la saisie.

Totalement écrit en langage machine, GFA ASSEMBLEUR est avec son débugger un outil unique.

Caractéristiques techniques :

- Programme pouvant résider simultanément avec le GFA Basic.
- Détection automatique des erreurs dès l'entrée avec affichage du message correspondant.
- Formatage automatique des entrées pour un gain de temps à la saisie.
- Plusieurs modes d'optimisation des programmes à la compilation.
- Codage automatique des entrées pour un gain de place mémoire (Tokenisation) en format normal ou codes ASCII.
- Moniteur de disquette intégré.
- Editeur puissant pouvant être employé comme éditeur de texte source universel.
- Assemblage conditionné et récursif.
- Nombreuses options d'impression.
- Grande souplesse de l'interface utilisateur : souris ou accès clavier paramétrable par l'utilisateur.
- Assembleur macro rapide.
- Débugger indépendant, rechargeable.

3 325121 030338

RÉF. ST 033
PRIX 590 F TTC

EDITIONS MICRO APPLICATION
58 RUE DU FAUBOURG-POISSONNIÈRE 75010 PARIS
TÉL. (1) 47 70 32 44

Téléchargé sur
Le Vieux Manuel

WWW.MANUELS.ABANDONWARE-FRANCE.ORG