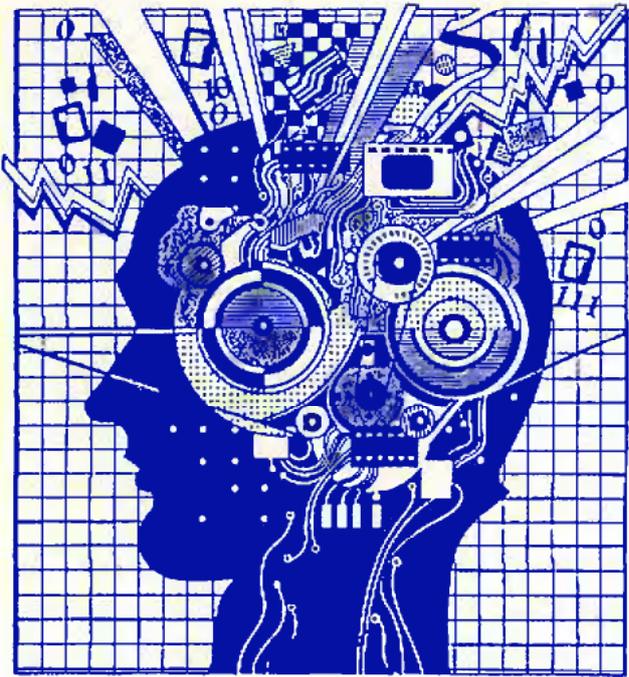


HiSoft Devpac



pour votre Atari ST

HiSoft
High Quality Software

HiSoft Devpac
pour votre Atari ST



HiSoft
High Quality Software

HiSoft DevpacST

Assembleur/Editeur/Débogueur

Configuration nécessaire:

Ordinateur Atari ST avec souris et lecteur de disquette.

Copyright © HiSoft 1988 pour la version anglaise,
Copyright © HiSoft 1990 pour la version française.

DevpacST Version 2, Octobre 1990

Première édition Août 1986 (ISBN 0 948517 04 2)

Deuxième édition Avril 1988 (ISBN 0 948517 11 5)

Version française Octobre 1990

DevpacST par Andy Pennell, David Nutkins, Alex Kiernan.

Traduit de l'anglais par Pierre Morel-Fourrier.

Correction et mise en page par Gilles Delcayre.

Les marques citées sont déposées par leurs propriétaires respectifs.

Tous droits réservés pour tous pays.

Ce logiciel et la documentation qui l'accompagne sont protégés par les lois du Copyright. Toute reproduction de la disquette ou du manuel à d'autres que l'utilisation privée du copiste est interdite.

Toute location est interdite.

Les informations figurant dans le présent ouvrage peuvent être sujettes à révision sans préavis de la part de HiSoft ou de Human Technologies. Le logiciel décrit dans ce document, ainsi que cette documentation, sont diffusés dans le cadre d'un accord de licence et de non divulgation et ne peuvent être utilisés ou copiés qu'en conformité avec les stipulations de l'accord. Toute copie, même partielle de ce logiciel sur bande magnétique, disque ou autre support, ou de sa documentation, à des fins autres que la sauvegarde de sécurité prévue par la loi est interdite.

Edité en France par:

HUMAN TECHNOLOGIES

87 rue de Billancourt

92100 Boulogne-Billancourt

Tél : (1) 46 04 88 71

Fax : (1) 46 04 82 24

Table des Matières

CHAPITRE 1 Introduction 1

Faites toujours une Copie de Sauvegarde	1
La Carte d'Enregistrement	1
Le Fichier LISEZMOI	1
Le Cycle de Développement	2
Contenu de la Disquette DevpacST	3
Comment utiliser ce manuel	4
Utilisateurs de DevpacST Version 1	4
Débutants	4
Utilisateurs Confirmés	5
Prise en main rapide	5

CHAPITRE 2 Editeur 9

Introduction	9
L'éditeur de texte	9
Quelques mots sur les boîtes de dialogue	10
Saisie d'un texte et déplacement du curseur	11
Touches curseur	12
Touche Tab	13
Touche Backspace	13
Touche Delete	13
Aller à une ligne donnée	13
Aller en début de fichier	14
Aller en fin de fichier	14
Quitter GenST	14
Effacement de texte	14
Effacer une Ligne	14
Effacer jusqu'à la fin de la ligne	14
Récupérer une ligne effacée	14
Effacer tout le texte	15
Opérations disque	15
Sauver un Texte	15

Sauver	15
Charger un Texte	16
Insérer un Texte	16
Recherche et Remplacement	16
Commandes de Bloc de Texte	17
Marquer un Bloc	17
Sauver un Bloc	17
Copier un Bloc	18
Détruire un Bloc	18
Copier un Bloc dans le Presse-Papier	18
Coller un Bloc	18
Imprimer un Bloc	18
Commandes diverses	19
A propos ...	19
Ecran d'aide	19
Préférences	19
Assembler et exécuter un programme	21
Assembler	21
Exécuter un programme	23
Débogueur	23
MonST	24
Exécution sous GEM	24
Correction des Erreurs	24
Lancer Autre...	25
Utilisation des Fenêtres et Accessoires de Bureau	25
Fenêtre GEM d'édition	25
Accessoires de bureau	25
Lancer GenST d'un double-clic	26
Utilisateurs de l'accessoire de bureau Saved!	26
CHAPITRE 3 Macro Assembleur	27
<hr/>	
Introduction	27
Appeler l'assembleur	27
A partir de l'éditeur	27
A partir du bureau GEM	29
Code de retour	31
Nom du Fichier Objet	31
Processus d'assemblage	31
L'Assemblage en Mémoire	32

Les types de fichier objet	33
Types de code	34
Syntaxe de l'Assembleur	35
Le champ Etiquette	35
Le champ Mnémonique	36
Le champ Opérande	36
Le champ Commentaire	36
Les Expressions	37
Étiquettes Locales	40
Jeu d'instructions	41
Alignement sur mot	41
Extensions au Jeu d'Instruction	42
Directives d'assemblage	43
Contrôle d'assemblage	43
Structure de répétition	51
Directive de contrôle du listing	52
Directives concernant les étiquettes	54
Les directives d'assemblage Conditionnel	56
Macro Instructions	58
Format des fichiers de sortie	64
Choisir le Bon Format de Fichier	65
Directives concernant le format des fichiers de sortie	66
Modules et Sections	66
Imports et Exports	68
Résumé des Directives	75
 CHAPITRE 4 Débogueur symbolique	 77

Introduction	77
Exceptions du 68000	78
Organisation de la Mémoire	79
Lancer MonST	81
À partir du bureau GEM	81
A partir de l'éditeur	81
Débogueur symbolique	82
Les boîtes de dialogue et les boîtes d'alerte de MonST	82
L'écran d'accueil	83
Le panneau de contrôle	83
Manement des fenêtres	84

Entrée des Commandes	84
Présentation de MonST	85
Guide de Référence de MonST	87
Expressions numériques	87
Commandes agissant sur les fenêtres	90
Touches Curseur	93
Echange d'écran	93
Interrompre l'Exécution des Programmes	94
Points d'arrêt	95
Historique	98
Quitter MonST	98
Chargement et Sauvegarde	99
Exécuter des Programmes	100
Recherche Mémoire	103
Divers	104
Version résidante de MonST	109
Résumé des commandes	110
Comment déboguer efficacement	112
Trucs et Astuces	112
Recherche des bogues	113
Analyse des exceptions	115
CHAPITRE 5 L'Editeur de liens	117
<hr/>	
Introduction	117
Lancer l'éditeur de liens	117
Ligne de Commande	117
L'exécution de LinkST	119
Fichiers de Contrôle	120
Lancer LinkST Automatiquement	122
Avertissements de LinkST	123
Messages d'erreur de LinkST	123

Appendice A Codes d'Erreur GEMDOS	125
--	------------

Appendice B Messages d'Erreur GenST	127
--	------------

Erreurs	127
Messages d'avertissements	130

Appendice C Organisation mémoire du ST	133
---	------------

Zone des Exceptions	133
Page de Base	134
Organisation mémoire	135

Appendice D Appeler le Système d'Exploitation	137
--	------------

GEMDOS - Entrées-sorties disque et écran.	137
En-Tête et Fin des Programmes	138
Résumé des Fonctions GEMDOS	139
BIOS - Basic I/O System (Entrées Sorties Élémentaires)	152
XBIOS - BIOS étendu	153
Bibliothèques GEM	154
Bibliothèque GEM AES	154
Bibliothèque Application	155
Bibliothèque Événement	156
Bibliothèque Menu	157
Bibliothèque Objet	157
Bibliothèque Boite de dialogue	158
Bibliothèque Graphique	158
Bibliothèque Presse-Papier	159
Bibliothèque Sélecteur de Fichiers	159
Bibliothèque Fenêtre	159
Bibliothèque Ressource	160
Bibliothèque Shell	160
Bibliothèque GEM VDI	161
Fonctions de Contrôle	162
Fonctions de Sortie	163
Fonctions d'attributs	164

Fonctions Raster	165
Fonctions d'Entrée	165
Fonctions de Renseignements	166
Squelette de Programme AES & VDI	166
Accessoires de Bureau	167
Châiner avec les Bibliothèques AES & VDI	168
Compilateur de Menus	168
Codes Ecran VT52	170

Appendice E Portage depuis d'autres Assembleurs

171

Atari MADMAC	171
GST-ASM	172
MCC Assembler	172
K-Seka	172
Fast ASM	172

Appendice F Bibliographie

173

Programmation du 68000	173
Manuels Techniques ST	173

Appendice G Support technique

175

Mise à jour	176
Suggestions	176

Appendice H Historique de DevpacST

177

Historique du Produit	177
Développement de Devpac	177
Résumé des améliorations de la version 2	177
L'éditeur	178
L'assembleur	179
Compatibilité	180
Le Débogueur	182
Intégration	182
L'éditeur de liens	183

CHAPITRE 1

Introduction

Faites toujours une Copie de Sauvegarde

Avant d'utiliser DevpacST, nous vous conseillons fortement de faire une copie de sauvegarde de votre disquette et de ranger l'original dans un endroit sûr. Ce logiciel n'est pas protégé contre la copie, ce qui vous permet de le copier facilement afin d'éviter certains désagréments. Cette disquette peut être copiée depuis le bureau GEM ou par tout utilitaire de copie. C'est une disquette simple face, mais elle peut bien sûr être utilisée dans un lecteur double face.

Avant de ranger votre disquette originale, notez dans le cadre ci-dessous le numéro de série qui y est inscrit. Il vous le sera demandé si vous désirez recourir à notre support technique.

Numéro de série : 29066 / 5 / 76

La Carte d'Enregistrement

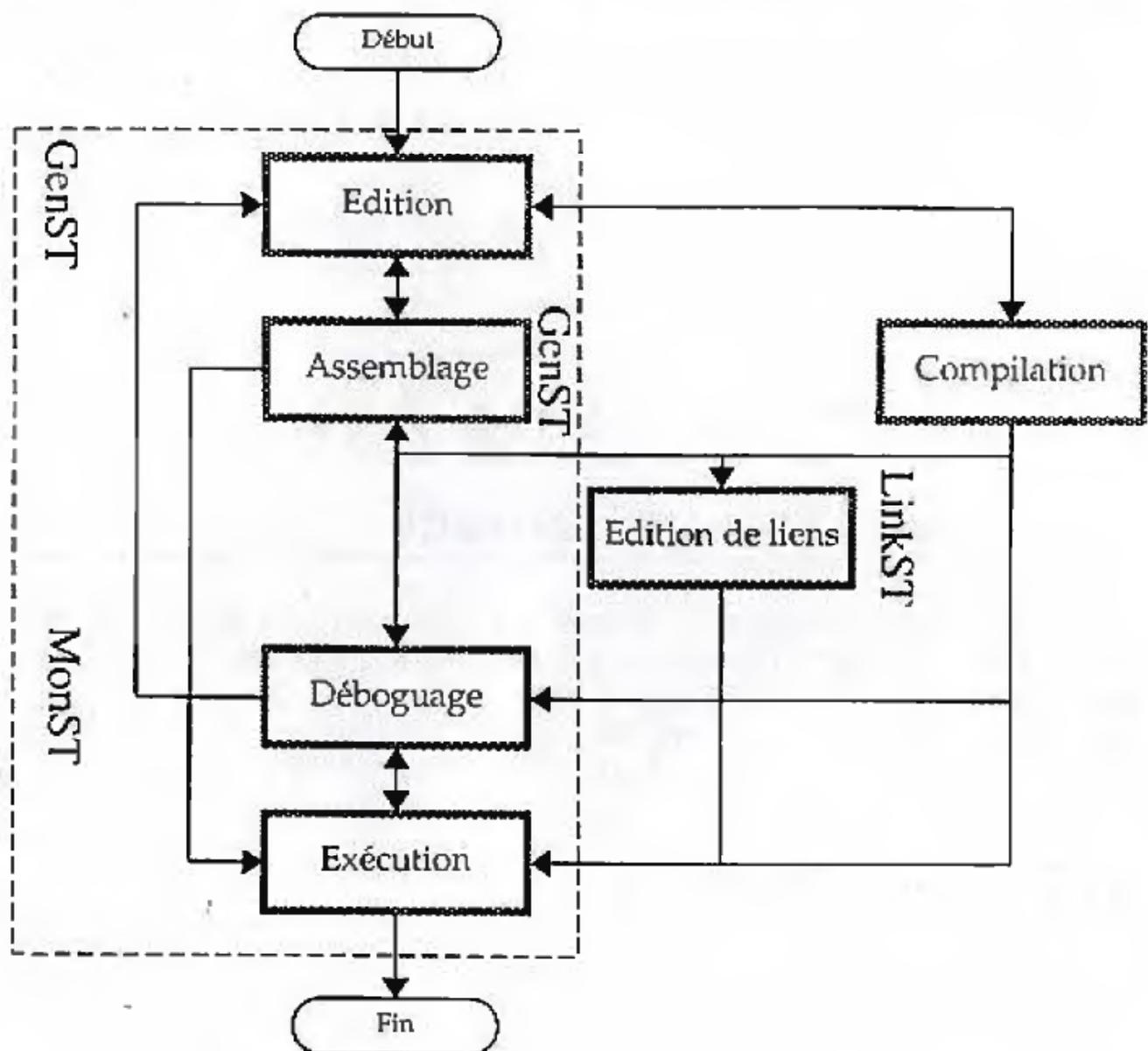
Une carte d'enregistrement est incluse dans ce manuel. Elle vous permet de bénéficier du support technique et des version ultérieures de ce logiciel (Voir **Appendice G**). Vous devez, pour cela, la remplir sans oublier le numéro de série et de version du logiciel, et nous la retourner. Vous trouverez également avec ce logiciel un mini guide détaillant le jeu d'instructions du 68000.

Le Fichier LISEZMOI

Comme tous les produits HiSoft, DevpacST est continuellement amélioré, et les dernières améliorations et corrections ne pouvant figurer dans ce manuel se trouvent dans le fichier LISEZMOI.S sur la disquette. Nous vous conseillons de lire le contenu de ce fichier maintenant, en double-cliquant sur son icône, puis en cliquant sur le bouton Voir. Vous pouvez imprimer ce fichier en cliquant sur le bouton Imprimer.

Le Cycle de Développement

La finalité d'un produit comme DevpacST est de vous permettre d'écrire des programmes en langage assembleur, de les traduire en code machine, et de les corriger lorsque, ou plutôt quand, ceux-ci ne fonctionnent pas. En fonction des applications, vous pourrez aussi, au moyen d'un éditeur de liens, construire un programme par petits modules, en y intégrant, pourquoi pas, des parties provenant d'un compilateur de langage évolué. Bien sûr, plus ce cycle de développement sera rapide, plus vite vous pourrez construire et exécuter vos programmes. DevpacST a été créé dans le but de permettre un cycle de développement aussi rapide et puissant que possible. Un bon schéma valant mieux qu'un long discours, voici représenté un cycle de développement classique :



Les étapes de compilation et d'édition de liens sont facultatives.

Contenu de la Disquette DevpacST

La disquette 3.5" simple-face fournie contient les fichiers suivants :

Programmes

- GENST2.PRG	Editeur de textes sous GEM et assembleur
- MONST2.PRG	Débogueur sous GEM
- MONST2.TOS	Débogueur sous TOS
< GENST2.TTP	Assembleur sans éditeur de texte
- AMONST.PRG	Débogueur résidant
- CHECKST.PRG	Programme de diagnostic
- LINKST.TTP	Editeur de liens au format GST
- NOTRACE.PRG	Inhibiteur de l'exception Trace
- MENU2ASM.TTP	Compilateur de menus

Fichiers Texte

- LISEZMOI.TXT	Dernières améliorations de DevpacST
- DEMC.S	Programme TOS très simple utilisé dans la section prise en main
- GEMTEST.S	Source d'une démo GEM simple.
- DESKACC.S	Exemple d'accessoire de bureau
- GEMMACRO.S	Macros d'interface VDI/AES
- AESLIB.S	Source de la bibliothèque AES
- VDILIB.S	Source de la bibliothèque VDI
- NOTRACE.S	Source du programme NOTRACE.PRG
- MENUTEST.S	Programme d'exemple d'utilisation de menus sous GEM
- MENUTEST.MDF	Fichier d'exemple de définition de menu
- MAKEGEM.S	Crée GEMLIB
- GEMLIB.LNK	Fichier de contrôle d'édition de liens pour GEMLIB

Fichier binaire

- GEMLIB.BIN	Bibliothèque AES & VDI
--------------	------------------------

Dossier

OLGEM	Exemples GEM actualisés de GenST 1
-------	------------------------------------

Comment utiliser ce manuel

Ce manuel n'a pas la prétention de vous apprendre à programmer en assembleur 68000 ni d'en détailler le jeu d'instructions. Pour cela, reportez vous à la bibliographie, ou au mini guide de référence du jeu d'instruction du 68000. Les Appendices donnent un aperçu des différents aspects techniques de l'Atari ST mais ne forment en aucun cas une description technique complète de la machine.

Ce manuel est constitué de cinq chapitres et de huit Appendices. Le premier chapitre constitue cette introduction, les autres concernent l'éditeur de textes, le macro assembleur, le débogueur et l'éditeur de liens. Les Appendices détaillent divers aspects complémentaires à DevpacST. Vous pouvez utiliser ce manuel de différentes façons selon que vous êtes :

Utilisateurs de DevpacST Version 1

Lisez l'Appendice H décrivant les nouvelles possibilités, puis la section **Guide de Référence de MonST** du **Chapitre 4** si vous désirez utiliser MonST qui a été complètement remanié. L'autre paragraphe que vous pouvez lire concerne le **Format des Fichiers Objets** du **Chapitre 3** si vous désirez générer du code destiné à l'édition de liens.

Débutants

Si vous débutez en assembleur, nous vous recommandons de lire l'un des livres cités en **Bibliographie** dans ce manuel.

A la fin de ce chapitre se trouve une prise en main qui vous familiarisera au produit.

Nous vous conseillons aussi le **Chapitre 2** qui détaille l'éditeur de textes, alors que la lecture d'une grande partie du **Chapitre 3** concernant l'assembleur ne vous apportera pas grand chose tant que vous ne vous serez pas familiarisé avec l'assembleur du 68000. Lisez aussi la **Présentation de MonST**, **Chapitre 4**. Par contre le **Chapitre 5** et les **Appendices** peuvent être mis de côté pour le moment. Vous pouvez enfin regarder les exemples de sources assembleur, mais ceux concernant la programmation de GEM n'ont pas été écrits pour des programmeurs débutants.

Utilisateurs Confirmés

Si vous avez une bonne expérience de la programmation en assembleur 68000, mais que vous ne connaissez pas DevpacST, il existe un moyen très simple d'assembler un fichier source :

Lancez GENST2.PRG, appuyez sur **Alt-L** et sélectionnez le fichier à charger dans l'éditeur de textes. Appuyez sur **Alt-A** et sélectionnez les options d'assemblage que vous désirez. Si vous désirez obtenir un fichier exécutable, cliquez sur le bouton Mémoire pour accélérer l'assemblage. Appuyez sur **Return** pour lancer l'assembleur qui peut être temporairement suspendu par les touches **Ctrl-S** et repris par **Ctrl-Q**. Tous les messages d'erreurs sont mémorisés, et de retour à l'éditeur, le curseur sera positionné à l'endroit de la première erreur. Pour visualiser les erreurs suivantes, il vous suffit d'appuyer sur **Alt-J**.

Pour exécuter votre programme s'il ne contient pas d'erreurs, appuyez sur **Alt-X** si vous l'avez assemblé en mémoire. Si vous l'avez assemblé sur disque, appuyez sur **Alt-O** et sélectionnez le nom de votre programme.

La prise en main qui suit va vous présenter rapidement le débogueur. Si vous avez un problème, veuillez lire la section correspondante du manuel avant de contacter notre support technique.

Prise en main rapide

Ces lignes vous permettront d'apprendre à vous servir de l'éditeur, de l'assembleur et du débogueur et de voir avec quelle facilité vous allez vous servir de DevpacST.

Nous allons assembler, exécuter et déboguer un programme simple qui contient deux erreurs. Ce programme affiche un message et attend la pression d'une touche au clavier.

Assurez vous que votre disque contient les fichiers **MONST2.PRG** et **DEMO.S** avant de lancer **GENST2.PRG** en double cliquant sur son icône. Après un court instant, une fenêtre vide s'affiche à l'écran. Chargez alors le fichier assembleur en cliquant sur l'option **Charger...** du menu **Fichier**. Le sélecteur de fichiers **GEM** standard apparaît. Pour charger le fichier **DEMO.S**, double-cliquez sur ce nom, ou tapez le au clavier suivi de **Return**.

Le début du fichier s'affiche dans la fenêtre. Pour le visualiser entièrement, cliquez sur la barre d'ascenseur ou utilisez les touches curseur.

Avec un petit programme, il vaut mieux assembler sans produire de listing ni de fichier binaire si l'on veut vérifier la syntaxe du source et supprimer les fautes de frappe.

Cliquez sur l'option **Assembler** du menu **Prg** pour faire apparaître la boîte de dialogue **Options d'assemblage**. Choisissez dans la section **Sortie** le bouton **Aucune**. Appuyez sur la touche **Return** pour lancer l'assemblage.

L'assembleur va afficher le message d'erreur `instruction non reconnue en ligne XX`. Appuyez sur une touche pour revenir à l'éditeur. Le curseur va se positionner sur la ligne en cause avec le message d'erreur affiché en rappel dans la ligne d'information de la fenêtre.

Changez `MOV.W` en `MOVE.W`. Relancez l'assembleur et cliquez sur le bouton **Mémoire** de la boîte de dialogue afin que le programme soit assemblé en mémoire plutôt que sur le disque. Cela permet une rapidité maximum et la possibilité de tester immédiatement ce que l'on fait. Appuyez sur **Return** pour lancer l'assemblage et appuyez une autre fois sur **Return** pour revenir à l'éditeur.

L'assemblage s'est bien passé. Cliquez sur l'option **Exécuter** du menu **Prg** pour exécuter votre nouveau programme. Il ne se passe pas grand-chose, excepté trois bombes qui apparaissent sur l'écran. Il y a un bogue semblé-t-il.

Pour traquer ce bogue, lancez le débogueur en cliquant sur **Déboguer** du menu **Prg**. Le débogueur est décrit en détail plus loin dans ce manuel, mais nous voulons juste exécuter notre programme pour traquer nos trois bombes. Appuyez sur **Ctrl-R**.

Le message **Erreur Adressage** apparaît à l'écran dans la fenêtre du bas. La fenêtre de désassemblage montre l'instruction en cause :

```
MOVE.W 1,-(A7)
```

Cette instruction produit une erreur d'adressage car l'adresse 1 est en mémoire protégée, et ne peut être lue en mode utilisateur. Nous avons oublié le caractère dièse devant le 1. Nous voulons en effet ranger la valeur immédiate 1 dans la pile. Appuyez sur **Ctrl-C** pour retourner à l'éditeur et corriger cette erreur.

Appuyez sur **Alt-T** pour vous déplacer au début du fichier. Cliquez sur **Trouver** du menu **Recherche**. Pour trouver l'instruction en cause, entrez :

```
move.w
```

et appuyez sur `Return` pour commencer la recherche. La première occurrence trouvée a un signe dièse. Appuyez sur `Alt-N` pour trouver la suivante. Le curseur est alors positionné sur la ligne :

```
move.w c_conin,-(sp)
```

Il s'agit bien de la ligne en cause! Ajoutez un caractère dièse :

```
move.w #c_conin,-(sp)
```

et assemblez de nouveau votre programme corrigé. Exécutez le programme une fois assemblé, et là, un message s'affiche à l'écran à la place des trois bombes. Appuyez sur une touche pour revenir à l'éditeur.

Cependant, vous avez remarqué que l'écran ne paraissait pas très "propre". Le message affiché recouvrait le motif du bureau, et peut-être le curseur de la souris. Tout cela parce que DEMO est un programme TOS qui est exécuté sur un écran GEM. Pour changer cela, cliquez sur Exécution sous GEM du menu Prg. La marque devant cette entrée de menu disparaît. Lancer à nouveau l'exécution et notez combien l'affichage se fait maintenant proprement ! Attention lorsque vous voulez exécuter des programmes GEM à bien vérifier que la marque est bien présente devant l'option du menu. Dans le cas contraire, des choses étranges peuvent apparaître!

Bien que notre programme fonctionne correctement, nous pouvons tracer en pas-à-pas grâce au débogueur MonST. Pour cela, cliquez sur Déboguer du menu Prg pour l'activer.

Plusieurs fenêtres sont présentes à l'écran. Celle du haut affiche les registres du 68000, la deuxième désassemble notre programme, la troisième affiche une partie de la mémoire et celle du bas contient diverses informations.

L'instruction courante est affichée dans la deuxième fenêtre :

```
MOVE.L #string,-(A7)
```

Les symboles du programme source apparaissent dans le débogueur car l'option Informations de débogage est sélectionnée par défaut.

Examinons la mémoire autour de l'adresse `string`. Appuyez sur `Alt-3` pour activer la fenêtre 3 (son titre passe en vidéo inverse). Appuyez sur `Alt-A` pour faire apparaître la boîte de dialogue vous demandant l'adresse de début de la fenêtre. Entrez :

```
string
```

(en minuscules) et appuyez sur `Return`. La fenêtre 3 affiche la mémoire à partir de l'adresse spécifiée, et le message apparaît à l'écran en ASCII et hexadécimal.

Appuyez sur `Ctrl-Z` pour exécuter l'instruction `MOVE`. L'écran est mis à jour et prend en compte les modifications sur `PC` et `A7`. Appuyez une seconde fois sur `Ctrl-Z` pour exécuter l'instruction `MOVE.W`. Regardez l'affichage hexadécimal à droite de `A7`. On y voit la valeur `9`. Ce qui s'est passé est conforme à ce que l'on attendait !

L'instruction suivante est `TRAP #1`. C'est un appel à GEMDOS pour imprimer une chaîne de caractères. Mais... que se passerait-il si une chaîne de caractères était affichée sur l'écran MonST ? En fait, MonST possède un deuxième écran d'affichage pour éviter les interférences avec votre programme. Appuyez sur `V` pour le voir. Un écran vide s'affiche. Appuyez sur une touche quelconque pour revenir au débogueur.

Appuyez donc sur `Ctrl-Z` pour exécuter l'instruction `Trap`. Vérifiez que le message est bien affiché à l'écran en appuyant sur la touche `V`.

Appuyez deux fois sur `Ctrl-Z` pour atteindre la deuxième instruction `Trap`. Celle-ci attend l'appui sur une touche. Appuyez sur `Ctrl-Z` et l'écran programme apparaît, attendant que vous appuyiez sur une touche. Appuyez sur la touche `q` par exemple. De retour à l'écran de MonST, regardez les 8 bits de poids faible du registre `D0` dans la fenêtre d'affichage des registres. Ils contiennent la valeur `$71`, code ASCII du caractère `q`. A droite de cette valeur est affiché le caractère `q` (à moins que vous ne soyez en basse résolution).

Le dernier `Trap` termine le programme. Pour le laisser se terminer tout seul, appuyez sur `Ctrl-R`. Vous retournez alors à l'éditeur de textes.

Veillez noter la façon dont nous avons utilisé la police de caractères `Courier` pour représenter les textes qui apparaissent à l'écran ou les touches qui doivent être entrées au clavier et la police `AvantGarde` pour les options de menus et les commandes. Bien sûr, `Ctrl-Z` implique l'appui sur la touche `Control` du clavier puis simultanément, l'appui sur la touche `Z` alors que `Return` signifie que vous devez appuyer sur la touche `Return` du clavier. Ces conventions sont utilisées tout au long du manuel.

CHAPITRE 2

Editeur de Texte

Introduction

Pour saisir et assembler vos programmes, vous avez besoin d'un éditeur de texte quelconque et d'un assembleur. GenST combine ces deux fonctionnalités en vous proposant un éditeur de texte rapide sous GEM et un assembleur intégré. Vous pouvez également exécuter les programmes assemblés en mémoire sans quitter l'éditeur ni accéder au disque. Vous accédez au débogueur en appuyant sur une touche. L'environnement intégré que vous propose GenST vous permet de corriger très rapidement vos programmes sans accès disque ou programme complémentaire.

Ce chapitre traite du fonctionnement de l'éditeur de texte et décrit la façon d'assembler les programmes - sans pour autant détailler complètement le fonctionnement de l'assembleur et du débogueur qui sera traité dans les chapitres suivants.

Pour lancer GenST, double-cliquez sur l'icône GENST.PRG à partir du bureau GEM. Une barre de menus et une fenêtre vide apparaissent à l'écran : vous pouvez maintenant saisir et assembler vos programmes.

L'éditeur de texte

Un éditeur de texte est un programme avec lequel vous pouvez saisir et modifier des lignes de texte au clavier, les sauvegarder et les charger depuis un disque. Il y a deux types d'éditeurs de texte. Les éditeurs ligne, qui traitent séparément chaque ligne de texte, et les éditeurs pleine page qui affichent le texte entier sur l'écran. Ces derniers sont beaucoup plus faciles à utiliser.

L'éditeur de GenST est un éditeur pleine page. Vous pouvez bien sûr saisir et modifier votre texte, le sauvegarder et le recharger à partir d'un disque. Vous pouvez également imprimer tout ou partie de votre texte, rechercher et remplacer des chaînes de caractères, ou utiliser les accessoires de bureau GEM. Il fonctionne sous GEM, ce qui signifie que vous bénéficiez de l'ergonomie de l'environnement GEM : Fenêtres, menus et souris. Toutefois, si vous avez l'habitude de travailler sans souris, sachez que presque toutes les options de l'éditeur sont disponibles au clavier.

L'éditeur de texte est orienté mémoire, c.a.d. que le fichier que vous êtes en train d'éditer se trouve entièrement en mémoire tant que vous y travaillez. Vous n'êtes pas pénalisé par des accès disque intempestifs chargeant votre texte bout par bout. Grâce à la taille mémoire conséquente du ST (au moins 512 K), vous pouvez éditer des fichiers de plus de 300 Ko ... si votre disque a assez de place pour eux ! De plus, toutes les fonctions d'édition sont accélérées puisque GenST travaille exclusivement en mémoire.

Une fois votre programme saisi, vous pouvez le sauver sur disque. L'éditeur possède de nombreuses options de chargement et de sauvegarde. Vous pouvez, par exemple, sauver tout ou partie d'un texte, ou charger un texte au milieu de celui que vous êtes en train d'éditer.

Il existe différentes façons d'utiliser l'éditeur. Vous pouvez accéder aux différentes fonctionnalités de la manière suivante :

- En appuyant sur une touche de fonction ou de déplacement de curseur.
- En cliquant sur une option de menu, comme Sauver.
- En utilisant un raccourci clavier, obtenu en pressant simultanément la touche Alternante (Alt) avec une autre, comme Alt-S pour Sauver sous....
- En pressant simultanément la touche Control (Ctrl) avec une autre, comme Ctrl-A pour déplacer le curseur d'un mot vers la gauche.
- En cliquant sur l'écran, par exemple sur une barre de défilement.

Les raccourcis clavier ont été choisis pour être facile à retenir. Les commandes associées à la touche Control sont compatibles avec WordStar et avec beaucoup d'autres.

A tout moment en cas de problème, l'appui de la touche Help provoquera l'affichage d'une boîte de dialogue récapitulant la signification de tous les raccourcis clavier ne figurant pas déjà dans les menus.

Quelques mots sur les boîtes de dialogue

Comme l'éditeur fait une utilisation intensive des boîtes de dialogue, il n'est pas superflu d'en récapituler le fonctionnement, en particulier pour la saisie de texte. Les boîtes de dialogue de l'éditeur se compose généralement de boutons, de boutons radio et de zones de texte éditables.

Il est possible de cliquer sur un bouton avec la souris, et cela a souvent comme effet de fermer la boîte. Il y a habituellement un bouton par défaut dont le contour est plus large que les autres. L'appui sur la touche `Return` revient à cliquer sur le bouton par défaut.

Les boutons radio sont des groupes de boutons dont un seul peut être sélectionné à la fois. Cliquer sur l'un d'entre eux désélectionne automatiquement les autres.

Les zones de texte éditables sont représentées par une ligne de tirets. Une barre verticale indique la position du curseur. Il est possible d'y saisir des caractères et d'en corriger le contenu grâce aux touches `Backspace`, `Delete` et curseur. L'appui sur la touche `Esc` permet d'effacer complètement la ligne. Lorsqu'il existe plusieurs zones de texte éditables dans une même boîte, vous pouvez passer de l'une à l'autre en utilisant les touches curseur haut et bas, ou en cliquant dessus avec la souris.

Certaines boîtes de dialogue n'autorisent que certains caractères dans leurs champs éditables. Par exemple, la boîte de dialogue `Aller ligne...` n'autorise que la saisie de chiffres.

Saisie d'un texte et déplacement du curseur

Après avoir lancé `GenST`, une fenêtre vide contenant une ligne d'information apparaît. Le *curseur* est un carré noir clignotant en haut à gauche de la fenêtre.

La ligne d'information en haut de la fenêtre vous informe sur la position du curseur (ligne, colonne) et sur le nombre d'octets mémoire libre. Initialement, ce nombre est égal à 59.980 octets (par défaut, la mémoire réservée pour un texte est fixée à 60.000 octets). Vous pouvez, si vous le désirez, modifier cette valeur en sélectionnant l'option `Préférences` (Voir plus loin). Les 20 octets "manquants" sont utilisés par l'éditeur pour ses besoins internes. La partie restante de la ligne d'information est utilisée pour y afficher les messages d'erreur qui seront généralement accompagnés d'un signal sonore (un 'bip'). De tels messages sont effacés par l'appui sur une touche.

Pour entrer du texte, utilisez le clavier. Les caractères tapés au clavier s'affichent dans la fenêtre alors que le curseur avance sur la ligne. A la fin de chaque ligne, appuyez sur `Return` (ou sur la touche `Enter` du pavé numérique) pour commencer la ligne suivante. Vous pouvez corriger des erreurs de frappe avec la touche `Backspace`, qui efface le caractère situé avant le curseur, ou la touche `Delete`, qui efface le caractère se trouvant sous le curseur.

Le principal avantage d'un éditeur de texte sur ordinateur par rapport à une machine à écrire est sa possibilité de corriger des textes saisis auparavant. Les nombreuses options de l'éditeur vous permettent de vous déplacer en toute liberté dans votre texte.

Touches curseur

Pour déplacer le curseur dans le texte afin de corriger des erreurs ou saisir de nouveaux caractères, utilisez les touches curseur désignées par CL, CR, CU et CD. Si vous déplacez le curseur au delà de la fin de la ligne, cela ne modifiera pas votre texte, mais si vous tapez des caractères à cet endroit, l'éditeur ajoutera le texte à la fin de la ligne. Si vous saisissez des lignes importantes, la fenêtre de visualisation se déplacera horizontalement si nécessaire.

Si vous appuyez sur CU en haut de la fenêtre, le texte se décalera vers le bas, ou le message `Début de fichier` apparaîtra si vous êtes sur la première ligne du texte. De la même manière, si vous appuyez sur CD en bas de la fenêtre, le texte se décalera vers le haut, ou le message `Fin de fichier` apparaîtra si vous êtes sur la dernière ligne du texte.

Vous pouvez déplacer le curseur caractère par caractère en cliquant sur les flèches des ascenseurs verticaux et horizontaux.

Si vous êtes un habitué de WordStar, les touches `Ctrl-S`, `Ctrl-D`, `Ctrl-E` et `Ctrl-X` sont équivalentes aux touches curseur.

Pour vous déplacer au début d'une ligne, appuyez sur `Ctrl CL`. Pour vous déplacer à la fin, appuyez sur `Ctrl CR`.

Pour vous déplacer d'un mot vers la gauche, appuyez sur `Shift CL`. Pour vous déplacer d'un mot vers la droite, appuyez sur `Shift CR`. Un mot est composé de n'importe quel caractère, et est encadré par des espaces, tabulations ou début/fin de ligne. `Ctrl-A` et `Ctrl-F` déplacent également le curseur d'un mot vers la gauche ou la droite.

Pour déplacer le curseur d'un écran vers le haut, vous pouvez cliquer sur la partie supérieure grise de l'ascenseur, ou appuyer sur les touches `Ctrl-R` ou `Shift CU`. Pour déplacer le curseur d'un écran vers le bas, vous pouvez cliquer sur la partie inférieure grise de l'ascenseur, ou appuyer sur les touches `Ctrl-C` ou `Shift CD`.

Vous pouvez déplacer le curseur à un endroit quelconque de la fenêtre en cliquant avec la souris à cet endroit (il n'existe malheureusement pas d'équivalent WordStar de cette option).

Touche Tab

La touche **Tab** insère un caractère spécial (code ASCII 9) dans le texte. A l'écran il apparaît comme une suite d'espaces - apparaît seulement. Car un appui sur **Tab** aligne le curseur à une position multiple de 8 colonnes. Si vous appuyez sur **Tab** au début d'une ligne, le curseur se déplace à la prochaine colonne multiple de 8, +1, soit colonne 9. Les tabulations sont très utiles pour aligner du texte verticalement. Sa principale utilité ici est d'aligner les lignes d'instructions assembleur. Effacer une tabulation équivaut à l'écran à effacer plusieurs espaces. Les tabulations ont l'avantage de n'utiliser qu'un seul octet en mémoire alors qu'elles sont représentées à l'écran comme plusieurs espaces. Vous pouvez changer la longueur de tabulation en faisant appel à l'option **Préférences** décrite un peu plus loin.

Touche Backspace

Cette touche efface le caractère qui précède le curseur. Si vous appuyez sur **Backspace** au début d'une ligne, le caractère "fin de ligne" invisible sera détruit et la ligne courante sera ajoutée à la fin de la ligne précédente. Si vous appuyez sur **Backspace** au delà de la fin de la ligne, le dernier caractère de la ligne sera effacé, à moins que la ligne ne soit vide; ce qui, dans ce cas, repositionnera le curseur en début de la ligne.

Touche Delete

Cette touche efface le caractère sous le curseur et n'a pas d'effet si le curseur se trouve au delà de la fin de la ligne.

Aller à une ligne donnée

Pour vous déplacer à une ligne donnée du texte, cliquez sur l'option **Aller ligne...** du menu **Options**, ou appuyez sur **Alt-G**. Tapez, dans la boîte de dialogue, le numéro de ligne à laquelle vous désirez aller. Appuyez sur **Return** ou cliquez sur **Ok** pour aller à la ligne spécifiée, ou bien cliquez sur **Annuler** pour interrompre l'opération. Cette opération déplace le curseur à la ligne spécifiée en rafraîchissant l'écran si nécessaire, ou affiche le message d'erreur **Fin de Fichier**, si la ligne spécifiée n'existe pas.

Aller en début de fichier

Pour aller en début de fichier, cliquez sur l'option Début de fichier du menu Options, ou appuyez sur la touche **Alt-T**. Le curseur se repositionnera sur la première ligne du texte en le réaffichant si nécessaire.

Aller en fin de fichier

Pour aller en fin de fichier, cliquez sur l'option Fin de fichier du menu Options, ou pressez **Alt-B**.

Quitter GenST

Pour quitter GenST, cliquez sur l'option Quitter du menu Fichier, ou appuyez sur **Alt-Q**. Si le texte a été modifié sans avoir été sauvé, une boîte d'alerte vous demandera confirmation pour quitter. Cliquez soit sur Annuler pour retourner à l'éditeur et faire une sauvegarde, soit sur Ok pour quitter GenST sans tenir compte des modifications.

Effacement de texte

Effacer une Ligne

Appuyez sur **Ctrl-Y** pour effacer la ligne courante.

Effacer jusqu'à la fin de la ligne

Le texte compris entre la position courante du curseur et la fin de ligne peut être effacé en appuyant sur **Ctrl-Q** (équivalent à la commande WordStar **Ctrl-Q Y**).

Récupérer une ligne effacée

Quand une ligne est effacée par l'une des deux commandes précédentes, vous pouvez la réinsérer dans le texte en appuyant sur **Ctrl-U** ou **Undo**. Cette opération peut être répétée plusieurs fois pour dupliquer une ligne de texte.

Effacer tout le texte

Pour effacer tout le texte, cliquez sur l'option Nouveau du menu Fichier, ou appuyez sur ALT-C. Si le texte a été modifié et non sauvé, une confirmation vous sera demandée par une boîte d'alerte. Cliquez alors sur Ok pour effacer le texte, ou sur Annuler pour abandonner l'opération.

Opérations disque

Il est indispensable de pouvoir enregistrer un texte que vous avez entré pour pouvoir le réutiliser ultérieurement. Pour cela, l'éditeur dispose de nombreuses fonctions de lecture et d'écriture disque.

Sauver un Texte

Pour enregistrer le texte que vous êtes en train d'éditer, cliquez sur l'option Sauver sous... du menu Fichier, ou appuyez sur ALT-S. Choisissez, au travers du sélecteur de fichier, le nom du fichier et cliquez sur Ok ou appuyez sur Return pour sauver le fichier sur le disque. Si le fichier existe déjà, il sera effacé et remplacé par le nouveau texte. Mais si l'option Copie de secours (*backup*) est sélectionnée dans les Préférences, l'ancien fichier sera renommé avec une extension .BAK (détruisant un éventuel fichier .BAK existant).

Si une erreur survient, une boîte d'alerte indiquant le numéro de l'erreur TOS est affichée. Voyez en **Appendice A** la signification de ces messages d'erreur. Si vous cliquez sur Annuler, le texte ne sera pas sauvé sur disque.

Sauver

Si vous avez déjà sauvé ou chargé un texte, GenST se souvient du nom que vous avez donné et l'affiche en guise de titre de la fenêtre principale. Vous pouvez sauver votre fichier sans passer par le sélecteur de fichiers en cliquant sur l'option Sauver du menu Fichier, ou en appuyant sur Shift-Alt-S. Le fichier sera sauvé sous son nom courant. Si vous essayez d'utiliser l'option Sauver avant que votre texte soit nommé, GenSt vous présentera le sélecteur de fichiers, de la même manière que pour Sauver sous....

Charger un Texte

Pour charger un nouveau texte, cliquez sur l'option Charger du menu Fichier, ou appuyez sur Alt-L. Si le texte que vous étiez en train d'éditer a été modifié sans être sauvegardé, un message d'alerte vous demandera confirmation. Choisissez ensuite le nom du fichier à charger grâce au sélecteur de fichier GEM. Cliquez sur Ok pour charger le fichier, ou sur Annule si vous vous êtes trompés. Si l'éditeur arrive à charger le fichier, le texte s'affichera dans la fenêtre principale. Dans le cas contraire, vous serez prévenu par un message d'alerte. Si le fichier ne tient pas en mémoire, essayez d'augmenter la taille de mémoire allouée au texte dans les Préférences, puis rechargez le fichier.

Insérer un Texte

Vous pouvez insérer le contenu d'un fichier à partir de la position courante du curseur en cliquant sur l'option Fusionner du menu Fichier, ou en appuyant sur Alt-I. Choisissez ensuite le nom du fichier dans la boîte de sélection de fichier et cliquez sur Ok. Le fichier sélectionné sera inséré si la mémoire est suffisante !

Recherche et Remplacement

Pour rechercher une chaîne de caractères dans le texte, cliquez sur l'option Trouver... du menu Recherche, ou appuyez sur Alt-F. Entrez ensuite les chaînes de caractères à chercher et/ou à remplacer dans la boîte de dialogue qui vous est proposée. Si vous cliquez sur Annuler, aucune recherche n'aura lieu. Cliquez sur Suivant ou appuyez sur Return, pour que la recherche s'effectue à partir du curseur vers la fin du texte, ou cliquez sur Précédent pour qu'elle s'effectue de la position courante du curseur vers le début du texte. Si vous ne voulez pas effectuer de remplacement, laissez le champ correspondant vide. Lorsque la recherche aboutie, le curseur se positionne à l'endroit où la première chaîne de caractères a été trouvée. Sinon un message Non Trouvé apparaît dans la ligne d'information de la fenêtre principale. Par défaut, aucune différence n'est faite entre les minuscules et les majuscules lors de la recherche. Ainsi par exemple, si vous désirez rechercher toutes les occurrences du mot test, vous trouverez également toutes celles de TEST ou Test. Cliquez sur MAJ/min pour faire la différence entre les minuscules et les majuscules.

Pour rechercher l'occurrence suivante, cliquez sur l'option Trouver suivant du menu Recherche, ou appuyez sur Alt-N. La recherche recommencera à la position qui suit juste celle du curseur.

Pour rechercher l'occurrence précédente, cliquez sur l'option Trouver précédent du menu Recherche, ou appuyez sur **Alt-P**. La recherche recommencera à la position qui précède juste celle du curseur.

Une occurrence peut être remplacée par un texte spécifié. Pour cela cliquez sur l'option Remplacer... du menu Recherche, ou appuyez sur **Alt-R**. Une fois la chaîne remplacée, l'éditeur cherchera l'occurrence suivante.

Si vous désirez remplacer toutes les occurrences d'une chaîne donnée par une autre chaîne, cliquez sur l'option Remplacer tout du menu Recherche. Le remplacement commence à partir de la position du curseur jusqu'à la fin du texte. Pendant le remplacement, la touche **Esc** peut être utilisée pour interrompre le processus, et la ligne d'information affichera combien de remplacement auront été effectués. Il n'y a délibérément pas de raccourcis clavier pour cette option pour éviter de l'activer accidentellement.

Commandes de Bloc de Texte

Un *bloc* est une partie marquée du texte qui peut être copiée, supprimée, imprimée ou enregistrée sur disque. Les touches de fonction sont utilisées comme raccourcis clavier.

Marquer un Bloc

Pour marquer le début d'un bloc, déplacez le curseur à l'endroit désiré et appuyez sur **F1**. Pour marquer la fin du bloc, déplacez le curseur à l'endroit désiré et appuyez sur **F2**. Vous n'êtes pas obligé de marquer d'abord le début du bloc avant sa fin si c'est plus pratique pour vous !

Un bloc marqué est représenté en vidéo inverse, blanc sur noir. Lorsque vous modifiez une ligne à l'intérieur d'un bloc, celle-ci est affichée en noir sur blanc. Mais elle sera affichée en blanc sur noir lorsque vous quitterez cette ligne ou choisirez une commande.

Sauver un Bloc

Lorsqu'un bloc est sélectionné, vous pouvez le sauver sur disque en appuyant sur **F3**. Si aucun bloc n'est sélectionné, le message Quel bloc ? est affiché sur la ligne d'information. Si le début d'un bloc se trouve après sa fin, le message Bloc Invalide ! apparaîtra. Ces deux erreurs interrompent la commande. Sinon, le sélecteur de fichiers s'affichera et vous permettra de sélectionner un nom de fichier. Si vous sélectionnez un fichier qui existe déjà, celui-ci sera écrasé - aucune copie de secours (*backup*) n'est créée par cette commande.

Copier un Bloc

Un bloc peut être dupliqué à un autre endroit du texte en déplaçant le curseur à l'endroit désiré et en appuyant sur F4, dans la mesure où la mémoire le permet. Si vous essayez de copier un bloc à l'intérieur de lui-même, le message *Bloc invalide* sera affiché dans la ligne d'information, et la copie sera abandonnée.

Détruire un Bloc

Un bloc peut être détruit (effacé du texte) en appuyant sur *Shift-F5*. La touche *shift* est délibérément imposée pour éviter les accidents. Un bloc détruit est mémorisé dans le *presse-papier* pour un usage ultérieur, si la mémoire le permet.

Note

Les versions de GenST antérieures à 2.0 utilisaient une autre touche.

Copier un Bloc dans le Presse-Papier

Un bloc marqué peut être copié dans le *presse-papier* en appuyant sur *Shift-F4*, si la mémoire le permet. Cela peut être très utile pour échanger des données entre différents fichiers. Chargez alors un fichier, marquez un bloc, copiez-le dans le *presse-papier*, chargez un autre fichier et collez le bloc.

Coller un Bloc

Le bloc se trouvant dans le *presse-papier* peut être collé dans le texte courant à la position du curseur en appuyant sur F5.

Note

Le contenu du *presse-papier* sera perdu si la taille du tampon d'édition est changée, ou si l'assembleur est lancé.

Imprimer un Bloc

Un bloc marqué peut être envoyé vers l'imprimante en cliquant sur l'option *Imprimer Bloc* du menu *Fichier*, ou en appuyant sur *Alt-W*. Une boîte d'alerte vous demandera confirmation. Cliquez sur *Ok* pour imprimer le bloc. Le port d'imprimante utilisé est celui qui est sélectionné dans l'accessoire de bureau *Panneau de contrôle*. Par défaut, le fichier est envoyé vers le port parallèle. Les tabulations sont envoyées comme une suite de blancs. L'impression donnera donc un meilleur résultat que si vous l'imprimiez depuis le bureau GEM.

Cette option vous permet également d'imprimer le texte en totalité si vous ne sélectionnez pas de bloc.

Les marqueurs de bloc restent actifs pendant l'édition. Ils ne sont effacés que par les commandes Nouveau, Effacer bloc et Charger....

Commandes diverses

A propos ...

Cliquez sur l'option A propos ... du menu Zz permet de faire afficher une boîte de dialogue contenant diverses informations sur GenST2. Appuyez sur **Return** ou cliquez sur **Ok** pour revenir à l'éditeur.

Ecran d'aide

Vous pouvez obtenir les équivalents clavier des commandes non présentes dans les menus, des commandes WordStar ainsi que la quantité de mémoire libre en appuyant sur la touche **Help** ou sur **Alt-H**.

Préférences

L'option Préférences... du menu Options vous permet de modifier certains paramètres de l'éditeur. Vous pouvez également y accéder en appuyant sur **Ctrl-T**.

Tabulations

Par défaut, la longueur d'une tabulation est de 8 caractères. Vous pouvez faire varier cette valeur de 2 à 16.

Taille du Texte

Par défaut, la zone mémoire tampon réservée pour le texte est de 60.000 caractères. Vous pouvez faire varier cette valeur de 4.000 à 999.000 octets. Ceci détermine la taille maximum de fichier qu'il est possible de charger et d'éditer. Veillez à laisser suffisamment de place pour l'assembleur et le débogueur. Appuyez sur la touche **Help** pour obtenir la mémoire disponible, et laissez au moins 100 Ko pour l'assembleur et le débogueur. La modification de la taille de la zone mémoire texte entraîne l'effacement du texte présent en mémoire, c'est pour cela qu'une confirmation vous est demandée lorsque vous n'avez pas sauvé un texte qui a été modifié.

Pavé Numérique

Cette option vous permet d'utiliser les touches de votre pavé numérique comme des touches de déplacement curseur, à la manière de celui d'un IBM-PC (Voir figure 2.2).

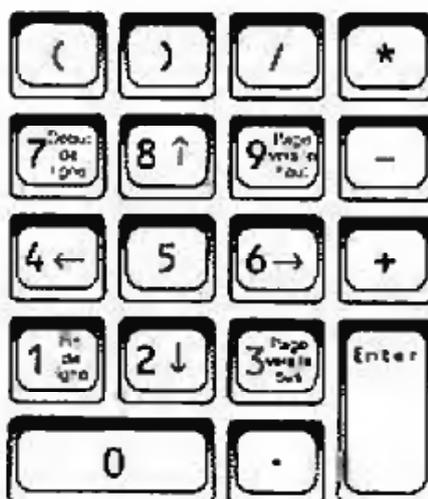


Figure 2.2 : Pavé numérique

Cette fonction peut être désactivée en cliquant sur le bouton Chiffres.

Copie de Secours (Backup)

Par défaut, l'éditeur ne crée pas de copie de secours de la version précédente du texte que vous sauvegardez, mais il suffit de cliquer sur le bouton Oui pour valider l'option.

Indentation Automatique

Il est particulièrement utile d'indenter automatiquement vos programmes quand vous les saisissez. Si cette option est active, l'éditeur ajoute une indentation au début de chaque nouvelle ligne quand vous appuyez sur Return. Les blancs ou tabulations du début de la ligne précédente sont repris et ajoutés automatiquement au début de chaque nouvelle ligne. Cliquez sur Non pour inactiver cette option.

Curseur

Par défaut, le curseur de GenST clignote. Cliquez sur Fixe si vous ne voulez pas que le curseur clignote.

Chargement de MonST

Par défaut, MonST est chargé lors du lancement de GenST. Cela vous permet d'y accéder en appuyant sur une touche. Si vous ne désirez pas charger MonST, cliquez sur Non. Vous économiserez ainsi environ 24 Ko de mémoire. Bien sûr, cette option n'est prise en compte que si vous sauvez les options et relancez GenST.

Sauver les Préférences

 Lorsque vos choix sont faits, cliquez sur Ok pour les valider jusqu'à ce que vous quittiez GenST. Cliquez sur Annuler si vous ne désirez pas valider les choix que vous avez faits, ou si, simplement, vous avez sélectionné cette option par erreur.

Si vous désirez que vos choix soient pris en compte lors de vos prochaines utilisations de GenST, cliquez sur Sauver. Un fichier `GENST2.INF` sera créé et écrit sur votre disque. L'éditeur sauvera également les options d'assemblage de la boîte de dialogue Options d'assemblage.

Assembler et exécuter un programme

Toutes les options d'assemblage et d'exécution se trouvent dans le menu Prg.

Assembler

Pour assembler un programme, cliquez sur l'option Assembler du menu Prg, ou appuyez sur `Alt-A`. La signification des diverses options ainsi que la procédure d'assemblage sont décrites dans le chapitre suivant. La seule option décrite ici est l'option **Sortie**.

 GenST peut assembler sur disque, en mémoire et même nulle part. Cet assemblage venu d'ailleurs, ou plutôt ne conduisant nulle part ne génère aucun code. Il sert à tester la syntaxe d'un programme. Vous pouvez ainsi tester très rapidement la validité de l'assemblage d'un texte en mémoire. Cliquez sur le bouton **Aucune** pour valider ce mode.

Lorsque vous assemblez en mémoire, vous devez spécifier la taille de la mémoire tampon réservée au programme. L'assembleur stockera dans cette zone le code généré. Utilisez le champ **Max:** pour spécifier une valeur en Ko. 20 Ko sont suffisants pour un programme moyen avec informations de mise au point (débogage), ou pour un gros programme sans informations de mise au point. Cliquez sur **Mémoire** pour choisir ce mode d'assemblage. Si le message **zone mémoire programme pleine** apparaît lors de l'assemblage, augmentez cette valeur et réassemblez. Il y a un inconvénient à ceci, plus le tampon programme est gros, plus petit est l'espace mémoire disponible pour l'assembleur lui-même et pour le stockage de ses variables internes. Si l'assembleur affiche le message **Pas assez de mémoire**, cela signifie qu'il n'a pas assez de mémoire pour terminer l'assemblage. Réduisez le tampon programme. Si cela ne marche pas non plus, il ne vous reste plus qu'à assembler sur disque.

Pour choisir l'assemblage sur disque, cliquez sur le bouton **Disque**. Dans ce cas, le texte source que vous éditez est assemblé et un fichier objet est créé sur le disque. Ce fichier peut être un programme exécutable, un accessoire, ou un fichier destiné à être chaîné (*link*). Par défaut, le nom du fichier que vous éditez sera repris pour le fichier objet. L'extension du fichier (.S généralement) sera changée pour obtenir par exemple un fichier **.PRG**. Si vous désirez changer ce nom par défaut, ou bien en spécifier un si le texte que vous éditez n'a pas de nom, entrez un nouveau nom dans le champ de texte à droite du bouton **Disque**. Si votre texte n'a pas de nom et que vous n'en spécifiez pas lors de l'assemblage, le nom **NONAME** lui sera attribué d'office. Notez que le tampon programme n'est pas utilisé dans le cas d'un assemblage sur disque. Tout l'espace est réservé à l'assembleur.

Cliquez sur le bouton **Assembler** ou appuyez sur **Return** pour lancer l'assemblage. Le processus d'assemblage est décrit en détails dans le chapitre suivant. Lorsque l'assemblage est terminé, appuyez sur une touche pour revenir à l'éditeur. S'il y a des erreurs, le premier message d'erreur est affiché dans la ligne d'information et le curseur est positionné sur la ligne comportant cette erreur. Cliquez sur l'option **Erreur** suivante du menu **Prg**, ou appuyez sur **Alt-J** pour afficher les erreurs suivantes.

Vous pouvez aussi lancer l'assembleur en appuyant sur **Shift-Alt-A** sans passer par la boîte de dialogue. Dans ce cas, l'assembleur utilisera par défaut les paramètres courants (les derniers saisis dans la boîte de dialogue **Option d'assemblage**).

Exécuter un programme

Cliquez sur l'option Exécuter du menu Prg, ou appuyez sur **Alt-X** pour exécuter le dernier programme qui a été assemblé en mémoire. Si aucun programme n'a été assemblé en mémoire, ou si il y a eu des erreurs lors de l'assemblage, l'exécution est impossible et un message d'erreur vous le signale.

Lorsque un programme bloque ou "plante", le retour dans l'éditeur n'est pas forcément possible. Nous vous conseillons donc de systématiquement sauver votre texte source avant d'exécuter le programme.

Note

Si seules des erreurs non fatales surviennent lors de l'assemblage (par exemple des symboles non définis), vous pouvez quand même exécuter le programme. Mais c'est à vos risques et périls!

A Noter

Lorsque vous cliquez sur Exécuter, la machine peut sembler bloquer et ne pas exécuter votre programme. Ceci arrive si la souris se trouve sur la barre de menus. Pour arranger cela, déplacez simplement la souris. De la même manière, cet inconvénient se reproduit à la fin de l'exécution et la machine semble ne pas vouloir retourner à l'éditeur. Là encore, déplacez la souris et tout rentrera dans l'ordre. Ceci est dû à une particularité de GEM et n'est pas de notre ressort.

Débogueur

Cliquez sur l'option Débogueur du menu Prg, ou appuyez sur **Alt-D** pour déboguer un programme déjà assemblé. MonST sera lancé pour déboguer votre programme. Vous pouvez déboguer sous GEM ou en mode texte selon la valeur de l'option Exécution sous GEM. Appuyez sur **Ctrl-C** pour terminer votre programme et retourner à l'éditeur.

Note

Si l'option Charge MonST de la boîte de dialogue Préférences n'est pas spécifiée, cette option n'est pas disponible et l'entrée du menu est affichée en gris.

MonST

Cliquez sur l'option MonST du menu Prg, ou appuyez sur ALT-M pour lancer MonST de la même manière que s'il était lancé depuis le bureau. Seulement, puisque MonST est résident, vous vous retrouvez instantanément sous le débogueur. Vous pouvez déboguer sous GEM ou en mode texte selon la valeur de l'option Exécution sous GEM. Appuyez sur CTRL-C pour terminer votre programme et retourner à l'éditeur.

Note

Si l'option Charge MonST de la boîte de dialogue Préférences n'est pas spécifiée, cette option n'est pas disponible et l'entrée du menu est affichée en gris.

Exécution sous GEM

Par défaut, les commandes Exécuter, Déboguer ou MonST, sont utilisées avec un écran type GEM constitué d'un fond tramé avec une ligne de menu blanche. Si vous désirez exécuter un programme TOS, vous pouvez obtenir un écran blanc avec un curseur clignotant en cliquant sur l'option Exécution sous GEM du menu Prg, ou en appuyant sur ALT-K. Une marque devant cette entrée de menu signifie mode GEM. Une absence de marque signifie mode TOS. Cette option est sauvegardée lorsque vous sauvez les préférences.

Note

Exécuter un programme TOS en mode GEM fonctionne, même si l'écran n'est pas très propre. Mais ne tentez pas d'exécuter un programme GEM en mode TOS. Cela peut "planter" la machine.

Correction des Erreurs

Lors de l'assemblage, tous les avertissements et erreurs sont enregistrés, et peuvent être rappelés depuis l'éditeur. Au retour dans l'éditeur, le curseur est positionné sur la ligne de la première erreur et le message d'erreur correspondant est affiché sur la ligne d'information de la fenêtre de l'éditeur. Cliquez sur l'option Erreur Suivante du menu Prg, ou appuyez sur ALT-J pour déplacer le curseur à l'endroit où se trouve l'erreur suivante. Vous pouvez ainsi corriger les erreurs d'assemblage rapidement et facilement. Lorsque toutes les erreurs ont été passées en revue, le message Plus d'erreurs apparaît. S'il y a aucune erreur, le message Quelles erreurs! est affiché.

Lancer Autre...

Cette option vous permet d'exécuter d'autres programmes depuis l'éditeur, et d'y retourner lorsque leur exécution est terminée. Le but principal de cette option est d'exécuter les programmes que vous avez assemblé sur disque, ou de lancer l'éditeur de liens (*linker*) sans quitter l'éditeur pour le bureau GEM. Vous pouvez exécuter des programmes TOS ou GEM, si la mémoire disponible le permet. Lorsque vous cliquez sur l'option Lancer Autre... du menu Prg, un message d'alerte vous prévient si vous n'avez pas sauvé votre texte. Le sélecteur de fichiers GEM apparaît. Sélectionnez le programme à exécuter. Si celui-ci a une extension .TOS ou .TTP, l'éditeur vous demandera la ligne de commande à passer au programme. Puis, le programme est exécuté avec un écran GEM si le programme est un .PRG, sinon avec un écran texte.

Note

L'initialisation de l'écran dépend du type de programme exécuté, non de l'état de l'option Exécution sous GEM.

Utilisation des Fenêtres et Accessoires de Bureau

Fenêtre GEM d'édition

La fenêtre utilisée par l'éditeur fonctionne comme les autres fenêtres GEM. Vous pouvez la déplacer en cliquant sur la barre de titre, changer sa taille en cliquant sur le bouton taille, ou la faire occuper tout l'écran en cliquant sur le bouton plein écran. Cliquer sur le bouton de fermeture est équivalent à cliquer sur l'option Quitter du menu Fichier.

Accessoires de bureau

Il existe des accessoires de bureau qui utilisent leur propre fenêtre, comme par exemple le panneau de contrôle. Lorsque vous sélectionnez cet accessoire, une fenêtre s'ouvre au milieu de l'écran. Vous pouvez déplacer cette fenêtre au dessus de celle de GenST. Vous pouvez recouvrir l'accessoire en cliquant sur la fenêtre de GenST. L'accessoire passe sous la fenêtre et n'est plus visible à l'écran. Pour voir l'accessoire, changez la taille de la fenêtre de GenST. Pour activer à nouveau l'accessoire, cliquez sur sa fenêtre, et il repassera sur la fenêtre de GenST. Attention, les menus de l'éditeur et le curseur clignotant ne fonctionnent que si la fenêtre de GenST est au dessus.

Lancer GenST d'un double-clic

Vous pouvez configurer GenST pour qu'il soit lancé automatiquement lorsque vous double-cliquez, depuis le bureau GEM, sur un fichier assembleur. Choisissez l'extension que vous désirez utiliser pour le fichiers assembleur. Nous recommandons .S. Vous pouvez aussi choisir .ASM par exemple. Ensuite cliquez une fois sur GENST2.PRG. Son icône est affichée en blanc sur noir. Cliquez sur l'option Installer une Application du menu Options. Entrez l'extension que vous avez choisie dans le champ Type de Document de la boîte de dialogue. Cliquez sur le bouton OK ou INSTALLER selon votre version du TOS (un seul de ces deux boutons est disponible. Son nom a simplement changé d'une version à l'autre).

Pour tester l'installation, double-cliquez sur un fichier ayant l'extension spécifiée, dans le même répertoire que GenST. Le bureau GEM lancera automatiquement GenST, et le fichier sur lequel vous avez cliqué sera chargé dans la fenêtre d'édition.

Note

Cliquez sur l'option Sauver le Bureau pour rendre cette option permanente.

Utilisateurs de l'accessoire de bureau Saved!

Si vous utilisez l'option PATH de l'accessoire de bureau Saved! édité par Hisoft, vous n'êtes plus obligé d'avoir vos fichiers de données dans le même répertoire que GenST. L'éditeur cherche le fichier GENST2.INF dans le répertoire courant, puis, s'il ne l'y trouve pas, le cherche dans les répertoires spécifiés par l'option PATH. Lorsque vous sauvez les préférences, le fichier GENST2.INF sera écrit dans le répertoire où il a été chargé, ou dans le répertoire courant s'il n'existe pas déjà.

Vous pouvez lancer Saved! depuis l'éditeur en appuyant sur les touches **Ctrl-Clr**. Ceci n'est toutefois possible que si un fichier SAVED!.ACC ou SAVED.ACC est présent sur votre disque de lancement (*boot disk*).

CHAPITRE 3

Macro Assembleur

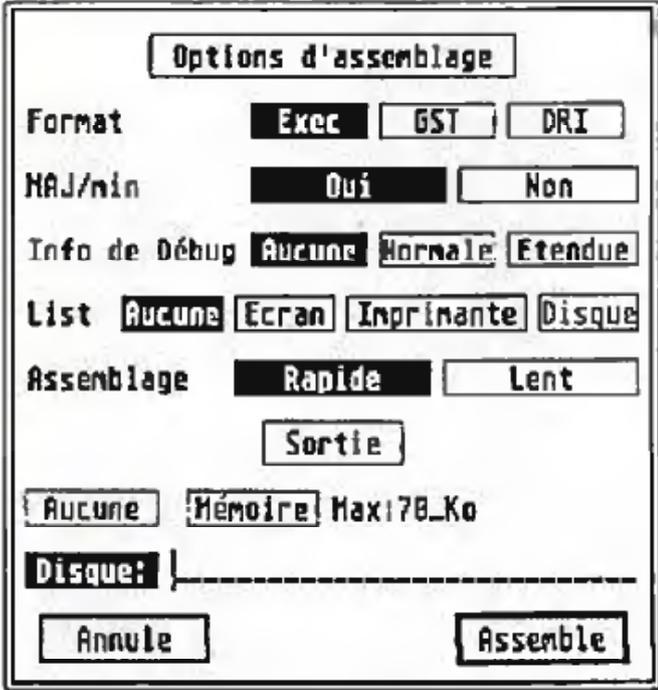
Introduction

GenST est un assembleur complet, puissant et rapide, accessible instantanément à partir de l'éditeur de texte intégré, ou directement à partir du bureau GEM en cliquant sur GENST2.TTP. Son rôle est de convertir un source assembleur entré ou chargé dans l'éditeur, éventuellement accompagnés de fichiers en-tête, en un fichier binaire destiné à être exécuté ou chaîné, ou bien en une image mémoire destinée à être exécuté immédiatement depuis l'éditeur.

Appeler l'assembleur

A partir de l'éditeur

L'assembleur est appelé depuis l'éditeur de textes en cliquant sur l'option **Assembler** du menu **Prg**, ou en appuyant sur **ALT-A**. La boîte de dialogue de la figure 3.1 apparaît.



Options d'assemblage

Format Exec GST DRI

HAJ/min Oui Non

Info de Débug Aucune Normale Etendue

List Aucune Ecran Imprimante Disque

Assemblage Rapide Lent

Aucune Mémoire Maxi78_Ko

Disque: _____

Figure 3.1 - La boîte de dialogue Options d'Assemblage

Format

Vous pouvez sélectionner le format du code généré : exécutable, CST ou DRI. Les différences entre ces formats sont décrites plus loin.

MAJ/Min

Cette option permet de demander à l'assembleur de faire la distinction ou non entre les majuscules et les minuscules lors de l'évaluation des symboles. Par exemple, si vous sélectionnez le bouton Oui, les symboles `Test` et `test` seront compris par l'assembleur comme deux symboles différents. Au contraire, si vous sélectionnez Non, ils seront considérés comme deux symboles équivalents.

Informations de débogage

Si vous désirez déboguer vos programmes en utilisant les noms des symboles, sélectionnez les modes de débogage Normal ou Étendu.

En mode normal, seul les 8 premiers caractères des symboles sont pris en compte dans les informations de débogage, alors que ce nombre atteint 22 caractères en mode étendu.

Listing

Sélectionnez le bouton Imprimante pour envoyer le listing de l'assemblage à l'imprimante, ou le bouton Disque pour l'envoyer sur disque dans un fichier portant le même nom que le fichier source mais ayant `.LST` pour extension.

Assemblage

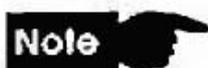
Cette option permet de déterminer la rapidité d'assemblage, Rapide ou Lent. Normalement, vous pouvez rester tout le temps en assemblage Rapide, excepté lorsque l'assembleur n'a plus assez de mémoire pour s'exécuter. Dans ce cas, le choix d'un assemblage Lent forcera l'assembleur à n'utiliser que le minimum de mémoire possible - ceci au détriment de la vitesse des accès disque.

Sortie

Cette option vous permet de spécifier comment sera stocké le code généré. Aucune signifie qu'il n'y aura pas de code généré - très utile pour vérifier rapidement la syntaxe d'un source. Sélectionnez Mémoire pour que la sortie se fasse en mémoire - vous permettant ainsi d'exécuter et de déboguer votre programme quasi instantanément sans quitter l'éditeur. Choisissez Disque pour que la sortie se fasse sur disque.

Les deux premières options peuvent aussi être spécifiées directement dans le source grâce à la directive OPT.

Une fois toutes les options spécifiées, cliquez sur le bouton **Assembler**, ou appuyez sur **Return** pour lancer l'assemblage. Appuyez sur une touche lorsque votre texte est assemblé et que vous voulez revenir à l'éditeur. S'il y a des erreurs, le curseur se positionnera automatiquement sur la première erreur. Voyez le paragraphe **Correction des Erreurs, Chapitre 2**.

**Note**

Vous pouvez aussi lancer directement l'assemblage en appuyant sur **SHIFT-ALT-A** sans passer par la boîte de dialogue. Dans ce cas, l'assembleur utilisera par défaut les paramètres courants (les derniers saisis dans la boîte de dialogue Option d'assemblage).

A partir du bureau GEM

Vous pouvez utiliser GenST pour assembler des fichiers sources assembleur que vous avez récupérés ou créés avec un autre éditeur que celui de DevpacST. Utilisez pour cela le programme GENST2.TTP depuis un environnement (*shell*) texte, ou en double-cliquant sur son icône depuis le bureau. Si vous ne spécifiez pas de ligne de commande, le programme vous demandera d'en entrer une, conformément à la syntaxe décrite ci-dessous.

Description de la ligne de commande

La ligne de commande de GENST2.TTP doit être de la forme :

```
fichier <-options> [-options]
```

Le **fichier** représente le nom du fichier qui doit être assemblé. Si vous omettez l'extension, l'assembleur prendra **.s** par défaut.

Les **options** doivent suivre ce nom de fichier. Elles se composent d'un tiret (-) suivi d'une lettre. La liste des options disponibles est décrite ci-dessous accompagnée de l'équivalence en directive OPT (lorsqu'elle existe).

- B** Aucun fichier objet binaire ne doit être créé
- C** Distinction majuscules/minuscules pour les symboles (OPT C-)
- D** Insertion des informations de débogage (OPT D+)

- E** Permet de définir des symboles à partir de la ligne de commandes. Cette option doit être suivie d'une ou de plusieurs affectations de symboles, séparées par des virgules ou des espaces (Par exemple, `gens2 test -e TYPE=4,OUTPUT=1`).
- L** Génération de code objet au format GST (OPT L+).
- L2** Génération de code objet au format DRI (OPT L2).
- M** Permet de choisir un assemblage lent (il est rapide par défaut).
- O** Permet de spécifier le nom du fichier de sortie (ce nom doit immédiatement suivre le caractère **O**).
- P** Permet de spécifier le nom du fichier listing (ce nom doit immédiatement suivre le caractère **P**) ; par défaut, le nom du source est utilisé avec l'extension `.LST`.
- Q** Permet de demander une pause avec appui sur une touche en fin d'assemblage.
- Txx** Permet de spécifier la taille des tabulations dans les listings, par exemple `-T10` fixe à 10 caractères la taille d'une tabulation.
- X** Insertion des informations de débogage étendu (OPT X+)

Par défaut, l'assembleur crée un fichier exécutable dont le nom est basé sur le nom du fichier source et du type de fichier de sortie, sans produire de listing, et en distinguant les minuscules des majuscules dans les symboles.

Par exemple :

```
test -b
```

assemble `test.s` sans créer de fichier binaire en sortie.

```
test -om:test.prg -p
```

assemble `test.s` en un fichier binaire `m:test.prg` en produisant un listing dans le fichier `test.lst`.

```
test -l2dpprn:
```

assemble le fichier `test.s` en un fichier objet au format DRI avec option de débogage. Un listing est envoyé sur le port parallèle (il pourrait facilement être envoyé au port série en spécifiant `AUX:` comme nom du fichier listing).

Code de retour

La version .T7P de GenST retourne au programme appelant un code d'erreur permettant de savoir ce qui s'est passé lors de l'assemblage. Ce code peut être égal à :

100	Fichier principal non trouvé
20	Grave Erreur
10	Erreur(s)
5	Avertissement(s)
0	Pas d'erreur.

Nom du Fichier Objet

Le nom du fichier de sortie créé par GenST est déterminé par certaines règles, en fonction des spécifications faites lors de l'assemblage (le champ Disque : du formulaire d'options d'assemblage, ou l'option -o de la ligne de commande et de la directive OUTPUT) :

Si un nom de fichier est spécifié explicitement à l'assemblage

Alors nom = nom de fichier explicite

sinon

Si la directive OUTPUT n'est pas utilisée

Alors nom = nom du fichier source + .PRG, .BIN ou .O

Sinon Si la directive OUTPUT spécifie une extension

Alors nom = nom du fichier source + extension d'OUTPUT

Sinon nom = nom spécifié par OUTPUT

Processus d'assemblage

GenST est un assembleur deux passes. Pendant la première passe, la table des symboles est créée en parcourant tout le source en mémoire ou en le lisant sur disque. Si des erreurs de syntaxe sont trouvées lors de la première passe, celles-ci sont affichées et la deuxième passe n'est pas exécutée.

Si, en revanche, tout s'est bien passé pendant la première passe, la deuxième commence. Les instructions assembleur sont converties en octets. Un listing peut être généré, ainsi qu'un fichier objet. Des erreurs ou avertissements peuvent survenir pendant la deuxième passe. La table des symboles peut également être affichée.

Pendant l'assemblage, l'affichage peut être interrompu en appuyant sur `Ctrl-S`. Pour reprendre l'assemblage, appuyez sur `Ctrl-Q`. Vous pouvez stopper l'assemblage en appuyant sur `Ctrl-C`. Attention toutefois si vous interrompez l'assemblage pendant qu'un fichier est généré, celui-ci sera incomplet et ne pourra être utilisé.

L'Assemblage en Mémoire

Dans le but de réduire les temps de développement, GenST peut assembler directement en mémoire pour vous permettre d'exécuter et de déboguer immédiatement vos programmes. Pour ce faire, une zone mémoire tampon est utilisée. Vous pouvez spécifier sa taille dans la boîte de dialogue Options d'assemblage. Si vous ne spécifiez pas d'option de débogage, la taille nécessaire correspond à celle du programme exécutable. Sinon, vous devez spécifier une taille plus grande pour permettre l'inclusion des informations destinées au débogueur.

Un programme qui s'exécute en mémoire est similaire à n'importe quel programme GEMDOS. Pour cela, vous devez terminer votre programme par un appel à la fonction GEMDOS *Pterm* ou *Pterm0* de la manière suivante :

```
clr.w    -(a7)
trap    #1
```

Un programme peut se modifier lui-même, mais si vous exécutez une deuxième fois votre programme, il sera réinitialisé dans son état originel. Les modifications apportées au code seront effacées.

Vous pouvez modifier la taille du tampon programme ainsi que toutes les options d'assemblage et conserver les valeurs modifiées en utilisant l'option Sauver dans la boîte de dialogue Préférences.

Les types de fichier objet

GenST peut générer six types différents de fichiers objet pour différents types d'applications. La distinction des différents types est possible grâce aux extensions de fichier :

- .PRG Application de type GEM (utilisation des fenêtres)
- .TOS Application de type TOS (pas d'utilisation de fenêtres)
- .TTP Application de type TOS avec ligne de commande
- .ACC Accessoire de bureau
- .BIN Fichier non exécutable destiné à être chaîné avec d'autres fichiers ou bibliothèques au format GST.
- .O Fichier non exécutable destiné à être chaîné à d'autres fichiers ou bibliothèques au format DRI.

Vous pouvez également assembler du code exécutable directement en mémoire en utilisant la version intégrée de l'assembleur, accélérant ainsi le cycle de développement.

Vous pouvez double-cliquer sur les trois premiers types de fichiers pour les exécuter depuis le bureau. Ce sont des fichiers *exécutables*. La différence entre ces trois types de fichier vient de l'initialisation faite avant leur exécution. Les fichiers .PRG affichent un écran vide similaire au bureau GEM. Les deux autres types de fichiers affichent un écran blanc et un curseur clignotant. La souris n'est pas visible. De plus, si vous double-cliquez sur un fichier .TTP, le bureau vous demandera d'entrer au clavier la ligne de commande destinée à ce programme.

Les fichiers .ACC sont des fichiers exécutables, mais ne peuvent être lancés en double-cliquant sur leur icône. Ils sont exécutés automatiquement pendant l'initialisation de la machine.

Les fichiers .BIN et .O ne peuvent être exécutés immédiatement. Vous devez utiliser un éditeur de liens pour les chaîner à d'autres modules. Ce sont des *modules objets chainables*. Il existe deux formats de fichier objet sur ST : le format GST (extension .BIN) et DRI (extension .O). Les différences entre ces deux formats seront détaillées ultérieurement.

Les extensions énoncées ci-dessus pour les fichiers exécutables, ne constituent pas une règle en soi. Par exemple, vous pouvez donner l'extension `.PRG` à un programme TOS, et utiliser l'option Installer une Application du bureau GEM pour spécifier une exécution TOS. Mais il est généralement plus simple de se conformer aux règles ci-dessus. Il existe toutefois une exception. Les programmes qui sont placés dans le répertoire `AUTO` doivent être des programmes TOS bien que l'extension `.PRG` leur soit imposée pour que la séquence d'initialisation puisse les exécuter.

Note

Certaines versions françaises de ROMs ST ne reconnaissent pas les fichiers `.TTP`. Vous devez donc les renommer en `.TOS` et les installer comme application de type TOS avec paramètres.

Types de code

Comme sur la plupart des systèmes 16-bits, et contrairement à ce qui se passe sur les systèmes 8-bits, un programme exécutable n'est pas chargé à une adresse spécifique, mais à une adresse qui dépend de l'organisation de la mémoire libre à un instant donné.

Pour palier à ce problème d'adressage absolu, un fichier exécutable sur ST contient des informations de relogement (*relocation information*) qui permettent au système d'exploitation de reloger le programme en mémoire avant son exécution. Par exemple, le segment de programme suivant :

```
Move.l #message,a0
:
:
:
message dc.b 'Appuyez sur une touche',0
```

place l'adresse absolue du message dans un registre, bien que lors de l'assemblage, l'adresse de message ne puisse être connue. Généralement, le programmeur considère cette adresse comme absolue, même si l'adresse réelle n'est pas connue. L'assembleur (ou l'éditeur de liens) s'occupe de générer les informations nécessaires au relogement du programme.

Note

Pour certains programmes - comme les jeux ou les développements spécifiques (par exemple, le *cross-development*) - spécifier une adresse absolue de démarrage peut être utile, c'est pour cette raison que la directive `ORG` existe.

Syntaxe de l'Assembleur

Chaque ligne destinée à être assemblée, exception faite des lignes de commentaires, commençant par un astérisque ou un point virgule, et des lignes vides, doit être de la forme :

Etiquette	Mnémonique	Opérande(s)	Commentaire
debut	move.l	d0,(a0)+	Stocke le resultat

Chaque champ doit être séparé par un espace, ou combinaison de plusieurs espaces et tabulations.

Le champ Etiquette

Normalement, une étiquette doit obligatoirement se trouver en début de ligne. Cependant, il est possible de les placer n'importe où sur une ligne en les faisant immédiatement suivre d'un deux-points (:). Les étiquettes sont autorisées devant toutes instructions 68000, mais sont interdites pour certaines directives d'assemblage, et obligatoires pour d'autres. Une étiquette est une suite de caractères alpha-numériques (y compris le caractère souligné '_', le point '.' et l'arobace '@', excepté lorsque ce dernier est immédiatement suivi d'un chiffre, ceci pouvant être interprété comme le début d'un nombre octal), mais doit obligatoirement commencer par une lettre ou un caractère souligné.

Les étiquettes commençant par un point sont considérées comme étant des *étiquettes locales* (Voir plus loin). Les noms de macros et les redéfinitions des noms de registres ne doivent pas comporter de point (Attention : des noms de macros peuvent très bien commencer par un point). Par défaut, les 127 premiers caractères d'une étiquette sont significatifs, mais vous pouvez réduire cette valeur. Les étiquettes ne peuvent porter le nom d'un registre, ni des mots réservés SR, CCR et JSP.

Par défaut, l'assembleur distingue les majuscules des minuscules lorsqu'il évalue les étiquettes (vous pouvez lui demander de ne pas le faire).

Voici maintenant quelques exemples de noms d'étiquettes valides :

```
test, Test, TEST, _test, _test.fin, test5, _5test
```

Et quelques exemples d'étiquettes illicites :

```
5test, _&e, test>
```

GenST se réserve certains symboles commençant par deux caractères soulignés, comme __LK, __RS et __G2.

Les étiquettes utilisées dans des expressions peuvent être soit relatives, soit absolues, en fonction de la façon dont elles ont été définies. Les étiquettes définies dans le source assemblé sont obligatoirement relatives. Les étiquettes définies par la directive EQU sont du même type que l'expression qu'elles représentent.

L'astérisque (*) représente la valeur du compteur programme (PC) au début de l'instruction ou de la directive qui est toujours une valeur relative.

Le champ Mnémonique

Le champ mnémonique suit celui de l'étiquette. Il peut contenir un nom d'instruction 68000, une directive d'assemblage ou encore un appel de macro. Certaines instructions et directives acceptent un attribut de taille d'opérande séparé du mnémonique par un point. Les tailles reconnues sont .B pour un octet (*Byte*), .W pour un mot de 16 bits (*Word*), .L pour un mot de 32 bits (*Long*), et .S pour court (*Short*). L'utilisation des minuscules ou les majuscules dans les attributs de taille, dépend de l'instruction ou de la directive concernée. GenST ne fait pas la distinction Majuscule/minuscule dans les instructions 68000 et les directives. Autrement dit, `move` est équivalent à `Move` ou à `MOVE`.

Le champ Opérande

Ce champ doit contenir, pour les instructions ou directives qui le nécessitent, une liste d'un ou de plusieurs paramètres séparés par des virgules. GenST ne fait pas la distinction Majuscule/minuscule pour les noms de registres. Ils peuvent indifféremment être écrits en minuscules, majuscules ou combinaison des deux.

Le champ Commentaire

Tout espace qui ne se trouve pas entre guillemets ou apostrophes après le champ de l'opérande est considéré comme le délimiteur du début du champ commentaire, champ qui d'ailleurs sera ignoré par GenSt lors de l'assemblage.

Exemple de lignes assembleur correctes

```
        move.l    d0,(a0)+           Le commentaire est
ici
boucle TST.W    d0
etiquette.solitaire
rts
```

* ceci est une ligne de commentaire

 indente: link A6,#-10 fait de la place

chaîne: dc.b 'espaces autorises ici' une chaîne

Les Expressions

GenST est capable de traiter des expressions complexes, avec parenthèses, opérateurs logiques, et en tenant compte des priorités d'opérations.

Les expressions peuvent être de deux types - Absolues ou relatives. Les expressions absolues sont des constantes calculées lors de l'assemblage. Les expressions relatives sont des adresses qui ne sont pas connues lors de l'assemblage puisque le chargeur de programme GEMDOS peut placer le programme où il veut en mémoire. Certaines instructions et directives n'autorisent qu'un seul type d'expression. Certains opérateurs ne peuvent être utilisés qu'avec un seul type d'expression.

Opérateurs

Voici la liste des opérateurs disponibles, dans l'ordre décroissant de priorité :

- moins (-) et plus (+) monadique
- opérateur de négation sur bit (~)
- décalage à gauche (<<) et à droite (>>)
- opérateurs sur bit Et (&), Ou (|, |) et Ou Exclusif (^)
- multiplication (*) et division (/)
- addition (+) et soustraction (-)
- égal (=), inférieur (<), supérieur (>)

Les opérateurs de comparaison tiennent compte du signe des nombres et retournent 0 si l'expression évaluée est fautive, et -1 (\$FFFFFFFF) si elle est vraie. Les opérateurs de décalage prennent deux paramètres : le premier représente la valeur à décaler, le second, le nombre de décalage à faire subir. Des zéros sont introduits lors du décalage pour remplacer les bits décalés.

La priorité des opérateurs peut être modifiée par des parenthèses '(' et ')'. Avec des opérateurs de même priorité, l'expression est évaluée de la gauche vers la droite. Les espaces ne sont pas autorisés dans les expressions (sauf s'ils sont entre apostrophes) car ils risquent d'être pris comme délimiteurs de commentaire.

L'évaluation des expressions est faite sur des entiers 32 bits signés. Aucun test de débordement n'est fait.

Les nombres

On peut spécifier des valeurs numériques de différentes manières :

constante décimale	:	1029
constante hexadécimale	:	\$12f
constante octale	:	@730
constante binaire	:	%1100010
constante caractère	:	'X'

Le signe \$ est utilisé pour signaler les nombres hexadécimaux, % pour signaler les nombres binaires, @ pour signaler les nombres octaux, et les ' ou les " pour signaler les chaînes de caractères.

Les constantes caractères

Utilisez des guillemets ou des apostrophes pour délimiter des chaînes de caractères. Il est possible de mettre des guillemets ou apostrophes dans une chaîne de caractères en répétant deux fois le caractère. Les constantes caractères peuvent contenir de un à quatre caractères. Si une constante fait moins de quatre caractères, sa valeur est cadrée à droite dans un entier long, et la valeur est complétée par des zéros. Voici quelques exemples de constantes caractères et leur équivalent ASCII et hexadécimal :

"Q"	Q	\$00000051
'hi'	hi	\$00006869
"Test"	Test	\$54657374
"it's"	it's	\$6974277C
'it's'	it's	\$6974277C

Les chaînes de caractères définies avec la directive DC.B sont codées légèrement différemment. Reportez vous à la description de cette directive un peu plus loin dans ce chapitre.

Combinaisons de types autorisés

Le tableau de la figure 3.2 récapitule, pour chaque opérateur, le type de résultat obtenu en fonction du type de paramètre utilisé (absolu ou relatif).

R : résultat relatif.

A : résultat absolu.

* : opération impossible.

	A op A	A op R	R op A	R op R
Opérateurs de décalage	A	*	*	*
Opérateurs sur bits	A	*	*	*
Multiplication	A	*	*	*
Division	A	*	*	*
Addition	A	R	R	*
Soustraction	A	*	R	A
Comparaisons	A	*	*	A

Figure 3.2 - Combinaisons de types autorisées

Les modes d'adressage

La table ci-dessous récapitule les modes d'adressage disponibles. Notez que GenST ne fait pas de différence entre les minuscules et les majuscules lorsqu'il évalue les modes d'adressage, ainsi D0 et a3 sont tous deux des noms de registres valides.

Forme	Mode d'adressage	Exemple
Dn	direct registre de donnée	D3
An	direct registre d'adresse	A5
(An)	indirect registre d'adresse	(A1)
(An)+	indirect reg. d'adresse avec post-incrémentation	(A5)+
-(An)	indirect reg. d'adresse avec pré-décrémentation	-(A0)
d(An)	indirect reg. d'adresse avec déplacement	20(A7)
d(An,Rn.s)	indirect reg. d'adresse avec index	4(A6,D4.I)
d.W	absolu court	\$0410.W
d.L	absolu long	\$12000.L
d(PC)	relatif au (PC) (avec déplacement)	SUIVANT(PC)
d(PC,Rn.s)	relatif au PC avec index	SUIVANT(PC,A2.W)
#d	immédiat	#26

où

- n numéro de registre de 0 à 7
- d nombre
- R registre d'index, A ou D
- s Taille opérande - *w(ord)* ou *L(ong)*. La valeur par défaut est *w(ord)*

Il est possible de ne pas spécifier de déplacement avec le mode d'adressage indirect registre d'adresse avec index :

```
move.l (a3,d2.l),d0
```

sera assemblé comme :

```
move.l 0(a3,d2.l),d0
```

Modes d'adressages spéciaux

CCR registre des codes de condition (Condition Code Register)
SR registre d'état (Status Register)
USP pointeur de pile utilisateur (User Stack Pointer)

En plus des trois registres ci-dessus, **SP** (*Stack Pointer*) peut être utilisé partout à la place de **A7**, c.a.d 4 (**SP**, **D3.W**).

Les noms des registres de données et d'adresses peuvent être remplacés par les symboles réservés **R0** à **R15**. Les symboles **R0-R7** sont équivalents aux registres **D0-D7**, et **R8-R15** à **A0-A7**. Cette particularité a été introduite pour garder la compatibilité avec d'autres assembleurs.

Étiquettes Locales

GenST accepte les étiquettes locales qui sont des étiquettes valables uniquement à l'intérieur d'une certaine zone de code source. Ces étiquettes commencent par un point et sont rattachées à la dernière étiquette non locale définie. Par exemple :

```
longueur1      move.l      4(sp),a0
.boucle        tst.b       (a0)+
               bne.s      .boucle
               rts

longueur2      move.l      4(sp),a0
.boucle        tst.b       -(a0)
               bne.s      .boucle
               rts
```

Il y a deux étiquettes locales **.boucle** dans ce source. La première est rattachée à **longueur1**, la seconde à **longueur2**.

Il est interdit de définir des étiquettes locales ayant pour nom **.L** ou **.W** afin d'éviter toute confusion avec la syntaxe de l'adressage absolu.

Le caractère point et les symboles

L'utilisation de symboles comprenant un point peuvent poser des problèmes avec la notation Motorola de l'adressage absolu court. En effet, le point est considéré comme faisant partie du nom du symbole. Par exemple, GenST générera une erreur sur la ligne :

```
move.l vecteur.w,d0
```

où `vecteur` est une adresse absolue, par exemple une variable système, en affirmant que le symbole `vecteur.w` n'est pas défini. Pour résoudre ce type de problème, l'expression - ici un symbole - peut être mis entre parenthèses, comme ci-dessous :

```
move.l (vecteur).w,d0
```

Par contre, le point peut être utilisé sans problème après une expression numérique, comme par exemple :

```
move.l $0402.w,d0
```

Note

La version 1 de GenST permettait l'utilisation de `\` à la place du point pour indiquer un adressage absolu court. GenST2 le permet aussi, mais nous ne vous le recommandons pas afin d'éviter toute confusion avec les paramètres de macros `\w` et `\l`.

Jeu d'instructions

Alignement sur mot

Toute instruction, excepté `DC.B` et `DS.B`, est toujours, lorsqu'elle est assemblée, alignée si un mot de seize bits. Il est maintenant possible de forcer l'alignement sur mot de directives `DC.B` et `DS.B` en plaçant la directive `EVEN` juste avant. Bien que toutes les instructions qui le nécessitent soient alignées sur des mots, certaines étiquettes peuvent très bien ne pas l'être (valeur impaire), comme dans l'exemple :

```
                nop      ceci est toujours aligne sur un mot
                dc.b    'impaire',0
debut
                tst.l   (a0)+
                bne.s  debut
```

où l'étiquette `debut` se trouve sur une adresse impaire - dû au nombre impair de caractères de la chaîne 'impaire'. Pour vous aider à dénicher ce genre de problème, GenSt générera un message d'erreur lorsqu'il détectera l'utilisation d'une adresse impaire comme opérande d'une instruction `BRA` ou `BSR`. Notez que de tels tests ne sont pas faits sur les autres instructions. Il est donc recommandé de faire précéder ce genre d'étiquettes par la directive `EVEN` si vous désirez qu'elles soient alignées. L'erreur la plus courante consiste à délibérément ne pas le faire, partant du principe que la longueur de la chaîne de caractères est connue. Cela marchera jusqu'au jour où vous modifierez la longueur de la chaîne.

Extensions au Jeu d'Instruction

Le jeu d'instruction complet du 68000 est reconnu, et certaines abréviations décrites ci-dessous sont automatiquement acceptées. Le jeu d'instruction complet et sa syntaxe sont décrits dans tout livre de référence du 68000 ou dans le guide de poche fourni avec Devpac.

Codes Conditions

Les codes conditions HS et LO sont reconnus dans les instructions Bcc, DBcc et Scc, et sont respectivement équivalents à CC et CS.

Instructions de Branchement

Pour forcer une instruction de branchement court, utilisez Bcc.B ou Bcc.S. Pour forcer un branchement relatif sur un mot, utilisez Bcc.W ou laissez le choix à l'optimisateur en utilisant Bcc tout court. Bcc.L est reconnu par compatibilité avec GenSTI, mais génère un avertissement car ce n'est pas à proprement parler une instruction 68000. Une instruction de branchement BRA.S à l'instruction suivante n'est pas autorisée, et est convertie, après affichage d'un avertissement, en une instruction NOP. Un BSR.S sur l'instruction suivante n'est pas autorisé et génère une erreur.

Instruction BTST

BTST est la seule instruction de test de bits à supporter l'adressage relatif au compteur de programme (PC).

Instruction CLR

CLR An n'est pas autorisé. Utilisez SUB.L An,An à la place. Notez que dans ce dernier cas, les drapeaux ne sont pas modifiés.

Instruction CMP

GenSt utilisera l'instruction CMPI si l'opérande source est une donnée immédiate, CMPA si l'opérande source est un registre d'adresse, et CMPM lorsque les deux opérandes utilisent le mode d'adressage post-incrémenté.

Instruction DBcc

DBRA est accepté comme équivalent à DBF.

Instruction ILLEGAL

Cette instruction génère le code \$4AFC-sur un mot.

Instruction LINK

Un avertissement est généré si le déplacement est positif ou impair.

Instruction MOVE from CCR

Cette instruction n'est disponible que sur le 68010 et plus. Elle sera donc convertie en MOVE from SR avec un avertissement.

Instruction MOVEQ

Un avertissement est généré lorsque la donnée est comprise inclusivement entre 128 et 255. Spécifiez une taille d'opérande long pour le supprimer.

Directives d'assemblage

Certains pseudo-mnémomoniques sont reconnus par GenST. Ces *directives d'assemblage* ne sont normalement pas assemblées et ne génèrent pas de code, mais elles indiquent à l'assembleur d'effectuer certaines opérations comme changer le format du code binaire généré ou le format de sortie du listing. Les directives sont analysées de la même manière que les lignes contenant des instructions assembleur, certaines peuvent (ou doivent) être précédées d'une étiquette, et peuvent être suivies d'un commentaire. Attention toutefois à ne pas mettre d'étiquette devant une directive qui n'en accepte pas, le résultat d'une telle opération risquant d'être indéfini ou carrément ignoré.

La signification de chacune de ces directives va maintenant être passée en revue. Veuillez simplement remarquer qu'une directive peut indifféremment s'écrire en minuscules ou en majuscules, même si nous utilisons généralement la notation en majuscule, les crochets '<' et '>' pour délimiter les paramètres facultatifs d'une directive, et les points de suspension '...' pour en indiquer la répétition possible.

Contrôle d'assemblage

END

Cette directive facultative indique la fin du source assembleur. L'analyse du source n'est pas poussée plus loin.

INCLUDE nom_de_fichier

Cette directive provoque la lecture et l'assemblage d'un fichier se trouvant sur le disque, exactement comme s'il était présent dans le fichier texte courant. La directive doit être suivie d'un nom de fichier GEMDOS valide.

Si le nom de fichier contient des espaces, vous devez mettre le nom de fichier entre apostrophes. Vous pouvez également spécifier un nom de lecteur, un chemin d'accès et une extension :

```
include b:\constant\entete.s
```

Vous pouvez imbriquer les inclusions de fichiers, c'est-à-dire qu'un fichier inclus peut en inclure d'autres tant que la mémoire le permet.

Si aucun chemin d'accès n'est spécifié, le fichier inclus sera chargé à partir du répertoire contenant le fichier source principal.

Note

Plus vous avez de mémoire mieux c'est. GenST lira le fichier inclus en une seule fois et le conservera en mémoire pour la deuxième passe.

INCBIN nom de fichier

Cette directive lit un fichier binaire et l'insère tel quel dans le fichier objet. Ce fichier binaire est toutefois aligné sur une adresse paire, et est complété par un octet NULL si sa taille est impaire. Vous pouvez utiliser cette option pour inclure des dessins d'écran, des *sprites* ou des fichiers texte.

OPT option <option> ...

Cette directive vous permet de contrôler diverses options de GenST. Chaque option est spécifiée par un caractère suivi d'un signe + ou -. Il est possible de modifier plusieurs options en les séparant par une virgule. Les options disponibles sont :

Option A - Adressage relatif au PC automatique (compteur de programme)

Cette option (OPT A+) permet de valider, lorsque cela est possible, le mode automatique d'adressage relatif au PC. Par exemple, lorsque l'option est active, la ligne :

```
move.l int_in,d0
```

est assemblée en :

```
move.l int_in(pc),d0
```

Prenez garde cependant, cette option ne garantit pas forcément la relogeabilité du code généré, mais permet d'en réduire la taille et d'en accélérer la vitesse d'exécution (Reportez-vous au programme GEMTEST pour en voir une des utilisations possibles). Si vous désirez annuler cette option sur une instruction particulière (pour pouvoir accéder à une adresse absolue, par exemple), utilisez la forme (expression).L. Dans ce cas, GenST considérera l'adressage spécifié dans la ligne :

```
move.l (int_in).L,d0
```

comme étant absolu.



Option C - Distinction Majuscule/minuscule et nombre de caractères significatifs

Par défaut, GenST évalue les symboles sur 127 caractères, en faisant la distinction entre les Majuscules et les minuscules (C-). Si vous désirez qu'il ne fasse pas de distinction MAJ/min, il suffit de spécifier l'option C-. Le nombre de caractères significatifs peut, quant à lui, être modifié en spécifiant la nouvelle valeur entre le C et le signe + ou - de l'option. Par exemple, C16+ indique à l'assembleur d'évaluer les symboles sur 16 caractères sans tenir compte des MAJ/min. Cette option peut être utilisée n'importe où dans le texte, mais n'a réellement de sens qu'en tout début de programme.

Option D - Informations de débogage

Grâce à la structure des fichiers exécutables GEMDOS, il est possible de rajouter, à la fin de ces fichiers, une table de symboles pouvant être récupéré par un débogueur, comme MonSt. Le rôle de cette table est de faciliter la mise au point des programmes. Par défaut, cette option n'est pas active ; elle peut être activée par D+ et désactivée par D-. Les informations destinées au débogueur se composent de la liste des étiquettes définies dans le programme, dont seuls les huit premiers caractères sont réellement inscrits - le standard DRI n'en autorisant pas plus (Voir plus loin l'option Informations étendues de débogage pour y remédier). Les étiquettes sont converties en majuscules lorsque la distinction MAJ/min n'est pas active.



Option E - Vérification des adressages absolus courts et longs

Lorsque cette option est valide (OPT E1), GenST vérifie que les adresses spécifiées lors d'adressages absolus courts ou longs, sont bien paires. Par exemple, la ligne :

```
move.l 5,a6
```

provoquera l'affichage d'un message d'erreur.

Option I - Vérification des adressages immédiats

Cette option, lorsque elle est valide, produit l'erreur '# probablement absent' à chaque fois qu'un adressage indirect ou absolu est utilisé dans une instruction. Par exemple, l'assembleur générera une erreur lorsqu'il rencontrera la ligne :

```
and.b $df,d1
```

vous signalant ainsi, peut-être, une erreur de frappe (il considère que vous vouliez taper `and.b # $df,d1` - adressage immédiat plutôt qu'absolu). Pour être bien sûr de ne pas faire de confusion entre ces modes d'adressages, mettez les références absolues entre parenthèses, suivies d'un attribut de taille `.L` ou `.W` (par exemple `move.l ($FFB000).L,d0`).

Option L - Format du fichier de sortie

Cette option permet de spécifier le format du code généré par GenST. Spécifiez l'option `L+` pour que GenST génère du code objet au format GST, `L2` pour obtenir du code objet DRI, ou `L-` pour obtenir du code exécutable (option prise par défaut). Cette option doit obligatoirement figurer sur la toute première ligne de votre fichier source.

Option M - Affichage intégral des macros

Quand un listing d'assemblage est généré, les appels aux macros apparaissent par défaut exactement comme dans le texte source. Si vous désirez voir les instructions générées par les macros, activez l'option par `M+` (désactivable par `M-`). Utilisez cette option aussi souvent que vous le désirez.

Option O - Optimisation

GenST peut se charger d'optimiser votre programme afin d'en réduire la taille et d'en accélérer le fonctionnement. Par défaut, GenST n'effectue aucune optimisation mais il est très facile de les activer (et les désactiver) individuellement.

OPT O1+ Optimise si possible les branchements arrières longs en branchements courts (`bne loop` en `bne.s loop`, par exemple).

OPT 02+ Optimise les adressages indirects avec déplacement en adressages indirects si le déplacement est nul. Par exemple:

```
move.l suivant(a0),d3
```

sera optimisé en :

```
move.l (a0),d3
```

si la valeur de `suivant` est nulle.

OPT 03+ Optimise les adressages absolus longs en adressages absolus courts si l'adresse se trouve dans l'espace \$FFFF8000-\$7FFF inclus.

OPT 04+ Optimise les instructions de la forme `MOVE.L #x,Dn` en `MOVEQ` si `x` est compris entre -128 et +127 inclus.

OPT 05+ Optimise les instructions `ADD #x` et `SUB #x` en `ADDQ` et `SUBQ` si `x` est compris entre 1 et 8 inclus.

OPT 06+ Ceci n'est pas réellement une optimisation ; son activation provoque l'affichage d'un avertissement lorsqu'un branchement avant long peut être transformé en branchement court et doit être utilisé en conjonction avec l'optimisation 1.

OPT 07+ Converti les `BRA.S` à l'instruction suivante en `NOB`. Ceci était toujours le cas dans les versions précédentes, même si l'optimisation n'était pas active. C'est maintenant considéré comme étant une erreur par défaut.

OPT 0+ Active toutes les options d'optimisation

OPT 0- Désactive toutes les options d'optimisation

OPT 01-, OPT 02-, etc ... Désactive l'optimisation spécifiée

Les messages d'avertissements produits par chacune des optimisations peuvent être contrôlés individuellement par l'option `OPT 0Wx+ / OPT 0Wx-`, où '`x`' représente le numéro de l'optimisation (Exemple : `OPT 0W3+` valide les avertissements produits par l'optimisation no 3).

OPT 0W- Désactive l'affichage des messages d'avertissements générés par toutes les optimisations. `0W+` active l'affichage.

GenST affiche, en fin d'assemblage, le nombre d'optimisations effectués et le nombre d'octets qui ont été économisés.

Option P - Test de *relogeabilité* du code

Cette option activée par `P+`, permet de demander à GenST de vérifier la *relogeabilité* du code généré - c.a.d. sa capacité à ne pas dépendre d'une référence mémoire absolue - et d'afficher, en cas d'insuccès, des messages d'erreur en regard des lignes suspectes. Cette option est par défaut inactive (`P-`).

Option S - Table des symboles

Cette option activée par `S+`, permet de demander à GenST d'inclure la table des symboles à la fin du listing, à condition que ce dernier ait été demandé. Par défaut, cette option est inactive (`S-`). Sachez que si vous utilisez cette directive plus d'une fois, GenST ne prendra en compte que la dernière.

Option T - Vérification de type

GenST offre une facilité intéressante, celle d'effectuer des vérifications de type sur certaines expressions. Mais dans certains cas, ceci peut s'avérer être plus une gêne qu'une aide. Pour cela, il est donc possible d'activer/désactiver cette vérification (`T+/T-`). Par exemple, GenST en assemblant le programme suivant :

```
main      bsr      initialise
          lea     main(a6),a0
          move.l  (main).w,a0
```

produira une erreur sur les deux dernières lignes, car `main`, expression relative, est utilisé en lieu et place d'expressions absolues. Maintenant, ce programme peut quand même être correct s'il est destiné, par exemple, à être exécuté sur une autre machine à base de 68000. Il est donc intéressant dans ce cas de désactiver la vérification de type.

Option U - Modification du préfixe des variables locales

Cette option permet, lorsque elle est activée (`U+`), de spécifier à GenST de considérer comme locales les variables commençant par un caractère souligné (`_`) - au lieu d'un point (`.`) par défaut.

Option W - Avertissements

Cette option permet d'activer (`W+`) ou de désactiver (`W-`) l'affichage des messages d'avertissements généré par GenST lors de l'assemblage. Vous pouvez l'utiliser autant de fois que vous le désirez.

Option X - Informations étendues de déboguage

Cette option similaire à l'option D, permet de demander à GenST d'inclure dans le fichier exécutable, des informations destinés à la mise au point du programme (table des symboles), en utilisant le format de déboguage étendu d'Hisoft. Ce format autorise jusqu'à 22 caractères significatifs pour les symboles.

Résumé des options

Les valeurs par défaut sont indiquées entre parenthèses après la description de chaque option :

- A Mode automatique d'adressage relatif au Compteur de Programme (A-)
- C Distinction Majuscules/minuscules et nombre de caractères significatifs. (C127+)
- D Insertion des informations de déboguage (D-)
- E Vérification des adressages absolus courts et longs (E+)
- L- Génération de code exécutable (défaut)
- L+ Génération de code au format GST
- L2 Génération de code au format DRI
- M Affichage intégral des macros dans le listing (M-)
- O Options d'optimisation (O-)
- P Vérification de la *relogeabilité* du code (P-)
- S Affichage de la table des symboles dans le listing
- T Vérification de type (T+)
- U Modification du préfixe des variables locales (U-)
- W Affichage des messages d'avertissements (W+)
- X Insertion des informations étendues de déboguage (X-)

Par exemple, la ligne :

```
opt m+,s+,w-
```

validera les options d'affichage intégral des macros, d'affichage de la table des symboles et désactivera l'affichage des messages d'avertissements.

<étiquette> EVEN

Cette directive force le compteur de programme (*Program Counter*) à pointer sur une adresse paire (aligné sur un mot). Comme GenST aligne toutes les instructions (sauf DC.B et DS.B) sur des adresses paires, cette directive n'a pas beaucoup d'utilité. Vous pouvez vous en servir pour aligner les chaînes de caractères sur des frontières de mots.

CNOP déplacement, alignement

Cette directive permet de positionner le compteur de programme (PC) en fonction de l'alignement et du déplacement spécifié. Un alignement de 2 permet d'aligner le PC sur une frontière de mots, 4 sur une frontière de mots longs, et ainsi de suite ... Par exemple,

```
cnop 1,4
```

alignera le PC sur la frontière de mots longs suivante + 1 octet.

```
<étiquette> DC.B expression<,expression>...  
<étiquette> DC.W expression<,expression>...  
<étiquette> DC.L expression<,expression> ...
```

Ces directives permettent de définir des constantes en mémoire. Elles peuvent avoir un ou plusieurs opérandes, séparés les uns des autres par des virgules. Les constantes seront alignées sur des frontières de mots pour DC.W, et sur des frontières de mots longs pour DC.L. Il n'est pas possible de réserver plus de 128 octets avec une seule directive DC.

DC.B traite les chaînes de caractères quasiment de la même manière que les constantes caractères décrites plus haut. La même syntaxe s'applique, mais des caractères nuls ne sont pas ajoutés pour compléter la valeur pour en faire un mot. La longueur maximum d'une chaîne est de 128 caractères.

Attention à ne pas mettre d'espace entre les différentes expressions derrière la directive DC.B car ils risquent d'être interprétés comme le début d'un commentaire. Par exemple, GenSt, pour la ligne :

```
dc.b 1,2,3,4
```

ne prendra en compte que les 3 premières expressions, et traitera l'expression , 4 comme un commentaire.

```
<étiquette> DS.B expression  
<étiquette> DS.W expression  
<étiquette> DS.L expression
```

Ces directives permettent de réserver de la place mémoire initialisée à zéro. Si une étiquette est placée devant la directive, elle référencera le début de la zone réservée. L'adresse de début d'une zone est toujours paire pour les directives DS.W et DS.L. Il n'y a pas de restriction sur la taille qu'il est possible de réserver, mais plus la zone sera importante, plus le fichier objet généré le sera.

Par exemple, ces trois lignes réservent 8 octets en mémoire :

```
ds .b      8
ds .w      4
ds .l      2
```

<étiquette> DCB.B nombre,valeur
<étiquette> DCB.W nombre,valeur
<étiquette> DCB.L nombre,valeur

Ces directives permettent de réserver de la place mémoire initialisée à la valeur spécifiée. nombre spécifie combien de fois la valeur doit être répétée en mémoire.

FAIL

Cette directive génère l'erreur `erreur utilisateur`. Elle peut être utilisée par exemple dans les macros pour signaler un mauvais passage de paramètres.

OUTPUT nom_de_fichier

Cette directive fixe le nom du fichier de sortie. Ce nom peut être également fixé lors de l'assemblage en le spécifiant dans les options d'assemblage. Ce dernier nom est alors prioritaire. Si le nom débute par un point, il est considéré comme une extension et est utilisé comme il l'a été décrit auparavant.

__G2 (symbole réservé)

__G2 est un symbole réservé représentant le numéro de version interne de l'assembleur. Il peut être utilisé pour des assemblages conditionnels en utilisant la directive IFD. Sa valeur dépend de la version de l'assembleur et reste toujours absolue.

Structure de répétition

Les structures de répétition permettent de demander à l'assembleur de dupliquer, dans le source, une ou plusieurs instructions un certain nombre de fois sans avoir à le taper à la main.

<étiquette> REPT expression
ENDR

Les lignes à dupliquer doivent être comprises entre les directives REPT et ENDR. Le nombre de duplications à effectuer est spécifié par la valeur de l'expression et doit obligatoirement être positif et non-nul. Il n'est pas possible d'imbriquer ces structures de répétition. Par exemple :

```
REPT      512/4      Copie rapide d'un secteur  
move.l   (a0)+, (a1)+  
ENDR
```

ces trois lignes vont générer 128 instructions `move` lors de l'assemblage.

Note

Il est interdit de définir des étiquettes à l'intérieur d'une telle structure afin d'éviter des définitions multiples d'étiquettes.

Directive de contrôle du listing

LIST

Cette directive permet d'activer, lors de la seconde passe de l'assembleur, la sortie d'un listing sur le périphérique sélectionné au démarrage de l'assemblage. Le listing s'interrompra sur une directive `END`, `NOLIST` ou en fin de fichier.

L'utilisation des directives `LIST+` et `LIST-`, pour, respectivement, activer et désactiver la sortie du listing, permet de bien mieux maîtriser ce processus. Mais attention, il faut tenir compte du nombre d'utilisation de chacune d'entre elle. En effet, GenST tient à jour un compteur interne, incrémenté par toutes directives `LIST+` et décrémenté par `LIST-`, qui lui permet de savoir si le listing est actif (lorsque ce compteur est supérieur ou égal à 0) ou non (lorsqu'il est négatif). Par défaut, ce compteur est égal à -1 en début d'assemblage (sortie de listing désactivée). Ce compteur est forcé à 0 si vous utilisez normalement la directive `LIST` ou à -1 pour `NOLIST`.

NOLIST

Cette directive stoppe la sortie d'un listing d'assemblage.

Lorsque GenST doit envoyer un listing sur disque ou sur imprimante, il le formate automatiquement par page. Chacune de ces pages comporte une en-tête, composée d'une première ligne, comportant la signature de GenST (Copyright, no de version, etc...), la date, l'heure et le numéro courant de la page, d'une seconde ligne comportant le titre du programme, d'une troisième ligne contenant le sous-titre, et enfin d'une quatrième ligne, vide. La date est affichée au format français JJ/MM/AA, à moins que vous n'utilisiez une machine américaine, auquel cas la date serait affichée au format MM/JJ/AA. Entre chaque page, un caractère de saut de page est envoyé (*Form Feed* : code Ascii 12).

PLEN expression

Cette directive fixe la longueur d'une page en nombre de lignes. La valeur par défaut est 60, mais vous pouvez spécifier une valeur comprise entre 12 et 255.

LLEN expression

Vous pouvez spécifier la longueur d'une ligne du listing d'assemblage en spécifiant une valeur comprise entre 38 et 255. La valeur par défaut est 132.

TTL chaîne

Cette directive vous permet de spécifier le titre qui est affiché au début de chacune des pages du listing. Mettez le entre apostrophes s'il contient des espaces. La première directive TTL fixera le titre de la première page. Si vous n'utilisez pas cette directive, le nom du fichier courant sera utilisé comme titre.

SUBTTL chaîne

Cette directive vous permet de spécifier le sous-titre qui est affiché au début de chacune des pages du listing. Mettez le entre apostrophes s'il contient des espaces. La première directive SUBTTL fixera le sous-titre de la première page.

SPC expression

Cette directive insérera le nombre de lignes vides spécifié par l'expression dans le listing d'assemblage.

PAGE

Effectue un saut de page dans le listing.

LISTCHAR expression<,expression> ...

Vous pouvez envoyer à l'imprimante ou sur disque des caractères de contrôle destinés à être interprétés par l'imprimante. Par exemple, sur les imprimantes EPSON, la ligne :

```
listchar 15
```

fera passer l'imprimante en mode compressé 132 colonnes.

FORMAT

paramètre<,paramètre> ...

Cette directive vous permet de contrôler au mieux le format d'une ligne de listing. Les paramètres pris par cette directive permettent de contrôler individuellement chaque champ d'une ligne de listing. Ces paramètres se composent d'un chiffre (0, 1, 2), indiquant le no du champ, suivi d'un signe + (pour valider le champ) ou - (pour l'invalider) :

- 0 numéro de ligne en décimal
- 1 nom/numéro de section et compteur de programme (PC).
- 2 données hexadécimales affichées par mots

Directives concernant les étiquettes

étiquette EQU expression

Cette directive permet d'affecter à l'étiquette spécifiée le résultat de l'expression. Il n'est pas possible d'utiliser de références avant dans l'expression, ni de faire appel à des références externes. Vous devez obligatoirement spécifier une étiquette, qui ne peut en aucun cas être locale, en face de cette directive.

étiquette = expression

Cette directive est équivalente à EQU.

étiquette EQUR registre

Cette directive vous permet de référencer, sous un autre nom, les registres de données ou d'adresses.

étiquette SET expression

Cette directive est similaire à EQU, la seule différence est qu'il est toujours possible de redéfinir une étiquette définie par SET alors que c'est impossible avec EQU. Il n'est pas possible d'utiliser de références avant dans l'expression. Cette directive est particulièrement pratique pour créer des compteurs dans les macros en écrivant, par exemple, la ligne suivante :

```
compteur SET compteur+1
```

(en supposant que ce compteur soit mis à zéro en début de source). A la fin de la première passe, les étiquettes définies par SET sont détruites. Leurs valeurs seront donc heureusement identiques lors des deux passes.

étiquette REG liste-de-registres

Cette directive vous permet d'initialiser un symbole avec une liste de registres qui pourra être utilisée dans les instructions MOVEM, diminuant de ce fait, la probabilité d'avoir une liste de registres différente entre le début et la fin d'une routine. Un symbole défini par REG peut être utilisé dans une instruction MOVEM, mais aussi au sein d'une expression moyennant un message d'avertissement de la part de GenST - la valeur utilisée dans ce cas là étant la même que celle utilisée dans un MOVEM..

```
<étiquette> RS.B    expression
<étiquette> RS.W    expression
<étiquette> RS.L    expression
```

Ces directives vous permettent de définir des listes de constantes pouvant être très utiles pour référencer des structures de données, des paramètres de fonction ou des variables locales. Illustrons ceci par des exemples.

Supposons que vous désiriez utiliser une structure de données composé d'un mot long, d'un octet et d'un deuxième mot long, dans cet ordre. Pour rendre le code plus lisible et plus facilement modifiable, vous pouvez définir cette structure comme suit :

```
                  rsreset
s_suivant rs.l      1
s_etat     rs.b     1
s_info     rs.l     1
```

et vous pouvez y accéder de la manière suivante :

```
                  move.l    s_suivant(a0),a1
                  move.l    s_info(a0),d0
                  tst.b     s_etat(a0)
```

Supposons maintenant que vous désirez utiliser des variables locales référencées par le registre A6 (comme dans MonST et GenST). Vous pouvez définir les références à vos variables locales de la façon suivante :

```
etat            rs.b      1
debut           rs.l      1
fin             rs.l      1
```

sachant que vous pouvez les utiliser de la manière suivante :

```
                  move.b    etat(a6),d1
                  move.l    debut(a6),d0
                  cmp.l     fin(a6),d0
```

Ces directives utilisent un compteur interne qui est mis à zéro au début de chaque passe. Chaque fois qu'une telle directive est rencontrée, l'étiquette reçoit la valeur du compteur. La valeur est éventuellement alignée sur un mot pour les directives RS.W et RS.L. La valeur du compteur est ensuite incrémentée en fonction de la valeur de l'expression et de la taille considérée. Dans cet exemple, etac vaut 0, debut, 2, et fin, 6.

RSRESET

Cette directive remet à zéro le compteur utilisé par les directives RS. Utilisez la si vous définissez plusieurs structures avec les directives RS.

RSSET expression

Initialise le compteur interne RS à une valeur donnée.

__RS (symbole réservé)

Ce symbole réservé contient la valeur du compteur interne utilisé par la directive RS.

Les directives d'assemblage Conditionnel

Cet ensemble de directives a pour but de permettre de faire dépendre l'assemblage d'une partie de programme du résultat d'une ou de plusieurs conditions (le résultat d'un calcul, la valeur d'une constante défini par EQU ou par SET, etc...). Ces directives sont très utiles pour, par exemple, écrire des programmes destinés à tourner sur des environnements différents, en contrôlant grâce à elle, l'assemblage de telle ou telle partie du code adaptée à tel ou tel environnement - la majeure partie du programme restant la même.

GenST possède de nombreuses directives d'assemblage conditionnel. Un bloc conditionnel doit être délimité, en son début, par l'une des nombreuses directives IF et, à sa fin, par la directive ENDC. Il est possible d'imbriquer des directives IF jusqu'à 65535 niveaux.

Il est interdit de placer des étiquettes en face des directives IF et ENDC.

IFEQ	expression
IFNE	expression
IFGT	expression
IFGE	expression
IFLT	expression
IFLE	expression

Ces directives évaluent l'expression spécifiée, la comparent à zéro, et autorise ou non l'assemblage du code se trouvant juste en dessous d'elle, en fonction du résultat de l'expression et du code condition spécifié. Les codes conditions sont exactement les mêmes que ceux du 68000 (IFEQ <==> IF Equal, IFNE <==> IF Not Equal, etc ...). Par exemple, si le symbole `DEBUG` est égal à 1 dans le source suivant :

```

login      IFEQ      DEBUG
           dc.b      'Entrez une commande :',0
           ENDC

           IFNE      DEBUG
login      opt      d+   info debug
           dc.b      'Hep, tape quelque chose :',0
           ENDC

```

GenST n'assemblera pas le premier bloc de code (`dc.b 'Entrez une commande :',0`) puisque `DEBUG` n'est pas nul, mais assemblera la seconde.

Note

La directive `IFNE` correspond à `IF` dans certains assembleurs ne disposant que d'une seule directive d'assemblage conditionnel.

IFD	symbole
IFND	symbole

Ces directives permettent de faire dépendre l'assemblage en fonction de l'état de définition d'un symbole, c.a.d défini ou non. Avec `IFD`, l'assemblage conditionnel se fait si le symbole est défini. Avec `IFND`, l'assemblage conditionnel ne se fait que si le symbole n'est pas défini. Utilisez ces directives avec prudence, car il est facile de générer un code différent entre les premières et deuxièmes passes. Vous pouvez également utiliser ces directives avec les symboles réservés.

IFC **'chaine1','chaine2'**

Cette directive permet de faire dépendre l'assemblage du résultat de la comparaison entre deux chaînes de caractères - l'assemblage conditionnel étant activé lorsque les deux chaînes sont identiques, il est désactivé sinon. La distinction entre les Majuscules et les minuscules est faite lors de la comparaison.

IFNC 'chaine1','chaine2'

Cette directive est exactement l'inverse de la précédente. A première vue, elle peut sembler sans intérêt, mais elle est en fait très utile au sein de macros (Reportez-vous au paragraphe suivant).

ELSEIF

Cette directive active l'assemblage conditionnel s'il est inactif, et vice-versa.

ENDC

Cette directive permet de terminer une directive d'assemblage conditionnel IFcc. GenST signalera une erreur en fin d'assemblage s'il trouve plus de IFs que de ENDCs.

IIF expression instruction

Ceci est une forme abrégée de la directive IFNE, permettant d'assembler une seule instruction ou directive si la condition spécifiée est différente de zéro. Ne pas utiliser ENDC avec cette directive.

Macro Instructions

GenST supporte entièrement les macro-instructions de type Motorola qui, utilisées en conjonction avec les directives d'assemblage conditionnel, facilitent l'écriture des programmes tout en les rendant plus lisible.

La macro-instruction représente le moyen, simple, d'associer à un nom, une série d'instructions ou de directives fréquemment utilisées ensemble dans un programme. A partir du moment où elle est définie, l'appel à une macro se fait de la même manière qu'une directive, en écrivant son nom, suivi de ses paramètres éventuels (jusqu'à 36 paramètres).

étiquette MACRO

Cette directive délimite le début de la définition d'une macro. Les instructions qui vont suivre seront copiées dans un tampon jusqu'à ce que l'assembleur rencontre la directive ENDM. Il n'est pas possible d'imbriquer les définitions de macros.

ENDM

Cette directive délimite la fin de la définition d'une macro (A n'utiliser qu'après une directive MACRO).

MEXIT

Stoppe prématurément l'évaluation de la macro courante (Voir plus loin l'exemple de la macro INC).

NARG (symbole réservé)

NARG n'est pas une directive mais un symbole réservé. Il contient le nombre de paramètres passés à la macro. A l'extérieur d'une macro, ce symbole a pour valeur 0. Si GenST est dans le mode 'distinction MAJ/min', NARG doit être écrit en majuscules.

Les Paramètres de Macros

Quand une macro a été définie, elle peut être appelée en utilisant son nom comme si c'était une directive. Une macro peut avoir jusqu'à 36 paramètres. Pour référencer des paramètres dans une définition de macro, utilisez le caractère backslash (\) suivi d'un caractère alphanumérique (1-9, A-Z, ou a-z) qui sera remplacé par le paramètre correspondant quand la macro sera utilisée, ou par rien du tout si aucun paramètre n'est passé à la macro. Il existe aussi un paramètre spécial, \0 qui représente l'attribut de taille éventuellement ajouté à la macro. La valeur de \0 peut être B (Byte), W (Word) ou L (Long) ; W est la valeur par défaut si aucun attribut de taille n'est spécifié.

Si un paramètre de macro contient des virgules ou des espaces, vous devez le mettre entre les caractères < et >. Dans ce cas, si vous voulez utiliser le caractère > ou < en tant que tel dans ce paramètre, vous devez le doubler (>> ou <<).

Vous pouvez convertir automatiquement une valeur numérique contenue dans un symbole en une suite de caractères numériques, en décimal ou en hexadécimal. Ceci en utilisant la syntaxe \<symbole> ou \\$<symbole>, cette deuxième forme indiquant la conversion hexadécimal. La valeur numérique contenue dans le symbole sera convertie en une chaîne de caractères numériques représentant cette valeur. Le symbole à convertir doit être défini et absolu.

Vous pouvez générer automatiquement des étiquettes uniques à chaque appel de macro en utilisant \@. Ces deux caractères seront remplacés par la séquence _nnn, où nnn représente un nombre qui est incrémenté à chaque appel de macro. Il peut être étendu jusqu'à cinq chiffres pour les programmes plus importants.

Vous pouvez utiliser le caractère \ dans une macro en l'écrivant \\
.

Vous pouvez écrire un appel de macro sur plusieurs lignes quand, par exemple, vous devez passer de nombreux paramètres. Pour cela, terminez la ligne par une virgule, et commencez la ligne suivante par `;`.

Par défaut dans le listing d'assemblage, vous ne voyez que l'appel de la macro et non le code qu'elle génère. Vous pouvez changer cela en utilisant la directive `OPT M` décrite précédemment.

Les appels de macro peuvent être imbriqués et utiliser la récursivité, tant que la mémoire le permet.

Les noms de macros peuvent commencer par un point. Ces noms sont stockés séparément, et ne sont donc jamais confondus avec d'éventuelles étiquettes qui porteraient le même nom.

Exemples de Macros

Exemple 1 - Appeler Gemdos

Une séquence type d'appel d'une fonction Gemdos est la suivante :

```
Empiler le numéro de fonction sur la pile
Appeler TRAP #1
Réinitialiser le pointeur de pile
```

Une telle macro pourrait prendre la forme suivante :

```
appel_gemdos      MACRO
    move.w         #\1,-(a7)  no de fonction
    trap          #1
    lea           \2(a7),a7  reinitialisation pointeur de pile
ENDM
```

Les directives sont en majuscule uniquement pour mieux les distinguer, ceci n'est pas obligatoire.

Si vous désirez maintenant appeler la fonction Gemdos 2 (écriture d'un caractère), vous pouvez écrire :

```
move.w  #'X',-(a7)
appel_gemdos 2,4
```

Quand la macro est appelée, `\1` est remplacé par 2, et `\2` par 4. Si le paramètre `\0` était utilisé dans la macro, il aurait pour valeur `w` puisqu'il n'est pas spécifié lors de l'appel. Le code généré par cet appel de la macro est donc dans ce cas :

```
move.w  #2,-(a7)
trap    #1
lea     4(a7),a7
```

Exemple 2 - une instruction INC

Le 68000 n'a pas d'instruction INC que d'autres processeurs possèdent, l'instruction ADDQ #1 est utilisé à la place. Ecrivons donc une macro qui simule l'instruction INC :

```
inc      MACRO
        IFC      '', '\1'
        FAIL    manque parametre!
        MEXIT
        ENDC
        addq.\0 #1, \1
        ENDM
```

Elle peut maintenant être appelée de la manière suivante :

```
inc.l   a0
```

qui générera le code :

```
addq.l #1, a0
```

La macro commence par comparer le premier argument avec une chaîne vide, en générant un message d'erreur à l'aide de la directive FAIL lorsqu'il y a égalité. La directive MEXIT est alors utilisé pour quitter la macro sans que l'évaluation de ce qu'il en reste soit fait. Dans le cas contraire, l'assembleur va générer une instruction ADDQ en utilisant le paramètre \0 comme attribut de taille d'opérande.

Exemple 3 - Macro factorielle

Cette macro permet d'obtenir la factorielle d'un nombre. Cependant, avant de nous intéresser à la macro proprement dite, voyons comment une factorielle peut être écrite dans un langage évolué tel que le Pascal :

```
function factor(n:integer):integer
begin
    if n>0 then
        factor := n*factor(n-1)
    else
        factor := 1
    end;
;
```

La macro factor utilise la directive SET pour effectuer la multiplication $n*(n-1)*(n-2)...$ de la manière suivante :

```

* parametre 1=etiquette, parametre 2='n'
factor MACRO
    IFND \1
\1      SET    1 defini le symbole s'il ne l'est pas
        ENDC
        IFGT \2
\1      factor \1,\2-1 appelle factor (n-1)
        set   \1*(\2)  n=n*factor(n-1)
        ENDC
        ENDM

```

```

*Exemple d'appel
    factor test,3

```

Le résultat de l'exemple ci-dessus affecte la valeur 3! (factorielle 3) à l'étiquette test. Nous avons mis des parenthèses autour du deuxième paramètre dans la deuxième directive SET car ce deuxième paramètre peut ne pas être qu'une simple valeur, mais une combinaison d'opérateurs. Si l'on ne met pas de parenthèses, la macro peut générer la ligne :

```
test    set    test*5-1-1-1
```

(c.a.d test*5-3) au lieu de :

```
test    set    test*(5-1-1-1)
```

(c.a.d test*2)

Exemple 4 - Instruction de Retour Conditionnel

Le 68000 ne possède pas d'instruction de retour conditionnel de sous-programme, comme on en trouve sur d'autres microprocesseurs. Implantons donc une instruction *retour si égalité* qui nous permettra de voir l'usage de \@.

```

rtseq   MACRO
        bne.s  \@
        rts
\@
        ENDM

```

le paramètre \@ a été utilisé pour générer une étiquette unique chaque fois que l'on appellera la macro. Les étiquettes générées seront par exemple _002 ou _017.

Exemple 5 - Substitution Numérique

Supposons que vous ayez une constante `version` contenant le numéro de version de votre programme, et que vous désiriez l'afficher dans un texte.

```
no_version MACRO
            dc.b      \1, '<version>', 0
            ENDM
```

```
version    equ      42
            no_version '<Programme version v>
```

générera la ligne suivante :

```
            dc.b      'Programme version v', '42', 0
```

La chaîne de caractères passée en paramètres est mise entre `<>` car elle contient des espaces.

Exemple 6 - Appel de Macro Complexe

Supposons que vous écriviez un programme comprenant une structure de tableau compliquée dont les éléments peuvent être de taille variable. Ecrivons une macro qui ne prenne en compte que les paramètres spécifiés :

```
table      MACRO
            dc.b      .fin\@-*          longueur de la table
            dc.b      \1                param tre obligatoire
            IFNC      '\2', ''
            dc.w      \2, \3            2e et 3e parametres associes
            ENDC
            dc.l      \4, \5, \6, \7
            IFNC      '\8', ''
            dc.b      \8                Texte
            ENDC
            dc.b      \9
            .fin\@  dc.b      0          Fin de la table
```

* Exemple d'appel

```
            table    $42,,,t1,t2,t3,t4,
&            <Entrez le nom :>,%0110
```

Cet exemple a pour but de vous expliquer la façon dont les macros peuvent considérablement simplifier la programmation lorsque des structures de données complexes doivent être utilisés.

Dans ce cas précis, le tableau est constitué d'un premier octet de longueur, calculé grâce à la création d'une étiquette locale avec \@ auquel la valeur de PC (*) est soustraite, de deux mots optionnels, de quatre longs, d'une chaîne de caractères optionnelle, d'un octet, et enfin d'un octet à zéro. Voici maintenant le code généré par cet exemple :

```
dc.b      .fin_001
dc.b      $42
dc.l      t1,t2,t3,t4
dc.b      'Entrez le nom : '
dc.b      %0110
.fin_001  dc.b      0
```

Format des fichiers de sortie

GenST a la possibilité de générer plusieurs types de fichiers binaires différents, vous offrant ainsi le maximum de flexibilité. Cette section a pour but de vous les expliquer en détails, et d'en examiner les avantages et inconvénients. Certaines directives ont un fonctionnement qui varie selon le format du fichier de sortie spécifié.

Fichiers exécutable

Ces fichiers sont directement exécutables, par exemple en double-cliquant sur leur icône depuis le bureau GEM. Ils peuvent contenir des informations de relogement et/ou de débogage. Les extensions des noms des fichiers exécutables sont .PRG, .TOS, .TTP et .ACC.

Avantages

Vraies sections BSS, temps de développement réduit.

Inconvénients

Problématique lorsqu'il y a plusieurs programmeurs sur le même programme.

Fichiers objets au format GST

Quand vous écrivez de longs programmes, ou quand vous désirez mélanger des modules assembleur avec des modules écrits en langages évolués, il est obligatoire de créer des fichiers chaînables. Le format GST est reconnu par la majorité des langages évolués provenant d'Angleterre (et d'autres) comme par exemple HiSoft BASIC et Lattice C. Les fichiers GST portent généralement l'extension .BIN.

Avantages

Grand degré de liberté - les variables importées peuvent être utilisées pratiquement n'importe où y compris dans des expressions arbitraires. Les bibliothèques peuvent être créées directement depuis l'assembleur, et les techniques d'importation de variables permettent à l'assembleur de détecter les conflits de type.

Inconvénients

Le format des bibliothèques implique une édition de liens lente. Les sections GEMDOS ne sont pas supportées en standard, bien que LinkST puisse créer de vraies sections BSS.

Fichiers objets au format DRI

C'est le format créé par Digital Research pour CP/M 68k, et repris initialement sur ST. Il est supporté, éventuellement via un utilitaire de conversion, par la majorité des langages évolués provenant des Etats-Unis. Ce type de fichiers porte généralement l'extension .O.

Avantages

Editions de liens plus rapide qu'avec GST, vraies sections GEMDOS.

Inconvénients

Usage restrictif des identificateurs importés, fichiers objets deux fois plus gros que l'exécutable, limite de 8 caractères par identificateur.

Choisir le Bon Format de Fichier

Si vous devez chaîner vos modules assembleurs avec un langage évolué, vous n'avez guère d'autres choix que de choisir le format utilisé par celui-ci.

Si vous écrivez de petits programmes tout en assembleur, le meilleur choix est l'exécutable. Rapide à assembler, pas d'édition de liens, et vous pouvez même chaîner en mémoire pour gagner encore du temps.

Si vous écrivez des programmes plus longs, par exemple plus de 32 Ko d'objet, ou si le programme est écrit par plusieurs personnes, il est plus raisonnable d'utiliser des fichiers chaînables. Nous recommandons le format GST plutôt que le format DRI à cause de sa plus grande flexibilité.

Directives concernant le format des fichiers de sortie

Cette section présente les directives dont le fonctionnement varie en fonction du format du fichier objet. Vous pouvez choisir le format de plusieurs façons : sur la ligne de commande en cliquant sur GENST2.TTP, avec les boutons de la boîte de dialogue d'option d'assemblage de l'éditeur, ou en utilisant la directive OPT L au début du fichier source.

Nous utiliserons des icônes pour indiquer les sections spécifiques à un format de fichier, c'est-à-dire :

Exec  Code exécutable, sur disque ou en mémoire

DRI  Code objet DRI

GST  Code objet GST

Modules et Sections

MODULE

nom_de_module

Cette directive définit le début d'un nouveau module. Le nom de ce module doit être mis entre apostrophes s'il contient des espaces. Cette directive n'est pas obligatoire car ANON_MODULE est le nom donné par défaut à tout module.

Exec  Cette directive est ignorée

DRI  Cette directive est ignorée

GST 

Cette directive permet de créer des bibliothèques assembleur en utilisant plusieurs modules. Chaque module est équivalent à un segment de programme, avec ses propres imports et exports.

Les étiquettes relatives sont locales au module. Vous pouvez donc utiliser deux fois la même étiquette dans deux modules sans conflit. Les étiquettes absolues (comme les constantes) sont globales à tous les modules.

SECTION nom_de_section

Cette directive permet de changer de nom de section. Un programme est composé de plusieurs sections qui sont regroupées, avec celles des autres modules, dans le fichier exécutable final. La section de départ par défaut est la section TEXT. Vous pouvez changer de section à tout moment. Les noms de sections TEXT, DATA et BSS peuvent être écrits tels quels, sans être précédées de la directive SECTION. Ils sont interprétés par GenST comme synonymes des directives SECTION TEXT, SECTION DATA et SECTION BSS respectivement.

Exec 

Les noms de section autorisés sont TEXT, section du programme, DATA pour les données initialisées, et BSS, une zone de mémoire réservée par GEMDOS au lancement du programme.

Cette zone est initialisée à zéro et ne prend pas de place dans le fichier exécutable. La seule instruction autorisée dans une section BSS est DS. Placez-y les variables qui n'ont pas besoin d'être initialisées, vous en économiserez d'autant la place disque.

DRI 

Les règles définies juste au dessus s'appliquent aussi ici.

GST 

Il n'y a pas de règles rigides à propos des noms de sections. Les sections de même nom provenant de différents fichiers sont regroupées par l'éditeur de liens. Les sections sont ordonnées dans le même ordre que dans le texte source.

Imports et Exports

Pour les deux types de fichiers objet, il est crucial de pouvoir importer et exporter des symboles relatifs (référencés au programme) et absolus (constantes). Le format GST distingue ces deux types, alors que le format DRI ne le fait pas. En utilisant le format GST, l'éditeur de liens vous signalera des erreurs de programmation que le format DRI ne trouvera pas.

XDEF export<,export>...

Cette directive permet de définir les variables qui doivent être exportées (c'est-à-dire visibles) vers d'autres fichiers ou modules. Si un export n'est pas défini, un message d'erreur le signalera. Il n'est pas possible d'exporter les étiquettes locales commençant par un point.

Exec 

Cette directive est ignorée

DRI 

Les identificateurs seront tronqués à 8 caractères sans message d'avertissement. Il est recommandé d'utiliser la directive OPT C8.

XREF import<,import>...

XREF.L import<,import>

Ces directives définissent les étiquettes à importer depuis d'autres fichiers ou modules. Si les imports sont aussi définis dans le module courant, un message d'erreur sera affiché. XREF sert à importer des étiquettes relatives (référencées au programme), XREF.L sert à importer des étiquettes absolues (constantes). Il n'est pas possible d'importer plusieurs fois la même étiquette dans le même module.

Exec 

Cette directive est ignorée

DRI 

Le format DRI ne fait pas de différence entre les deux types d'imports, mais il est important de distinguer les deux types pour que l'assembleur puisse les tester. Si vous ne distinguez pas les deux types d'import, supprimez le test de type en utilisant OPT T-1. Les étiquettes DRI n'ont que huit caractères significatifs.

GST

Attention à spécifier le bon type d'import pour ne pas générer d'erreur à l'édition de liens.

Utiliser des Imports dans des Expressions

Exec

Il n'y a pas d'import !

DRI

Un seul import peut être utilisé dans une expression. Une expression comprenant un import doit être du type import+nombre ou import-nombre. Le nombre peut être une expression complexe (sans import!) si elle précède l'import :

```
move.l    3+(1<<compteur+5)+import
```

GST

Jusqu'à dix imports peuvent être utilisés dans une expression. Ils ne peuvent être qu'additionnés ou soustraits entre eux. Ils peuvent être combinés avec une expression complexe (sans import!) si cette expression complexe précède les imports :

```
move.l    3+(1<<compteur+5)+import1-import2
```

Les cas, dans lesquels une expression comprenant un import peut être utilisée, dépendent du format du fichier objet, résumés dans la table ci-dessous :

Expression	GST	DRI	Exemple
PC-octet	O	N	move.w import(pc,d3,w)
PC-mot	O	O†	bsr.s import move.w import(pc),a0
octet	O	N	bsr import move.b #import,d0
mot	O	O	move.w import(a3),d0
long	O	O	move.l import,d0

†A la condition que ce ne soit pas une référence dans une autre section du même fichier, ce qui est interdit.

DRI

Une référence à une étiquette d'une autre section est considérée comme un import, et soumis aux règles ci-dessus.

GST

COMMENT

commentaire

Exec



Cette directive est ignorée

DRI



Cette directive est ignorée

GST



Cette directive inscrit la chaîne dans le fichier `.BIN` à destination de l'éditeur de liens qui l'affichera.

COMMENT HEAD=expression

Cette directive permet d'affecter n'importe quelle valeur au mot long de paramétrage se trouvant dans l'en-tête d'un fichier exécutable GEMDOS - attention, **HEAD** doit obligatoirement être écrit en majuscule. La valeur du bit de poids faible de ce mot long permet d'indiquer au TOS (version 1.4 et plus), lorsqu'il lance un programme, de ne pas initialiser à 0 (bit 0 = 1) toute la mémoire (TPA) se trouvant derrière la zone BSS. Ceci permet de gagner du temps au chargement du programme sur des machines ayant beaucoup de mémoire. La signification des autres bits de ce mot long a été réservée par Atari en vue d'extensions futures.

ORG expression

Cette directive force l'assembleur à générer un code non-translatable, en forçant le compteur de programme à la valeur indiquée. Les programmes GEMDOS standards n'ont pas besoin de cette directive, même si le code est non-translatable. Elle ne sert qu'à générer du code destiné à d'autres machines 68000, ou à mettre en ROM. Vous pouvez utiliser plusieurs directives **ORG** dans un fichier, mais les espaces non utilisés ne seront pas remplis.

Exec



A utiliser avec prudence car le fichier exécutable généré ne pourra probablement pas être exécuté par un double-clic. Le fichier produit aura bien en son début un en-tête GEMDOS mais sans informations de relogement.

DRI



Cette directive n'est pas autorisée, la génération de code absolu étant une option de l'éditeur de liens.



Cette directive est envoyée à l'éditeur de liens qui remplira le fichier de zéros jusqu'à l'adresse spécifiée.



Cette directive ne doit pas être utilisée pour l'assemblage en mémoire.

OFFSET <expression>

Cette directive oblige l'assembleur à générer, dans une section particulière, des références absolues. La valeur facultative de l'expression est affectée au compteur de programme (PC) au début de cette section. Cette directive ne génère aucun octet, et la seule instruction autorisée dans cette section est la directive DS. Elle peut être utilisée pour définir des variables systèmes du ST :

	OFFSET	\$400	
etv_timer	ds.l	1	aura pour valeur \$400
etv_critic	ds.l	1	404
etv_term	ds.l	1	408
ext_extra	ds.l	5	40C
memvalid	ds.l	1	420
memcntl	ds.w	1	424

__LK (symbole réservé)

Ce symbole réservé peut être utilisé pour savoir quel format de fichier objet est généré. Ce symbole peut valoir :

- 0 Exécutable
- 1 Format GST
- 2 Format DRI

Les autres valeurs sont réservées pour des extensions futures.

Option de Débogage DRI

Normalement, seules les étiquettes explicitement exportées avec XDEF sont incluses dans les informations de débogage. Mais le format autorise ce qu'il appelle des *étiquettes locales* (à ne pas confondre avec les étiquettes locales de GenST) qui ne sont pas des exports, mais qui seront quand même incluses dans les informations de débogage. L'option OPT D+ indique à GenST de traiter les étiquettes locales comme étiquettes locales DRI.

Ecrire des bibliothèques GST

Lorsque vous utilisez plusieurs modules pour écrire des bibliothèques au format GST, vous devez prendre garde aux références arrières des imports. Dans une bibliothèque, les routines de bas niveau doivent se trouver en premier, les routine de haut niveau en dernier. Par exemple, le squelette de bibliothèque ci-dessous ne sera pas chaîné :

```
MODULE bas_niveau
XDEF routine_bas_niveau
routine_bas_niveau
etc...

MODULE haut_niveau
XDEF routine_haut_niveau
XREF routine_bas_niveau
routine_haut_niveau
etc...
```

Ceci est dû au deuxième module qui fait référence à une étiquette défini dans un module antérieur, ce qui est interdit. La version correcte est :

```
MODULE haut_niveau
XDEF routine_haut_niveau
XREF routine_bas_niveau
routine_haut_niveau
etc...
MODULE bas_niveau
XDEF routine_bas_niveau
routine_bas_niveau
etc...
```

Exemples simples programmes pour chacun des formats

Cette section présente des exemples incomplets et non fonctionnels d'utilisation de chacun des formats de fichier.

Exécutable

```
SECTION TEXT
debut lea chaine(pc),a0
      move.l a0,sauve_chaine
      bsr ecrit_chaine
      bra quitte

SECTION DATA
chaine dc.b 'Entrez votre nom',0
```

```

SECTION TEXT
ecrit_chaine
    move.l    a0,-(sp)
    move.w    #9,-(sp)
    trap     #1
    addq.l    #6,sp
    rts

quitte    clr.w    -(sp)
    trap     #1

SECTION BSS
sauve_chaine    ds.l    1
END

```

Format DRI

```

XREF.L    quitte
SECTION TEXT
debut    move.l    #chaine,a0
    move.l    a0,sauve_chaine
    bsr     escrit_chaine
    bra     quitte

SECTION DATA
chaine    dc.b    'Entrez votre nom',0

SECTION TEXT
ecrit_chaine
    move.l    a0,-(sp)
    move.w    #9,-(sp)
    trap     #1
    addq.l    #6,sp
    rts

SECTION BSS
sauve_chaine    ds.l    1
END

```

Notez que la première instruction du programme a changé car le mode d'adressage relatif au PC n'est pas autorisé entre sections.

Format GST

```
MODULE PROGTEST
COMMENT Ce programme a encore besoin de
travail XREF.L quitte

SECTION TEXT
debut  lea   chaine(pc),a0
       move.l a0,sauve_chaine
       bsr   ecrit_chaine
       bra   quitte

SECTION DATA
chaine dc.b  'Entrez votre nom',0

SECTION TEXT
ecrit_chaine
       move.l a0,-(sp)
       move.w #9,-(sp)
       trap  #1
       addq.l #6,sp
       rts

SECTION BSS
sauve_chaine ds.l 1
END
```

Résumé des Directives

Contrôle d'assemblage

END	termine le code source
INCLUDE	inclut un fichier source
INCBIN	inclut un fichier binaire
OPT	contrôle d'options d'assemblage
EVEN	force le compteur de programme (PC) sur une adresse paire
CNOP	aligne le compteur de programme (PC)
DC	définit une constante
DS	réserve de la place mémoire
DCB	définit un bloc de constantes
FAIL	force une erreur d'assemblage

Structures de répétition

REPT	début de répétition
ENDR	fin de répétition

Contrôle du listing

LIST	active la sortie du listing d'assemblage
NOLIST	désactive la sortie du listing
PLEN	spécifie la longueur de page
LLEN	spécifie la longueur de ligne
TTL	spécifie le titre
SUBTTL	spécifie le sous-titre
SPC	insère des lignes
PAGE	commence une nouvelle page
LISTCHAR	envoie des codes de contrôle
FORMAT	définit le format du listing

Directives d'étiquettes

EQU	définit une étiquette
EQR	définit un nom de registre
SET	définit temporairement une étiquette
REG	définit une liste de registres
RS	réserve de la place
RSRESET	raz du compteur RS
RSSET	modification du compteur RS

Assemblage Conditionnel

IFEQ	assemble si nul
IFNE	assemble si différent de zéro
IFGT	assemble si supérieur
IFGE	assemble si supérieur ou égal
IFLT	assemble si inférieur
IFLE	assemble si inférieur ou égal
IFD	assemble si étiquette définie
IFND	assemble si étiquette non définie
IFC	assemble si chaînes de caractères égales
IFNC	assemble si chaînes de caractères différentes
ELSEIF	switche l'assemblage conditionnel
ENDC	fin de condition
IIF	IF immédiat

Macro Instructions

MACRO	début de définition de macro
ENDM	fin de définition de macro

Directives concernant le format du fichier de sortie

MODULE	début de module
SECTION	change de section
XDEF	exporte une étiquette
XREF	importe une étiquette
COMMENT	envoi d'un commentaire à l'éditeur de liens
ORG	assemblage absolu
OFFSET	défini un offset

Symboles réservés

NARG	nombre de paramètres d'une macro
__G2	numéro interne de version d'assembleur
__RS	compteur RS
__LK	type de fichier objet

CHAPITRE 4

Le débogueur symbolique

Introduction

Un *bogue* est une erreur de programmation. Ce mot vient de l'anglais *bug* qui signifie insecte, bestiole. Son utilisation en informatique date du temps où les ordinateurs fonctionnaient avec des lampes comme nos anciens téléviseurs. A cette époque, en effet, les papillons et autres insectes volants, attirés par la lumière, avaient la fâcheuse habitude de griller les lampes des ordinateurs en venant s'y brûler les ailes. Les programmeurs avaient coutume alors de dire qu'il y avait un *bug* (une bestiole) dans le programme. Les lampes ont disparu, mais l'expression est restée.

Plusieurs mots sont dérivés de *bug*. Le verbe *déboguer*, francisation plus ou moins réussie du verbe *to debug*, signifie *corriger des erreurs de programmation* ou *mettre au point un programme*. Un *débogueur* est un programme permettant de faciliter la tâche ô combien ingrate que sont la recherche et la correction des bogues.

Les programmes écrits en assembleur sont particulièrement sensibles aux erreurs de programmation - une simple erreur peut provoquer le blocage complet (ou plantage) de la machine. Il existe de nombreuses formes de bogues, de la plus simple (par exemple absence d'un retour chariot en fin de ligne à l'impression), en passant par les plus courants (un résultat incorrect) jusqu'aux plus sérieux (la machine se bloque complètement, produisant éventuellement un affichage bizarre).

Pour vous aider à rechercher toutes ces sortes d'erreurs, DevpacST est fourni avec MonST, qui est un débogueur symbolique et un désassembleur, avec lequel vous pouvez examiner la mémoire, exécuter vos programmes instruction par instruction, et récupérer les exceptions causées par des erreurs de programmation. Comme MonST est un débogueur symbolique, vous avez la possibilité de voir vos programmes avec les symboles et étiquettes qui y ont été définis - c'est quand même plus facile d'utiliser des symboles que de se battre avec des nombres hexadécimaux de 6 chiffres.

Bien que MonST soit un débogueur de bas niveau, n'affichant que des instructions 68000 et des octets mémoire, il peut aussi être utilisé pour déboguer des programmes écrits dans un langage évolué compilé. Si le compilateur possède l'option d'insertion de la table des symboles dans le code généré, il vous sera alors possible de visualiser les noms de vos procédures et de vos fonctions directement dans le code. Vous pouvez même demander à visualiser votre source directement. En ce qui nous concerne, nous avons utilisé MonST pour déboguer LinkST qui a été écrit en C. MonST et GenST ont, quant à eux, été écrits en assembleur.

Etant donné que MonST utilise sa propre mémoire écran, l'affichage de vos programmes n'est pas altéré par MonST lorsque vous tracez une routine pas à pas. Ceci est particulièrement utile pour les programmes graphiques comme les applications GEM ou les jeux. MonST utilise aussi un gestionnaire d'affichage qui lui est propre, rendant ainsi possible l'examen des routines AES ou BIOS sans affecter le débogueur.

Trois versions de MonST sont proposées sur le disque. Toutes trois offrent des possibilités similaires de débogage (V. plus loin).

Exceptions du 68000

MonST utilise les exceptions du 68000 pour interrompre un programme en cours d'exécution et pour le mode pas à pas. De ce fait, il serait peut-être judicieux de jeter un coup d'oeil sur le mode de fonctionnement des exceptions sur le ST.

Il existe plusieurs sortes d'exceptions qui peuvent survenir, soit délibérément, soit accidentellement. Lorsque l'une d'entre elles survient, le processeur va d'abord sauver certaines informations dans la pile, puis passer en mode superviseur, et enfin se brancher à une routine de traitement d'exception. Lorsque MonST est actif, il redirige certains traitements d'exceptions afin d'en prendre le contrôle si elles surviennent. La table ci-dessous décrit les différentes formes d'exceptions existantes, ce qu'elles produisent habituellement, et ce qu'il se produit lorsque MonST est actif :

No d'exception	Exception	Effet	MonST actif
2	erreur bus	bombes	interceptée
3	erreur d'adressage	bombes	interceptée
4	instruction illégale	bombes	interceptée
5	division par zéro	bombes	interceptée
6	instruction CHK	bombes	interceptée
7	instruction TRAPV	bombes	interceptée
8	violation de privilège	bombes	interceptée
9	trace	bombes	interceptée
10	émulateur ligne A	interface VDI	interface VDI
11	émulateur ligne F	TOS interne	TOS interne

32	trap #0	bombes	interceptée
33	trap #1	appel GEMDOS	appel GEMDOS
34	trap #2	appel AES/VDI	appel AES/VDI
35-44	trap #3-#12	bombes	redirigée
45	trap #13	appel BIOS	appel BIOS
46	trap #14	appel XBIOS	appel XBIOS
47	trap #15	bombes	interceptée

Les causes exactes de ces exceptions (et la façon d'y remédier) sont détaillées à la fin de cette section, mais, pour résumer :

Les exceptions 2 à 8 sont causées par des erreurs de programmation et sont interceptées par MonST

L'exception 9 peut indirectement être provoquée par une erreur de programmation. Elle est également utilisée par MonST pour le mode pas-à-pas.

Les exceptions 10, 11, 33, 34, 45 et 46 sont utilisées par le système et ne sont pas interceptées.

Les autres exceptions sont interceptées par MonST, et peuvent être utilisées par vos programmes si besoin est.

Les 'bombes' mentionnées ci-dessus indiquent que le ST essaiera de reprendre la main après l'exception, mais le succès n'est pas toujours garanti.

Quand une erreur survient et que MonST n'est pas chargé, le ST affiche un nombre de bombes (ou des *champignons atomiques* si vous avez un très vieux TOS) égal au numéro de l'exception. Le programme courant est interrompu, perdant des éventuelles données non sauvegardées et retourne au bureau GEM.

Occasionnellement, certains plantages graves remplissent l'écran de bombes. C'est très impressionnant, mais pas très utile !

Organisation de la Mémoire

Les versions normales de MonST sont co-résidentes en mémoire avec les programmes que vous désirez déboguer, c.a.d que vous devez d'abord lancer MonST puis charger le programme.

La figure 4.1 montre l'organisation logique de la mémoire du ST avec et sans MonST. L'organisation physique de la mémoire est décrit en Appendice C.

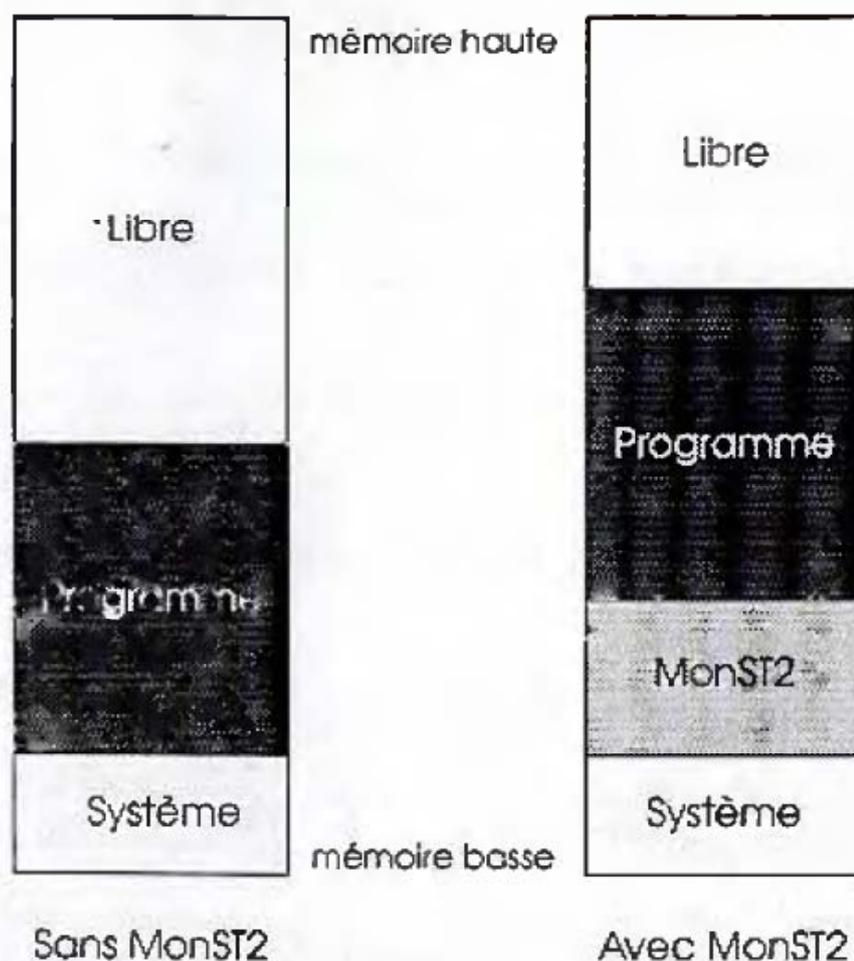


Figure 4.1 - Organisation logique de la mémoire

La taille du code de MonST est d'environ 23 Ko, mais nécessite 32 Ko de mémoire en plus. Cela peut sembler beaucoup, mais MonST doit avoir de la place pour une copie de la mémoire écran ; c'est une des caractéristiques la plus utile de MonST.

Les trois versions de MonST fournies sont :

MONST2.PRG	Version GEM
MONST2.TOS	Version TOS
AMONST2.PRG	Version résidante

Pour l'instant, nous allons décrire les deux premières versions ; la version résidante, quoique très similaire, le sera ultérieurement.

Lancer MonST

A partir du bureau GEM

Les deux versions de MonST sont rigoureusement identiques, exceptée l'extension de leur nom de fichier. La version GEM est utilisée pour les programmes GEM nécessitant une gestion de la souris ; la version TOS est utilisée pour les programmes TOS nécessitant la présence d'un écran blanc avec curseur clignotant. Les deux versions sont lancées en double-cliquant sur leur icône à partir du bureau GEM.

Note

Si vous déboguez un programme TOS avec la version GEM de MonST, le débogueur fonctionnera correctement malgré quelques problèmes mineurs d'affichage. En revanche, il est déconseillé de déboguer un programme GEM avec la version TOS - vous risquez d'avoir de gros problèmes.

A partir de l'éditeur

Lorsque GenST est lancé, il charge automatiquement le fichier MONST2.PRG en mémoire - à moins que cette option n'ait été désactivée dans les Préférences de l'éditeur. Le débogueur est ainsi, sous l'éditeur, instantanément disponible par pression d'une touche.

En appuyant sur **Alt-M**, ou en cliquant sur l'option MonST du menu Prg, vous lancerez MonST de la même manière que si vous l'aviez lancé depuis le bureau.

L'appui sur **Alt-D** ou la sélection de l'option Déboguer du menu Prg permettent aussi de lancer MonST qui, dans ce cas, s'occupera automatiquement de charger le programme précédemment assemblé dans GenST, y compris avec la table des symboles, prêt à être débogué.

Vous pouvez choisir d'exécuter votre programme aussi bien en mode GEM qu'en mode TOS en utilisant l'option Exécution sous GEM du menu Prg. Si une marque est présente devant cette entrée de menu, l'exécution se fera sous GEM, sinon, elle se fera en mode TOS. Les règles décrites plus haut concernant le bon choix du mode d'exécution s'appliquent bien sûr ici.

Débogueur symbolique

Une des caractéristiques principales de MonST est la capacité d'utiliser les symboles définis dans le source pour déboguer le programme. MonST reconnaît deux formats d'informations de débogage (ou table de symboles) : le standard DRI qui n'autorise que 8 caractères par symbole, et le *Format Etendu HiSoft* qui autorise jusqu'à 22 caractères par symbole. GenST et LinkST peuvent produire les deux types de formats, et quasiment tous les compilateurs et éditeurs de liens du marché peuvent générer des informations de débogage au format DRI. Nous essayons d'établir comme deuxième standard le Format Etendu HiSoft, mais à l'heure où nous écrivons, seuls trois produits le supportent (HiSoft BASIC, FTL-Modula 2 et Lattice C).

Les boîtes de dialogue et les boîtes d'alerte de MonST

MonST utilise intensivement les boîtes de dialogue et les boîtes d'alerte, mais ce ne sont pas exactement les mêmes que celles de GEM, même si elles ont un fonctionnement analogue. MonST n'utilise pas de véritables boîtes de dialogue GEM afin d'éviter d'éventuelles interactions, lors du débogage, avec les programmes qui utilisent les routines GEM. Pour la même raison, la souris n'est pas non plus disponible dans le débogueur lui-même.

Une boîte de dialogue type MonST se compose du message `Esc` pour abandonner juste au-dessus du coin supérieur gauche, d'un titre, généralement d'une ligne de saisie et d'un curseur. Vous pouvez à tout moment quitter la boîte de dialogue en appuyant sur `Esc` ou entrer des données. Les touches `curseur`, `Backspace` et `Delete` peuvent être utilisées pour modifier le texte saisi. Appuyez sur la touche `Clr` pour effacer la ligne, à la différence de GEM où vous devez appuyer sur `Esc`. Appuyez sur `Return` pour valider votre saisie. L'écran clignote lorsque la ligne saisie est erronée, vous invitant à la corriger. Lorsque une boîte de dialogue comporte plusieurs lignes de saisie, vous devez les entrer l'une après l'autre. Vous ne pouvez pas vous déplacer d'un champ à l'autre avec les touches `curseur` comme sous GEM.

Une boîte d'alerte MonST est une petite boîte affichant un message et vous demandant d'appuyer sur `Return`. Vous êtes ainsi informé d'une erreur. La boîte disparaîtra en appuyant sur `Return` ou `Esc`.

L'écran d'accueil

A moins que vous n'ayez lancé MonST avec l'option Déboguer depuis l'éditeur, une boîte de dialogue vous demandant d'entrer le nom du programme à déboguer apparaîtra à l'écran. Si vous désirez charger et déboguer un programme, vous devrez entrer son nom, son extension si celle-ci n'est pas .PRG, et appuyer sur Return. Vous pourrez ensuite sélectionner une commande pour commencer le débogage. Si vous désirez seulement examiner la mémoire sans charger de programme, appuyez sur Esc ou entrez une ligne vide.

Low Res

Certaines fonctionnalités de MonST fonctionnent différemment ou ne sont pas disponibles si vous exécutez MonST en basse résolution. Elles seront signalées par cette icône.

Le panneau de contrôle

L'écran principal de MonST se présente sous la forme d'un panneau de contrôle (*front panel*) montrant les registres, la mémoire et les instructions. Il est constitué de quatre fenêtres (Voir figure 4.2). En basse résolution, ces fenêtres sont organisées un peu différemment, en essayant d'exploiter au maximum le peu d'espace d'affichage autorisé par ce mode.

1 Registres		2 Déassemblage PC		3 Mémoire	
00:00000000	602E 0301 00E0 0030	A0:00000000	602E 0301 00E0 0030	00000000	602E 0301 00E0 0030
01:00000000	602E 0301 00E0 0030	A1:00000000	602E 0301 00E0 0030	00000004	00E0 0030 α 0
02:00000000	602E 0301 00E0 0030	A2:00000000	602E 0301 00E0 0030	00000008	0000 E904 297
03:00000000	602E 0301 00E0 0030	A3:00000000	602E 0301 00E0 0030	0000000C	0000 E90A 290
04:00000000	602E 0301 00E0 0030	A4:00000000	602E 0301 00E0 0030	00000010	0000 EA42 298
05:00000000	602E 0301 00E0 0030	A5:00000000	602E 0301 00E0 0030	00000014	0000 EA48 29H
06:00000000	602E 0301 00E0 0030	A6:00000000	602E 0301 00E0 0030	00000018	0000 EA4E 29H
07:00000000	602E 0301 00E0 0030	A7:00000000	602E 0301 00E0 0030	0000001C	0000 EA54 29T
SR:0000 U		A7':00000000	602E 0301 00E0 0030	00000020	0000 EA5A 29Z
PC:00E0003D MOVE.W #52700,SR				00000024	0000 EA60 29\
				00000028	00E0 6ADA κj0
				0000002C	00E0 DEF2 κ/2

MonST 1.2 © HISoft 1990

Figure 4.2 - Ecran principal de MonST

La fenêtre du haut (numéro 1) affiche les valeurs des registres de données et d'adresses, ainsi que la mémoire pointée par ces registres.

La fenêtre suivante (numéro 2) affiche le désassemblage de la mémoire sur plusieurs lignes. Par défaut, le désassemblage commence à partir de l'adresse du registre de compteur de programme (PC). Une petite flèche vers la droite indique la position de PC.

La troisième fenêtre affiche le contenu d'une partie de la mémoire en notation hexadécimale et ASCII à partir d'une adresse paire.

La dernière fenêtre tout en bas de l'écran est utilisée pour l'affichage de divers messages.

Une des caractéristiques les plus importantes de MonST est sa flexibilité dans l'utilisation des fenêtres. Vous pouvez créer jusqu'à deux nouvelles fenêtres, la taille des caractères peut être changée, et l'affichage des fenêtres peut être verrouillé sur la valeur d'un registre (Voir plus loin).

Maniement des fenêtres

MonST utilise le concept de *fenêtre courante* - signalé par l'inversion vidéo du titre de fenêtre. Vous pouvez changer de fenêtre courante en appuyant sur la touche `Tab` - chaque fenêtre devenant active à tour de rôle - ou en appuyant simultanément sur la touche `Alt` plus le numéro de la fenêtre que vous désirez sélectionner. Par exemple, l'appui sur `Alt-2` permet de sélectionner la fenêtre de désassemblage (les utilisateurs de claviers AZERTY ne doivent pas appuyer sur la touche `Shift` lorsqu'ils utilisent la touche `Alt` pour sélectionner une fenêtre). Veuillez remarquer qu'il n'est pas possible de sélectionner la dernière fenêtre qui n'est utilisée que pour l'affichage de messages.

Entrée des Commandes

Chacune des commandes de MonST peut être appelée par l'appui d'une seule touche, ceci afin d'obtenir une grande rapidité d'utilisation. Vous devrez, bien évidemment, vous familiariser à ces commandes. Les utilisateurs de HiSoft Devpac sur d'autres machines ne seront pas dépaysés, particulièrement sur Spectrum et QL, même si les commandes de fenêtres sont spécifiques à DevpacST.

En général, la touche `Alt` utilisée simultanément avec une autre touche agit sur la *fenêtre courante*.

Les commandes peuvent indifféremment être entrées en majuscule ou en minuscule. Les commandes qui peuvent avoir des effets désastreux nécessitent l'appui supplémentaire de la touche `Ctrl`. Les noms des commandes ont été choisis, dans la mesure du possible, pour être faciles à retenir. Les commandes sont exécutées immédiatement - il n'y a pas besoin d'appuyer sur `Return` pour les valider. Les commandes invalides sont ignorées. Les fenêtres du panneau de contrôle sont immédiatement mises à jour afin d'observer les changements effectués par la commande.

MonST est un outil puissant et parfois complexe. Chaque programmeur n'utilisera donc pas toutes les commandes disponibles. Pour cela, nous avons divisé ce chapitre sur MonST en deux parties. Nous décrirons d'abord les principales commandes de MonST, puis dans un deuxième temps, nous donnerons une liste complète des commandes destinées à être utilisées comme référence. Pour les débutants ou nouveaux utilisateurs, il est possible de prendre en main le débogueur en lisant simplement la paragraphe **Présentation** de MonST et de garder le paragraphe **Référence** pour plus tard.

Présentation de MonST

Pour commencer, vous devez *charger un programme* à déboguer. Si vous avez assemblé un programme en mémoire depuis l'éditeur, cliquez sur l'option **Déboguer**. Sinon, vous devez charger un programme depuis le disque. Entrez son nom dans la boîte de dialogue affichée au démarrage de MonST, ou bien utilisez la commande `Ctrl-L`.

Les symboles d'un programme, s'ils existent, seront utilisés par le débogueur. Un programme contient des symboles si vous utilisez les options d'assemblage **Information de déboguage Normale** ou **Étendue** - l'option étendue signifiant que vous pourrez disposer de symboles plus longs, l'option normale tronquant les symboles à huit caractères.

La commande la plus utilisée de MonST est probablement le *mode pas-à-pas*, que vous pouvez lancer en appuyant sur `Ctrl-Z` (ou `Ctrl-Y` si cela vous paraît plus pratique). Cette commande permet d'exécuter l'instruction pointée par le compteur de programme (PC) - celle comportant une flèche en regard dans la fenêtre 2. Après l'avoir exécuté, MonST mettra à jour les fenêtres registres et mémoires vous permettant de voir, instruction par instruction, l'évolution de l'état du processeur et de la mémoire en fonction de votre programme. L'exécution pas-à-pas d'un programme est la meilleure façon qui soit, de voir ce qui se passe dans une partie de programme, mais aussi, la plus lente. Si la section que vous désirez observer se trouve à la fin de votre programme, cela peut prendre pas mal de temps avant que vous n'arriviez au bon endroit en utilisant le mode pas-à-pas. Il existe bien sûr une solution à ce problème.

Un *point d'arrêt* est une instruction particulière que vous pouvez placer dans un programme pour en stopper l'exécution et activer MonST. Il existe plusieurs types de points d'arrêt, mais nous n'en examinerons que la forme la plus simple pour l'instant. Pour positionner un point d'arrêt, vous devez appuyer sur Alt-B et entrer l'adresse où vous désirez interrompre le programme. Vous pouvez entrer une valeur hexadécimale (la base utilisée par défaut), un symbole, ou une expression complexe (par exemple, vous pouvez entrer 1A2B0, debut_programme ou 10+donnees). Si vous entrez une expression incorrecte, l'écran clignotera et vous devrez ressaisir l'adresse.

Une fois votre point d'arrêt positionné, vous pourrez *exécuter* votre programme en appuyant sur Ctrl-R. L'exécution commencera à l'adresse pointée par PC, et stoppera quand le point d'arrêt sera rencontré, ou quand une exception surviendra.

MonST utilise son propre *écran d'affichage*, indépendant de celui utilisé par les programmes. Appuyez sur V pour voir l'écran dans lequel votre programme s'exécute, appuyez sur une touche quelconque pour revenir à MonST. Vous pouvez ainsi déboguer vos programmes sans en perturber l'affichage.

Vous pouvez effectuer un *zoom* sur une fenêtre, c'est-à-dire lui faire occuper tout l'écran en appuyant sur Ctrl-Z. Pour revenir à la taille initiale, appuyez à nouveau sur Ctrl-Z, ou sur la touche Esc. La touche Esc est aussi le meilleur moyen de sortir d'une commande que vous avez appelée par erreur. La commande de zoom, comme toutes les commandes Alt, porte sur la fenêtre courante. Utilisez la touche Tab pour changer la fenêtre active. Utilisez la commande Alt-P pour imprimer le contenu de la fenêtre courante sur imprimante.

Pour changer l'adresse à partir de laquelle une fenêtre affiche ses données, appuyez sur Alt-A, puis entrez la nouvelle adresse. Veuillez remarquer que la fenêtre de désassemblage affichera toujours la mémoire pointée par PC après une commande d'exécution pas-à-pas parce qu'elle est *verrouillée* sur le registre PC. Reportez vous au paragraphe de *Référence* pour connaître le fonctionnement du verrouillage de fenêtre.

Appuyez sur Ctrl-C pour *quitter* MonST. Aussi bizarre que cela puisse paraître, cette commande ne quittera pas toujours MonST. En fait, cette commande termine le programme en cours, que ce soit MonST ou votre propre programme. Vous savez que l'exécution de votre programme est terminée lorsqu'un message de confirmation apparaît en bas de l'écran. Appuyez alors de nouveau sur Ctrl-C pour quitter MonST. Si vous utilisez l'option *Déboguer* de l'éditeur, cette commande terminera alors votre programme en même temps que MonST.

Nous espérons que ce survol rapide de ces quelques options de MonST vous aura donné une idée de la manière de déboguer vos programmes. Une fois ces notions acquises, vous pourrez attaquer la partie **Référence** qui doit être appliquée à petites doses, comme toute thérapie.

Guide de Référence de MonST

Expressions numériques

MonST possède un évaluateur d'expressions, basé sur celui utilisé par GenST, prenant en compte la priorité des opérateurs. Les principales différences par rapport à celui de GenST concernent la base utilisée par défaut, qui est l'hexadécimale (les nombres décimaux peuvent être entrés s'ils sont précédés du caractère \), le type des expressions (relatif ou absolu) qui n'existe pas, l'opérateur * qui n'est plus utilisé que pour la multiplication, et la présence d'un signe *différent de*, noté <>.

Des symboles peuvent être utilisés dans les expressions puisque la distinction majuscules/minuscules est normalement effectuée et qu'ils peuvent avoir entre 8 ou 22 caractères de long selon l'option de débogage utilisée dans l'assembleur (utilisez les Préférences pour changer cette valeur).

Il est possible d'utiliser des registres dans les expressions en utilisant tout simplement leur nom, (comme A3, a7 ou D6). Pour ne pas confondre les noms de registres avec des valeurs hexadécimales, vous devez faire précéder les valeurs hexadécimales équivalentes à un nom de registre par le caractère \$ ou par un 0. Le registre A7 fait toujours référence au pointeur de pile utilisateur.

Les symboles réservés suivants peuvent également être utilisés dans les expressions, indifféremment écrits en majuscules ou en minuscules : TEXT, DATA, BSS, END, SP, SR et SSP. END représente l'adresse de fin de programme, soit un octet après la fin du segment BSS. SP représente le pointeur de pile utilisateur ou superviseur selon la valeur du registre d'état.

Il existe dix mémoires notées de M0 à M9. Elles peuvent être incluses dans les expressions comme étant des registres. Il est possible de les modifier en utilisant la commande Alt-R. Les mémoires de 2 à 5 inclu représentent les adresses de début des fenêtres de numéro correspondant. Modifier l'une de ces six mémoires changera l'adresse de début de la fenêtre correspondante.

L'évaluateur d'expression peut calculer des indirections, qu'il est possible d'imbriquer, en utilisant les accolades (et). Une indirection peut être réalisée sur un octet, un mot ou un long - valeur par défaut - en faisant suivre l'accolade fermante } par un point suivi de l'attribut de taille \exists ou ω . Si le pointeur est invalide, à cause d'une adresse incorrecte ou impaire, l'expression ne le sera pas non plus.

Par exemple, l'expression :

```
(debut_donnees+10).w
```

sera évaluée comme étant la valeur du mot se trouvant à l'adresse `debut_donnees+10`, en supposant que `debut_donnees` est paire.

Types de Fenêtres

Il existe quatre types de fenêtres. Leur signification et représentation à l'écran est décrite ci-dessous. Les types de fenêtres autorisés sont :

Fenêtre	Types Autorisés
1	Registres
2	Désassemblage
3	Mémoire
4	Désassemblage, Mémoire ou Code Source
5	Mémoire

Fenêtre d'Affichage des Registres

La valeur de chacun des registres de données est affichée en hexadécimal, avec la représentation ASCII de son octet de poids faible, suivi de l'affichage hexadécimal des huit octets mémoires sur lesquels il pointe. Les valeurs des registres d'adresses sont également affichées en hexadécimal, suivies des douze octets mémoires sur lesquels ils pointent. Lorsque un registre pointe sur une adresse impaire ou invalide, MonST affiche deux étoiles à la place (**).

La valeur du registre d'état est affichée en hexadécimal et en représentation où chacun des bit d'état est noté par une lettre. Un caractère U ou S est également affiché, et indique si le microprocesseur se trouve en mode utilisateur ou superviseur. A7' représente le registre de pile superviseur et est affiché de la même manière que les autres registres d'adresses.

La valeur du registre $\exists C$ est affichée en hexadécimal avec le désassemblage de l'instruction courante. Si cette instruction fait référence à des adresses effectives, celles-ci sont affichées en hexadécimal avec une suite d'octets sur lesquels ces adresses pointent. Par exemple, l'affichage :

indique que la valeur \$12A(A3) est \$1FAE, et que le mot pointé par cette valeur est \$0F01. Un exemple plus complexe :

```
MOVE.W $12A(A3),-(SP) ;00001FAE 0F01 0002AC08 FFFF
```

L'opérande source génère le même affichage que dans l'exemple précédent. L'adresse de destination est \$2AC08 qui pointe sur la valeur \$FFFF. Notez que l'affichage de la valeur pointée correspond toujours à la taille de l'opérande de l'instruction. Les données de l'instruction MOVEM sont représentée sous la forme de quatre mots. L'adresse effective tient compte d'une éventuelle pré-décrémentation.

Low Res

Aucune donnée pointée par les registres de données n'est affichée, et seuls quatre octets pointés par les registres d'adresses sont affichés. La ligne de désassemblage peut ne pas être assez longue pour afficher complètement les adresses effectives si le mode d'adressage est complexe.

Fenêtre de désassemblage

Cette fenêtre affiche le désassemblage de la mémoire. De gauche à droite apparaissent l'adresse, une étiquette éventuelle, puis l'instruction désassemblée. La valeur courante de PC est indiquée par une flèche.

Si un point d'arrêt est placé sur l'instruction désassemblée, des crochets ([]) sont affichés à côté de l'instruction. Entre ces crochets est affichée une information qui dépend du type de point d'arrêt. Pour les points d'arrêt Stop, c'est le nombre d'exécutions restantes, pour les points d'arrêts conditionnels, c'est un point d'interrogation suivi du début de la condition. Pour les points d'arrêts compteurs, c'est un caractère =, et pour les points d'arrêts permanents, c'est un caractère *.

Le format des codes-opérations du jeu d'instructions accepté par GenST est celui défini par Motorola. Les instructions sont écrites en majuscules (sauf les symboles en minuscules), les nombres sont écrits en hexadécimal, sauf les numéros de Trap en décimal. Les zéros non significatifs ne sont pas écrits, et le caractère \$ n'est présent que devant les nombres supérieurs à 10. La seule différence par rapport au standard Motorola concerne l'instruction MOVEM. Pour économiser de la place, le deuxième registre d'une liste a un nom abrégé. Par exemple l'instruction :

```
MOVEM.L d0-d3/a0-a2,-(sp)
```

sera désassemblée de la manière suivante :

```
MOVEM.L d0-3/a0-2,-(sp)
```

Un symbole sera affiché à la place de l'adresse hexadécimale, et sera limité à huit caractères.

Fenêtre d'affichage de la mémoire

Cette fenêtre affiche le contenu de la mémoire sous la forme d'une adresse hexadécimale, et de mots en hexadécimal et en ASCII. Si les adresses sont invalides, les caractères * * sont affichés à la place de la valeur des octets correspondants. Le nombre d'octets par ligne dépend de la largeur de la fenêtre - au maximum, 16 octets par ligne.

Fenêtre d'affichage du code source

Cette fenêtre affiche un fichier ASCII de la même manière qu'un éditeur d'écran. La longueur de tabulation par défaut est 8, mais il est possible de changer cette valeur en appuyant sur Alt-E.

Commandes agissant sur les fenêtres

La touche Alt est généralement utilisée pour agir sur la fenêtre courante. Le titre de la fenêtre active est en inverse vidéo. Il est possible de changer la fenêtre courante en appuyant sur Tab ou Alt plus un numéro de fenêtre.

La plupart des commandes fonctionnent dans tout type de fenêtre, quelle que soit sa taille, mais il ne se passera rien si la commande ne peut s'appliquer à la fenêtre courante.

Alt-A Adresse de début de fenêtre

Cette commande permet de modifier l'adresse de début de désassemblage ou d'affichage mémoire.

Alt-B Adresse de point d'arrêt

Cette commande vous permet de spécifier un point d'arrêt de n'importe quel type, décrits plus loin au paragraphe Points d'arrêt.

Dans une fenêtre d'affichage de mémoire, cette commande vous permet de modifier la mémoire en entrant des nombres hexadécimaux (1 à 9 et A à F) ou des caractères ASCII. Appuyez sur **Tab** pour sélectionner le mode choisi, hexadécimal ou ASCII. En mode ASCII, chaque touche appuyée aura sa valeur ASCII recopiée en mémoire. Les touches curseur peuvent être utilisées pour se déplacer dans la mémoire. Appuyez sur **Esc** pour quitter le mode d'édition.

Dans une fenêtre d'affichage de registres, cette commande est identique à **Alt-R**. Voir plus bas.

Dans une fenêtre d'affichage de texte source, cette commande change la longueur de tabulation de 8 à 4 caractères et vice-versa.

Vous pouvez choisir la taille des caractères affichés dans une fenêtre. En haute résolution, les hauteurs de caractères disponibles sont 8 et 16 points, en basse et moyenne, 8 et 6 points. En appuyant sur **Alt-F**, vous passez de l'une de ces valeurs à l'autre. Vous pouvez ainsi obtenir un plus grand nombre de lignes à l'écran, si votre moniteur peut le supporter.

Si vous changez la taille des caractères de la fenêtre des registres, la taille des autres fenêtres sera recalculée pour remplir l'espace disponible.

Vous pouvez verrouiller une fenêtre d'affichage de mémoire ou de désassemblage sur la valeur d'un registre. Après toute exception, l'affichage de la fenêtre sera mis à jour pour afficher la mémoire à partir de la valeur du registre spécifié. Cette commande s'applique sur la fenêtre courante.

Par défaut, la fenêtre 2 est verrouillée sur PC. Vous pouvez verrouiller plusieurs fenêtres sur la même adresse en spécifiant un nom de mémoire, comme par exemple **M2** qui contient l'adresse de début de la fenêtre numéro 2:

Cette commande vous demande une expression, et affiche sa valeur en hexadécimal, décimal, et éventuellement sous la forme d'un symbole.

Le contenu de la fenêtre courante est envoyée sur l'imprimante. Cette impression peut être interrompue par **Esc**.

Vous pouvez changer la valeur d'un registre en spécifiant son nom, le signe égal, et sa nouvelle valeur. Vous pouvez également utiliser cette commande pour modifier les mémoires M0 à M9. Par exemple, la ligne :

`a3=a2+4`

ajoutera 4 à la valeur de a2 et rangera cette valeur dans le registre a3.

Vous pouvez utiliser cette commande pour modifier une adresse de début de fenêtre en changeant la valeur de la mémoire associée, lorsque la fenêtre occupe tout l'écran. L'affichage sera mis à jour lorsque vous rendrez à la fenêtre sa taille normale.

Note

Ne modifiez pas M4 si la fenêtre 4 contient un texte source.

Cette commande divise une fenêtre en deux. Ainsi, la fenêtre 2 sera divisée en 2 et 4, et la fenêtre 3 en 3 et 5. Les nouvelles fenêtres sont indépendantes. Appuyez à nouveau sur **Alt-S** pour revenir à l'état initial.

Low Res

Cette commande n'a pas d'effet

Cette commande ne fonctionne qu'avec la fenêtre 4 (créée soit en divisant la fenêtre 2 - par **Alt-S** - soit en chargeant un fichier source). Cette commande change le type d'affichage de la fenêtre alternativement entre désassemblage, affichage mémoire et source (si vous en avez chargé un).

Cette commande permet d'agrandir la fenêtre courante à la taille de l'écran, sans en affecter son fonctionnement. Appuyez à nouveau sur **Alt-Z** ou sur **Esc** pour revenir à la taille normale.

Note

Appliquer cette commande à la fenêtre d'affichage des registres ne sert pas à grand-chose.

Touches Curseur

Ces touches agissent dans la fenêtre active. L'action de chacune d'entre elle dépend du type de la fenêtre.

Dans une fenêtre d'affichage de mémoire, ces touches modifient l'adresse d'affichage. Utilisez la flèche haute ou basse conjointement avec `Shift` pour vous déplacer d'un écran vers le haut ou vers le bas.

Dans une fenêtre de désassemblage, les flèches haute et basse déplacent l'affichage instruction par instruction, les flèches droite et gauche, mot par mot (2 octets).

Dans une fenêtre d'affichage de texte source, les flèches haute et basse décalent l'écran d'une ligne verticalement ou bien d'une page lorsqu'elles sont utilisées conjointement avec la touche `Shift`.

Echange d'écran

MonST utilise son propre écran et ses propres routines d'affichage pour éviter toute interférence avec les programmes que vous déboguez. Pour éviter un scintillement désagréable lorsque vous exécutez un programme en mode pas-à-pas, l'écran n'est échangé avec celui du programme qu'après 20 millisecondes. De plus, lorsque vous exécutez un programme en basse résolution, MonST peut s'exécuter en moyenne résolution.

V Visualisation de l'écran du programme

Cette commande permet d'afficher l'écran utilisé par le programme actuellement en train d'être exécuté. L'appui sur une touche quelconque permet de revenir à MonST.

Ctrl-O Changement de la résolution de l'écran

Cette commande permet de modifier la résolution de l'écran de MonST (uniquement sur moniteur couleur), alternativement de la basse vers la moyenne résolution. La taille des fenêtres et des caractères sont réinitialisés à chaque fois. Elle ne modifie pas le mode de l'écran du programme.

Note

Si votre programme change l'adresse d'écran ou la résolution avec une fonction XBIOS ou en modifiant les registres hardware, vous devez temporairement inhiber la temporisation lors de l'échange d'écran pendant l'exécution de ces lignes de code pour que MonST puisse prendre en compte les modifications (utilisez les Préférences).

L'écran de votre programme est généralement affiché pendant que vous faites des accès disque. En cas d'erreur signalée par une boîte d'alerte, vous pouvez ainsi cliquer sur cette boîte.

Interrompre l'Exécution des Programmes

Shift-Alt-Help

Interruption de programme

Vous pouvez interrompre l'exécution de votre programme en appuyant sur cette combinaison de trois touches. Une exception Trace sera générée à la valeur du PC. Si votre programme effectue de nombreux calculs, il se peut que l'exécution soit interrompue à l'intérieur du programme lui-même. Mais généralement, si votre programme effectue de nombreux appels au BIOS et à GEM, l'interruption va se produire en ROM ou dans les routines *line F* situé dans les adresses basses de la mémoire. Dans ce cas, placez un point-d'arrêt dans votre programme et reprenez l'exécution par `Ctrl-R`.

Si vous appuyez sur `Alt-Help` sans `Shift`, l'écran sera imprimé sur votre imprimante. Si vous avez appuyé sur ces touches par erreur, appuyez à nouveau sur `Alt-Help` pour interrompre l'impression.

Il est possible que, temporairement, cette commande ne fonctionne pas. Essayez alors une deuxième fois. Si vous appuyez sur ces touches quand vous êtes sous MonST, rien ne se passe.

Note

Ne terminez jamais un programme par `Ctrl-C` si vous en avez interrompu l'exécution en ROM. Le système pourrait se bloquer.

Points d'arrêt

Un point d'arrêt vous permet de stopper l'exécution de votre programme à un endroit particulier que vous pouvez spécifier. MonST accepte jusqu'à huit points d'arrêt simultanément dans un programme, chacun pouvant être un des cinq types différents. Ces points d'arrêts, lorsque ils sont rencontrés par le programme, permettent à MonST de reprendre la main et de décider s'il doit arrêter le programme, afficher le panneau de contrôle ou bien continuer l'exécution, ceci dépendant du type de point d'arrêt et de l'état des variables du programme.

Point d'Arrêt Simple.

Ce sont des points d'arrêt qui ne servent qu'une fois. Lorsqu'ils sont rencontrés, l'exécution est interrompue et MonST est activé, puis le point d'arrêt est automatiquement supprimé.

Point d'Arrêt Stop

Ce point d'arrêt doit être exécuté un nombre de fois précis avant d'interrompre l'exécution du programme. Un point d'arrêt simple est un point d'arrêt stop qui a une valeur de compteur égale à 1.

Point d'Arrêt Compteur

Chaque fois qu'un point d'arrêt de ce type est rencontré dans l'exécution du programme, un compteur est incrémenté et le programme n'est jamais interrompu.

Point d'Arrêt Permanent

Ils sont similaires aux points d'arrêt simple, mais ne sont pas effacés une fois qu'ils ont été rencontrés.

Point d'Arrêt Conditionnel

Ce type de point d'arrêt permet de n'interrompre le programme à une adresse spécifiée que si une condition particulière est remplie. A chacun de ce type de point d'arrêt est associé une expression (définie conformément aux règles précédemment énoncées concernant les expressions), laquelle sera évaluée à chaque fois que le programme rencontrera le point d'arrêt, afin de savoir si l'exécution devra se poursuivre (expression nulle ou fausse) ou s'interrompre (expression différente de zéro ou vraie).

Cette commande vous permet de placer ou d'effacer un point d'arrêt. La description du point d'arrêt se fait au travers d'une boîte de dialogue. Le format de la ligne à entrer varie en fonction du type de point d'arrêt :

<adresse>

place un point d'arrêt simple à l'adresse spécifiée.

<adresse>,<expression>

place un point d'arrêt stop. Ce point d'arrêt n'interrompra le programme qu'après avoir été exécuté **<expression>** fois.

<adresse>,<=>

place un point d'arrêt compteur. La valeur initiale du compteur est zéro. La valeur du compteur est affichée lorsque vous désassemblez le programme à l'endroit du point d'arrêt, ou en appuyant sur la touche Help.

<adresse>,*

place un point d'arrêt permanent à l'adresse spécifiée.

<adresse>,<?><expression>

place un point d'arrêt conditionnel. Il interrompra le programme lorsque l'expression sera différente de zéro.

<adresse>,-

efface un point d'arrêt de n'importe quel type à l'adresse spécifiée.

Il est impossible de placer un point d'arrêt sur une adresse impaire ou en ROM. Pour interrompre l'exécution d'un programme en ROM, utilisez la commande Exécution - Jusqu'à.

Chaque fois qu'un point d'arrêt est rencontré, que l'exécution du programme soit interrompue ou pas, l'état du programme est toujours enregistré dans l'Historique (Voir plus loin).

HelpAffichage d'informations diverses et des points d'arrêt

Cette commande affiche les adresses et longueurs des segments TEXT, DATA et BSS, ainsi que les points d'arrêt. Les commandes `Alt` sont utilisables dans cet affichage.

Ctrl-B

Spécifier un Point d'Arrêt

Disponible par souci de compatibilité avec MonST 1, cette commande place un point d'arrêt à l'adresse de début de la fenêtre courante qui doit nécessairement être une fenêtre de désassemblage. Si un point d'arrêt est déjà présent à cette adresse, il sera effacé.

U

Exécuter jusqu'à

Cette commande permet de placer un point d'arrêt à une adresse que vous devez spécifier, et de lancer l'exécution jusqu'à ce que le programme l'atteigne.

Ctrl-K

Effacement des points d'arrêt

Cette commande permet d'effacer tous les points d'arrêt précédemment placés dans le programme.

Ctrl-A

Spécifie un point d'arrêt et exécute

Cette commande place un point d'arrêt simple sur l'instruction qui suit celle pointée par PC, et continue l'exécution. Ceci est particulièrement utile dans les boucles DBF si vous ne désirez pas tracer toute la boucle, mais juste voir le résultat à la fin de la boucle.

Ctrl-D

Point d'arrêt GEMDOS

Vous pouvez interrompre l'exécution de votre programme chaque fois qu'une fonction GEMDOS particulière sera appelée. Spécifiez le numéro de fonction à interrompre, ou entrez une ligne vide pour effacer tous points d'arrêt GEMDOS existants.

Historique

MonST possède une zone mémoire dans laquelle il stocke l'état de la machine lors des exceptions et des points d'arrêt. L'historique est également modifié lorsque vous exécutez pas-à-pas votre programme, et lorsque vous utilisez les différentes formes de la commande Exécution.

Note

L'historique ne se souvient que de cinq événements. Quand des événements supplémentaires surviennent, les plus anciens sont effacés.

H

Affichage de l'Historique

Cette commande affiche une fenêtre plein-écran affichant le contenu de l'historique. Pour chaque événement, la valeur de tous les registres - y compris PC - est affichée, ainsi que le désassemblage de l'instruction courante.

Note

Si un point d'arrêt est affiché dans la fenêtre d'Historique, la valeur entre crochet correspond à la valeur du point d'arrêt au moment où vous affichez l'historique, non au moment de l'événement.

Quitter MonST

Ctrl-C

Terminer le Programme

Cette commande termine le programme en cours. Si vous avez chargé un programme, Ctrl-C en terminera l'exécution et affichera le message `Programme Terminé`. Vous pouvez alors en charger un autre.

Elle permet de sortir directement de MonST si aucun programme n'a été chargé.

Si l'option `Déboguer` est utilisée depuis l'éditeur, cette commande terminera l'exécution du programme en cours et quittera MonST en même temps.

Note

Terminer prématurément un programme GEM sans faire le ménage correctement - fermer proprement les fenêtres, la station de travail, etc... - peut fortement embrouiller l'AES ou le VDI et ainsi bloquer la machine.

Chargement et Sauvegarde

Ctrl-L

Chargement de programme

Cette commande vous permet de charger un programme à déboguer en lui spécifiant une ligne de commande. Il n'est pas possible de charger deux programmes, vous devez terminer le premier pour pouvoir charger le deuxième.

Le fichier à charger doit être exécutable, sinon vous risquez de voir apparaître le message d'erreur TOS numéro 66. Si toutefois vous le faisiez par inadvertance, nous vous conseillons de quitter MonST et de le relancer afin d'éviter des problèmes.

Note

Cette commande n'est pas disponible dans la version résidente de MonST, ou si vous avez appelé MonST par l'option Déboguer de l'éditeur.

B

Chargement de fichier binaire

Cette option vous permet de charger un fichier binaire en mémoire à partir d'une adresse (optionnelle) que vous pouvez spécifier. Vous devez d'abord taper le nom du fichier et l'adresse séparée par une virgule. Si vous ne spécifiez pas d'adresse, MonST allouera de la mémoire au travers de GEMDOS pour y placer le fichier. L'adresse mémoire de début de fichier se trouvera alors dans la mémoire M0, et l'adresse de fin dans M1.

S

Sauvegarde de fichier binaire

Cette commande vous permet de sauver une zone mémoire sur disque. Vous devez tout d'abord spécifier le nom de fichier de sauvegarde, l'adresse mémoire de début et de fin (inclusivement) de bloc, le tout séparé par des virgules. Pour sauvegarder une zone mémoire préalablement chargé par la commande précédente, vous pouvez écrire directement la ligne <nom de fichier>,M0,M1 à condition, bien entendu, que vous n'ayez pas modifié entre-temps les valeurs de M0 et M1.

Cette commande vous permet de charger un texte ASCII, normalement un fichier source, pour le visualiser. Une nouvelle fenêtre sera ouverte dans laquelle vous pourrez visualiser tout le fichier en utilisant les touches curseur. La mémoire à réserver pour ce fichier est allouée par GEMDOS - assurez vous que vous disposez d'assez de place. Nous vous conseillons de charger le source *avant* de charger un programme pour que vous ayez assez de mémoire.

Low Res

Vous pouvez charger un texte source en basse résolution, mais pas le visualiser. Vous devez pour cela passer en moyenne résolution avec `Ctrl-O`, puis appuyer sur `Alt-S`, et deux fois `Alt-T`.

Note

Un fichier ASCII chargé après un programme sera considéré comme lui appartenant, et sera donc détruit lorsque le programme se terminera ainsi que les toutes fenêtres qui auront pu lui être associées. Comme la version résidante de MonST ne peut détecter cela, soyez prudent en chargeant un fichier source avec AMonST.

Exécuter des Programmes

Ctrl-R**Reprendre l'exécution / exécuter**

Cette commande exécute le programme en utilisant les valeurs courantes des registres. Normalement, c'est de cette façon que vous pouvez reprendre une exécution interrompue par un point d'arrêt.

Ctrl-Z**Pas-à-pas**

Cette commande exécute uniquement l'instruction pointée par PC en utilisant les valeurs courantes des registres. Les instructions *Trap*, *line A* et *line F* sont considérée par défaut comme une seule instruction, et le traitement correspondant ne sera pas exécuté en mode pas-à-pas. Vous pouvez modifier ce choix en changeant les Préférences.

Ctrl-Y**Pas-à-pas**

Identique à `Ctrl-Z`.

Identique à **Ctrl-Z**.

Cette commande est très similaire à **Ctrl-Z**, à la différence qu'elle ne continue pas le mode pas-à-pas dans les routines appelés par des instructions **BSR**, **JSR**, **TRAP**, *line A* et *line F*. Elle les exécute tout simplement, sans mode pas-à-pas, et rends la main au retour du sous-programme, c.a.d à l'instruction suivant l'appel. Elle fonctionne indifféremment en mémoire vive ou morte (*RAM & ROM*).

Cette commande permet de sauter - de ne pas exécuter - l'instruction courante. Elle peut se révéler très utile lorsque vous désirez ne pas exécuter une certaine partie de programme dont vous savez pertinemment qu'elle ne fonctionne pas.

Cette commande permet d'exécuter un programme de plusieurs façons différentes. Vous devez appuyer sur une deuxième touche pour spécifier le type d'exécution :

Cette commande est identique à **Ctrl-R** et exécute normalement un programme à partir des valeurs courantes des registres.

L'exécution sera effectuée lentement, chaque instruction exécutée sera décrite dans l'historique.

Similaire à la commande **Exécution - Lente**, mais une valeur numérique doit être spécifiée, correspondant au nombre d'instructions à exécuter avant de retourner à **MonST**.

Vous devez entrer une condition qui sera évaluée après chaque instruction. Le programme sera exécuté jusqu'à ce que la condition soit différente de zéro. Vous retournerez alors sous MonST.

Par exemple, si vous tracez pas-à-pas une boucle DBF en ROM qui utilise D6 comme compteur, utilisez la condition `D6&FFFF=FFFF` pour sortir de la boucle (attendre que le mot de poids faible de D6 soit égal à `FFFF`). Vous pourriez également spécifier `PC=FC8B1A` par exemple, pour que l'exécution s'arrête lorsque le PC atteindra l'adresse `FC8B1A`.

Note

Prenez garde de ne pas confondre cette commande avec la commande U (Exécution - Jusqu'à) qui place un point d'arrêt à une adresse donnée et exécute le programme depuis la valeur courante de PC.

Lorsque vous lancez l'une de ces commandes (excepté Exécution - Go), MonST vous proposera d'Observer Y/N ? le déroulement du programme. Choisir `Y` (Oui) force MonST à remettre à jour son écran d'affichage après l'exécution de chaque instruction, vous permettant ainsi d'observer de près les modifications apportées à la mémoire et aux registres, ou bien d'arrêter le programme en appuyant simultanément sur les deux touches `Shift`. A l'inverse, MonST ne fera pas cette mise à jour et laissera l'affichage du programme se faire normalement à l'écran si vous choisissez `N` (Non). Dans ce cas, vous devez appuyez sur `Shift-Alt-Help` pour interrompre l'exécution.

Note

Si vous sélectionnez le mode Observer et que vous n'avez pas validé la temporisation lors de l'échange d'écran, vos yeux risquent de ne pas supporter le changement d'affichage après chaque instruction, particulièrement si vous avez un moniteur couleur.

Lorsque l'une de ces commandes est utilisée (exceptée Exécution - Go), MonST va enregistrer les informations associées à chacune des instructions dans l'historique. De plus, les TRAPs seront traités comme une seule instruction, à moins que vous ne l'ayez modifié dans les Préférences (Voir paragraphe Préférences - Suivre les Traps).

Lorsque vous exécutez des programmes en mode S, I ou U sans le mode Observer, vous allez remarquer que quelques points clignent en haut et à gauche de l'écran. Ce clignotement est normal, car comme l'exécution est fortement ralenti dans ces modes, il vous permet de savoir que votre programme continue de tourner.

Recherche Mémoire

G

Recherche mémoire

Cette commande sert à rechercher une séquence particulière en mémoire. Elle commence par afficher le message Recherche pour B/W/L/T/I ? (B pour rechercher des séquences d'octets, W pour des mots de 16 bits, L pour des mots longs de 32 bits, T pour du texte et I pour des instructions).

Si vous sélectionnez B, W ou L, vous aurez à entrer les nombres à rechercher séparés par des virgules. Par défaut, MonST ne tient pas compte des alignements mémoire sur mots, et vous pouvez, par exemple, rechercher des mots non alignés (démarrant sur des adresses impaires). Cependant, si vous finissez la séquence à rechercher par ,W ou ,L, MonST ne regardera que les séquences démarrant sur des frontières de mot ou de mot long respectivement. Par exemple :

```
G W
 1234,W
```

permet de rechercher le nombre \$1234 uniquement sur des frontières de mots.

Si vous sélectionnez T, vous aurez à entrer la suite de caractères ASCII à rechercher. Par défaut, la recherche fera la distinction entre les majuscules et les minuscules, mais vous pouvez la désactiver.

Si vous sélectionnez I, vous aurez à entrer une instruction désassemblée, ou bien une partie d'instruction. Par exemple, si vous désirez rechercher la séquence \$14(A, vous pourrez trouver l'instruction MOVE.L D2,\$14(A0) comme l'une des réponses. Faites attention aux minuscules/majuscules (à la différence de MonST 1), au format du désassembleur (écrivez toujours les nombres en hexadécimal), et n'utilisez pas SP mais A7.

Lorsque vous aurez entré la séquence à trouver, MonSt lancera la recherche en passant le contrôle à la commande Recherche occurrence suivante décrite ci-dessous.

Cette commande ne peut être utilisée qu'après la commande G pour trouver l'occurrence suivante de la séquence à chercher. Avec les options B, W, L et T, vous trouverez toujours au moins une occurrence, celle où MonST stocke votre texte. Vous pouvez aussi trouver votre séquence Texte dans le tampon clavier. La touche Esc est testée tous les 64 Ko pour vous permettre d'interrompre la recherche excepté lorsque l'option L est spécifiée, auquel cas elle sera testée tous les deux octets (ce qui est beaucoup plus lent).

La zone de recherche commence à l'adresse 0 et s'étend jusqu'à la fin de la mémoire vive, puis recommence de \$FA0000 jusqu'à \$FEFFFF (cartouche et ROM système), puis recommence à 0.

La recherche commence juste après l'adresse de début de la fenêtre courante (sauf pour les fenêtres registres). Si une occurrence est trouvée, la fenêtre est réaffichée à l'adresse correspondante.

Recherche dans une fenêtre code source

Seule l'option T de la commande de recherche est disponible dans cette fenêtre.

Divers

Ctrl-P

Préférences

Cette commande vous permet de contrôler diverses options de MonST. Les trois premières demandent une réponse par Y ou N (*Oui ou Non*). Appuyez sur Esc ou Return pour ne pas les modifier.

Echange d'Ecran

Par défaut, cette option est sélectionnée. Elle permet d'indiquer à MonST de générer un délai de 20 millisecondes avant de changer d'écran lorsque votre programme est exécuté. Si votre programme modifie l'adresse d'écran ou la résolution, supprimez ce délai en invalidant cette option.

Par défaut, le mode pas-à-pas et les différentes options de la commande **R** considèrent les *Trap*, les appels *line A* et *line F* comme une seule instruction. En sélectionnant cette option, vous tracerez pas-à-pas les routines ROM associées à ces instructions.

Note

Attention : cette option doit être utilisée avec précautions. Certaines routines de lecture ou écriture sur disque ne fonctionneront pas correctement si elles sont exécutées en mode pas-à-pas car elles doivent être exécutées en un temps précis. Il est possible de perdre des données ainsi. D'un autre côté, il peut être amusant de voir comment l'AES affiche les menus déroulants ou affiche une fenêtre.

Vous pouvez interrompre une routine qui s'exécute en ROM en appuyant sur **Shift-Alt-Help**, et en reprendre l'exécution par **Ctrl-R**. Cependant, l'AES et le VDI utilisent tous deux des appels *Line A* et *Line F*, et il se peut que des registres d'état soient empilés avec le bit de *Trace* positionné. Reprendre l'exécution d'un programme exécuté en mode *Trace* peut générer des exceptions *Trace* apparemment trompeuses. De telles exceptions peuvent survenir jusqu'à ce que vous quittiez la ROM et retourniez à votre programme.

Cela produit un effet de bord qui peut provoquer un blocage de votre machine. Si vous tracez une routine de gestion d'événement AES, il se peut que des événements destinés à des accessoires de bureau soient créés à ce moment là, et qu'elles empilent un registre d'état avec le bit *Trace* positionné. Si votre programme se termine avant que l'accessoire n'ait pu traiter cet événement, une exception *Trace* sera générée après la sortie de **MonST**, et la machine se bloquera. Ce problème peut être contourné en utilisant la version résidante de **MonST**, ou le programme **NOTRACE.FRG**.

Le Programme NOTRACE

C'est un tout petit programme destiné à être mis dans le répertoire **AUTO** de votre disque de lancement (*boot*). Il forcera le système à ignorer les exceptions *Trace* au lieu d'afficher une ribambelle de bombes. Le code source de ce programme est fourni.

Déplacements Relatifs

Cette option, active par défaut, affecte le désassemblage des instructions utilisant le mode d'adressage indirect par registre d'adresse avec déplacement, c'est-à-dire `xxx(An)`. Quand cette option est activée, la valeur du registre est ajoutée au déplacement, et MonST cherche s'il existe un symbole ayant cette valeur. Si oui, l'instruction sera désassemblée sous la forme `symbole(An)`. Cette option est très utile pour certaines formes de programmation en assembleur, comme pour les langages de haut niveau qui utilisent un registre de base pour accéder à des données. Par exemple, le **HiSoft BASIC** utilise le registre A3 comme pointeur de base vers les variables systèmes.

Options concernant les symboles

Ces options permettent de modifier la façon dont MonST évalue les symboles dans des expressions. Vous pouvez demander à ce que MonST ne fasse pas la distinction entre les majuscules et les minuscules dans les symboles en répondant `Y (Oui)` à la question `Ignore MAJ/min`. Vous pouvez aussi modifier la longueur maximale des symboles qui est normalement de 22 caractères - vous pouvez réduire cette valeur jusqu'à 8.

Fin de Mémoire

Vous pouvez modifier l'adresse de fin de mémoire vive. Si vous avez un 1040 ST et désirez exécuter un programme comme sur un 520, changez cette valeur en `$78000`.

Sauver les Préférences

Vous pouvez enregistrer vos choix dans le fichier `KONST.INF` pour les conserver lorsque vous relancerez MonST.

Ctrl-E

Réinstallation des exceptions

Cette commande permet de réinstaller les routines normales de traitement d'exceptions. Ceci peut s'avérer très utile pour les programmes utilisant eux-mêmes les exceptions.

I

Copie intelligente

Cette commande permet de copier un bloc mémoire d'une zone vers une autre. Les adresses de début, de fin et de début de copie de bloc doivent être entrées de la manière suivante :

<début>,<fin_inclusive>,<destination>

La copie est intelligente dans le sens où le bloc de mémoire peut être déplacé vers une zone qui recouvre la zone d'origine.

Note

Aucun test n'est fait sur la validité des adresses. Si vous copiez un bloc vers des adresses invalides, vous pouvez bloquer votre ordinateur ou altérer des données systèmes.

L

Liste des symboles

Cette commande ouvre une fenêtre plein-écran et y affiche tous les symboles chargés. Appuyez sur une touche pour afficher la page suivante, ou sur Esc pour sortir. Les symboles sont affichés dans l'ordre où ils sont lus sur disque (ou en mémoire si vous utilisez l'option Déboguer de l'éditeur).

Si vous n'avez pas spécifié d'option de débogage lors de l'assemblage, cette commande n'aura aucun effet.

W

Initialisation Mémoire

Cette commande initialise une zone de mémoire à une valeur particulière. Les paramètres doivent être entrés ainsi :

<début>,<fin_inclusive>,<octet_d_initialisation>

Comme pour la commande l, aucun contrôle de validité des adresses n'est effectué.

P

Désassemblage vers l'imprimante/disque

Cette commande vous permet de désassembler une zone de mémoire vers l'imprimante, ou vers le disque. Le désassemblage peut se faire soit avec les symboles originaux (s'ils existent), soit en en créant automatiquement d'autres.

Cette commande va vous demander d'entrer plusieurs lignes de paramètres. La première va vous servir à spécifier les adresses de début et de fin de désassemblage ; elle doit être de la forme :

<adresse_début>,<adresse_fin>

La suivante va vous servir à spécifier les adresses de début et de fin d'une zone de travail qui va servir à la génération automatique des symboles. Si votre programme possède déjà sa propre table de symboles, laissez la ligne vide en appuyant directement sur `Return`, sinon, spécifiez les caractéristiques de cette zone en tapant :

```
<adresse_début>,<adresse_fin>
```

Ensuite, vous pourrez définir une ou plusieurs zones, dans la partie que vous désirez désassembler, qui devront être traitées comme des données, c.a.d traduit sous forme d'instructions DC. Vous aurez à entrer les caractéristiques de ces zones en suivant le format ci-dessous et en appuyant sur `Return` entre chaque ligne. Pour terminer la liste, laissez la ligne vide et appuyez sur `Return`.

```
<début_des_données>,<fin_des_données>[,<taille>]
```

Le champ optionnel `taille` sert à spécifier l'unité de taille des données. Ce ne peut être que l'une des trois lettres B, W, ou L (pour un octet, un mot, un mot long respectivement) sachant que L est pris par défaut.

Enfin, MonST vous demandera d'entrer un nom de fichier pour y stocker le désassemblage sur disque. Laissez la ligne vide et appuyez simplement sur `Return` pour le sortir sur imprimante.

Si MonST doit générer automatiquement des symboles, il peut s'écouler un délai avant que le désassemblage ne commence. Les symboles automatiques sont créés sous la forme `Lxxxxxxx` où `xxxxxxx` représente son adresse (en hexadécimal).

Sortie sur imprimante

Le désassemblage se présente sous la forme d'un nombre hexadécimal de huit chiffres, puis jusqu'à dix octets de données hexadécimales, un symbole éventuel de douze caractères maximum, puis le désassemblage lui-même. Vous pouvez interrompre le désassemblage en appuyant sur `Esc`.

Sortie sur disque

Le désassemblage produit un format directement compatible avec GenST. Chaque ligne est constituée d'un symbole éventuel, une tabulation, puis le désassemblage lui-même, avec une tabulation séparant l'instruction de son opérande. Utilisez l'option XREF si vous désassemblez une zone mémoire sans symboles, sinon aucun symbole n'apparaîtra dans le fichier de sortie. Le désassemblage peut être interrompu par un appui sur `Esc` ou par une erreur disque.

M

Adresse de début de fenêtre

Cette commande est équivalente à `Alt-A`. Elle est incluse par souci de compatibilité avec MonST 1.

O

Calcul d'Expression

Cette commande est équivalente à `Alt-O`. Elle est incluse par souci de compatibilité avec MonST 1.

D

L'unité de disque et du répertoire courant

Cette commande permet de modifier l'unité de disque et le répertoire courant, celui d'où vous chargez ou sauvez par défaut les fichiers avec MonST.

Version résidante de MonST

Décrivons maintenant `AMONST2.PRG`, la version résidante de MonST. Ce programme doit être placé dans le répertoire `AUTO` de votre disque de lancement (*boot*), il sera ainsi toujours lancé au démarrage de la machine.

Une fois initialisé, cette version de MonST devient inactive en laissant normalement se dérouler la phase de lancement, mais par contre, elle est prête à être activée si une exception survient, comme par exemple une erreur d'adressage. Elle est surtout destinée aux programmeurs d'accessoires et de programmes `AUTO`. Si besoin est, vous pouvez placer une instruction illégale (par exemple `ILLEGAL`) en début du programme à déboguer, de sorte que `AMonST` puisse prendre la main au moment où ce programme sera exécuté par la phase de lancement de la machine.

Il est également possible de lancer `AMonST` depuis le bureau en double-cliquant sur son icône s'il n'est pas déjà résidant. Il s'initialisera de la même manière que s'il était placé dans le dossier `AUTO`.

Une fois activée, la version résidante de MonST est tout à fait similaire à la version normale. Les différences viennent des symboles qui ne peuvent être chargés, et des variables de la page de base qui sont mises à zéro puisqu'elles ne peuvent pas être connues. Lorsque vous appuyez sur `Ctrl-C` pour quitter `AMonST`, il reste présent en mémoire.

N'importe quel programme peut être interrompu en appuyant sur `Shift-Alt-Help` lorsque `AMonST` est résidant.

La seule façon de supprimer AMonST lorsqu'il est installé est de le désactiver du dossier AUTO (en modifiant l'extension .PRG en .PR ou en l'effaçant par exemple) puis de réinitialiser (*rebooter*) la machine.

Il est possible de charger la version normale de MonST lorsque AMonST est résidant, si la mémoire le permet. MonST peut alors être utilisé normalement, et la version résidante est ignorée jusqu'à ce que vous quittiez MonST.

Note

Ne lancez AMonST que depuis le bureau GEM. Si par exemple vous utilisez l'option Exécuter un Programme de GenST, des zones de mémoire système seront bloquées et inutilisables jusqu'à une réinitialisation de la machine.

Si vous maintenez appuyées les deux touches *Shift* pendant l'installation de la version résidante de MonST, le débogueur sera activé immédiatement. Vous pourrez alors examiner la mémoire ou positionner des points d'arrêt GEMDOS. Quittez alors le débogueur par *Ctrl-C*. Il restera résidant.

Résumé des commandes

Commandes Agissant sur les Fenêtres

Alt-A..... Adresse de début de fenêtre
Alt-B..... Adresse de point d'arrêt
Alt-E..... Edition d'une fenêtre
Alt-F..... Taille des caractères
Alt-L..... Verrouillage d'une fenêtre
Alt-O..... Calcul d'expression
Alt-P..... Recopie d'écran
Alt-R..... Modification de registre
Alt-S..... Division d'une fenêtre
Alt-T..... Changement de Type
S Alt-Z..... Zoom

Echange d'écran

V..... Visualisation de l'écran du programme
Ctrl-O..... Changement de la résolution de l'écran

Points d'Arrêt

Alt-B.....Spécifier un point d'arrêt
Help.....Affiche diverses informations et les points d'arrêt
Ctrl-B.....Spécifier un point d'arrêt
U.....Exécuter jusqu'à
Ctrl-K.....Effacement des points d'arrêt
Ctrl-A.....Spécifie un point d'arrêt et exécute
Ctrl-D.....Point d'arrêt GEMDOS

Chargement et Sauvegarde

Ctrl-L.....Chargement de programme
B.....Chargement de fichier binaire
S.....Sauvegarde de fichier binaire
A.....Chargement de fichier ASCII

Exécuter des Programmes

Ctrl-R.....Reprendre l'exécution / exécuter
Ctrl-Z.....Pas-à-pas
Ctrl-Y.....Pas-à-pas
Ctrl-W.....Pas-à-pas
Ctrl-T.....Interprétation d'une instruction (Trace)
Ctrl-S.....Sauter l'instruction courante
R.....Exécution

Recherche en Mémoire

G.....Recherche mémoire
N.....Recherche occurrence suivante

Divers

Ctrl-C.....Terminer le Programme
Ctrl-P.....Préférences
Ctrl-E.....Reinstallation des exceptions
I.....Copie intelligente
L.....Liste des symboles
W.....Initialisation mémoire
F.....Désassemblage vers l'imprimante/disque
M.....Adresse de début de fenêtre
O.....Calcul d'expression
D.....Modification de l'unité de disque et du répertoire courant
Shift-Alt-Help Interruption de programme
H.....Affichage de l'historique

Comment déboguer efficacement

Trucs et Astuces

Si vous avez interrompu l'exécution d'un programme par `Shift-Alt-#elp`, ou par la commande Exécuter Jusqu'à, et que vous vous retrouvez au milieu de la ROM, il existe un moyen simple de retourner à l'endroit où votre programme a appelé la routine en ROM. Assurez vous d'abord que l'option Tracer l'instruction Trap soit active, puis utilisez la commande Exécution - Jusqu'à avec comme condition d'arrêt `sp=a7`. Vous vous retrouverez alors forcément dans votre programme lorsque le programme repassera en mode utilisateur (à priori, juste après qu'il soit sorti de la routine ROM).

Si vous êtes dans un sous-programme qui ne vous intéresse pas, mais désiriez reprendre la main au retour de ce sous programme, utilisez la commande U qui positionne un point d'arrêt simple à une adresse spécifiée. Entrez l'expression `{sp}`, et un point d'arrêt sera placé à l'adresse de retour. Si la routine a rangé des données sur la pile, ou utilise un pointeur de pile local (cas des langages compilés), alors essayez Exécution - Jusqu'à (R-U) avec comme condition `{pc}.w=4E75` qui exécutera lentement votre programme jusqu'à ce qu'une instruction RTS soit rencontrée. Cela ne fonctionnera pas si la routine appelle un sous programme. Vous pouvez alors ajouter une condition supplémentaire comme par exemple `{:pc}.w=4E75)&(sp>xxx)` où `xxx` est la valeur du pointeur de pile moins 1.

Quand vous utilisez la commande Exécution - Jusqu'à, et savez que cela va prendre du temps, vous pouvez accélérer l'exécution en pré-calculant l'expression. Par exemple :

```
{a3>{3A400-\100+M1}}
```

peut être réduit à :

```
a3>xxx
```

où `xxx` est la valeur équivalente calculée pour vous par la commande `Alt-O`.

Ligne de Commande de MonST

Si vous lancez MonST depuis un environnement (*shell*) texte, vous pouvez lui passer des paramètres. Ils seront interprétés comme le nom du programme à charger suivi de ses propres paramètres.

Recherche des bogues

Il y a probablement autant de manière de chercher des bogues qu'il existe de programmeurs ! Il n'existe, cependant, qu'une seule manière de l'apprendre, c'est l'expérience, et ce n'est pas facile. Toutefois, voici quelques trucs que nous avons appris, à force d'expérience !

Tout d'abord, le meilleur moyen de trouver des erreurs dans un programme est de regarder son source et de réfléchir. Utiliser d'abord le débogueur, et ensuite regarder le source est une mauvaise habitude. En effet, vous n'aurez pas toujours à votre disposition ce type d'outils de débogage si vous êtes amenés à changer de machine et d'environnement.

Il est plus facile de corriger un programme qui bloque en générant des bombes, qu'un autre qui, à certains moments, ne fonctionne pas exactement comme il le devrait.

L'écrasement de données en mémoire est source de nombreux bogues. Lorsque cet écrasement est détectable, car il produit une erreur bus par exemple, un point d'arrêt conditionnel placé dans une ou plusieurs routines importantes peut grandement vous aider. Supposons que la variable globale `pointeur` devienne impaire pendant l'exécution, l'expression conditionnelle du point d'arrêt pourra être de la forme :

```
{pointeur;&1
```

Cependant, si cette méthode échoue et que ce pointeur viennent à être modifié de façon indétectable, il vous reste la solution de lancer l'exécution du programme par la commande Exécution - Jusqu'à ce que la condition précédente soit vrai. Le problème est que cela peut prendre un temps considérable. Et il n'est pas sûr que par cette méthode vous arriviez à trouver l'erreur, car elle peut être le fruit, par exemple, de l'exécution d'une routine en interruption survenant au moment où la pile se trouve dans une configuration bien précise.

Les *points d'arrêt compteur* sont une bonne façon de traquer les bogues avant qu'ils n'apparaissent. Par exemple, supposons qu'une routine soit soupçonnée ne pas se comporter comme elle devrait. Placez y un point d'arrêt compteur, et exécutez votre programme. Lorsque l'exécution stoppe, peut-être à cause d'une exception, examinez la valeur du compteur en appuyant sur `Help`. Terminez le programme et rechargez le. Placez au même endroit un point d'arrêt stop en spécifiant la valeur de l'ancien point d'arrêt. Lancez le programme. Vous pouvez maintenant examiner votre routine juste avant que l'exception ne soit générée.

Bon courage !

Les programmes du dossier AUTO

Si un tel programme se coince pendant l'initialisation, utilisez AMONST pour intercepter l'exception, ou ajoutez une instruction ILLEGAL au début de votre programme. Le programme AMONST doit être positionné dans le répertoire avant le programme à déboguer.

Les accessoires de bureau

Utilisez également AMONST pour déboguer les accessoires qui ne fonctionnent pas normalement. Pour trouver où votre accessoire est placé en mémoire, appuyez sur les touches Shift-Alt-Help pour entrer dans le débogueur. Recherchez en mémoire le nom en majuscules de votre accessoire, éventuellement complété à huit caractères par des espaces, en commençant à l'adresse 0. Ignorez les occurrences qui se trouvent dans des tampons de répertoire (le nom sera suivi de .ACC). Ignorez également l'occurrence qui se trouve dans le tampon de recherche de MonST (le nom sera précédé du caractère ASCII T). Dès que vous aurez trouvé la bonne chaîne de caractères en mémoire, examinez la valeur du pointeur long qui se trouve 12 octets après la chaîne de caractères. Cette valeur représente l'adresse de la page de base de votre accessoire - qui commence réellement 256 octets plus loin. En examinant cette zone mémoire, vous devez bien être capable de trouver la boucle principale et d'y placer un point d'arrêt au bon endroit. Appuyez alors sur Ctrl-R pour reprendre l'exécution suite à l'exception. MonST sera de nouveau activé quand le point d'arrêt sera atteint.

Si un accessoire comporte un bogue dans sa phase d'initialisation, vous devez interrompre son exécution tout au début, avant qu'il ne puisse faire quoi que ce soit. Réassemblez votre accessoire en ajoutant tout au début l'instruction ILLEGAL, et laissez AMONST récupérer cette exception.

Mais ceci n'est pas toujours possible. Nous décrivons ci-après une méthode destinée à stopper l'AES juste avant qu'il n'exécute votre programme. Cette méthode est assez compliquée et n'est pas recommandée aux débutants.

D'abord maintenez enfoncées les deux touches Shift pendant la phase de lancement (*boot*) pour entrer dans MonST. Positionnez un point d'arrêt sur la fonction GEMDOS \$3D (Fopen), puis appuyez sur Ctrl-C pour continuer la phase de lancement.

Vous réentrez dans MonST chaque fois qu'un fichier sera ouvert. Pour déterminer si le fichier qui est ouvert est celui qu'il convient, activez la fenêtre 3 et modifiez son adresse en spécifiant `:sp+2`. Si le nom affiché n'est pas celui de votre accessoire, appuyez sur `Ctrl-Z` pour exécuter l'ouverture du fichier, repositionnez un point d'arrêt sur la fonction GEMDOS \$3D, et appuyez sur `Ctrl-R` pour continuer. Si le nom est celui de votre accessoire, spécifiez un point d'arrêt sur la fonction GEMDOS \$4B et appuyez sur `Ctrl-R`. MonST sera réactivé juste avant le chargement de votre accessoire. Appuyez sur `Ctrl-Z` pour le charger. Positionnez un point d'arrêt par `Alt-B` à l'adresse `D0+100` qui représente l'adresse de la première instruction. Appuyez une dernière fois sur `Ctrl-R`, et lorsque MonST apparaîtra, le compteur de programme (PC) pointerà sur la première instruction de votre accessoire. Cette méthode est longue mais c'est bien souvent la seule possible pour trouver des bogues dans des accessoires.

Analyse des exceptions

Lorsque une exception survient dans un programme, il est très utile de savoir où et pourquoi elle s'est présentée, et éventuellement comment reprendre l'exécution.

Erreur bus

Si le compteur de programme (PC) pointe sur une zone mémoire inexistante, regardez dans la pile pour essayer d'y voir un indice. Il s'agit probablement de valeurs empilées qui n'ont pas été dépilées. Si le PC se trouve dans une zone de mémoire existante, l'erreur bus doit être causée par un accès à une adresse illégale ou protégée. Il n'est généralement pas possible de continuer l'exécution après une erreur de bus.

Erreur d'adresse

Si le PC contient une valeur étrange, la méthode ci-dessus peut être appliquée. Sinon, l'erreur a dû être causée par un accès à une adresse impaire. Corriger la valeur d'un registre peut être suffisant pour continuer l'exécution, du moins temporairement.

Instruction illégale

Si le compteur de programme (PC) pointe vers la mémoire basse, en dessous de l'adresse \$30, il est probable que l'exception ait été causée par un saut à l'adresse 0. Si vous examinez cette zone mémoire avec MonST, vous pourrez y voir un branchement court accompagné d'instruction ORI (en fait des pointeurs longs), et éventuellement d'une instruction illégale.

Violation de privilège

Cette exception est causée par une instruction privilégiée utilisée en mode utilisateur. En général, cela signifie que votre programme est complètement bloqué. Modifier la valeur de PC pour lui faire sauter l'instruction en cause n'apportera pas grand chose.

CHAPITRE 5

L'Editeur de liens

Introduction

Le travail d'un éditeur de liens est de combiner un ou plusieurs fichiers générés par GenST ou par un compilateur, afin de créer un seul fichier exécutable. Une de ses caractéristiques les plus intéressantes est la recherche de fonctions dans des bibliothèques. L'éditeur de liens examinera les bibliothèques et n'en extraira que les fonctions nécessaires limitant au maximum la taille du fichier exécutable.

Il existe malheureusement deux formats de fichiers objets sur ST. Ces sont les formats GST et DRI. GenST peut créer des fichiers de ces deux types, mais LinkST ne reconnaît que les fichiers GST. Si vous désirez chaîner des fichiers DRI, utilisez le programme Atari ALN, ou les programmes Digital Research LINK68 et RELMOD.

LinkST ne peut chaîner que des fichiers au format GST.

Note 

Lancer l'éditeur de liens

La manière la plus simple de lancer l'éditeur de liens depuis le bureau est de cliquer sur l'icône LINKST.TTP, et d'entrer une ligne de commande appropriée. Il existe une autre façon d'utiliser l'éditeur de liens en utilisant un fichier de contrôle contenant les options désirées.

La ligne de commande indique à l'éditeur de liens les informations nécessaires pour lire les fichiers au format GST et produire le programme exécutable.

Ligne de Commande

La ligne de commande doit être de la forme :

<nom_de_fichier> <-options> [nom_de_fichier] [-options]

Les options sont constituées d'un tiret - suivi d'une lettre. Les options disponibles sont :

- B Crée une vraie section BSS pour les sections de ce nom.
- D Déboguage. Les noms des symboles seront inclus en utilisant le standard DRI, soit huit caractères par identificateur (pour GenST ou d'autres débogueurs)
- F Force l'exécution de la deuxième passe de l'édition de liens lorsque des erreurs sont survenues pendant la première.
- L Les noms de fichier suivants cette option sont des bibliothèques.
- M Crée un fichier qui contiendra, dans l'ordre, la liste des sections et des symboles. Ce fichier portera le nom du fichier principal avec l'extension `.MAP`.
- O Le nom de fichier suivant cette option est le nom du programme généré. Un espace peut être placé entre cette option et le nom du programme
- Q Supprime la pause lorsque l'édition de liens est terminé.
- S Crée une table des symboles dans un fichier de même nom que le fichier principal, mais comportant l'extension `.SYM`.
- U Passe en majuscule tous les symboles importés ou exportés.
- X Déboguage étendu. Utilise le format de déboguage étendu d'Hisoft.
- W Spécifie un fichier de contrôle. Son extension est `.LNK` par défaut.

En général, tous les noms de fichiers donnés sont des fichiers en entrée, c'est-à-dire qu'ils sont lus par l'éditeur de liens. Ils ont par défaut l'extension `.BIN`. Si un fichier d'extension `.LNK` est spécifié, il sera considéré comme fichier de contrôle. Les noms de fichiers suivant l'option `-L` doivent être des bibliothèques.

Le nom du programme généré peut être spécifié après l'option `-O`, ou en utilisant la directive `OUTPUT` dans le fichier de contrôle. Si plusieurs noms sont spécifiés, seuls le plus récent sera pris en compte. S'il n'y en a pas, le premier nom de fichier spécifié sur la ligne de commande est repris avec l'extension `.PRG`.

Exemples de Lignes de Commande

```
PARTIE1 PARTIE2 -D
```

Lit les fichiers `PARTIE1.BIN` et `PARTIE2.BIN` comme fichiers d'entrée, et crée le fichier `PARTIE1.PRG` avec des informations de débogage.

```
PARTIE1 PARTIE2 -O TEST.PRG
```

Lit les fichiers `PARTIE1.BIN` et `PARTIE2.BIN` comme fichiers d'entrée, et crée le fichier `TEST.PRG`.

```
-O TEST.TOS DEBUT -L MABIB -S
```

Lit le fichier `DEBUT.BIN` comme fichier d'entrée, lit sélectivement la bibliothèque `MABIB.BIN`, et crée le fichier `TEST.TOS`. Un fichier contenant la liste des symboles utilisés sera créé sous le nom `TEST.SYM`.

L'exécution de LinkST

LinkST effectue son travail en deux passes. Dans la première, il crée une table des symboles de toutes les sections et modules. Le fichier objet est créé dans la seconde. Quand l'exécution commence, un message d'accueil est affiché ainsi que la liste des fichiers lus pendant les deux passes. Cela vous donne une idée des tâches réalisées et du temps qu'elles prennent ainsi que de l'endroit exact où d'éventuelles erreurs surviennent.

S'il y a assez de mémoire à la fin de la première passe, LinkST utilisera une mémoire cache pour stocker le fichier objet. Cette opération accélère grandement l'édition de liens. Si le cache est utilisé, le fichier n'est écrit sur disque qu'à la fin de la deuxième passe, en même temps que le nombre d'erreurs est affiché.

Lorsque l'édition de liens est terminé, vous devez appuyer sur une touche avant de quitter le programme. Vous pouvez ainsi prendre le temps de lire les messages affichés avant de revenir au bureau GEM. Vous pouvez supprimer cette pause en utilisant l'option `-q` particulièrement utile si vous utilisez un environnement (*shell*) texte ou un fichier *batch*.

LinkST a été écrit dans le but de travailler le plus rapidement possible. Evidemment, les accès disque ralentissent énormément le processus. Si vous ne possédez pas de disque dur, nous vous recommandons d'utiliser un *ram-disk*, mais laissez assez de mémoire libre pour que l'éditeur de liens puisse utiliser de la mémoire cache pour le fichier exécutable. Nous vous recommandons de mettre en priorité sur votre *ram-disk* les petites bibliothèques et les fichiers lus par LinkST.

Les messages d'erreur et d'avertissement sont affichés sur l'écran. Vous pouvez suspendre momentanément leur affichage en appuyant sur `Ctrl-S` et le reprendre par `Ctrl-Q`. Appuyez sur `Ctrl-C` pour quitter l'éditeur de liens et revenir au bureau GEM.

Vous pouvez envoyer les messages d'erreur et d'avertissement dans un fichier `NOM.TXT` en commençant la ligne de commande par :

`>NOM.TXT`

ou directement sur l'imprimante en commençant la ligne de commande par :

`>PRN :` (port parallèle) ou `>AUX :` (port série)

Si vous redirigez la sortie des messages d'erreur, vous devez spécifier l'option `-q` puisque vous ne verrez pas à l'écran le message vous demandant d'appuyer sur une touche.

Fichiers de Contrôle

Si vous en avez assez de taper au clavier une longue ligne de commande pour LinkST, ou bien si la ligne sur laquelle vous la tapez est trop courte pour accepter toutes vos options, vous pouvez créer un fichier de contrôle dans lequel vous spécifierez les options d'édition de liens ainsi que tous les fichiers que celui-ci doit charger.

L'extension généralement utilisée pour un tel fichier est `.LNK`. Vous pouvez créer ce fichier avec l'éditeur de MonST, mais n'essayez pas de l'assembler ! Pour indiquer à l'éditeur de liens que vous utilisez un fichier de contrôle, utilisez l'option `-w` (pour *with* - avec - dans la langue de Shakespeare) suivi du nom de fichier :

`-w nom.lnk`

Un fichier de contrôle est constitué d'une suite de directives. Les lignes vides sont ignorées, et il est possible d'inclure des lignes de commentaire en les faisant commencer par l'un de ces trois caractères `*`, `;` ou `!`. Voici les directives que vous pouvez placer dans ce type de fichier :

INPUT <nom_de_fichier>

Cette directive spécifie le nom du fichier objet que l'éditeur de liens doit lire. L'extension utilisée par défaut est `.31M` si elle n'est pas spécifiée.

OUTPUT <nom_de_fichier>

Cette directive permet de donner le nom du fichier exécutable généré. Il n'y a pas d'extension par défaut, vous devez la spécifier.

Pour ces deux directives, il est possible de spécifier un astérisque (*) à la place du nom de fichier. Dans ce cas, ce caractère sera remplacé par le premier nom de fichier de la ligne de commande. Cela peut servir à avoir un fichier de contrôle générique pour éditer les liens de programmes C par exemple.

LIBRARY <nom_de_fichier>

Cette directive spécifie le nom du fichier qui doit être utilisé comme bibliothèque. L'extension utilisée par défaut est `.BIN` si elle n'est pas spécifiée.

SECTION <nom_de_section>

Vous pouvez, en répétant cette directive, classer les sections dans un ordre précis.

DEBUG

Tous les noms de symboles inclus dans les fichiers `.BIN` sont recopiés dans le fichier exécutable. Cela vous permettra de déboguer plus facilement votre programme avec MonST.

XDEBUG

Cette directive est similaire à `DEBUG`, sauf qu'elle force l'éditeur de liens à utiliser le format de *Débogage Etendu HiSoft* qui autorise jusqu'à 22 caractères significatifs pour les symboles.

DATA taille(K)

Fixe la taille du segment `BSS`. Il est possible de spécifier la taille en octets ou en Ko (1024 octets). Cette option est particulièrement utile pour des compilateurs qui, comme le *Prospero Pascal*, stockent leurs variables dans ce segment.

BSS <nom_de_section>

Cette directive permet d'inclure la section dont le nom est spécifié dans le segment `BSS`. Cette section ne doit pas contenir de données initialisées ni d'instructions, sinon un message d'erreur sera généré. Cette directive ne doit pas être utilisée avec la directive `DATA`.

UPPER

Cette directive force l'éditeur de liens à passer en majuscule tous les symboles importés ou exportés.

TRUNCATE

Cette directive force l'éditeur de liens à tronquer les symboles à 8 caractères, ce qui est quelque fois utile lorsque vous avez à chaîner des modules assembleurs où les symboles sont long avec des modules de langage évolués où les symboles sont court.

Exemple de fichier de contrôle :

```
* Fichier de controle pour editer les liens d'un
programme C
INPUT STARTUP
INPUT *
XDEBUG
LIBRARY CLIB
```

En supposant que ce fichier de contrôle s'appelle `CPRG.LNK`, vous chaînez un fichier C de nom `TEST.C` avec la ligne de commande :

```
TEST -W CPRG
```

Les fichiers `STARTUP.BIN`, `TEST.BIN` seront lus. Les fonctions de la bibliothèque C appelées dans ces deux fichiers seront lues dans la bibliothèque `CLIB.BIN`. Un fichier programme `TEST.PRG` sera généré, puisque aucun nom n'a été donné. Des informations de débogage étendu seront ajoutées à ce fichier.

Si vous ne spécifiez pas de lecteur de disquette ou de chemin d'accès dans le fichier de contrôle ou sur la ligne de commande, les fichiers seront lus dans l'unité de disque et le répertoire par défaut. Si vous lancez LinkST depuis le bureau, le répertoire par défaut est celui dans lequel vous avez lancé LinkST. Si vous utilisez un environnement (*shell*) texte ou si vous lancez LinkST depuis l'éditeur de GenST, le répertoire par défaut peut être différent.

Lancer LinkST Automatiquement

Vous pouvez installer LinkST de manière à le lancer en double-cliquant depuis le bureau GEM sur un fichier `.LNK`. Utilisez l'option Installer une Application du bureau. Procédez de la même manière que pour GenST, mais spécifiez un type TOS avec paramètres avec `.LNK` comme extension.

Avertissements de LinkST

Les avertissements sont des messages indiquant que quelque chose pourrait être erronée, mais que ce n'est pas très grave en général.

le symbole "x" a déjà été défini

Le symbole est défini deux fois. Par exemple, Cela peut arriver si vous remplacez une fonction d'un module par la vôtre. L'éditeur de liens utilisera la première définition qu'il trouvera, et affichera cet avertissement pour la seconde.

le nom de module est trop long

Un nom de module ne doit pas excéder 80 caractères.

le commentaire est trop long

Les commentaires sont limités à 80 caractères.

des sections absolues se chevauchent

Deux sections absolues se recouvrent.

la section "x" n'est ni COMMON, ni SECTION

Un nom de section a été utilisé sans son type ait été défini. Il doit être soit SECTION soit COMMON.

Messages d'erreur de LinkST

Il existe quatre types de messages d'erreur : les erreurs générales, les erreurs d'entrée-sortie, les erreurs sur fichiers binaires, et les erreurs de l'éditeur de liens. Dans certains messages est inclus une chaîne de caractères que nous avons représentée par "x". Dans d'autres est inclus un nombre, que nous avons représenté ci-dessous par 99.

Les erreurs générales

symbole "x" non résolu dans le fichier "x"

Un symbole a été utilisé dans un fichier sans y avoir été défini. Le symbole peut être utilisé dans d'autres fichiers... cela vous donne quand même une piste !

XREF (utilisant "x") tronqué dans le module "x"

Une valeur définie par XREF est trop grande pour l'usage qui en est fait.
Cela peut venir d'un BSR à un symbole externe.

ligne de contrôle invalide "x"

Erreur de syntaxe dans une ligne du fichier de contrôle

donnée non-nulle dans la section BSS

La section BSS ne peut contenir que des données non initialisées.

Les erreurs d'entrée-sortie

fichier "x" non trouvé

impossible d'ouvrir le fichier de sortie "x"

impossible d'ouvrir le fichier map "x"

impossible d'ouvrir le fichier de symboles "x"

impossible d'ouvrir le fichier source "x"

erreur de lecture 99 dans le fichier "x"

erreur d'écriture disque

le nom de fichier "x" est trop long

Les erreurs sur fichiers binaires

Ces erreurs surviennent quand le format du fichier binaire en entrée est incorrect. Elles ne doivent normalement pas apparaître. Elles signifient que l'assembleur ou le compilateur a produit un code incorrect.

L'erreur :

directive SOURCE absente

peut survenir en essayant de chaîner un fichier qui n'est pas au format GST, comme par exemple un fichier DRI.

seules les valeurs LONGUES sont relogeable à l'exécution
tentative de redéfinition de l'identificateur du symbole "x"

mauvais code opération 0x99 dans une directive XREF

directive SOURCE mal placée

mauvaise directive 99

<id> 99 non défini comme une section mais utilisé comme tel

tentative de réutilisation de l'<id> 99 comme id de section

tentative de réutilisation de "x" comme nom de section

section COMMON mais utilisée comme si elle ne l'était pas

SECTION confondue avec COMMON

fin inattendu du fichier source

Appendice A

Codes d'Erreur GEMDOS

Cet Appendice détaille la signification des codes d'erreur GEMDOS. Les numéros utilisés ici sont ceux affichés par MonST et GenST. Ces valeurs seront négatives si vous appelez GEMDOS depuis vos propres programmes.

0	OK (pas d'erreur)	32	Numéro de fonction invalide
1	Erreur fondamentale	33	Fichier non trouvé
2	Lecteur pas prêt	34	Chemin d'accès non trouvé
3	Commande inconnue	35	Trop de fichiers ouverts
4	Erreur CRC	36	Accès non permis
5	Mauvaise requête	37	Handle invalide
6	Erreur de recherche	39	Mémoire insuffisante
7	Média inconnu	40	Adresse de bloc mémoire invalide
8	Secteur non trouvé	46	Unité disque invalide
9	Plus de papier	49	Plus de fichiers
10	Erreur en écriture	50	Disque plein (erreur MonST)
11	Erreur en lecture	64	Erreur de champ
12	Panne générale	65	Erreur interne
13	Disque protégé en écriture	66	Erreur de format dans un programme chargé
14	Disque changé	67	Erreur Setblock due à une restriction de taille
15	Périphérique inconnu		
16	Mauvais secteurs en formatage		
17	Insérer un autre disque		

Appendice B

Messages d'erreur GenST

GenST possède de nombreux messages d'erreur dont la plupart s'expliquent d'eux-mêmes. Cet Appendice en décrit la liste par ordre alphabétique, avec des explications pour les messages qui en nécessitent.

Etant donné que GenST est continuellement amélioré, cette liste peut partiellement ne pas correspondre avec les messages générés par votre programme. Il peut y avoir des messages non documentés ici.

Erreurs

Si des messages commencent par INTERNAL, signalez-le nous. Vous ne devriez normalement pas voir de tels messages.

.W ou .L exigé comme taille d'index

adresse impaire

parenthèse fermante manquante

apostrophe absente

code exécutable uniquement

seul du code exécutable peut être assemblé en mémoire.

combinaison de type illicite

condition erronée

Il y a plus de ENDC que de IF dans une macro.

conversion numérique invalide

Le symbole n'est pas défini ou est relatif ou bien c'est du à une division par zéro

donnée immédiate exigée

donnée trop large

ENDC absent

il y a plus de IF que de ENDC.

erreur de lecture fichier en-tête

erreur de syntaxe.

erreur pendant la sortie du listing

le listing sera interrompu à l'apparition du message.

erreur pendant l'écriture du fichier binaire

en principe, parce que le disque est plein.

l'expression absolue DOIT être évaluée

erreur utilisateur

causé par une directive FAIL.

expression incorrecte

normalement, c'est une erreur de syntaxe dans l'expression

expression IF invalide, ignorée

fichier en-tête répété

fichier introuvable

ignore instruction suivante

impossible de créer le fichier binaire

Il se peut que le nom de fichier soit erroné, ou que le disque soit protégé en écriture, etc...

impossible d'exporter le symbole

impossible d'imbriquer des MACROS ou des REPTS

impossible d'imbriquer des structures de répétitions

impossible d'importer le symbole

instruction BSR.S illicite

Changez le en BSR.

instruction non reconnue

ligne mal construite
liste de registre invalide
mauvais ENDC
mauvais ENDM ou MEXIT
mauvais ENDR
mauvais processeur
mode absolu non autorisé
mode d'adressage interdit
mode d'adressage inconnu
mode d'adressage invalide dans MOVEP
mode relatif non autorisé
nom de MODULE dupliqué
nom de section invalide, TEXT pris par défaut
nombre invalide
nombre trop grand
l'option doit se trouver au début
option invalide
ORG/RORG interdit
paramètre de FORMAT invalide
paramètre d'imprimante invalide
pas assez de mémoire
pas encore implanté
référence avant
registre d'adresse exigé
registre de données exigé
registre exigé

les sections BSS et OFFSET ne peuvent contenir des données

Les sections OFFSET et BSS ne peuvent que contenir des directives DS.

relogement non autorisé

restrictions dû au format de l'éditeur de liens

le format.DRI est très restrictif sur la façon dont il autorise les imports.

symbole défini 2 fois

symbole exigé

symbole importé interdit

symbole non défini

symbole supplémentaire trouvé en 2ème passe

un symbole est apparu pendant la deuxième passe alors qu'il n'y était pas lors de la première.

taille invalide

tampon programme plein

Modifiez la taille mémoire allouée à l'assemblage

utilisation incorrecte du symbole

variable locale interdite

virgule exigée

XREF interdit entre parenthèses

Messages d'avertissements

branchement ligne suivante converti en NOP

déplacement enlevé

xx(An) a été transformé en (An) par l'optimisation.

déplacement invalide dans l'instruction LINK

c.a.d négatif ou impair.

directive ignorée

extension du signe de l'opérande

une donnée dans un MOVEQ a nécessité une extension de signe pour que ce soit correct.

instruction 68010, convertie en MOVE SR

MOVE CCR n'est pas une instruction 68000.

la taille doit être .W

transformé en branchement court

par l'optimisation.

Appendice C

Organisation mémoire du ST

Cet appendice détaille certaines parties concernant l'organisation mémoire du ST :

1. Zone des exceptions
2. Page de base
3. Organisation mémoire matériel (*hardware*)

Zone des Exceptions

Quand une exception survient (en affichant des champignons atomiques ou des bombes), le ST enregistre une copie de l'état du microprocesseur dans une zone de mémoire qui n'est pas effacée par une réinitialisation de la machine. Après une exception, vous pouvez donc charger MonST et examiner la section de mémoire correspondante pour essayer de voir ce qui s'est passé. Si des exceptions arrivent souvent, utilisez plutôt la version résidante de MonST pour avoir une meilleure idée de ce qui s'est passé.

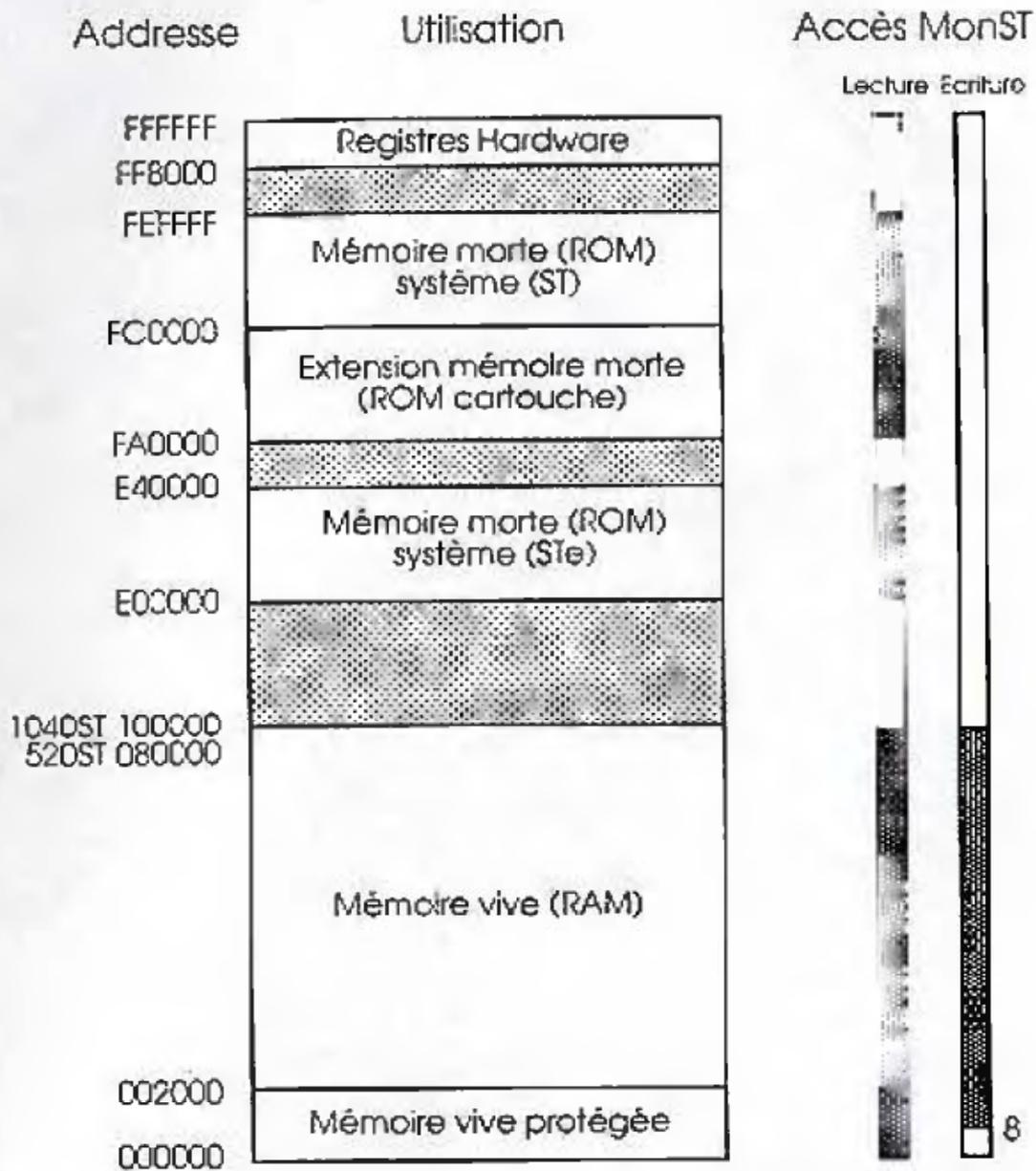
\$380	long	contient \$12345678 si valide
\$384	8 longs	valeur des registres D0-D7 au moment de l'exception
\$3A4	8 longs	valeur des registres A0-A7 (SSP) au moment de l'exception
\$3C4	octet	numéro d'exception
\$3C8	long	valeur du registre USP
\$3CC	16 mots	copiés à partir de SSP

Page de Base

Tout programme s'exécutant sous GEMDOS possède une page de base de 256 (\$100) octets de long, et contenant diverses informations :

Déplacement	Nom	Explication
\$00	p_lowtpa	Adresse de la page de base (ici!)
\$04	p_hitpa	Adresse de la fin de la page de base + 1
\$08	p_tbase	Pointeur vers le segment TEXT
\$0C	p_tlen	Longueur du segment TEXT
\$10	p_dbase	Pointeur vers le segment DATA
\$14	p_dlen	Longueur du segment DATA
\$18	p_bbase	Pointeur vers le segment BSS
\$1C	p_blen	Longueur du segment BSS
\$20	p_dta	Pointeur vers le tampon DTA
\$24	p_parent	Pointeur vers la page de base du programme parent (0 si accessoire de bureau)
\$28		Réservé
\$2C	p_env	Pointeur vers la chaîne d'environnement
\$80	p_cmdlin	Ligne de commande. Octet indiquant la longueur de la chaîne qui n'est pas toujours terminée par 0
\$100		Votre programme commence ici

Organisation mémoire



Appendice D

Appeler le Système d'Exploitation

Le système d'exploitation du ST est complexe et comporte différents niveaux. Cet Appendice a pour but de vous aider dans le développement de programmes appelant le système d'exploitation. Les procédures d'appels aux différentes routines sont décrits. Les programmes d'exemple fournis avec DevpacST sont également détaillés.

Les différents niveaux du système d'exploitation sont :

GEM AES	gestionnaire de fenêtres et d'événements
GEM VDI	routines graphiques indépendantes de la configuration machine
GEMDOS	Entrées-sorties disque et écran. Similaire à MS-DOS
BIOS	Entrées-sorties de bas niveau
XBIOS	Entrées-sorties de bas niveau étendues

Chaque section est maintenant décrite avec un degré d'explication différent.

GEMDOS - Entrées-sorties disque et écran.

GEMDOS provient de CP/M 68k, ressemble à CP/M par plusieurs aspects, et contient des possibilités supplémentaires (par exemple les sous-répertoires) empruntées à MS-DOS. Il gère la mémoire, les accès disque et les entrées-sorties caractères via l'écran, le clavier et les ports série et parallèle.

GEMDOS a été conçu pour être appelé directement en C. Les paramètres sont donc passés sur la pile et doivent y être supprimés après l'appel. La valeur éventuellement retournée par la fonction est placée dans le registre D0. En assembleur, la séquence d'appel type est :

```
move    ??,-(a7)    empile les parametres
move.w  #??,-(a7)  empile le numero de fonction
trap    #1          appelle GEMDOS
add.l   #??,a7     restaure la pile
```

Après l'appel de la fonction GEMDOS, la pile doit être corrigée pour supprimer les paramètres qui ont été empilés. ADD.L peut être utilisé comme ci-dessus, mais il existe deux manières de faire la même chose, plus rapide et exigeant moins de mémoire. Si huit octets ou moins ont été empilés, utilisez l'instruction :

```
addq.l #??,a7
```

qui n'utilise que deux octets. Si plus de huit octets ont été empilés, utilisez l'instruction :

```
lea.l ??(a7),a7
```

qui utilise quatre octets.

Une erreur courante est d'oublier de corriger la pile après l'appel de la fonction GEMDOS, ou de la corriger avec une mauvaise valeur.

En-Tête et Fin des Programmes

Quand un programme GEMDOS est lancé, toute la mémoire libre lui est attribuée. La mémoire depuis la fin du programme et jusqu'à la fin de la mémoire utilisable (généralement juste avant la mémoire écran) est attribuée au programme. La pile est placée à la fin de cette zone.

Si vous désirez utiliser les fonctions de gestion de la mémoire (comme Malloc), si vous désirez exécuter d'autres programmes en plus du vôtre, ou si votre programme s'exécute sous GEM, il est important de libérer une partie de la mémoire attribuée à votre programme. Il faut pour cela utiliser la fonction Mshrink au tout début de votre programme, en utilisant le fait qu'un pointeur vers la page de base de votre programme se trouve quatre octets après le sommet de la pile.

Voici un exemple d'en-tête de programme libérant la mémoire attribuée automatiquement au programme :

```

ds.l lng_pile reserve place pour la pile
ptr_pile equ *
extra equ xxx

...
move.l 4(a7),a3 page de base
move.l 5c(a3),d0 longueur segment text
add.l $14(a3),d0 longueur segment data
add.l $1c(a3),d0 longueur segment BSS
add.l #extra,d0 memoire a laisser eventuellement
                    au programme
add.l # $100,d0 longueur page de base
move.l #ptr_pile,a7 avant Mshrink

```

```

move.l d0,(-a7)    empile parametre de la fonction
                  Mshrink
move.l a3,-(a7)
clr.w  -(a7)
move.w #$4A,-(a7)
trap  #1           appelle Mshrink
lea.l 12(a7),a7   reinitialise la pile

```

Vous pouvez laisser une partie de la mémoire à votre programme en spécifiant le nombre d'octets à conserver dans extra. Notez que vous devez créer vous même une zone de pile pour votre programme avant d'appeler Mshrink, sinon au retour de Mshrink, la pile sera dans une zone mémoire non attribuée et sujette à modification. Un programme GEMDOS peut se terminer de trois façons différentes : Pterm0 que nous ne recommandons pas, Pterm, manière normale de terminer un programme, ou Ptermres si votre programme doit rester résidant.

Voici un exemple de la manière normale de terminer un programme :

```

clr.w  -(a7)           valeur de retour nulle
move.w #$4C,-(a7)     Pterm
trap  #1

```

Quand un programme est terminé, la mémoire allouée est libérée, et les éventuels fichiers ouverts sont fermés.

Résumé des Fonctions GEMDOS

Voici la liste des fonctions GEMDOS par ordre des numéros de fonctions. Pour chaque fonction, les paramètres accompagnés de leur taille sont spécifiés dans l'ordre où ils doivent être empilés. La valeur à additionner au pointeur de pile après l'appel est également indiquée.

Par exemple, l'appel de la fonction 2, Cconout pour imprimer un caractère x doit se faire de la manière suivante :

```

move.w #'X',-(a7)    le caractere
move.w #2,-(a7)     le numero de fonction
trap  #1            appel GEMDOS
addq.l #4,a7        corrige la pile

```

Actuellement, les fonctions GEMDOS ne modifient que les registres A0 et D0, mais cette particularité n'est pas documentée. Nous vous conseillons de supposer que les registres d0-d2/a0-a2 peuvent être modifiés, comme c'est le cas pour les autres couches du systèmes d'exploitation.

0 - Termine le programme (ancienne forme), Pterm0

Paramètre d'appel : aucun

Paramètre de retour : aucun

Pile : 2

Termine le programme en cours avec un code retour de 0. Nous vous recommandons d'utiliser plutôt Pterm (fonction s4c). Comme le programme ne revient jamais de cette fonction, il n'est pas nécessaire de corriger la pile.

1 - Lit un caractère au clavier, Cconin

Paramètre d'appel : aucun

Paramètre de retour : D0.L=code caractère

Pile : 2

Cette fonction attend l'appui sur une touche, écrit le caractère à l'écran et renvoie sa valeur. La valeur de retour contient le code ASCII du caractère dans les huit bits de poids faible alors que les bits de 16 à 23 contiennent le code clavier. Les autres bits sont à zéro.

2 - Ecrit un caractère à l'écran, Cconout

Paramètre d'appel : mot : caractère

Paramètre de retour : aucun

Pile : 4

L'octet inférieur du paramètre doit contenir le code ASCII du caractère à écrire à l'écran. L'octet supérieur doit être nul.

3 - Lit un caractère sur le port série, Cauxin

Paramètre d'appel : aucun

Paramètre de retour : D0.8 caractère lu

Pile : 2

Cette fonction attend qu'un caractère soit reçu sur la ligne série.

4 - Ecrit un caractère sur la sortie série, Cauxout

Paramètre d'appel : mot : caractère

Paramètre de retour : aucun

Pile : 4

Cette fonction envoie un caractère sur la ligne série. Comme pour la fonction 2, les huit bits de poids fort du mot doivent être nuls.

5 - Ecrit un caractère sur la sortie parallèle, Cprnout

Paramètre d'appel : mot : caractère

Paramètre de retour : D0.W=0 si échec, -1 si Ok

Pile : 4

Cette fonction envoie un caractère sur le port parallèle imprimante. Comme pour la fonction 2 et 4, les huit bits de poids fort du mot doivent être nuls.

6 - Entrée clavier brute ou affichage de caractère, **Crawio**

Paramètre d'appel : mot : caractère à écrire à l'écran, ou \$00FF pour en lire un
Paramètre de retour : D0.W si lecture caractère
Pile : 4

Si le paramètre d'appel est égal à \$00FF, la fonction retourne le code de la touche appuyée au clavier ou 0 si aucune touche n'a été appuyée. Sinon, le caractère passé en paramètre est affiché à l'écran.

7 - Entrée brute d'un caractère, **Crawcin**

Paramètre d'appel : aucun
Paramètre de retour : D0.L=caractère lu
Pile : 2

Attend l'appui d'une touche au clavier et retourne sa valeur sans afficher le caractère à l'écran.

8 - Entrée d'un caractère sans écho, **Cnecin**

Paramètre d'appel : aucun
Paramètre de retour : D0.L=caractère lu
Pile : 2

Attend l'appui d'une touche au clavier et retourne sa valeur sans afficher le caractère à l'écran. Les combinaisons de touches **Ctrl-Q**, **Ctrl-S** et **Ctrl-C** sont interprétés et ne provoquent pas de retour de la fonction. **Ctrl-C** interrompt le programme, **Ctrl-S** suspend l'affichage, et **Ctrl-Q** le reprend.

9 - Affiche une chaîne de caractères à l'écran, **Cconws**

Paramètre d'appel : long : adresse de la chaîne
Paramètre de retour : aucun
Pile : 6

Ecrit une chaîne de caractères terminée par 0 à l'écran

§A - Lecture d'une chaîne de caractères au clavier, **Cconrs**

Paramètre d'appel : long : adresse du tampon de stockage de la chaîne
Paramètre de retour : aucun
Pile : 6

Avant l'appel, le premier octet du tampon doit contenir la longueur maximale de la chaîne à saisir. Au retour, le deuxième octet contient la longueur de la chaîne lue. La chaîne elle-même commence à partir du troisième octet. Aucune marque de fin de chaîne n'est stockée, ni CR ni 0. Un **Ctrl-C** pendant la saisie interrompt le programme.

§B - Demande l'état du clavier, **Cconis**

Paramètre d'appel : aucun
Paramètre de retour : D0.L=-1 si caractère disponible, 0 sinon.
Pile : 2

Indique si une touche a été tapée au clavier. Utilisez une autre fonction pour lire la touche.

\$E - Fixe le lecteur de disque par défaut, Dsetdrv

Paramètre d'appel : mot : numéro du lecteur
Paramètre de retour : D0.L=carte des lecteurs du système
Pile : 4

Fixe le lecteur de disque par défaut : 0 = A:, 1 = B:, etc... La valeur retournée contient un bit à 1 pour chaque lecteur actif : bit 0 = A:, bit 1 = B:, etc...

\$10 - Teste l'état du périphérique de sortie standard, Cconos

Paramètre d'appel : aucun
Paramètre de retour : D0.L=-1 si prêt, 0 sinon
Pile : 2

Teste si le périphérique de sortie standard (écran!) est prêt à recevoir un caractère.

\$11 - Teste l'état de l'imprimante, Cprnos

Paramètre d'appel : aucun
Paramètre de retour : D0.L=-1 si prêt, 0 sinon
Pile : 2

Retourne -1 si l'imprimante parallèle est prête à recevoir un caractère. 0 sinon.

\$12 - Teste l'état du port série en entrée

Paramètre d'appel : aucun
Paramètre de retour : D0.L=-1 si un caractère est reçu, 0 sinon
Pile : 2

Teste si un caractère a été reçu par la ligne série et retourne -1 si oui, 0 sinon.

\$13 - Teste l'état du port série en sortie

Paramètre d'appel : aucun
Paramètre de retour : D0.L=-1 si prêt, 0 sinon
Pile : 2

Teste si un caractère peut être envoyé sur la ligne série. Retourne -1 si oui, 0 sinon.

\$19 - Demande le lecteur de disque par défaut, Dgetdrv

Paramètre d'appel : aucun
Paramètre de retour : D0.W=numéro du lecteur
Pile : 2

Retourne le lecteur de disque actif par défaut. 0 = A:, 1 = B:, etc...

\$1A - Fixe l'adresse de transfert disque, Fsetdta

Paramètre d'appel : long : pointeur vers le tampon
Paramètre de retour : aucun
Pile : 6

Fixe l'adresse d'un tampon de 44 octets utilisé par les fonctions Ffirst et Fnext.

\$20 - Changement de mode, Super

Paramètre d'appel : long : pointeur de pile, ou 0 ou 1

Paramètre de retour : D0.L, dépend de l'appel

Pile : 6

Cette fonction a deux usages : elle peut vous dire si vous êtes en mode superviseur ou utilisateur, et elle peut aussi vous permettre de changer de mode. Si vous désirez savoir dans quel mode vous êtes, appelez cette fonction avec la valeur 1 comme paramètre. Elle retournera 0 si vous êtes en mode utilisateur, -1 sinon. Pour changer de mode, passez en paramètre un nouveau pointeur de pile, ou la valeur 0 pour utiliser le pointeur de pile actuel. L'exemple suivant vous fait entrer en mode superviseur si vous êtes en mode utilisateur et spécifie un nouveau pointeur de pile :

```
move.l #nouvelle_pile, -(a7)
move.w #$20, -(a7)
trap #1
addq.l #6, a7
```

Lorsque vous passez en mode superviseur, l'ancienne valeur de SSP est retournée. Si vous désirez utiliser temporairement le mode superviseur pour lire de la mémoire protégée par exemple, utilisez plutôt la fonction XBIOS Supexec.

\$2A - Demande la date, Tgetdate

Paramètre d'appel : aucun

Paramètre de retour : D0.W=date

Pile : 2

La date retournée par cette fonction est compactée dans un mot de la manière suivante :

jour :	bits 0-4
mois :	bits 5-8
année :	bits 9-15 (0 = année 1980)

\$2B - Modifie la date, Tsetdate

Paramètre d'appel : mot : date

Paramètre de retour : aucun

Pile : 4

Modifie la date du système en utilisant le même format que pour la fonction Tgetdate.

\$2C - Demande l'heure, Tgettime

Paramètre d'appel : aucun

Paramètre de retour : D0.W=heure

Pile : 2

L'heure retournée par cette fonction est compactée dans un mot de la manière suivante :

secondes/2 : bits 0-4
minutes : bits 5-10
heures : bits 11-15

\$2D - Modifie l'heure, Tsettime

Paramètre d'appel : mot : date
Paramètre de retour : aucun
Pile : 4

Modifie l'heure du système en utilisant le même format que pour la fonction Tgettime.

\$2F - Demande l'adresse de transfert disque, Fgetdta

Paramètre d'appel : aucun
Paramètre de retour : D0.L : pointeur vers le tampon
Pile : 2

Demande l'adresse du tampon de transfert disque

\$30 - Demande le numéro de version de GEMDOS, Sversion

Paramètre d'appel : aucun
Paramètre de retour : D0.W=numéro de version
Pile : 2

Retourne le numéro de version de GEMDOS. Le numéro primaire est dans l'octet faible, et le numéro secondaire dans l'octet fort. Quelques numéros de version :

\$0D00	version 0.13 (ancienne version sur disque)
\$1300	version 0.19 (version ROM des STF)
\$1500	version 0.21 (TOS 1.4)
\$1700	version 0.23 (version ROM des STE)

\$31 - Fin d'un programme résident, Ptermres

Paramètres d'appel : mot : code de sortie, long : nombre d'octets à conserver
Paramètre de retour : aucun
Pile : 8

Autorise un programme à terminer son exécution mais en restant résident (du moins en partie). Cela peut servir à créer des extensions au système d'exploitation, comme un *ram-disk*. S'ils se terminaient normalement, ils seraient détruits par le prochain programme chargé en mémoire. La mémoire à réserver pour ce programme commence à la page de base. Sa longueur doit inclure les 256 octets de la page de base, la longueur nécessaire du programme, ainsi qu'éventuellement un espace pour la pile et les données.

\$36 - Demande l'espace libre du disque, Dfree

Paramètres d'appel : mot : code du lecteur, long : pointeur sur tampon

Paramètre de retour : aucun

Pile : 8

Retourne diverses informations sur le disque. Le code du disque doit être 0 pour le lecteur courant, 1 pour A, 2 pour B, etc... Le tampon doit être aligné sur un mot et contenir 16 octets. En retour, le tampon contient quatre mots longs : espace libre en blocs, nombre total de blocs, taille d'un secteur en octets, et nombre de secteurs par bloc.

\$39 - Crée un sous-répertoire, Dcreate

Paramètre d'appel : long : adresse du nom du répertoire

Paramètre de retour : D0.W=0 si OK, sinon code erreur

Pile : 6

Crée un nouveau sous-répertoire ou dossier. Le nom est spécifié par la chaîne de caractères terminée par zéro passée en paramètre.

\$3A - Supprime un sous-répertoire, Ddelete

Paramètre d'appel : long : adresse du nom du répertoire

Paramètre de retour : D0.W=0 si OK, sinon code erreur

Pile : 6

Détruit le sous-répertoire spécifié s'il ne contient aucun fichier ou sous-répertoire.

\$3B - Fixe le répertoire courant, Dsetpath

Paramètre d'appel : long : adresse du nom du répertoire

Paramètre de retour : D0.W=0 si OK, sinon code erreur

Pile : 6

Change le répertoire par défaut, spécifié par la chaîne terminée par zéro passée en paramètre.

\$3C - Crée un fichier, Fcreate

Paramètres d'appel : mot : attributs, long : adresse du nom de fichier

Paramètre de retour : D0.W=identificateur du fichier si succès, code d'erreur (sur un long) sinon

Pile : 8

Crée et ouvre un fichier. En cas de succès, un numéro de fichier est retourné. L'attribut du fichier doit être l'une des valeurs suivantes :

- 01 / lecture seule
- 02 / fichier caché
- 04 / fichier système
- 08 / fichier contenant le nom du volume dans ses onze premiers caractères

Le numéro de fichier retourné par cette fonction et la suivante est normalement un nombre à partir de 6. Les valeurs 0 à 5 correspondent à des fichiers standards déjà ouverts lorsque vous exécutez votre programme :

- 0 lecture clavier
- 1 écriture écran
- 2 port série
- 3 port parallèle

Il existe trois noms de fichiers systèmes correspondant à l'écran/clavier, la ligne série, et le port parallèle, respectivement CON:, AUX: et PRN:. Ces noms peuvent être utilisés avec cette fonction ou la suivante pour ouvrir des fichiers sur ces périphériques. La fonction Dcreate retourne alors une valeur négative sur un mot correspondant à l'identificateur de fichier. Il est important de tester entièrement le long retourné par la fonction avec TST.L / BMI pour déterminer si une erreur est réellement survenue.

\$3D - Ouvre un fichier, Fopen

Paramètres d'appel : mot ; attributs, long ; adresse du nom de fichier

Paramètre de retour : D0.W=identificateur du fichier si succès, code d'erreur (sur un long) sinon

Pile : 8

Ouvre un fichier existant en lecture, en écriture ou les deux. En cas de succès, un numéro de fichier est retourné. Sinon, un numéro d'erreur sur un long est retourné. L'attribut du fichier doit être l'une des valeurs suivantes :

- 0 ouverture en lecture seule
- 1 ouverture en écriture seule
- 2 ouverture en lecture et écriture

\$3E - Ferme un fichier, Fclose

Paramètre d'appel : mot ; numéro de fichier

Paramètre de retour : D0.W=0 si OK, sinon code erreur

Pile : 4

Ferme le fichier spécifié. Ne ferme pas un fichier standard.

Note

Cette fonction, comme toutes les fonctions de fichiers demandant un numéro de fichier, ne contrôle pas les valeurs des numéros de fichiers passés en paramètre. Si vous passez un numéro invalide, la fonction peut retourner un message d'erreur, ou bien planter !

\$3F - Lecture dans un fichier, Fread

Paramètres d'appel : long : adresse de stockage, long : nombre d'octets à lire, mot : numéro de fichier
Paramètre de retour : D0.L=nombre d'octets effectivement lus, ou code d'erreur négatif
Pile : 12

Lit le nombre d'octets spécifié dans un fichier. Si la fin de fichier est rencontrée avant d'avoir lu le nombre d'octets spécifié, la fonction ne retournera pas de code d'erreur mais le nombre d'octets réellement lus. Si cette valeur est différente du nombre d'octets spécifiés, vous avez rencontré la fin du fichier.

\$40 - Ecriture dans un fichier, Fwrite

Paramètres d'appel : long : adresse des données, long : nombre d'octets à écrire, mot : numéro de fichier
Paramètre de retour : D0.L=nombre d'octets écrits, ou code d'erreur négatif
Pile : 12

Ecrit le nombre d'octets spécifié dans un fichier. Si une erreur survient, la valeur retournée est négative. Si le disque est plein, aucun code d'erreur n'est retourné, mais le nombre d'octets écrits sera inférieur au nombre spécifié.

Note

Si vous passez une longueur négative, GEMDOS plantera salement.

\$41 - Efface un fichier, Fdelete

Paramètre d'appel : long : adresse du nom de fichier
Paramètre de retour : D0.W=0 si OK, code d'erreur négatif sinon
Pile : 5
Détruit le fichier spécifié du disque.

\$42 - Déplace le pointeur de fichier, Fseek

Paramètres d'appel : mot : mode, mot : numéro de fichier, long : position
Paramètre de retour : D0.L=position résultante absolue du pointeur de fichier
Pile : 10
Déplace le pointeur de fichier à une position spécifiée dans le fichier. Le mode doit être l'une des trois valeurs :

- 0 déplace le pointeur N octets à partir du début du fichier
- 1 déplace le pointeur N octets à partir de la position courante
- 2 déplace le pointeur N octets à partir de la fin de fichier

Si vous essayez de vous déplacer avant le début du fichier, la valeur retournée sera 0. Après la fin, la valeur retournée sera la longueur du fichier.

\$43 - Lit/fixe les attributs d'un fichier, Fattrib

Paramètres d'appel : nom : attributs, mot : lit/fixe, long : adresse
du nom de fichier

Paramètre de retour : D0.W=nouveaux attributs, ou code erreur

Pile : 10

Lit ou fixe les attributs d'un fichier dont le nom est passé en paramètre. Les différents attributs possibles sont :

01	lecture seule
02	fichier caché
04	fichier système
08	fichier contenant le nom du volume dans ses onze premiers caractères
\$10	sous-répertoire
\$20	fichier écrit puis refermé

L'autre mot indique si l'on veut lire (0) ou écrire (1) les attributs du fichier.

\$45 - Duplique un numéro de fichier, Fdup

Paramètre d'appel : mot : numéro de fichier standard

Paramètre de retour : D0.W=nouveau numéro, ou code d'erreur

Pile : 4

A partir d'un numéro de fichier standard (0-5), cette fonction retourne un autre numéro de fichier qui peut être utilisé pour adresser le même périphérique. Le nouveau fichier peut être fermé sans affecter le fichier standard.

\$46 - Force un numéro de fichier, Fforce

Paramètres d'appel : mot : numéro de fichier non standard, mot :
numéro standard

Paramètre de retour : D0.W=0 si OK, ou code d'erreur sinon

Pile : 6

Force un identificateur de fichier standard à pointer sur le même périphérique ou fichier que le fichier non standard. Peut être utilisé pour rediriger la sortie écran dans un fichier.

Note

Effectuez auparavant une copie du numéro de fichier avec Fdup pour pouvoir le restaurer ultérieurement.

\$47 - Lit le répertoire courant

Paramètres d'appel : mot : numéro du lecteur, long : pointeur vers
un tampon

Paramètre de retour : D0.W=0 si OK, ou code d'erreur sinon

Pile : 8

A partir du numéro de lecteur de disque (0=lecteur par défaut, 1=A, 2=B, etc...) cette fonction recopie dans le tampon de 64 caractères spécifié en paramètre un chaîne de caractères terminée par zéro contenant le nom du répertoire courant.

\$48 - Allocation mémoire, Malloc

Paramètre d'appel : long : nombre d'octets requis

Paramètre de retour : D0.L=adresse de la mémoire allouée, ou 0 si échec

Pile : 6

Alloue un bloc mémoire dans la zone système, si disponible. Quand un programme est terminé, toutes ses allocations sont effacées. Cette fonction peut également être utilisée pour obtenir l'espace mémoire disponible en passant -1 en paramètre.

Note

Quand GEMDOS utilise cette fonction, il teste si la valeur retournée par la fonction est paire. Nous vous recommandons donc de faire de même. Cette fonction retournait parfois, avec l'ancien TOS 0.13, une valeur impaire !

\$49 - Libération de mémoire, Mfree

Paramètre d'appel : long : adresse de la zone mémoire à libérer

Paramètre de retour : D0.W=0 si OK, ou code d'erreur sinon

Pile : 6

Libère un bloc mémoire alloué par Malloc.

\$4A - Réduit une zone de mémoire allouée, Mshrink

Paramètres d'appel : long : longueur à conserver, long : adresse de la zone, mot : 0

Paramètre de retour : D0.W=0 si OK, ou code d'erreur sinon

Pile : 12

Diminue par la fin la taille d'une zone mémoire allouée. Cette fonction est normalement utilisée lorsqu'un programme est lancé pour restituer à GEMDOS la mémoire qu'il n'utilise pas.

\$4B - Charge et exécute un programme, Pexec

Paramètres d'appel : long : pointeur vers une chaîne d'environnement, long : pointeur vers une ligne de commande, long : pointeur vers le nom de fichier, mot : mode

Paramètre de retour : D0.L=(dépend du mode)

Pile : 16

Charge et chaîne des programmes. Le mode peut prendre l'une des valeurs suivantes :

- 0 : charge et exécute
- 3 : charge sans exécuter
- 4 : exécute à partir de la page de base
- 5 : crée une page de base
- 6 : exécute à partir de la page de base, et libère mémoire

Pour le mode 0, la valeur retournée est soit un code d'erreur, soit la valeur retournée par le programme fils lorsque son exécution est terminée.

Pour le mode 3, la valeur retournée est soit un code d'erreur, soit l'adresse de la page de base du programme chargé.

La description des modes 4, 5 et 6 est hors de la portée de ce document.

La chaîne de caractères constituant la ligne de commande doit commencer par un octet indiquant sa longueur.

La chaîne de caractères constituant l'environnement est un pointeur vers une suite de chaîne de caractères terminées par un zéro. La liste de chaînes de caractères est elle-même terminée par deux octets nuls. Il est possible de transmettre au programme fils le même environnement que celui du programme appelant en passant la valeur 0 comme paramètre. Exemple d'environnement :

```
env      dc.b      'PATH=',0,'A:\',0,0
```

§4C - Termine un programme, Pterm

Paramètre d'appel : mot : valeur à retourner au programme appelant

Paramètre de retour : NS puisque pas de retour

Pile : NS

Termine le programme courant, et rend la main au programme appelant. La valeur à passer en paramètre est retournée au programme appelant. Elle doit être négative pour indiquer une erreur, ou nulle sinon. Spécifiez une valeur positive pour éviter une confusion avec un code erreur système.

§4E - Cherche la première occurrence d'un fichier, Ffirst

Paramètres d'appel : mot : attributs, long : pointeur vers le filtre de fichier

Paramètre de retour : D0.W=0 si trouvé, -33 sinon

Pile : 8

Cette fonction peut être utilisée pour lire un répertoire en utilisant un filtre (ou joker, comme "*.S") pour trouver tous les fichiers désirés. Cette fonction ne doit être appelée que pour trouver le premier nom de fichier. Utiliser Fnext pour les autres. Quand un fichier est trouvé, ses paramètres sont retournés dans le tampon DTA (voir fonction Fgetdta, Fsetdta, et le champ p_dta de la page de base). Le paramètre d'attributs détermine les fichiers à inclure dans la recherche (ces attributs peuvent être combinés).

Les valeurs d'attributs sont :

00	fichier normal
01	fichier en lecture seule
02	fichier caché
04	fichier système
08	retourne le nom du volume uniquement
S10	sous-répertoire
S20	fichier écrit puis refermé

Les données retournées dans le tampon DTA sont (par octet) :

0-20	réservé pour usage interne
21	attributs du fichier
22-23	heure du fichier compactée (voir Fgettime)
24-25	date du fichier compactée (voir Fgetdate)
26-29	taille du fichier (long)
30-43	nom du fichier, terminé par 0

\$4F - Cherche l'occurrence suivante d'un fichier, Fsnext

Paramètre d'appel : aucun

Paramètre de retour : D0.W=0 si trouvé, -33 sinon

Pile : 2

Après avoir appelé une fois Ffirst pour trouver la première occurrence d'une sélection de fichiers, cette fonction est utilisée pour rechercher les occurrences suivantes. Quand un fichier est trouvé, sa description se trouve dans le tampon DTA. Il est important de ne pas modifier les 21 premiers octets de la DTA entre les appels.

\$56 - Renomme un fichier, Frename

Paramètres d'appel : long : pointeur vers le nouveau nom, long :
pointeur vers l'ancien nom, mot : 0

Paramètre de retour : D0.W=0 si OK, sinon code erreur

Pile : 12

Cette fonction change le nom d'un fichier. Le nouveau nom de fichier ne doit pas exister.

\$57 - Lit ou modifie la date et l'heure d'un fichier, Fdalime

Paramètres d'appel : mot : 0 pour lire, 1 pour écrire, mot : numéro
de fichier, long : pointeur sur un tampon

Paramètre de retour : aucun

Pile : 10

Cette fonction peut être utilisée pour lire ou modifier la date et l'heure d'un fichier. Le tampon doit contenir deux mots. Le premier est l'heure compactée, le second la date compactée. Le format de compactage est détaillé aux fonction Fgettime et Fgetdate.

BIOS - Basic I/O System (Entrées Sorties Élémentaires)

Le BIOS du ST gère les entrées sorties de bas niveau vers l'écran, le clavier et les disques. L'appel des fonctions se fait de la même manière que pour GEMDOS, mais en utilisant l'instruction TRAP #13. Si vous désirez utiliser le BIOS, vous aurez besoin de plus d'informations que nous n'en fournissons ici puisque nous ne détaillons qu'une seule fonction. Voyez donc les livres de la bibliographie. Les fonctions BIOS ne modifient pas les registres D3-D7/A3-A7.

BIOS 5 - Modifie un Vecteur d'Exception, Setexc

Cette fonction très utile permet de modifier les vecteurs du système pour les faire pointer vers vos propres routines. Il est possible de modifier à la fois les vecteurs d'exception et les vecteurs système. La séquence d'appel est :

```
move.l    #ma_routine, -(a7)    adresse de la nouvelle
                                routine
move.w    #no_vecteur, -(a7)    numéro du vecteur
move.w    #5, -(a7)            numéro de la fonction BIOS
trap      #13
addq.l    #8, a7                restaure la pile
move.l    d0, vieille_routine  sauve l'ancienne valeur
```

Le numéro de vecteur doit être le numéro de l'exception (2 pour erreur bus, 3 pour erreur adresse, etc...), ou l'une des valeurs suivantes :

\$45	liste 200 Hz
\$100	interruption timer système
\$101	gestionnaire d'erreur critique
\$102	détournement de la routine de fin de programme

Au retour de la fonction, D0.L contient l'ancienne valeur du vecteur. Cette valeur doit absolument être restaurée avant de quitter votre programme.

Si vous spécifiez -1 comme valeur de vecteur, sa valeur ne sera pas modifiée, mais la valeur du vecteur sera retournée dans D0.L.

XBIOS - BIOS étendu

Le XBIOS est composé de 40 fonctions d'usages variés, comme l'accès au matériel (*hardware*), le contrôle de l'écran ou la modification de la carte du clavier. Là encore, nous vous laissons consulter les livres cités en bibliographie si vous voulez en savoir plus. Nous ne décrivons ici que cinq des fonctions XBIOS.

Aucune fonction XBIOS ne modifie les registres D3-D7/A4-A7. La séquence d'appel d'une fonction XBIOS est la même que pour les fonctions GEMDOS : il faut passer empiler les paramètres puis le numéro de fonction, exécuter une instruction TRAP #14 puis restaurer la pile.

XBIOS 2 - Lit l'adresse physique de l'écran, Physbase

Paramètre d'appel : aucun
Paramètre de retour : D0.L=adresse du début de l'écran
Pile : 2

Cette fonction retourne l'adresse de l'écran physique. C'est l'adresse de l'écran que l'on voit. La mémoire d'écran occupe 32000 octets, et est toujours alignée sur une adresse multiple de 256.

XBIOS 3 - Lit l'adresse logique de l'écran, Logbase

Paramètre d'appel : aucun
Paramètre de retour : D0.L=adresse du début de l'écran
Pile : 2

Cette fonction retourne l'adresse de l'écran logique. C'est l'adresse de l'écran qui est modifiée par les diverses fonctions du système d'exploitation.

XBIOS 4 - Lit la résolution de l'écran, Getrez

Paramètre d'appel : aucun
Paramètre de retour : D0.W=0 basse, 1 moyenne, 2 haute
Pile : 2

Retourne la résolution de l'écran.

XBIOS 5 - Modifie les adresses et la résolution de l'écran, Setscreen

Paramètres d'appel : mot : résolution, long : adresse physique,
long : adresse logique
Paramètre de retour : aucun
Pile : 12

Cette fonction modifie la résolution de l'écran et ses adresses physiques et logiques. Si une valeur -1 est spécifiée pour un paramètre, la valeur correspondante n'est pas modifiée. L'écran sera effacé si vous changez de résolution.

XBIOS \$26 - Exécution d'une routine en mode superviseur, Supexec

Paramètre d'appel : long : adresse de la routine

Paramètre de retour : selon la fonction

Pile : 6

La routine dont l'adresse est spécifiée est exécutée en mode superviseur. La routine doit se terminer par RTS. Les registres D3- D7/A3-A7 ne sont pas modifiés. La routine ne doit pas faire d'appel BIOS, XBIOS ou GEMDOS.

Bibliothèques GEM

GEM est constitué de deux parties : VDI et AES.

GEM VDI (Virtual Device Interface) est la partie qui dessine des graphiques et affiche du texte.

GEM AES (Application Environment Services) gère les fenêtres, les boîtes de dialogue, les menus et les événements.

Ce paragraphe a pour but de détailler les fichiers bibliothèques fournis, et les conventions d'appel de ces fonctions. Elle ne décrit pas l'utilisation de ces fonctions. Utilisez pour cela un des livres décrits en bibliographie. Toutefois, quelques informations non ou mal documentées sont fournies.

Bibliothèque GEM AES

L'appel des fonctions AES est basée sur l'utilisation de divers tableaux de mots et de longs. Ces tableaux sont définis en utilisant des directives DS :

```
control      mots
int_in       mots
addr_in      longs
int_out      mots
addr_out     longs
aes_param    longs
global      mots
```

Par exemple, la ligne de C suivante :

```
val = int_out[2] + int_out[3];
```

s'écrit en assembleur :

```
move.w  int_out+4,d0
add.w   int_out+6,d0.
```

Notez que les indices de tableaux sont multipliés par deux dans le programme assembleur puisqu'il s'agit d'un tableau de mots. Le C fait le calcul tout seul, pas l'assembleur. Pour un tableau de longs, l'indice doit être multiplié par 4.

Un fichier de macros, `GEMMACRO.S`, contient diverses macro-instructions destinées à être utilisées dans vos programmes GEM. Si vous générez du code exécutable avec GENST, incluez également le fichier `AESLIB.S` à la fin de votre programme.

Ces fichiers définissent des macros correspondant à chaque fonction AES et VDI. Elles prennent en compte les différents paramètres de chaque fonctions et les rangent dans les tableaux requis avant d'appeler la fonction d'appel AES.

Si vous passez des constantes comme paramètres aux macros de fonction AES, assurez vous de les faire précéder du caractère `#`. Par exemple, si vous passez les paramètres 3 et pointeur à la macro, elle générera un code du type :

```
move.w 3,init_in
move.l pointeur, addr_in
```

La première ligne provoquera une erreur à l'exécution, car le paramètre aurait dû être `#3`.

Il existe quelques macros AES qui ne prennent pas en compte tous les paramètres requis. Les autres paramètres doivent être placés dans d'autres tableaux. Après l'appel de la macro AES, `D0.W` contient la valeur de `int_out[0]`, et le registre d'état est affecté par cette valeur. D'autres valeurs retournées par la fonction se trouvent dans le tableau `int_out`.

Les descriptions suivantes supposent que tous les paramètres sont des mots, à moins qu'ils aient un suffixe `.L` qui indique un paramètre long.

Bibliothèque Application

appl_init

Doit être appelé au début de tout programme AES.

appl_read id,longueur,tampon.L

appl_wrtte id,longueur,tampon.L

appl_find nom.L

Recherche un programme par son nom. Normalement un accessoire de bureau.

appl_tplay *mémoire.L,nombre,échelle*

appl_trecord *mémoire.L,compteur*

appl_exit

Doit être appelé juste avant de terminer un programme AES. Envoie le message AC_CLOSE à tous les accessoires.

Bibliothèque Événement

evnt_keybd

evnt_button *clicks,masque,état*

La valeur retournée est le nombre de fois que le bouton est entré dans l'état spécifié. Les éléments 1 à 4 du tableau `int_out` contiennent respectivement les coordonnées X puis Y, l'état du bouton puis l'état du clavier à l'instant de l'événement.

evnt_mouse *flags,x,y,w,h*

Mêmes valeurs retournées que la fonction précédente.

evnt_mesag *tampon.L*

evnt_timer *compteur.L*

evnt_multl *flags,clicks,masque,état,m1flags,m1x,m1y,m1w,m1h,*

& m2flags,m2x,m2y,m2w,m2h,ptmessage.L,compteur.L

Tous les paramètres sont optionnels sauf le premier dans le cas où l'on spécifie un événement nul. Aucune valeur n'est alors placée dans le tableau `int_in`. La description de cette fonction ci-dessus est faite dans le format d'une macro sur plusieurs lignes, mais ce n'est pas obligatoire. Le tableau `int_out` contient respectivement en sortie les types d'événement, coordonnées X puis Y de la souris, bouton, état du clavier, code clavier et état du bouton.

evnt_dclick *nouveau,lit_écrit*

Bibliothèque Menu

menu_bar *arbre.L, montre*

menu_ichck *arbre.L, objet, marque*

menu_lenable *arbre.L, objet, autorise*

menu_tnormal *arbre.L, titre, normal*

menu_text *arbre.L, objet, texte.L*

menu_register *id, chaîne*

Note

Normalement, un menu est construit à l'aide d'un éditeur de ressources, ou bien à la main avec beaucoup de précautions. Vous pouvez aussi utiliser le compilateur de menus MENU2ASM décrit plus loin dans cette section.

Bibliothèque Objet

Les arbres d'objets peuvent être créés par un éditeur de ressource ou à la main. Les boîtes de dialogue sont plus faciles à définir à la main que les menus...

objc_add *arbre.L, parent, fils*

objc_delete *arbre.L, objet*

objc_draw *arbre.L, objdebut, profondeur, x, y, w, h*

objc_find *arbre.L, objdebut, profondeur, x, y*

objc_offset *arbre.L, objet*

Les indices 1 et 2 du tableau `int_out` contiennent les coordonnées X et Y.

objc_order *arbre.L, objet, nouvelle_pos*

objc_edit *arbre.L, objet, caractère, idx, type*

`int_out[1]` contient la nouvelle valeur de `idx`.

objc_change *arbre.L, objet, x, y, w, h, nouveau, redessine*

Bibliothèque Boîte de dialogue

form_do *arbre.L, objdebut*

Ne spécifiez jamais la valeur -1 pour *objdebut*, mais 0.

form_dial *flag, x1, y1, w1, h1, x2, y2, w2, h2*

form_alert *bouton, chaîne.L*

form_error *no_erreur*

Le numéro d'erreur doit être positif et inférieur à 64.

form_center *arbre.L*

Bibliothèque Graphique

graf_rubberbox *x, y, w, h*

int_out[1] contiendra la largeur finale, *int_out*[2] la hauteur.

graf_dragbox *w, h, x, y, bx, by, bw, bh*

int_out[1] contiendra la coordonnée X finale, *int_out*[2] la coordonnée Y.

graf_movebox *w, h, x, y, dx, dy*

graf_growbox *x, y, w, h, fx, fy, fw, fh*

graf_growbox *x, y, w, h, sx, sy, sw, sh*

graf_watchbox *arbre.L, objet, etat_entree, etat_sortie*

graf_slidebox *arbre, parent, obj, vh*

graf_handle

Le tableau *int_out* contiendra en sortie le handle VDI, la largeur et la hauteur du rectangle enveloppe de caractère et la largeur et la hauteur caractère (police système).

graf_mouse *nombre, adresse.L*

Le paramètre *adresse* est optionnel, il n'est nécessaire que lorsque vous définissez une nouvelle forme.

graf_mkstate

Le tableau `int_out` contiendra en sortie une valeur réservée, les coordonnées X puis Y de la souris, l'état du bouton de la souris puis l'état du clavier.

Bibliothèque Presse-Papier

`scrp_read tampon.L`

`scrp_write tampon.L`

Bibliothèque Sélecteur de Fichiers

`fsel_input chemin.L,nom_fichier.L`

Le paramètre `chemin` contient en entrée le chemin d'accès terminé par 0, comme par exemple `A:*.S`. Il sera remplacé par le nouveau chemin d'accès, assurez vous donc qu'il est assez grand. Le paramètre `nom_fichier` doit avoir une longueur de 13 octets, un maximum de 12 étant utilisés pour le nom de fichier. `int_out [1]` contiendra 0 en sortie si le bouton Annuler a été sélectionné. Si `D0.W` est égal à zéro en retour de fonction, il n'y a pas assez de mémoire pour appeler le sélecteur.

Bibliothèque Fenêtre

`wind_create type,x,y,w,h`

`wind_open handle,x,y,w,h`

`wind_close handle`

`wind_delete handle`

`wind_get handle,champ`

`wind_set handle,champ`

`wind_find x,y`

`wind_update debut_fin`

`wind_calc type,sorte,inx,iny,inw,inh`

Bibliothèque Ressource

rsrc_load nom_fichier.L

rsrc_free

rsrc_gaddr type,index

L'adresse en retour se trouvera dans *addr_out*.

rsrc_saddr type,index,adresse.L

rsrc_obfix arbre.L,objet

Bibliothèque Shell

shel_read commande.L,shell.L

shel_write chainage,graphique,gem,commande.L,shell.L

Nous n'avons jamais réussi à faire fonctionner correctement cette fonction.

shel_find tampon.L

Le tampon doit être d'au moins 80 caractères

shel_envrn valeur.L,chaîne.L

Débuguer les Appels AES

Contrairement aux fonctions VDI, les fonctions AES ne sont pas facilement repérables en mémoire depuis MonST car elles sont du type :

```
moveq    #??, d0    Numero de fonction AES
bsr      CALL_AES
```

Pour vous aider à vous retrouver parmi tous les numéros de fonctions AES, en voici la liste avec leur numéro :

A	appl_init	B	appl_read	C	appl_write
D	appl_find	E	appl_tplay	F	appl_record
13	appl_exit	14	evnt_keybd	15	evnt_button
16	evnt_mouse	17	evnt_mesag	18	evnt_timer
19	evnt_multi	1A	evnt_dclick	1E	menu_bar
1F	menu_lcheck	20	menu_enable	21	menu_normal
22	menu_text	23	menu_register	28	objc_add
29	objc_delete	2A	objc_draw	2B	objc_find
2C	objc_offset	2D	objc_order	2E	objc_edit
2F	objc_change	32	form_do	33	form_dial
34	form_alert	35	form_error	36	form_center
37	form_keybd	38	form_button	46	graf_rubberbox
47	graf_dragbox	48	graf_movebox	49	graf_growbox
4A	graf_shrinkbox	48	graf_watchbox	4C	graf_slidebox
4D	graf_handle	4E	graf_mouse	4F	graf_mkstate
50	scrp_read	51	scrp_write	5A	tsel_input
??	tsel_exinput	64	wind_create	65	wind_open
66	wind_close	67	wind_delete	68	wind_get
69	wind_set	6A	wind_find	68	wind_update
6C	wind_calc	6D	wind_new	6E	rsrc_load
6F	rsrc_free	70	rsrc_gaddr	71	rsrc_saddr
72	rsrc_obfix	78	shel_read	79	shel_write
7A	shel_get	7B	shel_put	7A	shel_find
78	shel_envrn				

Bibliothèque GEM VDI

L'appel des fonctions VDI est, comme les fonctions AES, basée sur l'utilisation de divers tableaux de mots et de longs. Ces tableaux sont définis en utilisant des directives DS :

```
control    mots
intin      mots
ptsin     mots
intout     mots
ptsout    longs
vdi_param  longs
```

Toutes les fonctions VDI sauf une nécessitent un handle VDI qui est par tradition un paramètre de toutes les fonctions. En fait, la majorité des programmes n'utilisent qu'un seul handle représentant l'écran. Les macros VDI fournies utilisent une variable appelée `current_handle` et la passent aux fonctions VDI. Cela économise une partie appréciable de code. Les bibliothèques du **HiSoft Basic** fonctionnent de même. Vous pouvez changer ceci puisque les sources des bibliothèques sont fournis.

Un fichier de macros, `GEMMACRO.S`, contient diverses macro-instructions destinées à être utilisées dans vos programmes GEM. Si vous générez du code exécutable avec `GENST`, incluez également le fichier `VDILIB.S` à la fin de votre programme.

Ces fichiers définissent des macros correspondant à chaque fonction AES et VDI. Elles prennent en compte les différents paramètres de chaque fonctions et les rangent dans les tableaux requis avant d'appeler la fonction d'appel VDI.

Il existe quelques macros VDI qui ne prennent pas en compte tous les paramètres requis. Les autres paramètres doivent être placés dans d'autres tableaux. Après l'appel de la macro les valeurs retournées par la fonction se trouvent dans les tableaux `intout` et `ptsout`.

Les descriptions suivantes supposent que tous les paramètres sont des mots, à moins qu'ils aient un suffixe `.L` qui indique un paramètre long.

Fonctions de Contrôle

v_opnwk **Ouvre station de Travail**

Cette fonction ne doit pas être utilisée à moins que GDOS ne soit installé. Le tableau `intin` doit être initialisé correctement. La variable `current_handle` est initialisée par cet appel.

v_clswk **Ferme Station de Travail**

v_opnvwk **Ouvre Station de Travail Virtuelle**

Cette fonction utilise `current_handle` pour ouvrir une autre station de travail, et modifie `current_handle` d'après le résultat. `intin` est généralement rempli de 10 mots à 1 et d'un mot à 2 (spécifiant des coordonnées RC).

v_clsvwk **Ferme Station de travail Virtuelle**

v_clrwk **Efface Station de Travail**

v_updwk **Mise à Jour Station de Travail**

vst_load_font Charge des Polices de Caractères

A n'utiliser que si GDOS est chargé.

vst_unload_font Efface des Polices de Caractères en Mémoire

Les polices doivent être effacées avant de fermer la station de travail.

vs_clip flag,x1,y1,x2,y2 Spécifie un rectangle de clipping

Fonctions de Sortie

v_pline compteur Polyline

Les coordonnées doivent être copiées dans intin avant l'appel

v_pmarker compteur Polymarqueur

Les coordonnées doivent être copiées dans intin avant l'appel

v_gtext x,y,chaîne.L Texte

La chaîne doit être terminée par zéro.

v_fillarea compteur Remplit une Surface

Les coordonnées doivent être copiées dans intin avant l'appel

v_contourfill x,y,index Remplissage Contour

vr_recl x1,y1,x2,y2 Remplissage Rectangle

v_bar x1,y1,x2,y2 Rectangle

v_arc x,y,rayon,debut,fin Arc

v_pieslice x,y,rayon,debut,fin Camembert

v_circle x,y,rayon Cercle

v_ellarc x,y,rayonx,rayony,debut,fin Arc d'Ellipse

v_ellpie x,y,rayonx,rayony,debut,fin Camembert Elliptique

v_ellipse x,y,rayonx,rayony Ellipse

<code>v_rbox x1,y1,x2,y2</code>	Rectangle arrondi
<code>v_rfbbox x1,y1,x2,y2</code>	Rectangle Rempli Arrondi
<code>v_justified x,y,chaîne.L,longueur,ws,cs</code>	Texte Graphique Justifié

Fonctions d'attributs

<code>vswr_mode</code>	Mode d'écriture
<code>vs_color index,rouge,vert,bleu</code>	Définit une couleur
<code>vsl_type style</code>	Type de Polyline
<code>vsl_udsty frame</code>	Définit une Trame de Ligne
<code>vsl_width largeur</code>	Largeur de Polyline
<code>vsl_color index</code>	Couleur de Polyline
<code>vsl_ends debut,fin</code>	Style de Fin de Polyline
<code>vsm_type symbole</code>	Type de Polymarqueur
<code>vsm_height hauteur</code>	Hauteur de Polymarqueur
<code>vsm_color index</code>	Couleur de Polymarqueur
<code>vst_height hauteur</code>	Hauteur de Texte, Mode Absolu
Le tableau <code>ptsout</code> contient la taille sélectionnée	
<code>vst_point hauteur</code>	Hauteur de Texte, Mode Points
Le tableau <code>ptsout</code> contient la taille sélectionnée	
<code>vst_rotation angle</code>	Orientation du Texte
<code>vst_font police</code>	Sélectionne Police de Texte
<code>vst_color index</code>	Couleur du Texte
<code>vst_alignment horizontal,vertical</code>	Alignement du Texte
<code>vsf_interior style</code>	Style de Remplissage Intérieur
<code>vsf_style index</code>	Index de Style de Remplissage
<code>vsf_color index</code>	Couleur de Remplissage

vsl_perimeter visu

Visibilité du Périmètre de Remplissage

vsl_udpat

Définir une Trame de Remplissage

Le tableau intin doit contenir la trame, et contrl[3] doit être initialisé correctement.

Fonctions Raster

vro_cpyfm mode,source.L,dest.L

Copie Ecran, Opaque

Cette fonction est généralement utilisée pour effectuer des défilements d'écran. Le paramètre mode a généralement la valeur 3 pour un mode remplacement. Les paramètres source et destination pointent vers des structures MFDB décrivant le format de la mémoire à déplacer. Une structure MFDB est constituée de dix mots :

- 0 mot de poids fort de l'adresse de la MFDB
- 2 mot de poids faible de l'adresse de la MFDB
- 4 largeur du bloc en pixels
- 6 hauteur en pixels
- 8 largeur en mots
- 10 mode, généralement 1
- 12 nombre de plans
- 14-18 réservés, à 0

L'adresse du bloc pointe soit vers l'écran soit vers une zone mémoire destinée à recevoir un bloc écran. Les largeur et hauteur doivent être compatibles avec la taille de l'écran. Le nombre de plans peut être obtenu en appelant la fonction vq_extnd 1 dans intout[4]. Pour effectuer des défilements, les pointeurs source et destination peuvent pointer vers la même MFDB.

Les rectangles source et destination doivent être placés dans le tableau ptsin sous la forme xh, yh, xb, yb pour chaque rectangle.

vr_cpyfm mode,source.L,dest.L,i1,i2

Copie Ecran, Transparent

vr_trnfm source.L,destination.L

Transforme Bloc Ecran

v_get_pixel x,y

Lit Couleur d'un Point

Fonctions d'Entrée

vex_timv nouveau_timer

Echange Vecteur d'Interruption Timer

v_show_c reset

Affiche Curseur

v_hide_c

Cache Curseur

<code>vq_mouse</code>	Etat du Bouton Souris
<code>vex_butv nouveau_bouton</code>	Echange Vecteur Bouton
<code>vex_curv nouveau_curseur</code>	Echange Vecteur Changement de Curseur
<code>vq_key_s</code>	Etat du Clavier

Fonctions de Renseignements

<code>vq_extnd etat</code>	Demande de Renseignements
<code>vq_color index,etat</code>	Représentation Couleur
<code>vql_attributes</code>	Attributs Polyline
<code>vqm_attributes</code>	Attributs Polymarqueur
<code>vqf_attributes</code>	Attributs de Remplissage
<code>vqt_attributes</code>	Attributs de Texte
<code>vqt_extent chaîne.L</code>	Attributs Texte Etendu
<p>La chaîne doit être terminée par 0. Le résultat se trouve dans ptsout.</p>	
<code>vqt_width caractere</code>	Largeur Caractère
<code>vqt_name nombre</code>	Nom et Index de police
<code>vqt_fontinfo</code>	Infos sur une Police

Squelette de Programme AES & VDI

La structure d'un programme-type GEM est :

Initialisation :

- Réduire la mémoire
- Appeler `appl_init`
- Initialiser `current_handle` à partir du résultat de `graf_handle`
- Ouvrir une station virtuelle en utilisant `current_handle`
- Eventuellement ouvrir une fenêtre

Programme Principal :

- Attendre et gérer les événements

Fin :

Fermer les fenêtres
Fermer la station virtuelle
Appeler `appl_exit`
Appeler `Pterm`

Accessoires de Bureau

Un accessoire de bureau est un fichier exécutable avec l'extension `.ACC` et qui est chargé lors de l'initialisation de l'AES. Nous n'avons jamais vu de documentation officielle sur les accessoires de bureau, et les informations suivantes ont été acquises par l'expérience, et en programmant notre accessoire de bureau `Saved!`.

La première chose à prendre en compte est qu'un accessoire n'est pas un programme standard GEMDOS. Lors du lancement d'un accessoire, tous les registres sont à zéro, y compris `A7`, et sauf `A0` qui pointe sur la page de base. Un accessoire doit contenir toute la mémoire dont il a besoin, en la plaçant par exemple dans le segment `BSS`. Il est interdit de faire appel aux fonctions GEMDOS `Mshrink` ou `Pterm`.

La boucle principale d'un accessoire est, comme pour toute application GEM, une boucle attendant et traitant des événements. Notez que les numéros des événements `AC_OPEN` et `AC_CLOSE` sont respectivement 40 et 41, contrairement à ce qui est spécifié dans de nombreuses documentations.

De nombreux programmeurs d'accessoires ont rencontré des problèmes en utilisant les fonctions `VDI`. Nous ne vous conseillons d'ouvrir une station de travail virtuelle que si vous en avez besoin (par exemple avant d'ouvrir une fenêtre) et de la refermer quand vous n'en avez plus besoin (en refermant la fenêtre ou en recevant `AC_CLOSE`). L'exemple d'accessoire fourni n'utilise pas de fonction `VDI`. Paranoïa, quand tu nous tiens !

Si votre accessoire est activé par des événements timer, n'appellez des fonctions GEMDOS (`Trap #1`) que si votre fenêtre est active et au-dessus des autres. Sinon des bombes apparaissent et un plantage est fortement probable.

Le fichier `DESKACC.S` est un programme source d'accessoire. Il affiche simplement la mémoire libre dans une boîte d'alerte. Il contient le symbole `RUNNER` qui, lorsque sa valeur est à 1, génère un programme normal au lieu d'un accessoire. Cela peut être d'une aide inestimable pour déboguer votre programme. En effet, déboguer un programme normal est beaucoup plus facile que déboguer un accessoire. Et vous pourrez déboguer votre programme avec ses symboles, chose impossible dans le cas d'un accessoire.

Chaîner avec les Bibliothèques AES & VDI

Lorsque vous utilisez les fonctions GEM dans vos programmes, vous devez inclure au début le fichier GEMMACRO.S. Si vous assemblez directement votre programme sous la forme d'un programme exécutable, vous devez également inclure les fichiers AESLIB.S et VDI.LIB.S à la fin de votre programme. Mais si vous assemblez votre programme sous la forme de fichier objet, n'incluez pas ces deux fichiers. Regardez l'exemple GEMTEST.S qui vous montrera comment déterminer automatiquement si ces deux fichiers doivent être inclus ou pas.

Si vous développez une application sous GEM, nous vous recommandons d'assembler directement vos fichiers sous forme d'exécutable pour gagner du temps. Mais la taille de votre exécutable peut être fortement réduite si vous créez un fichier objet, que vous chaînez sélectivement avec la bibliothèque GEMLIB.BIN.

Par exemple, si vous assemblez GEMTEST.S sous la forme d'un fichier objet GST, vous le chaînez avec la bibliothèque GEM en passant la ligne de commande suivante à LinkST :

```
gemtest -wgemlib
```

Le fichier de contrôle GEMLIB.LNK s'occupera de tout.

Si vous désirez vraiment réduire la taille de votre exécutable au strict minimum, vous pouvez supprimer certaines parties de la bibliothèque GEM, car son source est fourni.

Compilateur de Menus

Ce programme vous permet de créer des menus sans utiliser d'éditeur de ressource. MENU2ASM.TTP transforme un fichier de description de menu en source assembleur destiné à être inclus dans vos programme.

Un fichier de description de menu est un fichier texte comportant l'extension .MDF. Voici un exemple de tel fichier :

```
[ Bureau | A Propos... ]  
[ Fichier | Nouveau \ Charger \ (-----\ Quitter ]  
[ Recherche | Chercher \ Remplacer ]  
etc...
```

Les fins de lignes sont ignorées.

Chaque titre de menu accompagné de ses sous-menus est encadrée par des crochets [et]. Une barre verticale | sépare un titre de son sous-menu. Les différentes entrées d'un sous-menu sont séparées par un caractère \. Les entrées de menu invalides (affichées en gris) doivent être précédées d'une parenthèse ouvrante (. Le premier menu est toujours le menu bureau (de nom Bureau).

Les accessoires de bureau seront ajoutés par l'AES. Cette syntaxe est la même que celle utilisée par nos compilateurs BASIC, et ce n'est pas une coïncidence !

Nous vous recommandons de faire précéder chaque entrée de menu de deux espaces et d'ajouter au moins un espace à la fin. Les titres de menu doivent être précédés et suivis d'un espace.

Pour compiler un menu, double-cliquez sur MENU2ASM.TTP et entrez le nom de fichier à compiler sans extension. Il produira un fichier source assembleur possédant l'extension .MNU destiné à être inclus dans vos programmes.

Le fichier MENUTEST.MDF contient un exemple de définition de menu destiné à être utilisé avec le programme MENUTEST.S.

Anciennes Bibliothèques GenST AES & VDI

Le dossier OLDGEN contient les versions mises à jour des fichiers sources fournis avec DevpacST 1. Les conventions d'appels des fonctions GEM sont différentes. Ces fichiers sont fournis pour les utilisateurs de DevpacST 1 qui possèdent maintenant la version 2.

Codes Ecran VT52

Lorsque vous écrivez à l'écran en utilisant les fonctions BIOS ou GEMDOS, le gestionnaire d'écran émule les séquences de caractères VT52. Les codes de contrôle sont envoyés en utilisant des séquences d'échappement, ce qui signifie que vous devez envoyer le caractère d'échappement (27 en décimal, ou \$1B) suivi d'un ou de plusieurs autres caractères.

ESC A	curseur haut; aucun effet sur la première ligne
ESC B	curseur bas; aucun effet sur la dernière ligne
ESC C	curseur droit; aucun effet sur la dernière colonne
ESC D	curseur gauche; aucun effet sur la première colonne
ESC E	efface l'écran et place le curseur en haut à gauche de l'écran
ESC H	place le curseur en haut à gauche de l'écran
ESC I	déplace le curseur d'une ligne vers le haut; si le curseur est sur la première ligne, fait défiler l'écran d'une ligne vers le bas.
ESC J	efface l'écran depuis le curseur jusqu'à la fin de l'écran
ESC K	efface la fin de la ligne
ESC L	insère une ligne en déplaçant les lignes suivantes vers le bas; le curseur est positionné au début de la nouvelle ligne
ESC M	détruit une ligne et remonte les lignes suivantes
ESC Y	change la position du curseur; doit être suivi de deux caractères représentant les numéros de ligne et de colonne. La numérotation des lignes et colonnes commence en haut à gauche de l'écran à la position (32,32)
ESC b	couleur caractère; doit être suivi d'un caractère indiquant la couleur dont seuls les quatre bits de poids faible sont pris en compte
ESC c	couleur du fond; similaire à ESC b
ESC d	efface l'écran depuis l'origine jusqu'à la position du curseur
ESC e	autorise le curseur
ESC f	supprime le curseur
ESC j	sauve la position du curseur
ESC k	restaure la position du curseur sauvée par ESC j
ESC l	efface une ligne et place le curseur en début de ligne
ESC o	efface le début de la ligne jusqu'à la position du curseur
ESC p	inversion vidéo
ESC q	fin d'inversion vidéo
ESC v	autorise le retour de ligne automatique
ESC w	supprime le retour de ligne automatique

Appendice E

Portage depuis d'autres Assembleurs

La plupart des assembleurs sur ST suivent à leur manière le standard Motorola. Les instructions sont les mêmes, mais la syntaxe des symboles, commentaires et directives varient. Cet Appendice décrit les changements à apporter à vos programmes écrits avec d'autres assembleurs pour les faire fonctionner avec DevpacST, que ce soit des programmes que vous avez vous-mêmes écrits ou que vous avez recopiés dans des magazines. En fait, l'assembleur standard utilisé dans les magazines est DevpacST. Vous n'aurez donc pas de problème pour les faire assembler par GenST !

Nous n'étudierons pas ici les différences d'utilisation ou les différentes options des autres assembleurs, mais simplement la manière de convertir vos programmes.

Atari MADMAC

GenST ne nécessite pas de caractère deux-points après les noms de symboles. Les commentaires ne commencent pas nécessairement par un point-virgule, mais il n'autorise pas les instructions commençant en colonne 1.

La syntaxe des symboles locaux est la même, mais GenST n'autorise pas les caractères \$ et ? dans les symboles. L'utilisation des \ dans les chaînes doit être modifiée, et la priorité de quelques opérateurs est différente.

MADMAC autorise les directives à commencer par un point. Si vous supprimez ce point, la plupart des directives sont identiques. Pour celles qui diffèrent, voici leur équivalent GenST :

BSS=SECTION BSS, DATA=SECTION DATA, TEXT=SECTION TEXT, ABS=OFFSET, ELSE=ELSEIF, ENDIF=ENDC, EXITM=MEXIT, GLOBL et EXTERN=XREF ou XDEF, EJECT=PAGE, TITLE=TTL, NLIST=NOLIST.

INIT peut être transformé en directives DC ou DCB. CARGS peut être remplacé par des directives RS.

La syntaxe des macros de MADMAC est particulière et ses paramètres par noms doivent être modifiés. L'équivalent de \~ est \@, et \# est à remplacer par NARG. \? doit être émulé par IFC ou IFNC. Les options 6502 de MADMAC ne sont pas supportées.

GST-ASM

Les symboles GST ne sont significatifs que sur huit caractères, et aucune différence n'est faite entre les minuscules et les majuscules, donc utilisez l'option OPT C8- de GenST. Les règles d'évaluation d'expressions sont similaires bien que \$ ne soit pas autorisé dans les étiquettes GenST.

La plupart des directives sont identiques sauf PAGEWID=LLEN et PAGELEN=PLEN. Les définitions de macros devront être revues, ainsi que les symboles locaux.

Les fonctions intégrées et les structures ne sont pas supportées.

MCC Assembler

Très peu de modifications sont nécessaires. Seule la syntaxe des étiquettes locales est différente, et vous devez ajouter .L aux directives XREF concernant les symboles absolus.

K-Seka

GenST ne nécessite pas de caractère deux-points après les noms de symbole et il n'autorise pas les instructions commençant en colonne 1. Plusieurs directives Seka utilisent par défaut des tailles Octet (B) au lieu de Mot (W). Les directives à modifier sont :

D=DC, BLK=DS, CODE=SECTION CODE, DATA=SECTION DATA, IF=IFNE, ELSE=ELSEIF, ENDIF=ENDC.

La syntaxe des macros implique que les ? doivent être remplacés par des \, sauf pour ?0 qui doit être remplacé par \@.

Fast ASM

La syntaxe de Fast ASM reprend celle de GenST 1.2. Peu de changements doivent être faits. Vous devez convertir les fichiers sources compactés en ASCII pour les charger dans GenST. Les commentaires commençant par \ doivent être modifiés pour commencer par * ou ;. Les commentaires suivant les instructions n'ont pas besoin d'être modifiés.

L'utilisation des nombres flottants n'est pas supportée par GenST.

Appendice F

Bibliographie

Cette bibliographie contient quelques suggestions de lecture d'ouvrages sur le 68000, le ST et GEM. Les points de vue exprimés sont les nôtres, mais rien ne peut remplacer l'examen du livre dans une bibliothèque pour faire votre propre choix.

Programmation du 68000

Mise en oeuvre du 68000 de C.Vieillefond
Édité par Sybex

Si vous avez toujours voulu tout savoir sur le 68000 sans jamais oser le demander ...

M68000 Programmer's Reference Manual
Publié par Prentice Hall

C'est le guide officiel de chez Motorola sur le jeu d'instructions du 68000. Pour ceux que l'anglais ne rebute pas.

Manuels Techniques ST

Manuel de programmation GEM Volume 1 & 2 - VDI et AES
par Digital Research

C'est l'ouvrage de référence du GEM. Malheureusement, il n'est disponible qu'aux développeurs enregistrés chez Digital Research.

Manuel de Référence de l'Atari ST - Tome 1 - Documentation
Système
par Atari France

C'est l'ouvrage de référence du système ATARI (BIOS, XBIOS, GEMDOS, line A, etc...). Il faut s'adresser à Atari France pour l'obtenir.

Le livre du GEM sur Atari ST
Édité par Micro Applications

Cet ouvrage est la traduction du Manuel de programmation GEM citée plus haut agrémentée de quelques exemples simples. A noter qu'il existe aussi chez le même éditeur d'autres ouvrages sur l'Atari ST.

Programmer en assembleur 68000 sur l'Atari ST par Olivier Hard
Édité par Cedlc/Nathan

Trucs, astuces et exemples simples.

Appendice G

Support technique et mise à jour

Dans le but de maintenir la qualité de notre support technique, voici quelques conseils à suivre pour vous permettre de profiter de tous les avantages que nous réservons aux utilisateurs enregistrés. Nous pourrions ainsi mieux vous aider, et corriger les bogues quand vous nous les signalez pour éviter de faire rencontrer les mêmes problèmes aux autres utilisateurs.

Vous pouvez accéder à notre support technique de trois manières différentes (**Attention, ces supports ne sont accessibles qu'à condition de nous retourner votre carte de garantie**) :

Le support télématique : composez 3615 sur votre Minitel puis le code **HUMAN** et vous pourrez consulter les réponses d'autres utilisateurs ou bien poser les vôtres. Ce service est accessible 24 heures sur 24 et les réponses sont données le soir même. Ce serveur vous informe également sur nos nouveaux produits et sur l'évolution de Devpac. Vous pourrez aussi télécharger des démonstrations de nos logiciels, créer une boîte aux lettres (texte et binaire) ou converser avec d'autres utilisateurs dans des forums.

Le support téléphonique : il se fait exclusivement le mardi et le jeudi matin au (1) 46 04 88 71. Ces horaires étant réservées au support de Devpac, nous ne pourrions répondre à vos questions en dehors de ces deux matinées. D'autre part, il est très fortement conseillé avant de nous appeler de vérifier sur le support télématique s'il n'a pas déjà été répondu à une question similaire à la votre.

Le support par courrier : A n'utiliser qu'en dernier recours. N'oubliez pas d'inscrire lisiblement vos noms et adresses sur la lettre. Tous documents ou disquettes qui nous auront été envoyés ne seront pas retournés.

Quelle que soit la méthode que vous utilisez, indiquez toujours le numéro de série se trouvant sur votre disque principal, et le numéro de version de votre programme. Nous nous réservons le droit de refuser tout support si vous ne pouvez nous fournir ces informations.

Si vous nous signalez un bogue, exécutez le programme CHECKST.PRG et indiquez nous les informations renvoyées, ainsi que le détail de votre configuration comme les accessoires et les programmes résidents utilisés.

Si vous pensez avoir trouvé un bogue, essayez de créer un petit programme reproduisant ce problème et envoyez le nous. Il est toujours plus facile pour nous de répondre aux questions sur des exemples précis que lorsque cela reste dans le vague.

Mise à jour

Comme pour tous nos produits, DevpacST est continuellement amélioré. Périodiquement, de nouvelles versions sont disponibles. Une faible participation aux frais est demandée pour les mises à jour, sauf dans le cas où une nouvelle documentation est incluse auquel cas le prix est un peu plus élevé. Tous les utilisateurs qui nous renvoient leur carte d'enregistrement seront avertis des nouvelles versions de nos programmes.

Suggestions

Nous accueillons volontiers tout commentaire ou suggestion sur nos produits. Et pour être sûr que nous nous en souviendrons, il est préférable que vous nous les fassiez parvenir par courrier.

Appendice H

Historique de DevpacST

Historique du Produit

DevpacST 0.50 est sorti fin 1985, avec de nombreuses restrictions concernant l'éditeur et la production de code objet. La principale version suivante fut la 0.91 améliorée sous bien des points, suivie par la 0.99f, dernière version qui ne pouvait pas produire de code objet. La version 1.0 est sortie en avril 1986 et ne comporta que peu de modifications avant la version 1.22 de juin 1986 comportant pour la première fois une documentation dans un classeur à anneaux. Après diverses versions mineures, la version 2.0 nettement améliorée est sortie en avril 1988. La version française est sortie en septembre 1990.

Développement de Devpac

DevpacST est basé sur DevpacQL, notre assembleur sur QL. GenST et MonS ont été écrits en assembleur sur QL, puis chargés sur ST en utilisant une ligne série. LinkST a été écrit en C avec le compilateur Lattice sur ST. Le développement a été porté sur ST à partir de la version 1.24 comportant quelques modifications, jusqu'à la version 1.26. Des versions internes ont été écrites pour tester le code objet ou le format de débogage étendu jusqu'à la version 1.57. Ensuite, MonST et GenST ont été complètement réécrits pour la version 2. L'éditeur de GenST a été considérablement modifié. Il provient de Hisoft Basic puis Power BASIC. L'éditeur est écrit entièrement en assembleur.

Résumé des améliorations de la version 2

Le contenu de ce paragraphe a pour but de donner un tour d'horizon rapide des principales améliorations apportées par rapport à la version 1.2 de DevpacST. Pour de plus amples informations, reportez vous au paragraphe correspondant dans ce manuel.

L'éditeur

Voici résumé les principales améliorations apportées à l'éditeur de GenST:

- Amélioration de la rapidité d'affichage.
- Traitement de ligne pouvant avoir jusqu'à 240 caractères.
- Fonctionnement en basse résolution.
- Présence d'une barre de défilement horizontal afin de pouvoir visualiser les lignes dans toute leur ampleur.
- *Standardisation* du fonctionnement de la barre de défilement vertical.
- Configuration par défaut des touches du pavé numérique en touche curseur type IBM.
- Possibilité de modifier la taille de la zone mémoire d'édition directement à partir du programme. Ceci, grâce conjointement à la sauvegarde des Préférences rend obsolète le programme d'installation de la version 1.
- Possibilité de lancer d'autres programmes à partir de l'éditeur en utilisant l'option Lancer autre.
- L'effacement de bloc est maintenant accessible par `Shift-F5` au lieu de `Shift-F3` (au grand bonheur des gauchers).
- Présence d'un processus de Couper/Copier/Coller sur les blocs (une ligne effacée peut, par exemple, être rappelé autant de fois que nécessaire).
- Visualisation à l'écran des blocs sélectionnés (en inverse vidéo).
- Toutes les commandes des menus possèdent un raccourci clavier, à l'exception de Effacer fichier et Remplacer tout pour des raisons de sécurité bien compréhensible.

L'assembleur

Voici résumé les principales améliorations apportées à l'assembleur :

- Possibilité d'avoir pour les symboles jusqu'à 127 caractères significatifs.
- Possibilité d'utiliser des symboles locaux.
- La directive INCBIN en permettant d'inclure directement dans le source un fichier binaire facilite grandement l'insertion dans un programme d'un nombre de données important (comme par exemple des copies d'écran, des icônes, etc...).
- Amélioration de la rapidité d'assemblage
 - Assemble maintenant jusqu'à 75.000 lignes par minute en absolu, soit 35.000 lignes par minute sur de véritables programmes.
 - Les accès disque sont limités au maximum (utilisation d'une mémoire cache, particulièrement sensible lorsque l'assemblage s'effectue sur disquette), entre autre lors de la génération du fichier en sortie.

Les fichiers en-tête ne sont plus lus qu'une seule fois

Améliorations des algorithmes de recherche et de traitement de la table des symboles.

- Possibilité d'utiliser une extension au format DRI de la table des symboles (*Débogage Etendu Hisoft*), permettant de déboguer avec des symboles jusqu'à 22 caractères significatifs.
- Les appels de macros et les insertions de fichiers en-tête peuvent être imbriqués autant de fois que la mémoire reste suffisante pour le faire.
- Les sections TEXT, DATA et BSS sont maintenant correctement supporté lorsque l'on génère du code exécutable.
- Modules et sections multiples - le format d'objet GST est beaucoup mieux supportés, autorisant les modules et les sections multiples.
- Possibilité d'assembler et de créer des fichiers objets au format DRI.
- Les symboles externes peuvent maintenant être utilisé dans les expressions, y compris entre eux si besoin est.

- L'optimisation est maintenant capable d'agir sur les branchements (qu'il passe en court lorsque cela est possible), sur les adressages (transformation d'adressage long en adressage court), sur les instructions (transformation d'un MOVE #x,Dn par un MOVEQ si $-128 \leq x \leq 127$, d'un ADD #x ou d'un SUB #x par un ADDQ ou SUBQ si $1 \leq x \leq 8$), etc ...
- Les macros acceptent dorénavant jusqu'à 36 paramètres et supportent les substitutions numériques. La mémoire tampon affecté aux macros est dynamique au lieu d'utiliser la partie inutilisée de la zone mémoire d'édition.
- GenST accepte maintenant les structures de répétitions (REPT), la directive REG (pour entrer une liste de registre - destinée par exemple à l'instruction MOVEM), les nombres en octal.
- L'évaluateur d'expression possède des opérateurs de comparaisons.
- Considérable augmentation des options de la directive OPT.
- Les registres peuvent aussi être référencés par R0 - R15.
- Une version *en ligne* de l'assembleur est fourni pour ceux qui utilisent un environnement texte (*shell*) ou des fichiers *batch*.

Compatibilité

La plupart des sources devrait être assemblé avec sans ou peu de modifications. Vous devrez pour cela porter votre attention sur les points suivants :

- Les sections BSS - les directives RSBSS et DSBSS n'existent plus, vous devez les remplacer par SECTION BSS puis utiliser des directives DS.
- Les symboles sont significatifs sur 127 caractères et sont évalués, par défaut, en tenant compte des majuscules et des minuscules.
- La directive OPT N doit être remplacée par FORMAT qui permet un meilleur contrôle du format de sortie du listing.
- La directive ORG a été modifiée.
- Utilisateurs de HiSoft Basic - Les fichiers objets générés par GenST2 ne sont pas acceptés par les versions du programme BUILDLIB antérieure à 1.4.

- Les instructions `BRA.W` et `BRA.B` sont maintenant acceptées. Comme elle ne l'étaient pas dans la version 1, vous avez été forcé d'utiliser la forme `BRA.L` qui est à proprement parlé une instruction 68020 et qui va provoquer dorénavant un message d'avertissement de la part de GenST2.
- Certaines modifications mineures ont été apportées à l'analyse des instructions vous forçant à reconsidérer certaines notations.
 - Pour réaliser un adressage court dans une instruction au sein d'une macro, utilisez la notation `(expression).W` (standard Motorola) au lieu de `(expression)\W`. En effet `\W` et `\L` vont être considéré comme paramètre de la macro et seront sans doute remplacé par 0 lors de son évaluation. De même, ne les utilisez pas non plus comme attributs de taille derrière des substituts de registres, utilisez la notations `.W` ou `.L` (Par exemple, si `buf` est un substitut de registre, vous devez écrire `move.b d0,0(a6,buf.1)`)
 - Les substituts de registre peuvent être utilisés dans les instruction `MOVEM`.
 - La priorité de l'opérateur `=` a été modifiée.
- Le programme d'exemple `GEM` a été modifié (il utilise maintenant une véritable section `BSS` et fonctionne correctement sous `GDOS`) et peut être trouvé dans le répertoire `OLDGEM`. De nombreux autres fichiers d'exemple, comme un accessoire de bureau et de nouvelles bibliothèques `AES` et `VDI` vous sont également fournis.
- GenST signale maintenant en première passe les erreurs de syntaxe, et ne commence pas la deuxième s'il en trouve. Il utilise désormais les routines `GEMDOS` d'affichage, qui peuvent être suspendu par `Ctrl-S`, relancé par `Ctrl-Q` et interrompu par `Ctrl-C`.
- La présence de nouvelles directives peut potentiellement générer des conflits avec les noms de vos macros contenus dans des sources GenST1. Ces directives sont : `COMMENT`, `DCB`, `ELSEIF`, `ENDR`, `FORMAT`, `IIF`, `INCBIN`, `OFFSET`, `OUTPUT`, `REG`, `REPT`, `RSSET`, `SUBTTL`.
- La présence de nouveaux symboles réservés peut conduire au même résultat. Ils commencent tous par 2 caractères souligné et sont : `__LK`, `__RS` et `__G2`.

Le Débogueur

MonST a été amélioré sur de nombreux points incluant le multi-fenêtrage qui a considérablement modifié et simplifié l'interface. Reportez vous au paragraphe **Référence** du **chapitre 4** avant de commencer tout travail un tant soit peu sérieux avec cette nouvelle version. Voici maintenant résumé les principales améliorations apportées au débogueur :

- Multi-fenêtrage
- Possibilité de temporiser les échanges d'écrans afin d'éviter des clignotements intempestifs.
- Evalueur d'expressions complet y compris avec des indirections.
- Autorise jusqu'à 22 caractères significatifs pour les symboles
- Fonctionnement multi-résolution, permettant, par exemple, de déboguer en moyenne résolution un programme tournant en basse (et vice versa).
- Possibilité de visualisation du fichier source.
- Désassemblage (au format GST) imprimante ou disque avec génération automatique de symboles.
- Points d'arrêts conditionnels.
- Historique.
- Possibilité d'interruption de programme en cours d'exécution.
- Les versions GEM et TOS ne diffèrent plus que par l'extension de leurs fichiers.

Intégration

L'intégration des composantes est probablement la plus importante amélioration apporté à Devpac. L'assembleur (comme avant) et le débogueur sont maintenant accessible à partir de l'éditeur par pression d'une touche. Il est possible d'assembler et d'exécuter directement en mémoire du code exécutable, sans faire un seul accès disque. Il est même possible d'y inclure des informations de déboguage destinées à MonST, accélérant considérablement le cycle développement: assemblage/déboguage.

Les messages d'erreurs et d'avertissements sont maintenant récupérés par l'éditeur, vous permettant ainsi de les passer en revue en appuyant juste sur une touche (en l'occurrence Alt-J) sachant qu'après un assemblage, l'éditeur se positionnera automatiquement sur la ligne contenant la première erreur. Les erreurs ne sont plus perdues lorsque le nombre de ligne est modifié (elles ne sont cependant pas recalculées).

L'éditeur de liens

Il supporte maintenant le format de *Déboguage Etendu d'HiSoft* et est plus rapide que ses prédécesseurs. Il accepte aussi l'ordonnancement explicite des sections et de véritables sections BSS. Veuillez maintenant remarquer que LinkST ne fonctionne qu'avec des fichiers objets au format GST et que si vous désirez chaînez des fichiers objets au format DRI, vous devez utiliser les éditeurs de liens ALN d'Atari ou LINK68 de Digital Research.

HiSoft Devpac Atari

Assembleur/Editeur/Débugueur/Editeur de Liens Version 2

Devpac 2 de HiSoft (High Quality Software) est l'Assembleur pour votre Atari. Il existe en deux versions : l'une pour Atari ST, l'autre pour Atari TT. Il comprend un macro assembleur 68000 (68030 en version TT), un éditeur de texte plein écran, un puissant désassembleur et débogueur, et un éditeur de lien rapide.

Devpac est l'Assembleur le plus utilisé sur Atari, par le néophyte désireux s'initier au langage Assembleur, mais aussi par les sociétés d'édition de jeux ou de logiciels, dont Human Technologies. La version 2 de Devpac apporte beaucoup de nouvelles fonctions et d'améliorations rarement vues sur Atari:

- ✓ Assemble jusqu'à 70 000 lignes par minute (35 000 lpm pour un fichier volumineux comprenant de nombreux symboles) - aucune attente pour le programmeur!
- ✓ Nouvelles fonctionnalités d'assemblage: étiquettes locales, sections multiples, jusqu'à 127 caractères significatifs pour les étiquettes et meilleur contrôle du listing.
- ✓ Editeur ultra-rapide comprenant un véritable sélecteur de fichiers, sélection de bloc en vidéo inverse; facilement reconfigurable.
- ✓ Les modules, assembleur, débogueur et éditeur, sont totalement intégrés pour développer très facilement et rapidement.
- ✓ Le débogueur est entièrement nouveau et permet le multi-fenêtrage, les points d'arrêt conditionnels, l'évaluation d'expression, le désassemblage sur disque et beaucoup plus...

Devpac Atari version 2 est livré avec un manuel de plus de 180 pages en français détaillant l'utilisation de tous les modules avec un résumé du système d'exploitation et un aide-mémoire sur le jeu d'instructions du 68000 (68030 pour la version TT).

Pour la découverte ou pour une utilisation professionnelle de l'Assembleur sur Atari, Devpac 2 est l'outil de développement qu'il vous faut!

