

**DANS
CE LIVRE :
LA DISQUETTE**

TRUCS ET ASTUCES

**360 KO
DE PROGRAMMES
SUR LA DISQUETTE**

EDITIONS MICRO APPLICATION



LIVRE DATA BECKER

Martin Pauly
Frank Schepers
Jürgen Schulz

Trucs et Astuces ST II

Editions Micro Application

MICRO APPLICATION

58, Rue du Faubourg Poissonnière
75010 PARIS

© Reproduction interdite sans l'autorisation de
MICRO APPLICATION

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

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

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

ISBN : 2-86899-193-9

© 1988 DATA BECKER
Merowingerstrasse, 30
4000 Düsseldorf - RFA

Auteurs : Martin Pauly, Frank Schepers, Jürgen Schulz
Traduction française assurée par : Monsieur et Madame Baudin

© 1989 MICRO APPLICATION
58 Rue du Faubourg Poissonnière
75010 PARIS

Collection dirigée par Mr Philippe OLIVIER
Edition réalisée par Frédérique BEAUDONNET

IBM PC, IBM PC XT, IBM PC AT sont des modèles déposés de International Business Machines.
Atari, ST, Méga ST et GDOS sont des marques déposées par Atari Corp.
C64 et Amiga sont des modèles déposés de Commodore Electronics Limited.
GEM et GEM Desktop sont des marques déposées de Digital Research.
First Word et First Word Plus sont des marques déposées de GST Holding Limited.
Motorola est une marque déposée et MC68000 est une marque de Motorola Inc.
Mégamax C est une marque déposée de Mégamax Inc.
GFA BASIC et GFA ASSEMBLEUR sont des marques déposées de GFA Systemtechnik GmbH.
Degas est une marque déposée de Batteries Inc.
STAD est une marque déposée de Application Systems.
Devpac II est une marque déposée de Hisoft.
MS DOS est une marque déposée de Microsoft Corporation.

Préface

Depuis 1985, date de commercialisation du 520 ST, la petite machine de Jack Tramiel a conquis sans effort une foule d'adeptes. Depuis les spécialistes de la programmation en passant par les éditeurs de logiciels et les revues spécialisées, l'on pourra dire qu'une littérature abondante aura été publiée sur le sujet. Si bien qu'aujourd'hui, peu d'informations techniques échappent encore aux investigations des "ST-Maniaques".

Néanmoins l'utilisateur avait à souffrir d'une trop grande dispersion des informations. Longs à retrouver, éparpillés parmi des centaines d'ouvrages et d'articles, connus parfois des seuls initiés, certains bons "tuyaux" ne franchissaient guère les cercles étroits des clubs informatiques. Cela ne pouvait plus durer ; tout utilisateur un tant soit peu programmeur se devait d'avoir accès à cette mine de trésors enfouis, en redécouvrant par la même occasion les possibilités secrètes de leur machine favorite.

L'ouvrage que voici accomplit le tour de force de réunir les meilleurs trucs et astuces se rapportant à l'Atari ST. Certains sont étonnants ; d'autres si pratiques que l'on imagine difficilement après coup avoir pu vivre un instant sans eux.. TRUCS ET ASTUCES II vous permettra, sans négliger les connaissances de base, d'accéder rapidement à l'état de grâce en matière de développements. Vous accéderez même à certains conseils que l'on trouverait difficilement ailleurs.

Ce livre s'adresse sans distinction à tous les utilisateurs de la série ST y compris le Méga ST. Sans trop risquer le ridicule, l'on peut décemment prévoir qu'il trônera plusieurs années durant sur votre table de travail juste à côté de votre ordinateur, et ce pour trois raisons :

- ① Les trucs et astuces ne se limitent pas à un seul langage de programmation. Les programmes, exemples ou utilitaires proposés vous sont d'une part parfaitement expliqués, et font d'autre part appel à différents langages de programmation, aisément adaptables selon vos préférences en la matière.
- ② Nous avons essayé de réunir les informations les plus intéressantes dans le but de faciliter votre travail quotidien d'utilisateur ou de développeur. Nous nous sommes efforcés par ailleurs à ne pas vous accabler d'informations trop techniques. Au fil des pages, votre savoir s'enrichira progressivement, sans heurts.

Nous avons enfin choisi de travailler avec les langages GFA Basic, C et Assembleur. Vous trouverez de plus des conseils destinées aux débutants, des astuces concernant les logiciels standard du marché, un savoir pratique approfondi pour exploiter pleinement les ressources du système d'exploitation de l'ATARI.

◆ **Important !**

Les programmes donnés dans ce livre ont été écrits à l'aide des langages suivants:

- ⇨ Basic-GFA 2.0 et 3.0.
- ⇨ Mégamax-C-Compiler 1.1
- ⇨ GFA Assembleur et Devpac II

◆ **Utilisation de la disquette jointe à ce manuel**

Afin que vous puissiez disposer immédiatement des programmes contenus dans ce livre, il était indispensable d'y adjoindre une disquette Cette dernière propose des programmes ou accessoires compilés afin de ne pas vous pénaliser dans le cas où vous ne disposeriez pas de l'interpréteur ou du compilateur nécessaire.

Une seconde disquette, plus complète peut vous être fournie sur demande.

Sommaire

1. Le chargement automatique des programmes Le dossier AUTO	11
1.1. La mise-à-jour et la sauvegarde du calendrier	12
1.2. L'installation automatique de l'imprimante	17
1.3. L'installation du GDOS et des accessoires	19
1.4. Le chargement automatique des applications GEM	21
2. Les utilitaires de gestion de l'unité de disque dur ou de disquette	27
2.1. Dissimuler des fichiers	27
2.2. Encoder des fichiers	29
2.3. Un 'floppy-speeder'	33
2.4. Renommer un dossier	35
2.5. Programme de copie automatique dans disque RAM	43
3. Programmes de conversion	53
3.1. Conversion de 1st Word Plus en ASCII	59
3.2. Programme de conversion ASCII ordinaire ASCII vers 1st-Word-Plus	61
3.3. Conversion de n'importe quelle sorte de fichiers	62
4. Trucs et astuces concernant les logiciels standard	67
4.1. Un programme 'REM-Killer' pour le Basic-GFA	67
4.2. Générateur de lignes Data	70
4.3. Compiler des programmes GFA avec protection List	73
4.4. Programmation de touches	74
5. Trucs et astuces : généralités	87
5.1. Faire une copie d'écran	87
5.2. L'affichage des caractères ASCII ayant un code inférieur à 32	88
5.3. Déterminer la position du pointeur de la souris	89
5.4. Activer et désactiver le pointeur de la souris	90
5.5. Une souris à la demande	91
5.6. Régler la vitesse d'exécution de la souris	92
5.7. Un nouveau pointeur pour la souris	95

5.8. Contrôle de l'état de la manette de jeu en Basic-GFA 2.xx	104
5.9. Contrôle de l'état de la manette de jeu en Assembleur	105
5.10. Freezer et reset par une simple touche	109
5.11. Pour épargner votre écran	113
5.12. Le véritable multitasking sur l'Atari ST	116
5.13. Utilitaire d'impression	121
5.14. Pour recevoir des messages d'erreur à la place des bombes	136
5.15. Pour passer un moniteur couleur de 50 à 60 Herz	140

6. Trucs et astuces concernant le GEM 143

6.1. Les jeux de caractères du GDOS et du GEM	143
6.2. Pour confectionner votre propre bureau GEM	148
6.3. Un accessoire de recherche automatique de fichier dans le disque dur	152
6.4. Programmation simple des fenêtres	170
6.5. Confection d'un accessoire	186

7. Trucs et astuces de programmation 189

7.1 Le clavier	189
7.1.1 Les touches spéciales	190
7.1.2 Les touches de fonction et les touches fléchées	192
7.1.3 Pour modifier le jeu des touches	193
7.1.4 Une mémoire-tampon pour le clavier	195
7.2 Les fichiers batch	198
7.3 Les programmes-résidents et le vbl-queue	199
7.3.1 Le retour-image (VBL)	202
7.3.2 Le retour de ligne (HBL)	208
7.3.3 L'horloge du MFP	213
7.3.4 La commande Trap	216
7.3.5 La RAM de l'écran	219
7.3.6 L'appel du système d'exploitation à partir des programmes résidents	220
7.4 L'heure à la seconde près	221
7.5 Movem	224
7.6 Rechargement de programmes	226
7.7 Les sous-programmes en assembleur	231
7.7.1 Paramètres et variables locales	231
7.7.2 Entrée et sortie numérique	240
7.7.3 Exemples de récursion	245
7.8 Le Basic-GFA et l'Assembleur	249
7.9 Simulation d'une roulette	256
7.10 Programmation sonore sur l'Atari	270
7.10.1 La commande sound en Basic GFA	271
7.10.2 Variez les sons en jouant sur la période de l'onde sonore	272
7.10.3 Nouvelles possibilités par 'Wave'	272

7.10.4	Sound en langage C et en assembleur	.275
7.10.5	Musique en interrupt	.277
7.11	Programmation graphique	.279
7.11.1	Affichage d'Images en n'importe quel format	.280
7.11.2	Insérer des "Doodles" dans vos propres programmes	.287
7.11.3	Des graphiques stables	.291
7.11.4	Bande annonce	.296
7.11.5	Affichage clignotant	.300
7.11.6	Fondu enchaîné des images	.305
7.11.7	Soft-scrolling	.307
7.11.8	Programmation de symboles 'Sprite'	.311

8. Trucs et astuces pour le Hardware **321**

8.1	Comment allumer votre configuration complète à partir d'un seul Interrupteur	.322
8.2	Pour passer du TOS ordinaire au Blitter-TOS	.324
8.3	Régler la vitesse du processeur	.328

9. Annexe **331**

9.1.	Tableau des codes ASCII	.331
9.2.	Tableau des codes SCAN du clavier	.332

Index **333**

Chapitre 1

Le chargement automatique des programmes

Le dossier AUTO

L'Atari ST vous offre la possibilité de charger et de lancer automatiquement à partir du disque d'initialisation un système d'exploitation. Il permet aussi de charger automatiquement les programmes qui se trouvent dans un dossier AUTO. Insérez la disquette qui vous sert à l'initialisation (disquette 'boot') et affichez son répertoire. Cliquez sur "nouveau dossier" dans le menu "fichier": vous voyez apparaître une boîte de dialogue dans laquelle vous entrez le nom de dossier 'AUTO'. Si vous provoquez l'initialisation du système à partir d'un disque dur, votre dossier AUTO doit se trouver dans la partition C.

Vous pouvez alors copier dans ce dossier les programmes que vous voulez charger automatiquement soit lors de l'allumage soit après un 'reset'. Comme c'est bien souvent le cas, la difficulté vient d'un tout petit détail. En effet, les programmes se trouvant dans le dossier AUTO doivent porter l'extension .PRG mais ne pas faire appel au GEM, puisque ce dernier n'est pas encore installé au moment où le système traite le dossier AUTO. Ceci peut paraître bizarre, puisque d'ordinaire seuls les programmes GEM portent l'extension .PRG alors que les programmes TOS (ceux qui ne font pas appel au GEM) portent l'extension .TOS. Nous reviendrons plus en détail sur la question de ces programmes GEM dans le chapitre 1.4. Etudions tout d'abord le dossier AUTO.

1.1. La mise-à-jour et la sauvegarde du calendrier

Nous allons d'abord écrire un petit programme qui intervient après l'allumage ou après un 'reset' pour contrôler si le calendrier (date et heure système) a bien été mis-à-jour; si ce n'est pas le cas, le programme va le demander. Les utilisateurs possédant un système équipé d'une horloge permanente n'ont naturellement pas besoin de ce programme. Pour ceux qui n'ont pas cette chance, nous supposons qu'il leur est déjà arrivé souvent, durant l'élaboration d'un programme, de devoir en rechercher la dernière version après en avoir écrit plusieurs au cours d'une session de travail. Ceci n'est plus un problème si on prend la peine de mettre à jour correctement le calendrier à chaque nouvelle version.

Pour que le programme fonctionne bien, il est essentiel que l'accessoire de contrôle ne se trouve pas sur l'unité servant à l'initialisation, car il réécrirait l'ancien calendrier en écrasant le nouveau.

Ce programme vous montre que vous pouvez placer dans le dossier AUTO des programmes écrits en GFA dans leur version compilée du moment que vous ne faites pas intervenir d'objets graphiques. Vous pouvez d'ailleurs vérifier cela avec certains de vos propres programmes.

```

'
' Programme: AUTODATE.BAS
'
' Lors du demarrage, ce programme contrôle la date
' et demande eventuellement sa mise-à-jour
'
'
D.ate%=Xbios(23)           ! Rechercher la date et
l'heure
A.nnee%=D.ate% Div 2^25    ! Isoler l'annee
If A.nnee%<>48             ! La valeur initiale
existe encore
'
'                               ! lors du demarrage
T.ime%=D.ate% And &HFFFF   ! Isoler l'heure
D.ate%=D.ate% Div &HFFFF   ! Isoler la date
Void Gemdos(&H2D,T.ime%)   ! puis la saisir
Void Gemdos(&H2B,D.ate%)   !
Else                         ! sinon prendre l'ancienne
date
Do                           ! oui, entrer la date
actuelle
Cls                           ! Vider l'ecran
' BOX 35,60,252,200        ! seulement si le

```

```

                                programme n'est
'   LINE 35,84,252,84           ! pas dans le dossier AUTO
,
Print At(6,5);"Indiquer la date et l'heure"
Print At(7,8);"Date: ___.__.____"
Print At(6,10);"Heure: ___.__.____"
,
Print At(14,8);                 ! Entrer la date
@Z_saisie(2)                    ! Jour sur deux chiffres
Date$=Res$+"."
Print ". ";
@Z_saisie(2)                    ! Mois sur deux chiffres
Date$=Date$+Res$+"."
Print ". ";
@Z_saisie(4)                    ! Annee sur quatre chiffres
Date$="Date$+Res$"
,
Print At(14,10);               ! Entrer l'heure
@Z_saisie(2)                   ! Les heures sur deux chiffres
Heure$=Res$+":"
Print ":";
@Z_saisie(2)                   ! Les minutes sur deux chiffres
Heure$=Heure$+Res$+":"
Print ":";
@Z_saisie(2)                   ! Les secondes sur deux chiffres
Heure$=Heure$+Res$
,
Print At(6,14);"valider?(o/n) :",Chr$(8);
Outflag%=1                     ! Flag du curseur on/out
Compteur%=0                    ! Delai d'attente
Do
  Let Touche$=Upper$(Inkey$)    ! Verification de la
  touche actionnee
  Exit If Touche$="O" Or Touche$="N" ! est-ce la bonne
  touche?
  Add Compteur%,1               ! Sinon incrementer le
  compteur
  If Compteur%=100             ! Delai termine?
    Compteur%=0                ! Remettre le compteur à zero
    If Out_flag%=1             ! Le curseur etait-il mis?
      Print " ";Chr$(8);      ! Affichage d'espaces vides
      Out_flag%=0              ! Flag du curseur desactive
    Else
      Print " ";Chr$(8);      ! Afficher un tiret bas
      Out_flag%=1              ! Flag du curseur active
    Endif
  Endif
Loop
,
If Touche$="O"                  ! La date existante est-elle juste?
  Settime Heure$,Date$        ! Mettre à jour le calendrier

```

```

    If Date$=Date$ And Left$(Heure$,5)=Left$(Time$,5)
    End
    Endif
Endif
'
Loop
'
! Repetition jusqu'à ce que la bonne
! date soit entree
Endif
'
Procédure Z_saisie(Quantite%)
! Saisie de nombres
composés de
'
! Quantite% de chiffres
'
Res$=""
! Effacer le string du resultat
For Boucle%=1 To Quantite%
! en tout Quantite% de chiffres
Outflag%=1
! Placer le flag du curseur
Compteur%=0
! Effacer le compteur en attente
Do
Let Touche$=Inkey$
! Contrôler la touche activee
Exit If Touche$>="0" And Touche$<="9" ! Quel chiffre?
Add Compteur%,1
! Incrementer le compteur
If Compteur%=100
! Delai termine?
Compteur%=0
! Effacer le compteur
If Out_flag%=1
! Curseur active?
Print " ";Chr$(8);
! Afficher des espaces vides
Out_flag%=0
! Desactiver flag du curseur
Else
! Curseur desactive?
Print "_";Chr$(8);
! Afficher un tiret bas
Out_flag%=1
! Activer flag du curseur
Endif
Endif
Endif
Loop
'
Print Touche$;
! Afficher le caractère saisi
Res$=Res$+Touche$
! et l'ajouter au string resultant
Next Boucle%
Return

```

Le programme commence par rechercher le calendrier existant à l'aide de la fonction Xbios(23) du système d'exploitation. Si vous venez d'allumer votre ordinateur, la date a la valeur 48. Si la date n'a plus cette valeur 48, c'est qu'elle a déjà été modifiée une fois au moins (par exemple par un 'reset'): il suffit alors de la mettre à jour à l'aide des deux fonctions GEMDOS. Sinon, le programme vous donne la possibilité de saisir entièrement le nouveau calendrier. Nous avons pour cela une petite procédure qui permet de fixer le nombre de chiffres désirés pour chaque nombre du calendrier. Cette procédure se charge de faire clignoter le curseur en attendant la saisie, puis transmet le résultat à la variable Res\$.

Après avoir correctement saisi les données, vous répondez par "oui" à la question qui s'affiche. Nous utilisons alors une particularité de la commande Settime qui refuse d'enregistrer une date impossible (par exemple 29.02), ce qui permet de contrôler facilement la validité des données saisies sans avoir besoin d'écrire pour cela une routine spéciale.

Nous allons vous expliquer pourquoi nous devons alors enregistrer le calendrier simultanément sous plusieurs routines du système d'exploitation. Voici tout d'abord les fonctions utilisées, avec la description exacte de leurs paramètres:

```
GFA: Void Xbios(22,L:Tida%)
C : long tida;
    Settime(tida);          ou:  Xbios(22,tida);
ASS: move.l  tida,-(sp)
      move.w  #22,-(sp)
      trap   #14
      addq.l  #6,sp
```

Le paramètre tida se compose d'informations encodées bit par bit représentant l'heure et la date de la façon suivante:

Bit	Signification
0- 4	Les secondes sur deux chiffres
6-10	Les minutes
11-15	Les heures
16-20	Le jour
21-24	Le mois
25-31	L'année moins 1980

Le contrôle de l'heure se fait de façon inversée à l'aide de la fonction Gettime Xbios(23). La valeur longue obtenue en retour a la même structure.

Les routines du GEMDOS ne traitent que des mots: c'est pourquoi il existe deux routines différentes pour le calendrier:

```
Date:
GFA: Void Gemdos(&H2B,D.ate%)
C : int date;
    Tsetdate(date);          ou  gemdos(0x2B,date);
ASS: move.w  date,-(sp)
      move.w  #$2B,-(sp)
      trap   #1
      addq.l  #4,sp
```

Date est une valeur de 16 bits ayant la signification suivante:

Bit	Signification
0- 4	Jour
5- 8	Mois
9-15	Année moins 1980

La fonction inverse, c'est-à-dire la lecture de la date, se fait par GEMDOS(0x2A).

```

Heure:
GFA: Void Gemdos (&H2D, T.time%)
C : int time;
    Tsettime(time);      ou  gemdos(0x2D, time);
ASS: move.w   time, -(sp)
     move.w   #$2D, -(sp)
     trap    #1
     addq.l   #4, sp
  
```

Time est une valeur de 16 bits ayant la signification suivante:

Bit	Signification
0- 4	Les secondes sur deux chiffres
5-10	Les minutes
11-15	Les heures

Ici aussi, la structure de la fonction XBIOS est identique, mais avec un mot différent. La fonction inverse, la lecture de l'heure, se fait par GEMDOS(0x2C).

A présent, vous vous demandez certainement pourquoi nous avons utilisé à fond les deux possibilités de mise à jour du calendrier. L'Atari ST possède en fait deux horloges. La première, située dans le processeur du clavier travaille en temps réel et poursuit le comptage après un 'reset': il s'agit d'une véritable horloge, que l'on règle à l'aide de fonctions XBIOS. Il existe une deuxième horloge, qui n'existe que sous la forme de programme, que l'on peut régler à l'aide de fonctions GEMDOS. Elle ne résiste pas à un reset (qui la fait repartir de zéro), mais c'est elle qui sert au système d'exploitation pour dater par exemple les fichiers. Après la réinitialisation de l'ordinateur par un reset, son horloge XBIOS continue à indiquer l'heure (sans se remettre à zéro), si bien que notre programme peut se limiter à relire cette horloge pour transmettre les données obtenues aux fonctions GEMDOS, puisque de surcroît leur structure est rigoureusement identique.

Lorsque le programme doit intervenir juste après l'allumage de la configuration, la date à saisir existe sous forme de string. Pour vous épargner sa reconversion, nous utilisons la commande 'settime' qui se charge de transmettre les nouvelles valeurs aux deux horloges. Voilà donc pourquoi nous avons dû utiliser les deux possibilités d'écriture de l'heure dans notre programme. Ceci vous montre par ailleurs qu'il faut beaucoup de connaissances théoriques pour écrire ne serait-ce qu'un tout petit programme!

1.2. L'installation automatique de l'imprimante

L'accessoire d'installation de l'imprimante s'avère souvent peu satisfaisant. La plupart du temps, il ne prend pas en compte la possibilité de régler le nombre de points par ligne, car en recourant à l'option 1280 points, on ne peut plus faire une copie d'écran qu'avec une imprimante admettant du papier format A3. De plus, de nombreux logiciels de traitement de texte prennent leurs informations (par exemple au sujet du papier en continu ou feuille à feuille) à partir des valeurs standards. Si l'on s'en tient à cela, il faut obligatoirement disposer d'un accessoire d'installation de l'imprimante, ce qui prend de la place mémoire (c'est l'un des six accessoires) et empêche notre programme de mise à jour de l'horloge de fonctionner.

Il y a deux façons de procéder pour rendre accessibles ces valeurs au système d'exploitation. La première possibilité, que nous allons exposer ici, consiste à lire les informations dans le fichier DESKTOP.INF. Il vous suffit pour cela, après avoir fixé les valeurs à l'aide de l'accessoire d'installation de l'imprimante, de cliquer sur "sauvegarder le bureau" dans le menu "options". C'est là également que sont sauvegardées d'autres informations comme par exemple les unités de disques installées, le réglage du port de sortie RS232 etc. Vous pouvez soit effacer purement et simplement l'accessoire d'installation de l'imprimante se trouvant sur votre disquette d'initialisation soit changer le nom de ce fichier accessoire. Recopiez alors le programme ci-dessous dans le dossier AUTO après l'avoir compilé.

```
'  
'  
' Programme AUTODR.BAS  
'  
' Ce programme recherche les paramètres concernant l'imprimante  
' dans le fichier DESKTOP.INF et transmet les variables adéquates  
'
```

```

OPEN "i",#1,"DESKTOP.INF" !Ouvrir le fichier pour le lire
DO
EXIT IF EOF(#1)           !Fin, après avoir lu toutes les lignes
,
LINE INPUT #1,ligne$      !Lire des lignes entières
,
IF LEFT$(ligne$,2)="#b" !Parametrage de l'imprimante
valeur%=0                !Convertir en decimal les nombres binaires
FOR boucle%=3 TO 9       !3 à 9 signes
MUL valeur%,2           !Multiplier par deux le nombre precedent
,
ADD valeur%,VAL(MID$(ligne$,boucle%,1)) !Additionner le
NEXT boucle%             !nouveau nombre jusqu'à ce que tous
,                          !les chiffres aient defile
VOID XBIOS(33,valeur%) !Entrer les paramètres
ENDIF
,
LOOP

```

Les valeurs destinées à la routine du système d'exploitation sont identiques à celles qui se trouvent dans le fichier DESKTOP.INF: il suffit de les convertir en une valeur numérique. Voici la signification des bits du nombre de deux octets:

Bit n	0	1
0	Imprimante à matrice	Imprimante à marguerite
1	Imprimante noir et blanc	Imprimante couleurs
2	Imprimante Atari	Imprimante Epson
3	Mode test	Mode qualité
4	Interface centronics	Interface sériel RS232
5	Papier en continu	Feuille à feuille
6-14	Réservé	
15	Toujours sur 0	

La deuxième possibilité d'adaptation de l'imprimante consiste à transmettre directement au système d'exploitation les valeurs adéquates pour l'imprimante. Ceci raccourcit le programme d'une ligne et vous dispense de devoir placer le fichier DESKTOP.INF sur votre disquette d'initialisation. Vous pouvez vous-même retrouver la valeur numérique en combinant les numéros de bits d'après le tableau ci-dessus. Pour appeler la fonction voulue, vous écrivez:

```
Void Xbios(33,Valeur%)
```

Vous devez aussi compiler ce petit "programme" puis le copier dans le dossier AUTO.

1.3. L'installation du GDOS et des accessoires

Ce programme GDOS est indispensable pour certains logiciels, comme par exemple les logiciels graphiques ou certains traitements de texte, mais il peut par contre en empêcher d'autres de tourner. Il est cependant très désagréable, spécialement pour les possesseurs de disque dur, de devoir constamment recopier et effacer ce programme dans le dossier AUTO selon le logiciel utilisé. Nous allons remédier à cela et écrire un programme qui fasse d'une pierre deux coups: il vous permettra en plus de sélectionner les accessoires que vous désirez charger lors du processus d'initialisation.

Voici d'abord quelques explications au sujet de ce programme. Vous savez déjà que seuls les programmes portant l'extension .PRG sont chargés automatiquement depuis le dossier AUTO. Les accessoires quant à eux doivent absolument se trouver en dehors du dossier AUTO et porter l'extension .ACC. Notre programme va jouer sur ces possibilités. Il vous demande d'abord si vous allez travailler avec le GDOS: si ce n'est pas le cas, il renomme le fichier GDOS.PRГ sous le nom GDOS.PR, ce qui empêche son chargement automatique. Nous allons procéder de même pour les accessoires, en dotant de l'extension .AC ceux que vous ne désirez pas utiliser momentanément.

Pour renommer ces fichiers, il faut d'abord savoir que les programmes contenus dans le dossier AUTO sont traités l'un après l'autre en suivant l'ordre dans lequel ils ont été recopiés, qui n'est pas forcément l'ordre alphabétique ni chronologique. C'est pourquoi il est important de recopier le programme ci-dessous dans le dossier AUTO avant d'y copier le programme GDOS.PRГ. Si le programme GDOS porte un autre nom, vous devez soit le renommer, soit utiliser ce nom dans le programme ci-dessous.

Notre programme ne contient pas de représentation graphique puisque le dossier AUTO ne peut contenir de programmes faisant appel à des éléments graphiques ou à des boîtes de dialogue.

```
'  
' Programme GDOSMAKE.BAS  
'  
' Installation du GDOS et des ACCESSOIRES  
'
```

```

Print "    Installer le GDOS ? (o/n) "; !Question
Do
  U$=Upper$(Inkey$)                !Une touche actionnee ?
  Exit If U$="O" Or U$="N"          !Quitter si 'o' ou 'n'
Loop
'
Chdir "\auto\"                      !Aller dans le dossier AUTO
'
If Exist("GDOS.PRG")=0 And Exist("GDOS.PR")=0 !L'un ou l'autre
existe?
'
  Print "' GDOS.PR?' non trouvé"!Affichage message d'erreur
  U=Inp(2)                          !Attendre l'action d'une touche
Else
'
  If U$="O"                          !Installer le GDOS?
    If Exist("gdos.prg")=0           !Le nom est déjà correct
      Name "gdos.prg" As "gdos.pr"  !Renommer si 'non'
    Endif
  Else
    If Exist("gdos.pr")=0           !Le nom est déjà correct
      Name "gdos.pr" As "gdos.prg"  !Renommer si 'non'
    Endif
  Endif
'
Chdir "\"                            !Retourner au repertoire principal
'
Dim A_name$(15)                     !15 noms d'accessoires
Dim A_flag$(15)                     !15 Flag pour
ecrire/effacer
'
Cls                                  !Vider l'ecran
Print "Installation des accessoires"
'
Dta$=Space$(50)                     !creer un buffer
Void Gemdos (&H1A,L:Varptr (Dta$))  !pour search
Search$="*.AC?"                      !Searchname
Presence%=Gemdos (&H4E,L:Varptr (Search$),0) !Premier fichier
Acc%=0
While (Presence%=0 And Acc%<15)     !Maximum 15 accessoires
  A_name$(Acc%)=Mid$(Dta$,31,14)     !Lire les noms dans le DTA
  Print "    ";A_name$(Acc%);" (o/n)"; !Afficher les noms
'
Do
  U$=Upper$(Inkey$)
  Exit If U$="O" Or U$="N"
Loop
'

```

```

Print U$                                !Afficher la touche actionnee
If U$="O"                                !o ?
    A_flag$(Acc%)=1                      !Insérer flag
Else
    A_flag$(Acc%)=0                      !Effacer flag
Endif
Inc Acc%                                 !Incrementer le compteur
Presence%=Gemdos(&H4F)                   !Lire le nom de l'acc suivant
Wend
,
For Acc1%=0 To Acc%                      !Tous les accessoires
    If A_flag$(Acc1%)=0 And Instr(A_name$(Acc1%),".ACC")<>0 !déjà
        ,
!efface?
        Point%=Instr(A_name$(Acc1%),".") !Rechercher le point
        Nouveau_nom$=Left$(A_name$(Acc1%),Point%)+".ACC" !Extension AC
        Name A_name$(Acc1%) As Nouveau_nom$!Renommer
    Endif
    ,
    If A_flag$(Acc1%)=1 And Instr(A_name$(Acc1%),".ACC")=0 !déjà
        ,
!present?
        Point%=Instr(A_name$(Acc1%),".") !Rechercher le point
        Nouveau_nom$=Left$(A_name$(Acc1%),Point%)+".ACC"!Extension ACC
        Name A_name$(Acc1%) As Nouveau_nom$!Renommer
    Endif
Next Acc1%

```

Avant d'en venir à un programme de luxe auto-starter depuis le GEM, nous voudrions encore attirer votre attention sur le point suivant: vous aurez souvent besoin de plusieurs des programmes écrits dans ce livre, si bien qu'il serait astucieux de les réunir en un seul programme auquel vous pourriez ensuite attribuer un code de confidentialité (mot de passe) dans la mesure où l'initialisation se fait à partir du disque dur.

1.4. Le chargement automatique des applications GEM

Comme nous l'avons déjà vu ci-dessus, vous ne pouvez pas lancer à partir du dossier AUTO les programmes exploitant les possibilités du GEM, ce qui vous interdit également de lancer directement depuis ce dossier un logiciel de traitement de texte ou de gestion de fichier, puisqu'ils font appel généralement aux possibilités du GEM. Et pourtant, comme ce serait idyllique de pouvoir insérer une disquette nommée "graphique", pour ensuite réinitialiser l'ordinateur et se mettre à dessiner deux minutes plus tard...

Que doit faire l'utilisateur pour lancer une application GEM? Nous pouvons décomposer sa démarche en quatre étapes :

- ◆ il attend de recevoir le bureau GEM
- ◆ il ouvre une fenêtre contenant le répertoire d'une unité de disque
- ◆ il pointe le curseur de la souris sur le programme à lancer
- ◆ il exécute alors un double-clic.

Admettons que l'utilisateur se soit facilité les choses par le biais de l'option "sauvegarder le bureau" qui lui permet d'obtenir directement une fenêtre ouverte avec l'icône du programme située en plein milieu de l'écran, ce qui lui épargne les étapes deux et trois. Il lui suffit alors d'un programme TOS dans le dossier AUTO qui prenne en charge les étapes un et quatre. Ce n'est pas bien difficile comme vous allez le voir.

Il nous faut tout d'abord faire usage du VBL, non pas au niveau du VBL-Queue mais directement au niveau du vecteur adéquat (numéro 28, adresse \$70). Ce procédé peut paraître surprenant mais il est légitime et justifié par le fait que nous n'avons besoin du VBL que durant quelques secondes. Les slots (zones d'entrées) destinés aux programmes plus complets restent donc disponibles. (Vous trouverez de plus amples informations au sujet du VBL dans le chapitre 7.3 consacré aux programmes résidents et au VBL-Queue).

Cette routine VBL se borne à contrôler s'il y a déjà quelque chose d'affiché à l'écran, plus exactement si le dernier octet de la RAM-vidéo est occupé. Si c'est le cas, il ne peut s'agir que du fond gris ou vert du bureau GEM. Il est encore trop tôt pour exécuter le double-clic, car l'ordinateur a besoin d'un petit délai pour afficher le répertoire. C'est pourquoi nous installons une deuxième routine VBL (la précédente est désactivée, nous n'en avons plus besoin), qui attend 140 VBL, c'est-à-dire deux secondes dans le cas des moniteurs monochromes.

Nous n'avons rencontré jusqu'ici aucun problème, mais comment allons-nous simuler un double-clic? C'est également très simple. Si vous avez un répertoire des routines XBIOS, regardez un peu ce qui se trouve sous le numéro 34 (kbdvbase). On trouve sous cette fonction les adresses de quelques routines-système bien utiles, parmi lesquelles se trouvent celles qui gèrent la souris. Nous prenons cette adresse et l'entrons quatre fois: souris avec touche appuyée, puis touche relâchée, touche appuyée, touche relâchée, ce qui nous donne bien un double-clic. Dans A0, nous attribuons à la routine un pointeur sur les données de la souris. Le tour est joué car pour le reste, c'est-à-dire le lancement du programme, nous nous en remettons au système d'exploitation.

Nous en avons ainsi terminé en ce qui concerne le fonctionnement du programme. Que reste-t-il à faire pour lancer automatiquement un programme sous GEM?

- ◆ créez, si ce n'est déjà fait, un dossier AUTO sur votre disquette boot
- ◆ copiez le programme AUTOGEM.PRG dans ce dossier
- ◆ arrangez-vous pour amener le répertoire sur l'écran de façon à ce que l'icône du programme à lancer se trouve en plein milieu, là où se trouve le pointeur de la souris juste après le démarrage.
- ◆ cliquez sur "sauvegarder le bureau" dans le menu "options".

Voilà, c'est tout. Appuyez sur le bouton 'reset' et dégustez l'effet de ce petit programme. Il y a une chose qu'il ne faut absolument pas faire: déplacer la souris dans les deux secondes suivant le lancement du programme.

Voici le texte du programme en assembleur:

```

;
; Lancement automatique des applications GEM : AUTOGEM.S
;      MP      29-04-88
;
gemdos      = 1
xbios      = 14
keep       = $31
supexec    = 38
kbdvbase   = 34
vbl        = $70
_v_bas_ad  = $44e
last_byte  = 31999

        .TEXT

mémoire   movea.l   4(sp),a0           ; Calcul de la place
          move.l   #$100,d6
          add.l   12(a0),d6
          add.l   20(a0),d6
          add.l   28(a0),d6

superviseur  pea    init_vbl           ; Passage en mode
          move.w   #supexec,-(sp)
          trap    #xbios
          addq.l   #6,sp

          clr.w   -(sp)                ; Fin, reste résident
          move.l   d6,-(sp)

```

```

        move.w    #keep, -(sp)
        trap     #gemdos

init_vbl: move.l    vbl,old_vbl    ; Pas d'utilisation de la VBL
        move.l    #new_vbl,vbl    ; Méthode directe
        move.l    old_vbl,j_vbl+2 ;
        rts

new_vbl: move.l    a0,savereg      ; Sauver les registres
        movea.l   _v_bas_ad,a0
        tst.b     last_byte(a0)
        beq      weiter          ;

        move.l    #newest,vbl     ; Activer le nouveau VBL
        move.w    #140,counter    ; 2 Secondes (monochrome)
        bra.s     weiter

newest:  move.l    a0,savereg      ; Remettre les registres...
        subi.w    #1,counter      ; Décrémenter compteur
        bne.s     weiter

        move.l    old_vbl,vbl     ; VBL remise en état

souris  move.w    #kbdvbase, -(sp) ; gestionnaire de la
        trap     #xbios
        addq.l   #2,sp
        movea.l   d0,a0           ;
        movea.l   16(a0),a0
        move.l    a0,j1+2
        move.l    a0,j2+2
        move.l    a0,j3+2
        move.l    a0,j4+2

j1:      lea.l    pressed,a0       ; Simuler un double-clic
        jsr     $12345678         ; Adresse fausse

j2:      lea.l    released,a0
        jsr     $12345678

j3:      lea.l    pressed,a0
        jsr     $12345678

j4:      lea.l    released,a0
        jsr     $12345678

weiter:  movea.l   savereg,a0      ; Remettre les registres
j_vbl:   jmp     $12345678

```

```
        .DATA
pressed: .DC.b $fa,0,0           ; Souris actionnée
released: .DC.b $f8,0,0        ; Souris relachée

        .BSS
        .EVEN

old_vbl: .DS.l 1
savereg: .DS.l 2
counter: .DS.w 1

        .END
```


Chapitre 2

Les utilitaires de gestion de l'unité

de disque dur ou de disquette

Nous voudrions dans ce chapitre vous transmettre un certain savoir-faire concernant l'utilisation des disquettes, et vous faire découvrir des utilitaires qui vous rendront sûrement de grands services.

2.1. Dissimuler des fichiers

Ce programme va vous servir à clarifier l'affichage de vos répertoires bien encombrés. Il sert à dissimuler des fichiers de telle sorte qu'ils n'apparaissent plus dans l'affichage ordinaire d'un répertoire. Naturellement, vous pourrez quand même les réafficher si bon vous semble.

Tout fichier se trouvant dans une mémoire de masse est flanqué d'un fichier-attribut contenant les caractéristiques particulières à ce fichier. Le fichier-attribut mémorise par exemple le statut du fichier en question (lecture seule ou lecture/écriture), caractéristique que vous pouvez vous même attribuer à un fichier à partir du bureau GEM. Il existe de même un attribut transformant un fichier normal en fichier caché. Vous accédez à ces fichiers cachés tout à fait normalement, sans pourtant qu'ils n'apparaissent dans les répertoires de la disquette. En dissimulant tous les fichiers sauf celui du programme à lancer, on

reçoit à l'affichage un répertoire (apparemment) presque vide, ce qui donne une impression de grande clarté.

La question essentielle est donc de savoir comment changer l'attribut d'un fichier. Il existe pour cela une fonction GEMDOS nommée "Change Mode", ou "Fattrib" en langage C. Les différents attributs portent chacun un numéro. Seul 0 (fichier normal) et 2 (fichier caché) sont intéressants pour nous dans ce chapitre. Il suffit de transmettre l'un après l'autre le nouvel attribut, un flag, un pointeur sur le nom du fichier (éventuellement aussi sur le chemin d'accès) et pour terminer, le numéro de la fonction (\$43). D0 vous renvoie toujours la nouvelle valeur de l'attribut. Le flag indique si le nouvel attribut doit être enregistré (1) ou s'il doit reprendre son ancienne valeur (0). Mais si D0 est négatif en quittant la fonction, c'est qu'il y a une erreur.

Le programme ci-dessous en assembleur vous fournit toutes les explications complémentaires. Il nécessite l'emploi de la routine de saisie INPUT.S.

```

;
; Dissimuler / rechercher un fichier HIDE.S
;      MP'   13-05-88
;
gemdos   = 1
conin    = 7
print    = 9
chmode   = $43

        .TEXT

        pea      annonce           ; Afficher le titre
        move.w   #print, -(sp)
        trap     #gemdos
        addq.l   #6, sp

choose:  move.w   #conin, -(sp)     ; Attendre une touche
        trap     #gemdos
        'addq.l  #2, sp
        cmpi.w   #'0', d0         ;
        beq.s    ok
        cmpi.w   #'1', d0
        bne.s    choose

ok:      subi.w   #'0', d0         ; ASCII --> Binaire
        asl.w    #1, d0           ; * 2
        move.w   d0, and_now      ; et sauver

        move.w   #30, -(sp)       ; Boucle d'attente
        pea      filename

```

```

move.w #4,-(sp) ; ligne
move.w #5,-(sp) ; colonne
bsr rdstr
adda.l #10,sp

move.w and_now,-(sp) ; Change Mode
move.w #1,-(sp) ; Modifier Attribut
pea filename
move.w #chmode,-(sp)
trap #gemdos
adda.l #10,sp
tst.w d0 ; Erreur ?
bpl.s quit

pea err_text ; Oui, donc afficher message
move.w #print,-(sp)
trap #gemdos
addq.l #6,sp

move.w #conin,-(sp) ;
trap #gemdos
addq.l #2,sp

quit: clr.w -(sp) ; -> Desktop
trap #gemdos
.PATH 'A:\'
.INCLUDE 'INPUT.S' ; Eingabe-Warteschleife

.BSS
and_now: .DS.w 1
filename: .DS.b 31

.DATA
annonce: .DC.b 27,'E *** FICHER 1 = dissimuler ou 0 =
retrouver ***',13,10,10,0
err_text: .DC.b 27,' Fichier inexistant (TOUCHE)',0

.END

```

2.2. Encoder des fichiers

Tout le monde parle aujourd'hui de la sécurité des enregistrements informatiques. Un programme d'encodage des fichiers peut s'avérer très utile, même si vous ne travaillez pas en réseau ou si vous n'êtes pas en liaison permanente avec le Pentagone. Vous en aurez peut-être besoin pour empêcher votre petite soeur de

jouer à Arkanoid ou pour dissimuler à vos amis une de vos trouvailles sauvegardée sur une disquette.

Notre programme d'encodage est très simple, mais il suffit pour empêcher une personne étrangère d'avoir accès à vos données si elle ne connaît pas les clés du code. Ce programme encode un fichier en le liant à un mot de passe octet par octet par un ou-exclusif. Ce qui signifie qu'un bit positif dans l'octet du mot de passe inverse la valeur d'un bit du fichier. Ce processus très simple a un grand avantage: il permet d'utiliser le même programme et la même clé pour encoder ou décoder les données.

Ce programme est très efficace lorsqu'on encode un même fichier plusieurs fois avec des clés longues et variées. En utilisant deux clés comprenant x et y caractères, on obtient le même résultat qu'avec une clé de z caractères, z étant alors le plus petit commun multiple de x et y. Les fichiers sont ensuite décodés avec les mêmes clés, en procédant par ordre inverse.

Bien sûr, ce programme ne représente pas une garantie absolue et son code peut être retrouvé. Si vous n'avez rien de mieux à faire, demandez à un de vos amis de vous encoder un de vos textes et essayez vous-même de trouver les clés d'accès. Un dernier conseil: faites attention de ne pas égarer ou oublier vos clés d'accès, car sinon...

```
;
; Encoder/Décoder un fichier ENCODER.S
;      MP      11-05-88
;
gemdos    = 1
conin     = 1
print     = 9
setdta    = $1a
open      = $3d
create    = $3c
close     = $3e
sfirst    = $4e
read      = $3f
write     = $40
          .TEXT
          pea      annonce           ;Afficher le titre
          move.w   #print,-(sp)
          trap     #gemdos
          addq.l   #6,sp
          move.w   #30,-(sp)       ;Boucle d'attente de
saisie
          pea      filename
```

```

move.w    #8,-(sp)           ;ligne
move.w    #5,-(sp)           ;colonne
bsr       rdstr
adda.l    #10,sp
pea       annonce2
move.w    #print,-(sp)
trap     #gemdos
addq.l    #6,sp
label:    move.w    #30,-(sp)   ;entrer le code du fichier
pea       code
move.w    #14,-(sp)
move.w    #5,-(sp)
bsr       rdstr
adda.l    #10,sp
tst.b     code               ;nom de code entré?
beq.s     label              ;si non, recommencer
pea       dta                 ;créer un nouveau DTA
move.w    #setdta,-(sp)
trap     #gemdos
addq.l    #6,sp
clr.w     -(sp)              ;Sfirst
pea       filename
move.w    #sfirst,-(sp)
trap     #gemdos
addq.l    #8,sp
clr.w     -(sp)              ;Ouvrir le fichier pour le lire
pea       filename
move.w    #open,-(sp)
trap     #gemdos
addq.l    #8,sp
tst.w     d0
bmi       error              ;message d'erreur
move.w    d0,handle
pea       place               ;lire le fichier
move.l    taille,-(sp)
move.w    handle,-(sp)
move.w    #read,-(sp)
trap     #gemdos
adda.l    #12,sp
move.w    handle,-(sp)
move.w    #close,-(sp)
trap     #gemdos
addq.l    #4,sp
move.l    taille,d0          ;octet à encoder
lea.l     place,a0
loop1:    lea.l     code,a1
loop2:    tst.b     (a1)        ;fin du nom de code?
beq.s     loop2              ;oui, commencer au début
move.b    (a1)+,d1           ;sinon rechercher l'octet
eor.b     d1,(a0)+          ;et l'encoder (liaison EOR)
subi.l    #1,d0

```

```

    bne.s    loop1
    clr.w    -(sp)          ;créer fichier
    pea     filename
    move.w   #create,-(sp)
    trap    #gemdos
    addq.l   #8,sp
    move.w   d0,handle
    pea     place          ;réécrire le fichier
    move.l   taille,-(sp)
    move.w   handle,-(sp)
    move.w   #write,-(sp)
    trap    #gemdos
    adda.l   #10,sp
    move.w   handle,-(sp)  ;fermer le fichier
    move.w   #close,-(sp)
    trap    #gemdos
    addq.l   #4,sp
    bra.s    quit
error:     pea     err_text  ;Afficher le message d'erreur
    move.w   #print,-(sp)
    trap    #gemdos
    addq.l   #6,sp
    move.w   #conin,-(sp) ;attend l'action d'une touche
    trap    #gemdos
    addq.l   #2,sp
quit:     clr.w    -(sp)          ;fin du programme
    trap    #gemdos
    .PATH 'A:\'
    .INCLUDE 'INPUT.S'      ;insérer la boucle d'attente
    .DATA
annonce:  .DC.b 27,'E',10,'*** Encoder/Décoder un fichier
***',13,10,1
          .DC.b 'Nom du fichier:',0
annonce2: .DC.b 13,10,10,'Nom de code:',0
err_text: .DC.b 13,10,10,'Fichier inexistant ! (TOUCHE)',0
          .BSS

dta:     .DS.b 26
taille:  .DS.l 1
name:    .DS.b 14
filename: .DS.b 31
code:    .DS.b 31
handle:  .DS.w 1
place:

        .END

```

2.3. Un 'floppy-speeder'

Nous sommes ici contraints de vous annoncer une bonne et une mauvaise nouvelle. Voici d'abord la mauvaise: pour pouvoir profiter du programme ci-dessous, vous devez disposer du TOS en ROM. La bonne nouvelle maintenant: ce TOS peut être soit l'ancienne version ou le nouveau Blitter-TOS.

Le titre du paragraphe vous dit déjà qu'il s'agit d'un programme capable d'accélérer la vitesse de travail de l'unité de disquette. Effectivement, il peut réduire d'un tiers le délai de chargement nécessaire pour les grands programmes ou fichiers. Il peut aussi réduire considérablement les délais de recopiage d'une disquette dans un disque RAM.

Après avoir positionné sa tête de lecture/écriture sur une piste, votre unité de disquette vérifie encore une fois qu'il s'agit bien de la bonne piste. Pour cela, le 'floppy-controller' doit naturellement lire les coordonnées de la piste en question, ce qu'il fait automatiquement. Ceci se traduit par une grosse perte de temps et semble tout à fait superflu. Une unité de disquette de bonne qualité est capable de positionner correctement sa tête de lecture/écriture sans que son 'chef' (= le 'floppy-controller') soit toujours sur son dos.

Nos efforts vont donc consister à supprimer ce contrôle. C'est d'ailleurs très simple, puisqu'il suffit de modifier un seul octet (!) du système d'exploitation. Il y a cependant un os: l'octet en question se trouve dans la mémoire ROM, où on ne peut en principe le modifier. Solution à ce problème: nous recopions toutes les routines de l'unité de disquette dans la mémoire vive et y modifions l'octet en question. Ne vous tracassez pas: ceci ne vous prendra que 8 Koctets de votre précieuse RAM.

Nous voudrions encore vous expliquer la fonction que remplit ce mystérieux octet que nous voulons modifier. Il contient la commande du floppy-controller permettant à la tête de lecture/écriture de se positionner sur une piste présélectionnée. L'un des bits de cette commande (le numéro 2) fait que la position atteinte est recontrôlée (le bit 2 est positif) ou non (bit 2 annulé). Dans le TOS d'origine, cette commande est symbolisée par \$14 (en binaire: 00010100, le bit 3 est positif). Nous remplaçons cette commande par \$10 (binaire: 00010000). Le bit numéro s'appelle 'verify-flag'.

Comme nous recopions le contenu de la ROM dans la RAM, nous devons procéder à quatre autres modifications. Il s'agit de modifier les adresses dans les commandes JSR, que nous devons transférer de la ROM vers la RAM (naturellement, en restant

dans les mêmes routines, puisque nous avons recopié toutes les routines floppy) pour que notre modification concernant l'octet \$10 soit rendue active. Si vous avez sous la main le texte du programme en ROM, vous trouvez cet octet dans une commande MOVEQ sous l'adresse \$FC1B8E (ou \$FC1D8E dans les blitter-TOS).

Il ne nous reste plus qu'à signaler à l'ordinateur que nous avons une nouvelle routine d'écriture/lecture pour les disquettes. L'adresse de lancement du programme est écrite dans la variable hdv_rw, utilisée habituellement pour les drivers des disques RAM et les disques durs. Il faut ici distinguer deux cas: soit notre programme est le premier installé sous hdv_rw, soit il y a déjà eu installation auparavant d'un disque RAM, d'un disque dur ou d'un cache-memory. Si c'est le cas (hdv_rw n'est plus dirigé vers la ROM) vous devez vérifier dans votre version du TOS quel est l'appareil installé. Les numéros des périphériques se trouvent dans la pile, sous les paramètres habituels (0=unité de disque A, 1=unité de disque B etc). Si vous désirez entrer un numéro d'unité de disque supérieur à 1, laissez dans la routine la tâche qui était déjà attribuée à hdv_rw.

```

;
;Floppy-speeder           SPEEDER.S
;  MP  26-06-88
;
gemdos      = 1
xbios      = 14
keep       = $31
superexec  = 38
hdv_rw     = $476
           .TEXT
           movea.l 4(sp),a0           ;Calculer la place
mémoire nécessaire
           move.l  #100,d6
           add.l  12(a0),d6
           add.l  20(a0),d6
           add.l  28(a0),d6
           pea   init           ;Provoquer l'initialisation dans
           move.w #superexec,-(sp) ;le superviseur
           trap  #xbios
           addq.l #6,sp
           clr.w  -(sp)           ;programme en résident
           move.l d6,-(sp)       ;nombre d'octets nécessaires
           move.w #keep,-(sp)
           trap  #gemdos
init:      lea.l  $fc0000,a0      ;adresse dans la ROM
           lea.l  libre,a1
           move.w #1f06/2-1,d0 ;1f06 copier les octets de la ROM
loop:     move.w  (a0)+,(a1)
           dbra  d0,loop
           cmpi.w #c46,$fc001e   ;Date TOS

```

```

    bne.s    blitter ;(contrôler la version TOS utilisée)
    move.l   #libre+$167c,libre+$1232      ;adresses JSR
    move.l   #libre+$18ce,libre+$1260      ;à adapter
    move.l   #libre+$159e,libre+$1296
    move.l   #libre+$159e,libre+$13c8
    move.b   #$10,libre+$1b8f ;Patch: verify off
    lea.l    libre+$10d2,a0 ;adresse de positionnement
                                ;pour la lecture/écriture

    bra.s    go_on
blitter:   move.l   #libre+$1858,libre+$1416 ;Répéter pour le
    move.l   #libre+$18ce,libre+$1444 ;blitter-TOS
    move.l   #libre+$1782,libre+$147a
    move.l   #libre+$1782,libre+$15ac
    move.b   #$10,libre+$1d8f
    lea.l    libre+$12b6,a0
go_on:    cmpi.b   #$fc,hdv_rw+1
    bne.s    ramdisk ;non, il y a certainement
                                déjà un disque RAM ou un
                                disque dur

    move.l   a0,hdv_rw ;nouvelle routine lect/écriture
    rts
ramdisk:  move.l   hdv_rw,jmp_old+2 ;remarquer l'ancienne
adresse   move.l   #new_rw,hdv_rw ;'détour' par new_rw
    move.l   a0,floppy+2
    rts
new_rw:   cmpi.w   #2,14(sp) ;device-number
    bcs.s   floppy ;device < 2 --> unité de disquette
jmp_old:  jmp      $12345678
floppy:   jmp      $12345678
.EVEN
.BSS
libre:   .DS.b $1f06

.END

```

2.4. Renommer un dossier

Dans sa forme d'origine, le TOS ne permet pas de renommer un dossier. Il faut pour cela soit disposer d'un 'diskmonitor' (logiciel capable de lire la disquette secteur par secteur) et renommer artisanalement le dossier, soit créer un nouveau dossier dans lequel on recopie le contenu de l'ancien dossier qui est ensuite détruit. Vous trouverez ci-dessous un programme vous permettant d'éviter ces procédés compliqués et vous permettant de renommer confortablement un dossier. Le seul hic étant qu'il faut ensuite réinitialiser votre ordinateur, afin d'enregistrer et de valider la nouvelle structure du répertoire.

Ce programme est conçu pour être un accessoire, ce qui permet d'avoir sous les yeux le nom du dossier à modifier lorsqu'on appelle le programme et d'éviter ainsi de confondre deux noms. Une fois lancé, le programme vous demande d'abord de préciser l'ancien puis le nouveau nom du dossier à renommer. Il faut pour cela que l'unité de disque dans laquelle se trouve le dossier soit l'unité actualisée. C'est-à-dire que vous devez, lorsque vous lancez le programme, vous trouver déjà au moins dans le répertoire principal de la disquette ou de la partition du disque dur à modifier.

Les noms une fois entrés, le système détermine quelle est l'unité de disque actuelle et compte le nombre de secteurs (à l'aide des informations contenues dans le bloc des paramètres BIOS, le BPB). La disquette est alors lue et enregistrée secteur par secteur dans un espace mémoire tampon (buffer), dans lequel le programme va rechercher le dossier à renommer. Un fois identifié, son nouveau nom est inscrit dans le buffer qui est ensuite recopié sur la disquette.

Nous vous indiquons ici la structure du BPB afin que vous compreniez mieux comment se fait le comptage des secteurs:

```
typedef struct bpb
{
    int secksize;      /* taille du secteur en octets */
    int clustsize;    /* taille du cluster en secteurs */
    int clustbysize; /* taille du cluster en octets */
    int dirsize;      /* taille du répertoire en secteurs */
    int fatsize;      /* taille du File-Allocation-Table */
    int fatsek;       /* No de secteur du deuxième FAT */
    int clustsek;     /* No de secteur du premier cluster de données */
    int clustnbre;    /* Nombre total des clusters sur la disquette */
    int flags;        /* Flags */
}BPB;
```

Getbpb()

BIOS-No 7

Appel: BPB*Getbpb(disk)
 int disk;

Cette fonction vous indique l'adresse de l'unité de disque retenue sous 'disk' par un pointeur sur le bloc des paramètres BIOS.

La fonction Rwabs constitue une autre routine importante pour le programme; voici son fonctionnement:

Rwabs()**BIOS-No 4**

Appel: long Rwabs(flag,buffer,quantité,firstek,disk)
 int flag;
 long buffer;
 int quantité, firstek, disk;

La fonction Rwabs permet de lire et d'écrire des secteurs sur la disquette.

flag peut accepter les paramètres suivants:

- 0 lire un secteur
- 1 écrire un secteur
- 2 lire un secteur sans déclarer de changement de disquette
- 3 écrire un secteur sans déclarer de changement de disquette

buffer contient l'adresse du buffer dans lequel les données vont être lues ou à partir duquel elles vont être réécrites sur la disquette. L'adresse doit être paire, sinon la vitesse de transfert des données est diminuée.

quantité indique le nombre de secteur à lire puis à réécrire.

firstek désigne le secteur logique de début, à partir duquel le transfert doit commencer.

disk désigne l'unité de disque utilisée.

Voici maintenant le texte du programme : **RENAME.C**

```

                /* Renommer un dossier */

#include <osbind.h>
#include <gemdefs.h>
#include <gembind.h>
#include <obdefs.h>
#include <string.h>

#define EOS      '\0'      /* Fin de chaîne */
#define TRUE    -1
#define FALSE   0

int  contrl[12],
     intin[128],

```

```

    ptsin[128],
    intout[128],
    ptsout[128],
    work_in[11],
    work_out[57],
    msgbuff[8],
    vdi_handle,           /* VDI-Handle */
    phys_handle,         /* Workstation-Handle physique */
    appl_id,             /* Id de l'application */
    menu_id;            /* Id de l'accessoire */

char name_old[12],
    name_new[12];
    char string1[]="<< Renommer un dossier >>"; /*string de la
                                                    boîte de dialogue 1*/

char string2[]="@";
char string3[]="Ancien : _____ . ____";
char string4[]="FFFFFFFFFFFF";
char string5[]=" Suite ";

TEDINFO teds1[]={
    string2, string3, string4, 3, 6, 0, 0x1180, 0x0, -1, 12, 24};
/*Tedinfo de la boîte de dial 1*/
OBJECT tree1[]={
    -1, 1, 3, G_BOX, NONE, OUTLINED, 0x21100L, 150, 100, 300, 160,
    2, -1, -1, G_STRING, NONE, NORMAL, string1, 40, 16, 200, 16,
    3, -1, -1, G_FTEXT, EDITABLE, NORMAL, &teds1[0], 50, 90, 192, 16,
    0, -1, -1, G_BUTTON, SELECTABLE|DEFAULT|EXIT|LASTOB, NORMAL,
    string5, 120, 130, 64, 16};

char string6[]=">> Renommer un dossier <<"; /*string de la boîte
                                                    de dialogue 2*/

char string7[]="@";
char string8[]="Nouveau : _____ . ____";
char string9[]="FFFFFFFFFFFF";
char string0[]=" Envoi ";

TEDINFO teds2[]={
    string7, string8, string9, 3, 6, 0, 0x1180, 0x0, -1, 12, 24};
/*Tedinfo de la boîte de dial 2*/
OBJECT tree2[]={
    -1, 1, 3, G_BOX, NONE, OUTLINED, 0x21100L, 150, 100, 300, 160,
    2, -1, -1, G_STRING, NONE, NORMAL, string6, 40, 16, 200, 16,
    3, -1, -1, G_FTEXT, EDITABLE, NORMAL, &teds2[0], 50, 90, 192, 16,
    0, -1, -1,
G_BUTTON, SELECTABLE|DEFAULT|EXIT|LASTOB, NORMAL, string0, 120, 130, 64, 16};
main()
{
    appl_id=appl_init();           /*Annoncer le programme */
    open_vwork();                 /*ouvrir workstation*/
    menu_init();                   /*annoncer l'option dans le menu*/
}

```

```

    loop();                                /*programme principal*/
}

open_vwork()
{
    register int i;                        /*index actuel*/
    int dummy;                             /*dummy*/
                                           /*workstation-handle physique*/
    phys_handle=graf_handle(&dummy,&dummy,&dummy,&dummy);
    vdi_handle=phys_handle;
    for(i=0;i<10;work_in[i++]=1);         /*initialiser le
                                           tableau*/

    work_in[10]=2;
    v_opnvwk(work_in,&vdi_handle,work_out); /*ouvrir le
                                           'virtual screen workstation'*/
}

menu_init()
{
    menu_id=menu_register(appl_id," Renommer Dossier");
                                           /*Annoncer l'accessoire*/
    if(menu_id==-1)                        /*erreur*/
    {
        form_alert
(1, "[1][Plus de place dans la barre de menus][ OK ]");
        appl_exit();                       /*fin du programme*/
    }
}

loop()
{
    char buff1[513],
        buff2[513],
        ring[14];

    register int xx,tt,zz;

    int secteurs,
        longueur,
        debut,
        disc,
        flag,
        *biospbptr;

    while(TRUE)                            /*boucle sans fin*/
    {
        graf_mouse(0,0L);
        evnt_mesag(msgbuff);               /*gchetteur d'évènement*/
        if((msgbuff[0]==AC_OPEN) && (msgbuff[4]==menu_id))
                                           /*accessoire
                                           sélectionné*/

```

```

{
    wind_update(TRUE);           /*ne pas tenir compte de
                                l'évènement*/
    dialog();                     /*afficher nom du
                                dossier*/
    wind_update(FALSE);          /*autoriser l'évènement
                                engendré*/

    flag=FALSE;
    disc=Dgetdrv();               /*unité de disque actuelle*/
    biospbptr=(int *)Getbpb(disc); /*charger les données du
                                disque dans BPB-buff1*/
                                /*nombre total des
                                secteurs sur le disque*/

    secteurs=((*(biospbptr+1))*(*(biospbptr+7)))+(*(biospbptr+6));
    xx=0;
    while((xx<secteurs)&&(flag==FALSE))
    {
        Rwabs(0,buff1,1,xx,disc); /*lire à chaque fois un
                                secteur*/
        longueur=strlen(name_old); /*longueur de l'ancien
                                nom du dossier*/

        for(zz=0;zz<512;++zz)
            *(buff2+zz)=*(buff1+zz); /*sauver contenu du
                                buffer*/

        tt=0;
        while((tt<512)&&(flag==FALSE)) /*contrôler le contenu
                                du buffer*/
        {
            for(zz=0;zz<longueur;++zz)
                *(ring+zz)=*(buff2+tt+zz); /*selon la longueur du
                                nom*/

            *(ring+zz+1)='\0'; /*copier les octets
                                dans le buffer intermédiaire*/
            if((strcmp(name_old,ring))==0)/*et le comparer avec
                                les noms*/
            {
                debut=tt; /*nom retrouvé ->
                                écrire adresse de départ*/
                flag=TRUE; /*dans le buffer et
                                placé un flag d'identification*/
            }
            else
                flag=FALSE;
            tt++;
        }
        xx++;
    }
    if(flag==TRUE) /*nom retrouvé*/
    {
        zz=0;
    }
}

```

```

        while((* (name_new+zz)) != '\0') /*recopier le nouveau
                                         nom dans le buffer*/
        {
            *(buff1+debut+zz)=*(name_new+zz);
            zz++;
        }
        Rwabs(1,buff1,1,xx-1,disc); /*puis recopier le
                                     secteur modifié*/
        form_alert(1,"[1][Dossier renommé] | RESET !!![ OK ]");
    }
else
    form_alert(1,"[1][Dossier non trouvé] [ OK ]");
}
}
}

dialog()
{
    GRECT box; /*coordonnées de la
                boîte de dialogue*/

    char buffer[15];
    register int xx; /*index actuel*/
    int longueur1,
        longueur2;

    for(xx=0;xx<12;+xx) /*vider la zone des
                          noms des fichiers*/
        name_old[xx]=EOS;

    /*centrer la boîte de
    dial*/
    form_center(tree1,&box.g_x,&box.g_y,&box.g_w,&box.g_h);
    graf_mbox(20,20,50,16,320,200); /*dessiner un rectangle
                                       allongé*/
    /*réservé de la place
    sur l'écran*/

    form_dial
(FMD_START,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
    /*dessiner un rectangle
    allongé*/

    form_dial(FMD_GROW,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
    /*boîte de dialogue*/

    objc_draw(tree1,0,MAX_DEPTH,box.g_x,box.g_y,box.g_w,box.g_h);
    form_do(tree1,0); /*traiter les réponses
                       fournies*/

    tree1[3].ob_state &= SELECTED; /*redessiner un bouton*/
    /*dessiner un rectangle
    à coins arrondis*/

    form_dial
(FMD_SHRINK,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);

```

```

                                                    /*libérer de la place
                                                    sur l'écran*/
form_dial
(FMD_FINISH,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
strcpy(buffer,(&teds1[0])->te_ptext);
strcpy(name_old,buffer);

for(xx=0;xx<12;++xx)                                /*effacer la zone des
                                                    noms des
fichiers*/
    name_new[xx]=EOS;
/*centrer la boîte de dial*/
form_center(tree2,&box.g_x,&box.g_y,&box.g_w,&box.g_h);
graf_mbox(20,20,50,16,320,200); /*dessiner un rectangle
allongé*/
                                                    /*réservé de la place
                                                    sur l'écran*/
form_dial
(FMD_START,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
                                                    /*dessiner un rectangle
allongé*/
form_dial(FMD_GROW,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
                                                    /*boîte de dialogue*/
objc_draw(tree2,0,MAX_DEPTH,box.g_x,box.g_y,box.g_w,box.g_h);

form_do(tree2,0);                                    /*traiter les réponses
fournies*/
tree2[3].ob_state &= SELECTED; /*redessiner un bouton*/
                                                    /*dessiner un rectangle
à coins arrondis*/
form_dial
(FMD_SHRINK,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
                                                    /*libérer de la place
sur l'écran*/
form_dial
(FMD_FINISH,320,200,20,20,box.g_x,box.g_y,box.g_w,box.g_h);
strcpy(buffer,(&teds2[0])->te_ptext); /*sauver le texte
saisi*/
strcpy(name_new,buffer);

if((longueur1=strlen(name_new))<11) /*remplir les noms
avec des espaces vides*/
{
    longueur2=11-longueur1;
    for(xx=0;xx<longueur2;++xx)
        *(name_new+longueur1+xx)='\ ';
}
}

```

2.5. Programme de copie automatique dans disque RAM

Depuis qu'il existe des micro-ordinateurs personnels disposant d'une mémoire vive de plusieurs centaines de K-octets, l'usage d'un disque RAM est apparu comme une alternative très rapide et pratique par rapport au travail constant avec l'unité de disquette. Ce disque RAM peut s'installer automatiquement lors du processus d'initialisation, et il serait encore plus confortable de pouvoir y charger directement lors du démarrage les programmes et fichiers les plus utilisés (traitement de texte, compilateur etc). C'est justement le but du programme que nous vous donnons ci-dessous avec son mode d'emploi.

Voici sommairement la façon dont il fonctionne:

1. il faut d'abord fabriquer un fichier texte contenant les noms des fichiers à recopier (ce que vous pouvez faire avec n'importe quel traitement de texte), qui devra porter le nom LISTE.LST et se trouver dans le répertoire principal de la disquette boot.
2. les fichiers à recopier devront eux-aussi se trouver dans le répertoire principal de la dqt.
3. Après le démarrage, l'ordinateur lit le fichier répertoire LISTE.LST et le recopie dans un buffer. Puis il installe un buffer-RAM dans lequel il enregistre un par un les fichiers (en suivant l'ordre dans lequel ils sont mentionnés). Il transfère ensuite le contenu du buffer-RAM dans le disque RAM. Le programme ci-dessous prévoit un buffer RAM de 200Koctets: cette taille peut être augmentée en tenant compte des capacités de votre mémoire vive).
4. Le buffer est vidé une fois les fichiers recopiés dans le disque RAM et le programme prend fin.

Avant d'expliquer plus en détail les particularités de ce programme, voici une description des routines du système d'exploitation utilisées:

Malloc()

GEMDOS-numéro 0x48

Appel: long Malloc(quantité)
 long quantité;

Si 'quantité' est égal à -1, Malloc communique le nombre d'octets encore libre dans le plus grand des blocs. Si 'quantité' prend une valeur différente de -1, le nombre d'octets ainsi affiché par 'quantité' est réservé. Malloc renvoie un pointeur sur cet espace libre. Si la taille mémoire ne suffit pas, le résultat retourné est 0L.

Fopen() **GEMDOS-numéro 0x3D**

Appel: int Fopen(filename,mode)
 char *filename
 int mode;

La fonction Fopen sert à ouvrir un fichier de nom 'filename'. C'est la valeur de 'mode' qui détermine ce qui doit advenir de ce fichier. 'mode' peut prendre les valeurs suivantes:

0	seulement lecture
1	seulement écriture
2	lecture et écriture

Si le processus a bien lieu, la valeur retournée de la fonction est le numéro handle du fichier; sinon le programme transmet un numéro d'erreur négatif.

Fread() **GEMDOS-numéro 0x3F**

Appel: long Fread(handle,quantité,buffer)
 int handle;
 long quantité;
 char *buffer;

Cette fonction permet de reprendre dans le fichier portant le numéro 'handle' le nombre d'octets précisé par 'quantité' et de les enregsitrer dans le buffer désigné sous 'buffer'. Si le processus a bien lieu, la valeur retournée de la fonction représente le nombre d'octets lus; sinon le programme transmet un numéro d'erreur négatif.

Fwrite() **GEMDOS-numéro 0x40**

Appel: long Fwrite(handle,quantité,buffer)
 int handle;
 long quantité;
 char *buffer;

Cette fonction permet de reprendre dans le buffer indiqué par 'buffer' le nombre d'octets précisés par 'quantité' et de les enregistrer dans le fichier désigné par le numéro 'handle'. Si le processus a bien lieu, la valeur retournée de la fonction représente le nombre d'octets enregistrés; sinon le programme transmet un numéro d'erreur négatif.

Fcreate()

GEMDOS-numéro 0x3c

Appel: int Fcreate(filename,attribut)
 char *filename;
 int attribut;

La fonction Fcreate permet de créer un fichier portant le nom 'filename'. La valeur fixée sous 'attribut' peut avoir une des significations suivantes:

0x00	fichier normal
0x01	seulement lecture
0x02	fichier dissimulé
0x04	fichier système
0x08	nom d'une dqf

Si le processus a bien lieu, la valeur retournée de la fonction est le numéro handle, sinon le programme transmet un numéro d'erreur négatif.

Fclose()

GEMDOS-numéro 0x3c

Appel: int Fclose(handle)
 int handle;

La fonction Fclose permet de clore un fichier portant le numéro 'handle'. Si le processus a bien lieu, la valeur retournée de la fonction est 0, sinon le programme transmet un numéro d'erreur négatif.

Mfree()

GEMDOS-numéro 0x49

Appel: long Mfree(pointer)
 long pointeur;

La fonction Mfree permet de vider la zone mémoire réservée par Malloc et dont le début est indiqué par 'pointeur'. Si le processus a bien lieu, la valeur retournée de la fonction est 0L, sinon le programme transmet un numéro d'erreur négatif.

Fsetdta()**GEMDOS-numéro 0x1A**

Appel: void Fsetdta(buffer)
 char *buffer;

La fonction Fsetdta sert à préciser l'adresse de transfert sur disque (DTA = disc-transfer-address). Les fonctions Ffirst et Fnext enregistrent leurs informations dans ce buffer. Normalement, c'est le TOS qui fixe ce buffer DTA d'une taille de 44 octets. Mais lorsqu'on veut faire traiter les informations contenues dans ce buffer par un autre programme, la fonction Fsetdta permet de transformer en DTA une zone quelconque de la mémoire déterminée par le programme. Dans le programme ci-dessous, c'est la structure DTAREC qui s'en charge; elle est construite de la façon suivante:

```
typedef struct dtarec
{
  char reserved[21];
  char attribut;
  int time;
  int date;
  long size;
  char name[14];
}DTAREC;
```

Les 21 premiers octets sont réservés pour le système d'exploitation. Vient ensuite l'attribut du fichier, qui se compose d'une combinaison des possibilités suivantes:

0x00	accès normal (lecture/écriture)
0x01	accès normal (protégé contre l'écriture)
0x02	saisie dissimulée
0x04	saisie-système dissimulée
0x08	nom de la dqt
0x10	sous-répertoire
0x20	fichier ayant été écrit puis refermé (statut archivé)

Les deux enregistrements suivants indiquent l'heure et la date du fichier. Sous 'size' se trouve la taille du fichier sous forme de string long. Sous 'name' enfin se trouve le nom du fichier et son extension avec l'octet nul terminal (format C-string).

Ffirst()**GEMDIS-numéro 0x4E**

Appel: int Ffirst(name,attribut)
 char *name;
 int attribut;

Ffirst recherche dans le répertoire actuel la première mention du fichier indiqué par 'name'. On peut ici utiliser les masques ? ou *. Les données saisies sous 'attribut' peuvent servir à limiter la recherche. Vous trouvez sous Fsetdta des précisions en ce qui concerne la façon de se servir des valeurs de l'attribut. Si le processus a bien lieu, la valeur retournée de la fonction est 0, -33, lorsque la fonction ne peut plus trouver de fichier (le répertoire n'est pas vide mais il ne contient plus de fichier portant le nom voulu); la valeur de retour est -49 lorsqu'il n'y a plus de fichier, lorsque le répertoire est vide.

Après avoir décrit les routines du système d'exploitation les plus utilisées dans ce programme, nous allons voir de plus près la façon dont il se déroule.

Commençons par la définition de la structure décrite ci-dessus qui doit nous servir de buffer DTA et dans laquelle la fonction Ffirst va enregistrer les informations propres au fichier dont nous avons besoin.

Vient ensuite l'installation d'un buffer (datbuffer) qui va servir à enregistrer le contenu du fichier-répertoire LISTE.LST. Puisque nous en parlons, c'est le moment d'expliquer la structure de ce fichier.

La première ligne contient l'identificateur du disque RAM, suivi d'un double-point et d'un backslash, par exemple: F:\. Viennent ensuite à partir de la deuxième ligne les noms des fichiers à recopier, précédés eux-aussi de l'identificateur de l'unité de disque concernée, par exemple:

```
A:\PROGRAM1.PRG
A:\PROGRAM2.PRG
```

Un fichier-répertoire complet aura donc l'allure suivante:

```
F:\
A:\PROGRAM1.PRG
A:\PROGRAM2.PRG
```

Ce sera tout au sujet de ce fichier-répertoire, poursuivons maintenant la description du déroulement du programme. La fonction surch_file sert à vérifier si le fichier-répertoire se trouve bien sur la disquette boot. La fonction Fsetdta sert

ensuite à installer le buffer DTA. La fonction `Fsfirst` recherche alors le fichier nommé sous 'filename', que nous avons ici nommé `LISTE.LST`, dans le répertoire actuel qui est ici le répertoire principal de la disquette. Lorsque le fichier est retrouvé, la fonction prend la valeur 0, sinon une valeur inférieure.

S'il ne trouve pas le fichier-répertoire, le programme s'interrompt en envoyant un message d'erreur, sinon il transmet la taille de ce fichier au moyen des informations qui sont transmises auparavant par le biais de `Fsfirst` et chargées dans le buffer DTA. Nous avons limité la taille de ce fichier à 4000 octets, mais vous pouvez la modifier comme bon vous semble. Ici aussi, en cas de dépassement de la taille autorisée, le programme renvoie un message d'erreur et s'interrompt.

Si tout s'est bien passé, nous en sommes à l'intervention de la fonction `Malloc`, qui sert à réserver un buffer RAM de 200 Koctets qui va servir au transit des fichiers à recopier. S'il n'y a pas assez de place-mémoire disponible, le programme renvoie un message d'erreur et s'interrompt. Naturellement, vous pouvez changer vous-même la taille de ce buffer RAM: il vous suffit d'inscrire la taille désirée sous forme de valeur longue dans la fonction `Malloc`.

La fonction `Fopen` permet d'ouvrir le fichier-répertoire, `Fread` de l'enregistrer dans le buffer installé auparavant et enfin `Fclose` de le refermer proprement.

Deux boucles servent ensuite à lire toutes les lignes du fichier-répertoire une par une et à les formater sous forme de chaîne de caractères C-String (chaque ligne transmise reçoit une marque de fin de string '\0'). Un pointeur désignant le début de chaque ligne est enfin transmis à la fonction `filecopy()`. Ce processus se répète autant de fois qu'il le faut pour que toutes les lignes soient traitées, c'est-à-dire pour que l'index courant 'index1' soit égal à la longueur de fichier mentionnée sous 'longueur'.

La fonction `filecopy()` utilise la fonction `surch-file()` que nous avons déjà décrite, pour contrôler si le fichier transmis se trouve aussi sur la disquette boot. Si la recherche reste infructueuse, le programme envoie un message d'erreur.

Si le fichier existe bien, le programme indique la taille du fichier qui va être traité; il prend cette information dans le buffer DTA qui a lui même reçu les données spécifiques aux fichiers grâce à la fonction `Fsfirst`.

Les fonctions `Fopen`, `Fread` et `Fclose` permettent ensuite de lire le fichier se trouvant dans le buffer RAM

Pour recopier dans le disque RAM un fichier se trouvant dans le buffer RAM, il faut le créer dans le disque RAM par la fonction Fcreate. Pour ce faire, il faut avoir l'intitulé complet du chemin d'accès qui se compose de l'identificateur de l'unité de disque, plus le chemin, plus le nom du fichier. Il faut donc modifier l'intitulé du chemin d'accès existant, puisque son identificateur d'unité de disque est celui de la disquette boot, c'est-à-dire dans notre exemple A:\PROGRAM1.PRG portant donc l'identificateur A. Il faut modifier l'identificateur A pour qu'il corresponde à celui du disque RAM, dans notre exemple F. Comme nous avons indiqué l'identificateur de l'unité de disque cible dans la première ligne du fichier-répertoire, cet identificateur se trouve à la première position du buffer 'datbuffer'. Comme le nom d'un secteur est en même temps l'adresse de son premier élément, il nous suffit de transmettre le contenu de cet élément à la place appropriée de la ligne actuelle. On écrit la ligne de programme :

```
*filename=*datbuffer.
```

L'intitulé du chemin d'accès est alors complet, ce qui permet de créer le fichier sur le disque dur à l'aide de Fcreate. Le numéro handle ainsi obtenu est repris dans la fonction Fwrite pour recopier dans le disque RAM le fichier qui se trouvait dans le buffer RAM. Après l'avoir fermé par Fclose, le fichier est dorénavant disponible dans le RAMdisque. Ce processus se répète pour chaque ligne du fichier-répertoire jusqu'à ce que tous les fichiers aient été copiés.

Nous espérons que nous vous avons ainsi fourni toutes les explications nécessaires sur le fonctionnement du programme de recopiage qui vous simplifiera le travail avec le disque RAM. Voici ce programme : COPYRAM.C

```
/*Programme de recopie automatique dans le disque RAM*/

#include <stdio.h>
#include <osbind.h>
#include <ctype.h>
typedef struct dtarec          /*Buffer DTA dans lequel sera */
                               /*enregistré le fichier*/
{
    char reserved[21];         /*réservé*/
    char attribut;             /*attribut du fichier*/
    int time;                  /*heure-système*/
    int date;                  /*date-système*/
    long size;                 /*taille du fichier*/
    char name[14];            /*nom du fichier*/
} DTAREC;

DTAREC dtabuffer;
```

```

char datbuffer[1000];          /*buffer pour le
                                fichier-répertoire*/
long rambuffer;                /*buffer pour les fichiers à*/
                                /*recopier*/

main()
{
    long longueur;             /*longueur du
                                fichier-répertoire*/

    int fhandle,                /*handle du fichier-répertoire*/
        index1,                 /*1er index courant*/
        index2,                 /*2ème index courant*/
        ret;                    /*retour de la fonction
                                recherche*/

    char liste[10];            /*string pour le nom du fichier-
                                répertoire*/

    index1=3;
    printf("Recherche de LISTE.LST...\n"); /*début du
                                                programme*/

    strcpy(liste,"LISTE.LST"); /*Nom du fichier-répertoire*/
    ret=surch_file(liste);     /*le fichier existe?*/
    if(ret!=0)                  /*si non, alors*/
        exit(1);                /*clôre le programme*/
    else                         /*si oui, alors*/
    {
        printf("Début de copie dans la RAM !\n");
        longueur=dtabuffer.size; /*indiquer la longueur du
                                    fichier-répertoire*/
        if(longueur>1000)        /*si longueur supérieure au buffer*/
        {
            printf("\nvotre liste est trop longue!\n");
            printf("\nTouche\n");
            Cconin();            /*attendre appui sur une touche*/
            exit(1);            /*clôre le programme*/
        }
        rambuffer=Malloc(51200L); /*indiquer l'adresse de début du
                                    buffer de 200Ko*/
        if (rambuffer==0L)       /*si place mémoire insuffisante*/
        {
            printf("\n pas assez de place mémoire! \n");
            printf("\nTouche\n");
            Cconin();            /*attendre appui sur une touche*/
            exit(1);            /*clôre le programme*/
        }
        fhandle=Fopen(liste,0); /*ouverture du fichier-répertoire
                                    et communication du handle*/
        Fread(fhandle,longueur,datbuffer); /*enregistrer le
                                    fichier-répertoire dans le buffer*/
        Fclose(fhandle);        /*fermer le fichier répertoire*/
        while(index1 < longueur) /*tant que 1er index
                                    courant<longueur du fichier-répertoire*/

```

```

{
    while(((*(datbuffer + index1)) < 32) && (index1 <
longueur))
        ++index1;          /*augmenter le ler index courant*/
    index2=index1;         /*sauver position du ler index*/
    if(index1 < longueur) /*si ler index < longueur du
                           fichier-répertoire*/
    {
        while(*(datbuffer+index1) > 32) /*pas de code de
            commande que buffer contient un caractère*/
            ++index1;          /*augmenter ler index*/
        *(datbuffer+index1)=0;      /*ajouter fin de string '\0'*/
        filecopy((datbuffer+index2)); /*indiquer nom du
                                       fichier à recopier*/
    }
}
Mfree(rambuffer);      /*vider le buffer-fichiers*/
exit(1);                /*clôre le programme*/
}

surch_file(filename)
char *filename;
{
    int ret;

    Fsetdta(&dtabuffer); /*déterminer l'adresse de début du
                           buffer dta*/
    ret=Ffirst(filename,0); /*rechercher le nom de fichier*/
    if(ret!=0)              /*fichier non-trouvé*/
    {
        printf("\nfichier %s non trouvé \n",filename);
        printf("\nTouche\n");
        Cconin();          /*attendre appui d'une touche*/
    }
    return(ret);          /*fichier trouvé - oui (==0) ou
                           non (!=0)*/
}

filecopy(filename)
char *filename;          /*nom du fichier à recopier*/
{
    long longueur;      /*longueur du fichier*/
    int fhandle,        /*handle de l'ancien fichier*/
        rhandle,       /*handle du fichier recopié*/
        ret;
    ret=surch_file(filename); /*fichier trouvé?*/
    if(ret == 0)          /*si oui, alors*/
    {
        longueur=dtabuffer.size; /*indiquer longueur du fichier*/
        fhandle=Fopen(filename,0); /*ouvrir le fichier*/
        Fread(fhandle,longueur,rambuffer); /*enregist. fichier

```

```

                                dans buffer*/
Fclose(fhandle);                /*fermer le fichier*/
*filename=*datbuffer;          /*indiquer nouvelle unité de
                                disque*/
if((rhandle=Fcreate(filename,0))<=0) /*si le fichier ne
                                        peut être créé*/
{
    printf("\nle fichier %s n'a pu être créé !\n",filename);
    printf("\nTouche\n");
    Cconin();                    /*attendre appui sur une
                                touche*/
}
else                            /*le fichier a pu être créé*/
{
    Fwrite(rhandle,longueur,rambuffer); /*enregistrer le
                                        fichier à recopier depuis le buffer*/
    Fclose(rhandle);            /*fermer le fichier*/
}
}
}
```

Chapitre 3

Programmes de conversion

La conversion de données est un des problèmes importants en matière de traitement informatique. La plupart des programmes et logiciels rangent les données d'une façon qui leur est tout à fait propre, ce qui les rend illisibles sous un autre logiciel. Ces derniers temps, presque tous les logiciels offrent la possibilité d'enregistrer les données dans des fichiers ASCII par les commandes nommées Exporter, Importer etc., mais certains problèmes apparaissent alors par exemple avec 1st Word Plus. On appelle fichier ASCII un fichier contenant des données sans aucun signe ou code de commande, qui ne contient donc que des lettres ou des chiffres, ainsi que les CR (carriage Return) et LF (LineFeeds) qui sont les signes d'espacement.

Pour pouvoir écrire un programme de conversion, il faut donc tout d'abord pouvoir analyser les fichiers sources et cibles. Nous vous donnons donc ci-dessous un programme qui vous permet de visualiser toutes les données d'un fichier sous divers modes de représentation.

```
'
' Programme: READBIN.LST
'
' Ce programme peut afficher le contenu d'un fichier quelconque
' ainsi que l'éditer
'
' Hidem                !désactiver la souris
Dim Valeur%(25)
Dossier1$=Space$(64)   !string du chemin d'accès
actuel
A=Gemdos(71,L:Varptr(Dossier1$),0) !routine du système
                                d'exploitation
'                               !pour le chemin d'accès
```

```

For I%=0 To 63
  Exit If Mid$(Dossier1$,I%,1)=Chr$(0) !les strings se terminent
  ,                                     !par 0
Next I%
,
If I%=0                                !string vide ?
  Dossier$="\*.*"                       !nom par défaut
Else                                     !sinon
  Dossier$=Left$(Dossier1$,I%-1)+"\*.*" !reprendre chemin actuel
Endif
,
Do
  Cls
  Fileselect Dossier$, "", Name$        !choix du fichier
  Exit If Name$=""                      !jusqu'à ANNULLATION
  If Exist(Name$) <> 0
    ' uniquement si le fichier existe
    Open "u", #1, Name$                 !ouvrir fichier
    If Lof(#1)=0                        !fichier vide?
      Alert 1, "fichier=0 octet", 1, "ANNULER", Dummy% !message puis
      ,                                  !interruption
    Else                                  !sinon
      Longueur%=Lof(#1)                !noter la longueur du fichier
      Erase Date%( )                   !détruire ancien tableau
      Dim Date%(Int (Longueur%/4)+1)   !dimensionner le tableau
      Bload Name$, Varptr(Date%(0))    !charger le fichier
      Dat_start%=Varptr(Date%(0))     !adresse de début du tableau
      Adresse%=0                       !compteur dans le fichier
      Cls                               !vider l'écran
      Box 2*8-5, 2*16-6, 41*8+5, 23*16+5 !créer le masque d'écran
      Box 50*8-5, 7*16-6, 70*8+5, 8*16+5
      Line 2*8-5, 3*16-5, 41*8+5, 3*16-5
      Line 10*8+8, 2*16-5, 10*8+8, 23*16+5
      Line 17*8+8, 2*16+5, 17*8+8, 23*16+5
      Line 24*8+8, 2*16+5, 24*8+8, 23*16+5
      Line 37*8+8, 2*16+5, 37*8+8, 23*16+5
      Text 3*8, 2*16+8, "Adresse"
      Text 12*8, 2*16+8, "Dec"
      Text 19*8, 2*16+8, "Hex"
      Text 26*8, 2*16+8, "Binaire"
      Text 40*8, 2*16+8, "A"
      Text 50*8, 11*16-3, "Touches fonctions:"
      Text 50*8, 13*16-8, "Return page suivante"
      Text 50*8, 14*16-8, "N page suivante"
      Text 50*8, 15*16-8, "L page précédente"
      Text 50*8, 16*16-8, "Home page un"
      Text 50*8, 17*16-8, "Insert dernière page"
      Text 50*8, 18*16-8, "A nouvelle adresse"
      Text 50*8, 20*16-8, "E éditer"
      Text 50*8, 21*16-8, "X nouveau fichier"
      Text 50*8, 22*16-8, "Q fin"

```

```

'
Let Name1$=""
Let Name_len%=Len(Name$) !longueur du nom du fichier"
While (Mid$(Name$,Name_len%,1)<>"\") And Name_len%>0
  Let Name1$=Mid$(Name$,Name_len%,1)+Name1$
  Dec Name_len%
Wend
Text 50*8,2*16+8,"Fichier: "+Name1$
@Affichage
Do
  Let Touche$=Upper$(Inkey$) !Touche activée
  '
  If Touche$="Q" !Fin du programme
    Goto E.nd
  Endif
  '
  If Touche$="X" !traiter le nouveau fichier
    Goto E.nl
  Endif
  '
  If Touche$="A" !afficher à partir nouvelle
    adresse
    Do
      Print At(3,1);" ";
      Print At(3,1);"nouvelle adresse: ";
      Input Start%
      Exit If Start%>=0 And Start%<=Longueur%
    Loop
    Print At(3,1);" ";
    If Start%+20>Longueur% !dépassement
      Adresse%=Longueur%-20
    Else !sinon reprendre nouvelle
      adresse
      Adresse%=Start%
    Endif
    @Affichage !et afficher les données
  Endif
  '
  ' page suivante
  '
  If
  (Touche$=Chr$(13) Or Touche$="N") And Adresse%+20<Lof(#1)
    If Adresse%+40<Longueur%
      Add Adresse%,20
    Else
      Adresse%=Longueur%-20
    Endif
    @Affichage
  Endif
  '
  If Touche$="L" And Adresse%>0 !afficher page précédente

```

```

    If Adresse%<20                                !début
      Adresse%=0                                  !commence au début
    Else
      Sub Adresse%,20                            !sinon une page en arrière
    Endif
    @Affichage                                    !et afficher
  Endif
  '
  If Touche$="E"                                  !éditer les données
    @E.dit                                       !dans procédure Edit
  Endif
  '
  If Left$(Touche$,1)=Chr$(0)                   !Touches de fonctions
  '
  If Mid$(
    (Touche$,2,1)=Chr$(&H50) And Adresse%+20<Lof(#1) !cursor
  '
    'down
    Inc Adresse%                                !descendre d'une ligne
    If Adresse%>=Longueur%                      !dépasse la fin
      Adresse%=Longueur%-1                    !si oui, aller à la fin
    Endif
    Get 3*8,4*16,42*8,23*16-1,I.mage$        !défilement par
    '                                           GET et
    '                                           !PUT
    Put 3*8,3*16,I.mage$,3
    Get 51*8,7*16-3,70*8,8*16,I.mage$
    Put 50*8,7*16-3,I.mage$,3
    @Outdown                                    !donner une nouvelle ligne
  Endif
  '
  If Mid$(Touche$,2,1)=Chr$(&H48) And Adresse%>0 !cursor up
    Dec Adresse%                                !une ligne plus haut
    If Adresse%<0                              !déjà au début
      Adresse%=0                               !si oui, aller au début
    Else
      Get 3*8,3*16,42*8,22*16-1,I.mage$      !défilement par
      '                                           GET et
      '                                           !PUT
      Put 3*8,4*16,I.mage$,3
      Get 50*8,7*16-3,69*8-1,8*16,I.mage$
      Put 51*8,7*16-3,I.mage$,3
      @Outup                                    !donner une nouvelle ligne
    Endif
  Endif
  '
  If Mid$(Touche$,2,1)=Chr$(&H47) And Adresse%>0 !Home
    Adresse%=0                                  !Début du fichier
    @Affichage                                    !afficher une page
  Endif
  '
  If Mid$

```

```

(Touche$,2,1)=Chr$(&H52) And Adresse%+20<Longueur%
,
    Adresse%=Longueur%-20      !Insert
    @Affichage                 !Fin du fichier
    Endif                     !afficher une page
,
Endif
,
Loop
Endif
Endif
E.n1:
Close #1                       !fermer le fichier
Loop
E.nd:
Close #1
Edit
,
Procedure Affichage             !Affiche une page entière
For I%=0 To 19                 !20 lignes en tout
Exit If Adresse%+I%>=Longueur% !arrêter à la fin
Valeur%(I%)=Peek(Dat_start%+Adresse%+I%) ! chercher octets
Print At (3,I%+4);"";
Print Using "#####",Adresse%+I%      !Afficher l'adresse
Print At (13,I%+4);"";
Print Using "###",Valeur%(I%)         !affichage décimal
Print At (20,I%+4);"";
Print
"$";Left$("00",2-Len(Hex$(Valeur%(I%))))+Hex$(Valeur%(I%))
,
    !affichage hexadécimal
Print At (27,I%+4);"";
Print
"$";Left$("0000000",8-Len(Bin$(Valeur%(I%))))+Bin$(Valeur%(I%))
,
    !Binaire
Text 40*8,I%+4*16-3,Chr$(Valeur%(I%)) ! ASCII
Text 50*8+I%*8,8*16-3,Chr$(Valeur%(I%)) ! String
Next I%
Return
,
Procedure Outdown              !ajouter de nouvelles lignes
I%=1
Valeur%(I%)=Peek(Dat_start%+Adresse%+19) !chercher octets
Print At (3,23);"";
Print Using "#####",Adresse%+19      !afficher l'adresse
Print At (13,23);"";
Print Using "###",Valeur%(I%)         !décimal
Print At (20,23);"";
Print
"$";Left$("00",2-Len(Hex$(Valeur%(I%))))+Hex$(Valeur%(I%))!Hexa
Print At (27,23);"";

```

```

Print
"%";Left$("00000000",8-Len(Bin$(Valeur%(I%))))+Bin$(Valeur%(I%))
,
!Binaire
Text 40*8,23*16-3,Chr$(Valeur%(I%)) !ASCII
Text 50*8+19*8,8*16-3,Chr$(Valeur%(I%)) !String
Return
,
Procedure Outup !Ajoute de nouvelles lignes en haut
I%=1
Valeur%(I%)=Peek(Dat_start%+Adresse%) !chercher octets
Print At(3,4);"";
Print Using "#####",Adresse% !Afficher l'adresse
Print At(13,4);"";
Print Using "###",Valeur%(I%) !Décimal
Print At(20,4);"";
Print
"$";Left$("00",2-Len(Hex$(Valeur%(I%))))+Hex$(Valeur%(I%))!Hexa
Print At(27,4);"";
Print
"%";Left$("00000000",8-Len(Bin$(Valeur%(I%))))+Bin$(Valeur%(I%))
,
!Binaire
Text 40*8,4*16-3,Chr$(Valeur%(I%)) !ASCII
Text 50*8,8*16-3,Chr$(Valeur%(I%)) !String
Return
,
Procedure E.dit !Edition des données
Do !Rechercher l'adresse de début
Print At(3,1);" ";
Print At(3,1);" ";
Input "Adresse de début :";Start%
Exit If Start%>=0 And Start%<=Longueur% !pas plus petit que
début
!ni plus grand que fin
Loop
Print At(3,1);" ";
U=Peek(Dat_start%+Start%) ! rechercher ancien octet
Do
Print At(3,1);" ";
Print At(3,1);" Ancienne valeur: ";U;"Nouvelle valeur: ";
Input Valeur% ! entrer nouvelle valeur
Exit If Valeur%>=0 And Valeur%<=255
Loop
Print At(3,1);" ";
Seek #1,Start% !Pointeur de fichier dirigé sur l'
' !octet à modifier
Out #1,Valeur% !enregistrer octet
Poke Dat_start%+Start%,Valeur% ! et le reprendre dans le
buffer
Adresse%=Start% !prendre la nouvelle adresse

```

```
@Affichage          !et afficher une nouvelle pleine
                    page
Return
```

Vous voyez s'afficher sur votre écran un petit mode d'emploi qui vous évite tout problème d'utilisation.

3.1. Conversion de 1st Word Plus en ASCII

Nous allons maintenant vous montrer comment procéder pour écrire un programme de conversion en prenant pour exemple le logiciel 1st Word Plus.

Le logiciel 1st Word Plus vous permet de sauvegarder des fichiers sous forme ASCII en désactivant l'option TT du menu "Traitement"; l'enregistrement permet alors de créer un fichier ne contenant presque plus de codes de commande. Regardons un de ces fichiers: saisissez vous-même quelques lignes de texte libre, puis enregistrez-les en mode ASCII. Remarquez que le programme de conversion doit pouvoir repérer tous les signes particuliers, comme les paragraphes, les sauts de page etc. Pour les fichiers de texte, il faut aussi tenir compte de lettres particulières, comme le 'ç' ou le 'ù' sur lesquels nous reviendrons plus loin.

Le programme READBIN vous permet de lire le fichier texte ASCII: on remarque immédiatement que les signes CR et LF figurent à la fin de toutes les lignes. Ils ne devraient cependant servir qu'à marquer les sauts de ligne et de paragraphe, et c'est pourquoi les autres logiciels ne peuvent pas reprendre des fichiers ASCII générés sous 1st Word Plus. On ne peut plus par exemple modifier la largeur du texte, et les logiciels utilisant les caractères proportionnels (par exemple Signum) ont besoin de la justification qui est très pénible à enlever.

Voici donc un petit programme qui détruit les CR et LF en fin de ligne tout en les maintenant lorsqu'ils signalent un saut de paragraphe ou de ligne.

```
,
, Programme: CONV1ASC.BAS
,
, Ce programme convertit les fichiers ASCII issus de 1st Word
, en fichiers ASCII ordinaires
,
DO
FILESELECT "\*.asc", "", oldname$ !choix du fichier source
EXIT IF oldname$="" !jusqu'à ANNULATION
,
```

```

OPEN "I",#1,oldname$           !ouvrir fichier
longueur%=LOF(#1)              !indiquer la longueur du
                                fichier
CLOSE #1                       !et fermer le tableau
DIM date%(INT(longueur%/4)+1)  !dimensionner le tableau
BLOAD oldname$,VARPTR(date%(0)) !enregistrer les données
compteur%=1
DO
    point%=INSTR(compteur%,oldname$,".") !Rechercher le point
                                        pour
                                        !!l'extension
    EXIT IF INSTR(point%,oldname$,"\\")=0 !Point dans le nom du
fichier
    compteur%=point%+1           !sinon dans le nom du
                                dossier
LOOP
'
name_old$=LEFT$(oldname$,point%)+".OLD" !donner l'extension
"OLD" à
NAME oldname$ AS name_old$      !l'ancien fichier
'
OPEN "o",#1,oldname$           !fichier cible sous le même nom
'
start%=VARPTR(date%(0))        !adresse de début des données
lauf%=start%                    !variable dans le tableau des
données
i%=0                             !variable sur une ligne
DO
    EXIT IF lauf%>=start%+longueur%    !répéter jusqu'à avoir
                                        atteint la
                                        '
                                        !fin
PRINT AT(10,10);"CONVERSION DU FICHER : ";oldname$
IF PEEK(lauf%+i%)=13             !signe équivalent à CR?
    IF PEEK(lauf%+i%+2)<>13        !pas d'autre CR?
        FOR j%=0 TO i%-1         !archiver tous les signes sans CR
                                '
                                !ni LF
            OUT #1,PEEK(lauf%+j%)
        NEXT j%
        ADD lauf%,(i%+2)         !Pointeur sur une nouvelle ligne
        i%=0                     !Remettre à zéro compteur de ligne
    ELSE
        !sinon chercher les autres CR
        FOR j%=0 TO i%+3         !archiver tous les signes y compris
                                '
                                !les CR et LF
            OUT #1,PEEK(lauf%+j%) !Pour fins de paragr et lignes
                                vides
        NEXT j%
        ADD lauf%,(i%+4)         !Pointeur sur une nouvelle ligne
        i%=0                     !Remettre à zéro compteur de
                                lignes
DO

```

```

EXIT IF PEEK(lauf%) <> 13 !d'autres lignes vides?
      OUT #1,13           !si oui, archiver les CR
      OUT #1,10          !enregistrer les LF
      ADD lauf%,2        !Pointeur sur une nouvelle ligne
    LOOP
  ENDIF
ELSE                                     !pas de CR donc
  ADD i%,1                               !incrémenter le compteur de lignes
ENDIF
LOOP
CLOSE #1                                 !fermer le fichier
LOOP

```

3.2. Programme de conversion ASCII ordinaire

ASCII vers 1st-Word-Plus

Un petit problème apparaît également en traitant des textes ASCII avec 1st Word Plus. Essayez de charger sous ce logiciel un texte en ASCII standard. Vous voyez tout de suite que la largeur du texte n'est pas bonne. Ceci vient tout simplement de ce que 1st Word Plus n'identifie pas correctement les signes d'espaces vides ayant la valeur CHR\$(32): il les prend pour des lettres et ne peut donc pas couper correctement; pour ce faire, il aurait besoin de CHR\$(30) pour bien séparer les mots. Il est possible de remédier à cela en cherchant et remplaçant chaque espace, pour ensuite reformater le texte. Pour les textes un peu longs, cela pourrait vous demander un temps considérable; il est bien plus simple d'utiliser le programme ci-dessous:

```

'
' Programme: CONVASC1.BAS
'
' Ce programme convertit l'ASCII standard en ASCII de 1st Word Plus
'
' Hidem                               !désactiver la souris
dossier1$=SPACE$(64)                   !string du chemin d'accès actuel
a=GEMDOS(71,L:VARPTR(dossier1$),0)    !routine du système
                                        d'exploitation
                                        !pour le chemin d'accès
FOR i%=0 TO 63
  EXIT IF MID$(dossier1$,i%,1)=CHR$(0) !les strings se terminent
  '                                     !par 0
NEXT i%
'

```

```

IF i%=0                                !string vide ?
  dossier$="\"*.*)"                      !nom par défaut
ELSE                                    !sinon
  dossier$=LEFT$(dossier1$,i%-1)+"\"*.*)" !reprendre chemin actuel
ENDIF
'
DO
  CLS
  FILESELECT dossier$,",",name$        !choix du fichier
  EXIT IF name$=""                     !jusqu'à ANNULATION If
  IF EXIST(name$)<>0                    !uniquement si le fichier existe
    OPEN "i",#1,name$                  !ouvrir fichier
    longueur%=LOF(#1)                  !longueur du fichier
    CLOSE #1                            !fermer le fichier
    Erase Date%()                       ' détruire ancien tableau
    DIM date%(INT(longueur%/4)+1)       !dimensionner le tableau
    BLOAD name$,VARPTR(date%(0))        !charger le fichier
    start%=VARPTR(date%(0))             !adresse de début du tableau
    ,
    FOR lauf%=start% TO start%+longueur% !tout le fichier
      IF PEEK(lauf%)=32                  !signe vide
        POKE lauf%,30                    !à remplacer
      ENDIF
    NEXT lauf%
    ,
    BSAVE name$,start%,longueur%        !Archiver le fichier
  ENDIF
LOOP

```

3.3. Conversion de n'importe quelle sorte de fichiers

Vous disposez maintenant d'un programme utilisable avec un logiciel précis, 1st Word Plus; nous allons maintenant vous donner un programme universel de conversion; il permet la sauvegarde de n'importe quel texte en le dépouillant de tous les signes spéciaux et des codes de commande.

Vous devez indiquer les signes à modifier dans les lignes 'data' figurant en fin de programme: mentionnez d'abord le signe à changer puis le signe de remplacement. Vous pouvez remplacer un seul signe par autant d'autres que vous le voulez: par exemple, vous pouvez remplacer le "a" par l'expression "ceci est un exemple". Il faut clore l'expression par un zéro, ainsi que la liste des lignes 'data'. Vous vous doutez donc que le zéro est le seul signe qui ne soit pas modifiable!

```

'
' Programme: Conv_A_A.BAS
'
' Ce programme convertit des fichiers de n'importe quelle
' origine en un
' nouveau fichier selon le tableau Data en fin du programme
'
dossier1$=SPACE$(64)           !string du chemin d'accès
                                !actuel
a=GEMDOS(71,L:VARPTR(dossier1$),0) !routine du système
                                !d'exploitation
                                !pour le chemin d'accès
'
FOR i%=0 TO 63
  EXIT IF MID$(dossier1$,i%,1)=CHR$(0) !les strings se terminent
  '                                !par 0
NEXT i%
'
IF i%=0                        !string vide ?
  dossier$="\*.*"              !nom par défaut
  ELSE                          !sinon
  dossier$=LEFT$(dossier1$,i%-1)+"\*.*" !reprendre chemin actuel
ENDIF
'
DIM cible$(256)                !place pour les nouveaux strings
source$=""                     !détruire le fichier source
FOR i%=0 TO 255
  cible$(i%)=""                !détruire les strings de
  remplacement                 !remplacement
NEXT i%
compteur%=0                    !Compteur des données
'
DO
  READ q%                       !lire la date source
  EXIT IF q%=0                  !0 terminal, donc fin
  source$=source$+CHR$(q%)      !sinon ajouter au string
DO
  READ z%                       !lire la cible
  EXIT IF z%=0                  !0 terminal, donc fin
  cible$(compteur%)=cible$(compteur%)+CHR$(z%) !sinon ajouter
  '                                au string
LOOP                            !jusqu'au prochain 0
'
  INC compteur%                 !incrémenter le compteur
LOOP
'
DO
  FILESELECT dossier$,"",source$ !sélection du fichier
  '                                !à convertir
  EXIT IF source$=""            !pas de saisie donc fin
  '
  IF EXIST(source$)<>0          !uniquement si le fichier

```

```

OPEN "i", #1, source$           existe
longueur%=LOF(#1)              !ouvrir fichier
                                !indiquer la longueur du
                                fichier
CLOSE #1                        !puis le refermer
DIM date%(INT(longueur%/4)+1)  !réserver assez de place
'
start%=VARPTR(date%(0))        !adresse de début
BLOAD source$, start%          !charger le fichier
'
compteur%=1
DO
  point%=INSTR(compteur%, source$, ".") !Rechercher le point
  '                                     pour
  EXIT IF INSTR(point%, source$, "\")=0 !l'extension
  '                                     !Point dans le nom du
  compteur%=point%+1            fichier
  '                                     !sinon dans le nom du dossier
LOOP
source_old$=LEFT$(source$, point%)+ "OLD" !donner l'extension
'                                     "OLD" à
NAME source$ AS source_old$        !l'ancien fichier
'
OPEN "o", #1, source$           !ouvrir le fichier cible
'
FOR i%=0 TO longueur%-1        !archiver tous les signes
  q%=PEEK(start%+i%)           !Prendre les signes dans
  '                               la mémoire
  found%=INSTR(source$, CHR$(q%)) !contenu dans le fichier
  '                               source?
  IF found%>0                   !oui
  PRINT #1, cible$(found%-1);    !écrire alors le string
cible
  ELSE                           !sinon
  OUT #1, q%                     !écrire la ligne
  ENDF
NEXT i%
'
CLOSE #1                         !fermer le fichier cible
'
ENDIF
LOOP                             !autres fichiers
END
'
' Ne mettre aucun commentaire dans les lignes Data qui suivent!!
'
DATA 142,91,0
DATA 153,92,0
DATA 154,93,0
DATA 158,126,0

```

```
DATA 132,123,0  
DATA 148,124,0  
DATA 129,125,0  
DATA 221,64,0  
DATA 0
```

Dans cet exemple, nous avons converti un fichier en ASCII Atari pour faire un fichier imprimable directement sur une imprimante Epson ou compatible.

Chapitre 4

Trucs et astuces concernant

les logiciels standard

Quelques petites astuces suffisent souvent à améliorer considérablement l'utilisation des logiciels et quelques petits trucs rendent possibles des fonctionnalités qui paraissaient inaccessibles. Vous l'avez déjà constaté au sujet de 1st Word Plus dans le chapitre précédent. Nous allons vous donner maintenant quelques indications qui vous simplifieront le travail avec certains logiciels.

4.1. Un programme 'REM-Killer' pour le Basic-GFA

Plus un programme est long et plus le programmeur a tendance à y insérer des lignes de commentaires qui devraient lui permettre en principe de s'y retrouver dans son programme en le relisant. Les professionnels de l'informatique n'hésitent pas à écrire quelquefois deux ou trois lignes de commentaires pour une ligne de programmation. Notre petit Atari n'est pas un ordinateur géant, mais ceci ne doit pas vous empêcher d'écrire quelques remarques dans vos programmes. Alors que les commentaires ne prennent aucune place dans le programme courant sous un langage compilé, il n'en va pas de même avec l'interpréteur Basic. Si vous écrivez autant de lignes de commentaires que de programmation, la taille de votre programme en est doublée et prend donc deux fois plus de place tant dans la

mémoire vive que dans la mémoire de masse, sans compter que la vitesse de traitement peut aussi s'en trouver affectée dans une certaine mesure. C'est pourquoi il est conseillé, lorsqu'on utilise un langage interprété, de bien commenter le texte du programme, d'en conserver une version avec toutes les remarques, mais de se servir ensuite d'une version expurgée de tout commentaire pour le travail courant. C'est pour vous éviter d'avoir à faire ce travail ligne par ligne que nous vous donnons ci-après un programme REM-Killer (en anglais: REM=remarque et Kill=supprimer).

A quoi ressemble le texte du commentaire en Basic GFA? Soit la ligne débute par une apostrophe ou REM et elle se compose entièrement d'un commentaire, soit elle comprend du texte de programme suivi d'un point d'exclamation introduisant le commentaire. Selon le cas, il nous faut donc supprimer soit une ligne entière soit le restant d'une ligne. Etudiez d'abord le texte de ce programme :

```

'
' REM-Killer pour le Basic-GFA      REMKILL.BAS
'   MP   20-05-88
'
DIM z$(2000)
'
a$=" *** REM-Killer ***| (c) Data Becker| "
a$=a$+" Voulez-vous vraiment | détruire les commentaires ? "
ALERT 0,a$,0," Oui | Non  ",x%
'
IF x%=1
FILESELECT "\*.LST","",filename$
IF filename$="" OR filename$="\ "
END
ELSE
OPEN "I",#1,filename$                !lire les lignes de prg
'                                     après Z$( )
i%=1
WHILE NOT EOF(#1)
INC i%
LINE INPUT #1,z$(i%)
WEND
CLOSE #1
'
x%=0
WHILE x%<=i%
@commentaire
@remline
INC x%
WEND
'

```

```

OPEN "O",#1,filename$           !réécrire le programme
FOR x%=0 TO i%
  PRINT #1,z$(x%)
NEXT x%
CLOSE #1
ENDIF
ENDIF
END
'
PROCEDURE remline
  signe%=1
  WHILE MID$(z$(x%),signe%,1)=" "   !ignorer les espaces
    ,                               vides
    INC signe%
  WEND
  ,
  IF (MID$(z$(x%),signe%,1)="" ) OR
(MID$(z$(x%),signe%,3)="REM") !ligne de commentaire?
  DEC i%
  FOR z%=x% TO i%
    z$(z%)=z$(z%+1)
  NEXT z%
  , DEC x%
  ENDF
RETURN
'
PROCEDURE commentaire
IF z$(x%)>" "
  str!=0           !pas de string
  found!=0        !rien trouvé
  FOR z%=1 TO LEN(z$(x%))
    IF MID$(z$(x%),z%,1)=CHR$(34)
      str!=NOT str!   !changer le string-flag
    ENDF
    IF (MID$(z$(x%),z%,2)=" !") AND NOT str! AND NOT found!
      found!=-1
      position=z%
    ENDF
  NEXT z%
  IF found!
    z$(x%)=LEFT$(z$(x%),position-1)
  ENDF
ENDIF
RETURN

```

Vous avez certainement remarqué que notre programme ne peut traiter que des fichiers *.LST. Vous devez donc sauvegarder le programme à expurger par SAVE,A avant de lancer le Rem-Killer. Ce dernier commence alors à enregistrer les lignes de programme dans le tableau String-Array Z*(). Grâce aux

sous-programmes 'Remline' et 'Commentaire' il cherche ensuite tous les commentaires et les élimine. Il lui suffit alors de sauver le programme ainsi traité. Pour le charger, ôtez le REM-Killer de la mémoire vive et cliquez sur Merge.

Nous attirons encore une fois votre attention sur le fait que vous devez d'abord sauvegarder une version commentée de votre programme qui sera votre copie de sécurité. En effet, si vous deviez un jour y apporter quelques modifications, cela vous sera bien plus facile si vous avez vos commentaires dans le programme, même s'il s'agit d'un programme que vous avez écrit vous-même.

4.2. Générateur de lignes Data

Vous avez certainement déjà vu dans des revues d'informatique des programmes se composant essentiellement de colonnes de chiffres. Il est très astreignant de les recopier et on risque en plus de laisser passer quelques erreurs bien difficiles à retrouver ensuite. Comment engendre-t-on de telles quantités de chiffres?

Habituellement, il s'agit de programmes écrits en Assembleur. Ce langage est pénible à utiliser car il faut souvent plusieurs lignes d'écriture pour une seule opération relativement simple. Si bien que les programmes prennent vite des proportions démesurées, et sont bien trop longs pour qu'une revue accepte de les publier. D'où l'idée de publier ces programmes sous leur forme chiffrée, puisque finalement un programme une fois traité par un Assembleur ne consiste plus qu'en une suite de chiffres: le langage machine. Cette méthode représente un gain de place considérable; vous n'avez même pas besoin d'un Assembleur pour saisir le programme, car ces chiffres peuvent être traités à l'aide d'un interpréteur Basic. Vous pouvez utiliser ce procédé non seulement en Assembleur mais aussi en langage C, du moment que le programme est flanqué de son fichier ressource.

Ces séquences de chiffres sont inscrites dans le programme sur des lignes 'data'. Pour en faire un véritable programme, il faut les traiter à l'aide d'un autre programme qui s'appelle Basic-Loader; le programme qui transforme un programme opérationnel en Basic Loader s'appelle un générateur de lignes data. Nous allons vous présenter ce générateur afin que vous puissiez aussi tirer partie de vos programmes en assembleur (dans la mesure où vous en possédez!).

Nous avons choisi le Basic GFA comme langage de programmation même pour notre Basic-Loader. Pour simplifier, nous créons un fichier en ASCII (*.LST) dans notre mémoire de masse, fichier transformé ensuite par l'interpréteur en un véritable fichier-Basic grâce à Merge. La première mission de notre programme

consistera à générer un Header chargé d'ouvrir le fichier pour afficher le programme opérationnel et qui enregistre dans ce fichier les valeurs numériques reprises dans le Loader. Ce header est affiché dans les commandes Print#1.

Il faut ensuite générer les lignes Data qui se terminent par un astérisque (*). Nous chargeons ensuite tout le fichier source dans le tableau (array) X%. Pour retrouver facilement les fautes de frappe dans ce loader, nous calculons à chaque ligne une somme de test (check). Il faut pour cela numéroter les lignes, ce qui n'est pas habituel en Basic GFA, mais indispensable si on veut pouvoir recevoir des messages du genre 'la ligne xxx contient une erreur'. C'est pourquoi nous écrivons ces numéros de ligne derrière les données data. Pour éviter de confondre les deux chiffres qui se suivent, on multiplie le deuxième chiffre systématiquement par deux lors du calcul de contrôle. Ceci rend les fautes de frappe quasiment impossibles.

Certaines personnes ne jurent que par la saisie en hexadécimal pour les Loader du Basic, en affirmant qu'on s'épargne ainsi un tiers de la dactylographie (les nombres hexadécimaux ne comptent que deux positions contre trois pour les nombres décimaux). Nous réfutons cette argumentation pour deux raisons: premièrement, parce qu'un Loader en Basic contient beaucoup de nombres à deux ou même une seule position et deuxièmement parce qu'un programmeur moyennement doué en dactylographie saisira beaucoup plus vite des nombres décimaux pris sur le pavé numérique du clavier que des nombres décimaux pris sur le grand clavier. Pour gagner du temps, nous écrivons jusqu'à 16 nombres par ligne. Ceci engendre certes des lignes longues, mais réduit la quantité de sommes-tests à effectuer ainsi que l'écriture de numéros de lignes.

◆ Mode d'emploi du générateur

Vous devez d'abord placer le programme que vous voudriez avoir sous forme de Basic Loader dans le répertoire principal ou au moins dans le dossier contenant GFABASIC.PRG. Lancez alors le BASICGFA et chargez le générateur DATAGEN.BAS, puis précisez le nom du programme voulu et attendez un petit moment. Une fois que l'ordinateur a terminé, vous quittez le générateur et cliquez sur Merge. Chargez alors le fichier *.LST qui vient d'être créé dans la mémoire vive et archivez-le ensuite par Save comme un fichier Basic ordinaire: et voilà, c'est fait!

Un dernier conseil: le programme créé par le Loader peut avoir une longueur supérieure d'au moins 15 octets à celle du programme original, car l'enregistrement d'une ligne data prend toujours 16 octets complets. Mais ceci n'a aucun effet perceptible sur le fonctionnement du programme et ne vous fait perdre aucune place sur une disquette, car la place mémoire y est découpée en blocs de 512 octets (512 étant un multiple de 16). DATA.BAS :

```

'
' Générateur de lignes (BASIC-GFA)
'   MP 03-05-88
'
DIM x%(2000,15)
FILESELECT "\*.prg", "", filename$
a=INSTR(filename$, ".")-1
IF a=0
  a=LEN(filename$)
ENDIF
OPEN "R", #2, filename$, 1
FIELD #2, 1 AS bytes$
OPEN "O", #1, LEFT$(filename$, a) + ".LST"
'
' Ecrire loader-Header
PRINT "***** Création du Chargeur Basic *****"
'
PRINT #1, "' ***** "
PRINT #1, "' Chargeur Basic : "; UPPER$(LEFT$(filename$, a))
PRINT #1, "' "
PRINT #1,
  "? "+CHR$(34)+" ***** Création du programme ***** "+CHR$(34)
PRINT #1, "Open "+CHR$(34)+"R"+CHR$(34)+"", #1, "+CHR$(34)";
PRINT #1, UPPER$(filename$)+CHR$(34)+"", 16"
PRINT #1, "Field #1, 16 As Code$"
PRINT #1, "Do"
PRINT #1, "Ligne$="; CHR$(34)+CHR$(34)
PRINT #1, "Read Z$"
PRINT #1, "Exit If Z$="+CHR$(34)+"*"+CHR$(34)
PRINT #1, "Checksumme=0"
PRINT #1, "For I=0 To 15"
PRINT #1, "Read Byte"
PRINT #1, "Add Checksomme, (I Mod 2+1)*Byte"
PRINT #1, "Ligne$=Ligne$+Chr$(Byte)"
PRINT #1, "Next I"
PRINT #1, "Read Somme_juste"
PRINT #1, "If Somme_juste<>Checksumme"
PRINT #1, "print "+CHR$(34)+"Erreur dans la ligne"+CHR$(34)+"", Z$"
PRINT #1, "End"
PRINT #1, "Endif"
PRINT #1, "Lset Code$=Ligne$"
PRINT #1, "Put #1, Val(Z$)"
PRINT #1, "Loop"
PRINT #1, "Close #1"
PRINT #1, "Print "+CHR$(34)+"Terminé!" +CHR$(34)
PRINT #1, "End"
PRINT #1, "' "
'
' créer datas
'

```

```

FOR i=0 TO LOF(#2)-1
  GET #2,i+1
  x%(i DIV 16,i MOD 16)=ASC(byte$)
NEXT i
FOR i=LOF(#2) TO LOF(#2)+15
  x%(i DIV 16,i MOD 16)=0
NEXT i
,
FOR i=0 TO (LOF(#2)-1) DIV 16
  checksumme=0
  PRINT #1,"Data ";i+1," ";
  FOR j=0 TO 15
    PRINT #1,x%(i,j);";";
    ADD checksumme,(1+j MOD 2)*x%(i,j)
  NEXT j
  PRINT #1,checksumme
NEXT i
PRINT #1,"Data *"
CLOSE #1
CLOSE #2
END

```

4.3. Compiler des programmes GFA avec protection List

Nous avons pu nous mêmes constater qu'il n'était pas toujours possible de récupérer intégralement des programmes archivés par Psave, mais nous nous sommes rendus compte qu'une petite astuce permettait de reprendre ces programmes sous un compilateur. Il suffit de changer la valeur du premier octet qui est \$FF et de lui redonner la valeur normale \$0. Cependant, il faut savoir que certaines instructions ne peuvent être compilées.

Commencez par recopier le programme protégé sur une disquette puis lancez le programme ci-dessous:

```

,
, Programme: COMP.BAS
, Compiler les programmes protégés
,
Fileselect "\*.BAS","",Filename$
If Filename$="" Or Filename$="\ " ! ne pas indiquer de nom
  End
Else
  Open "I",#1,Filename$ !ouvrir le fichier pour
, le lire

```

```

U=Inp(#1)           !lire le premier signe
Close #1           !et refermer le fichier
If U=0             !valeur nulle déjà présente
  Alert 1," ce programme n'est pas | protégé! ",1," Dommage",Dummy&
End
Else               !sinon
  Open "U",#1,Filename$ !ouvrir le fichier à
  '                modifier
  Out #1,0         !Ecrire un zéro
  Close #1        !puis refermer le fichier
Endif
Endif

```

4.4. Programmation de touches

Nous sommes convaincus que le programme qui suit représente un véritable régal. Vous savez certainement que certains logiciels vous permettent de programmer des touches, c'est à dire que certains points des menus sont sélectionnables non seulement en cliquant l'option désirée dans le menu déroulant mais aussi en appuyant tout simplement sur une touche quelconque (la plupart du temps en combinant l'action de cette touche avec <CONTROL> ou <ALTERNATE>). Cela représente souvent une amélioration sensible des conditions de travail sur l'ordinateur, puisque vos mains ne quittent plus le clavier pour aller chercher la souris. Tous les logiciels n'offrent malheureusement pas cette possibilité.

Avec le programme ci-dessous, vous pouvez ajouter cette option dans tous les programmes GEM possédant un fichier ressource (*.RSC). Il vous permet aussi d'attribuer des formules de texte à certaines touches, ce qui est bien utile tant dans un traitement de texte que dans un éditeur de programme. Pour parfaire tout cela, vous avez même la possibilité d'appeler la date à l'aide d'une simple touche. Naturellement, vous n'avez aucunement besoin de savoir programmer pour pouvoir utiliser ce programme!

Comme il s'agit d'un programme assez complexe et difficile à comprendre, nous allons diviser en deux les explications fournies: d'une part, un mode d'emploi et d'autre part, les explications sur le fonctionnement du programme. Les lecteurs qui se limitent à utiliser les programmes donnés vont y rencontrer des choses qu'ils ne comprendront guère. C'est inévitable car le fonctionnement de ce programme est assez complexe et nous ne pouvons dans le cadre de cet ouvrage nous livrer à une initiation complète en programmation. Mais si vous vous en tenez très exactement au mode d'emploi fourni, tout devrait bien se passer. Commençons donc par le mode d'emploi.

Pour pouvoir tourner, ce programme a besoin d'un fichier de paramètres nommé TOUCHES.PAR sur la disquette. Ce fichier contient toutes les informations précisant les actions déclenchées par telle ou telle touche. C'est un fichier qui possède le format suivant:

```

;les commentaires sont
;introduits par point-virgule
;
SS AA FF 00 X1 X2
SS AA FF 00 X1 X2
.
.
.
SS AA FF 00 X1 X2
00

```

Tous les nombres doivent être saisis en hexadécimal et les nombres de A à F en majuscules.

Voici la signification des variables :

SS et AA	Codes Scan et ASCII des touches à programmer
FF	Numéros des fonctions (0=simulation d'un menu et 1=simulation d'une touche)
00	Dummy (octet sans signification)
X1 et X2 pour FF=0	X1=nom du menu et X2 texte des options (voir ci-dessous)
X1 et X2 pour FF=1	Code Scan (X1) et ASCII (X2) des touches concernées

En fait, c'est encore plus compliqué qu'il n'y paraît. Si vous n'avez pas compris le formatage du fichier, vous ne faites pas pour autant parti des utilisateurs sous-doués. Comme tout ceci est un peu ardu, nous joignons un petit programme en Basic GFA qui se charge de confectionner ce fichier. KEYBOARD.LST :

```

'
' Programme pour créer le fichier 'TOUCHES.PAR'
' MP 27-12-87
'
IF NOT EXIST("TOUCHES.PAR")
OPEN "O", #1, "TOUCHES.PAR" ! Créer le fichier
ELSE
OPEN "A", #1, "TOUCHES.PAR" ! Editer le fichier
ENDIF
'
DO
PRINT

```

```

PRINT "Appuyez sur la touche à programmer (Fin=Q)..."
LET touche=GEMDOS(1)
EXIT IF touche=270401617      ! ** Attention...cette valeur
                               varie suivant
                               les ST, STF ou Méga ST **
ADD touche,&H1000000
LET touche$=MID$(HEX$(touche),2,2)+" "+RIGHT$(HEX$(touche),2)
ALERT 2,"Formule ou menu?",1,"Formule|Menu",ret
SUB ret,1
IF ret=0
  PRINT "Entrez les touches dans l'ordre...terminez par UNDO"
  DO
    key=GEMDOS(1)
    EXIT IF key=6356992
    ADD key,&H1000000
    key$=MID$(HEX$(key),2,2)+" "+RIGHT$(HEX$(key),2)
    PRINT #1;touche$;" 01 00 ";key$
  LOOP
ELSE
  INPUT "Titre, Index : ",titre,index
  titre$=RIGHT$(HEX$(titre+256),2)
  index$=RIGHT$(HEX$(index+256),2)
  PRINT #1;touche$;" 00 00 ";titre$;" ";index$
ENDIF
LOOP
PRINT #1,"00"
CLOSE #1

```

Après avoir lancé le programme, vous pouvez entrer la première combinaison de touches avec sa signification. Vous pouvez utiliser toutes les touches de fonction (en position ordinaire et en position shift), toutes les lettres avec <CONTROL> (seule exception: <CONTROL> + D) ainsi que les touches fléchées du curseur et <ESCAPE>. Les autres combinaisons seront enregistrées mais resteront sans effet, en particulier on ne peut utiliser la touche <ALTERNATE>. Vous pouvez ensuite déterminer si vous voulez la programmer comme une option d'un menu ou comme une formule. Dans ce dernier cas, le programme vous demande d'indiquer les touches concernées, dans le premier cas vous devez entrer deux nombres identifiant le menu (voir ci-dessous). Vous pouvez même utiliser la même touche pour ces deux choses en entrant la même touche deux fois de suite.

Le seul problème restant à éclaircir concerne justement les deux nombres identifiant le menu. Le premier nombre désigne l'index du menu et le deuxième l'index de l'option. Nous ne voudrions pas ici entrer dans le détail des arborescences des menus, si bien que nous allons vous donner un petit programme qui se chargera de vous fournir les deux nombres adéquats. Avant de le lancer, vous devez renommer le fichier-ressource (*.RSC) du programme pour lequel vous voudriez programmer le contenu des menus et lui donner le nom TEST.RSC.

Venons-en maintenant au véritable programme: copiez `clavier.prg` et `touches.par` dans le dossier AUTO de votre disquette boot. Voilà, c'est tout. Vous pouvez appeler les commandes affectées aux touches à tout moment, sauf dans les boîtes de dialogue. En appuyant <CONTROL>+D vous verrez apparaître la date actuelle telle que vous l'avez saisie dans le tableau de contrôle.

Nous conseillons aux non-programmeurs d'arrêter ici la lecture de ce chapitre, car cela va devenir ardu. Ce programme réclamerait tant d'explications qu'elles ne pourraient tenir dans le cadre de ce volume. C'est pourquoi nous allons nous limiter aux points les plus intéressants.

Le principe de base est le suivant: lorsqu'une application GEM attend une réponse de l'utilisateur, elle se sert généralement de la fonction `evnt_multi`. Il convient donc de capter cette fonction. Nous la laissons s'exécuter et nous contrôlons alors si l'utilisateur a actionné une touche programmée. Si tel n'est pas le cas, notre programme se contente de transmettre la valeur de la touche appuyée au programme principal. Si tel est par contre le cas, nous remplaçons la touche appuyée par une ou plusieurs options qui lui ont été affectées. C'est en principe tout ce qu'il suffit de comprendre.

Si vous avez déjà parcouru le chapitre consacré aux programmes résidents, vous vous attendez à trouver TRAP #2 dans notre initialisation, la fonction qui se charge normalement de tous les GEM-Calls (appels dans le GEM). Nous n'avons pas utilisé ce principe: nous intervenons au niveau de VBL-Queue. Le vecteur TRAP #2 n'est modifié qu'à ce niveau, et ceci pour deux raisons: premièrement, nous voulons que notre programme puisse se lancer automatiquement. Il se trouve que lorsque les programmes AUTO sont lancés, le GEM n'est pas encore installé, si bien que nous ne pouvons pas encore nous en servir.

Deuxièmement, le système d'exploitation achemine ce vecteur GEM pour des raisons assez mystérieuses vers les routines originales (donc vers la ROM) lorsque l'écran passe de la représentation texte à la représentation graphique, donc par exemple lors de l'affichage d'un fichier à l'écran. C'est pourquoi nous devons contrôler dans le VBL-Interrupt si ce vecteur a déjà été initialisé (si tel n'est pas le cas, le premier octet porte le numéro 34) et s'il est bien pointé vers notre propre routine ou non. Voilà pour ce qui concerne l'initialisation.

Notre routine résidente intervient donc maintenant à chaque appel du GEM. Les registres D0 et D1 contiennent toutes les informations concernant les types d'appels possibles. Nous commençons par contrôler si l'application avait ou non appelé la fonction `evnt_multi`. Si tel n'est pas le cas, nous laissons intervenir le GEM. Si c'est le cas, c'est là que la question devient brûlante. Nous devons appeler

evnt_multi puis revenir dans notre routine, car c'est le seul moyen de savoir quelle touche a été appuyée. C'est pourquoi nous écrivons une adresse de retour représentant le début de la routine à parcourir ainsi que le registre des états dans le stack, pour ensuite appeler la routine originale; ceci revient à simuler un TRAP.

Tout accessoire ainsi que le programme principal représentent un programme en-soi, et le système passe de l'un à l'autre à chaque appel de la bibliothèque AES. L'AES prend bonne note des adresses de retour et des registres des programmes avant de passer de l'un à l'autre, afin de pouvoir ensuite retrouver son chemin. Cela n'a rien d'extraordinaire, mais fait que tous les appels evnt_multi (qu'ils proviennent d'un accessoire ou du programme principal) passent par notre routine résidente, et que l'AES va toujours recevoir la même adresse de retour: celle qui pointe vers notre routine! Imaginez un instant le chaos résultant de ce que l'ordinateur tente obstinément de réactiver ces programmes tout en ne recevant qu'une seule et même adresse!

Nous devons donc veiller à ce que l'AES ne revienne dans notre routine que si l'appel provient effectivement de l'application en cours. Comme le GEM attribue à tous les programmes en cours un numéro d'identification (ID), nous pouvons trier les appels. Comme les accessoires ne nécessitent en principe jamais le recours à une barre de menus particulière, nous guettons un appel menu_bar et devons noter immédiatement le numéro ID du programme demandeur. Par la suite, nous n'envoyons une adresse de retour sur le stack que si l'appel de evnt_multi provenait bien du programme principal en cours. C'est là le problème le plus complexe que nous ayons eu à résoudre lors de l'élaboration de ce programme. Si vous en comprenez toutes les subtilités, vous pouvez sans complexe vous vanter de faire partie de l'élite des programmeurs.

Le restant est très simple. Si vous avez des doutes, procurez-vous un manuel consacré au GEM. Si vous en avez envie, vous pouvez rajouter des options dans ce programme, comme par exemple l'affichage du jour de la semaine lié à la date. Les as de la programmation pourront tenter d'automatiser les réponses aux boîtes de dialogue ou aux choix offerts par les menus. Si vous parvenez à comprendre tout le déroulement de ce programme, ceci ne devrait offrir aucune difficulté pour vous!

```

;
; Touches préprogrammées CLAVIER.S
; MP 22-12-87
;
gemdos = 1
xbios = 14
conin = 7

```

```

get_date = $2a
keep     = $31
open     = $3d
close    = $3e
print    = 9
read     = $3f
trap2_adr      = $88
_vblqueue     = $456
nvbls        = $454
aes          = $c8
evnt_multi   = 25
menu_bar     = 30
supexec      = 38

        .TEXT
        movea.l 4(sp),a0          ;Calculer la place-mémoire
;                                     nécessaire
        move.l  # $100,d6
        add.l  12(a0),d6
        add.l  20(a0),d6
        add.l  28(a0),d6          ;Ne nous servira que plus
;                                     tard
        bsr   readfile          ;Charger le fichier des
;                                     paramètres
        pea   init_vbl          ;Déclarer la routine VBL
        move.w #supexec,-(sp)   ;en mode superviseur
        trap #xbios
        addq.l #6,sp
        tst.w  init_err          ;Erreur d'initialisation ?
        bne   error
        clr.w  -(sp)            ;Pas d'erreur donc
        move.l d6,-(sp)        ;maintenir le programme en
;                                     résident
        move.w #keep,-(sp)
        trap #gemdos

init_vbl: move.w  nvbls,d0          ;Installer la routine VBL
        lsl.w  #2,d0             ;* 4
        movea.l _vblqueue,a0
        moveq.l #4,d1
;                                     ;Nous commençons par la
s_loop:  tst.l   0(a0,d1.w)        ;Libre ?
        beq.s  found
        addq.w #4,d1             ;Si non, saisie suivante
        cmp.w  d0,d1            ;Toutes déjà traitées ?
        bne.s  s_loop
        move.w #1,init_err       ;si oui, consigner les
;                                     erreurs
        rts
found:   move.l  #vbl,0(a0,d1.w)  ;modifier position du

```

```

;                                     pointeur
vbl:      rts
          cmpi.b   #34,trap2_adr      ;TRAP #2 (vecteur n34)
;
          beq.s    finish_vbl         ;si pas encore, attendre
          cmpi.l   #enter_gem,trap2_ad ;pointe déjà sur
;                                     une autre routine ?
          beq.s    finish_vbl         ;si oui, ne rien
;                                     faire
          move.l   trap2_adr,old_vector ;si non, consigner
          move.l   #enter_gem,trap2_ad ;et le diriger vers
;                                     notre programme
finish_vbl:
          rts
enter_gem:
          cmpi.w   #aes,d0            ;Nouvelle entrée dans le
;                                     GEM
          bne.s    normal             ;Pas d'AES? alors exécution
;                                     normale
          movem.l  d2/a0-a3,save      ;Sauver le registre
          movea.l  d1,a0              ;Si non, rechercher le
;                                     pointeur sur
;                                     le bloc de paramètres
          movea.l  (a0),a1            ;A1: pointeur sur
;                                     Contrl-Array
          cmpi.w   #evnt_multi,(a1)   ;Moment d'intervenir?
          bne.s    test_2             ;Non, alors se retirer...
          movea.l  4(a0),a1           ;Global-array pour Ap_Id
          move.w   4(a1),d2
          cmp.w    apid,d2            ;Pas d'accessoire ?
          bne.s    norm_getreg        ;Si oui, pas d'adresse de
;                                     retour
          tst.w    action              ;Un évènement s'est produit ?
;
          bne      message             ;Si oui, pas d'appel du GEM
          move.l   a0,saveadr         ;Le PB est encore
;                                     nécessaire
          pea      testkbd             ;Adresse de retour
          move.w   sr,-(sp)
norm_getreg:
          movem.l  save,d2/a0-a3      ;Revenir au registre
normal:   movea.l  old_vector,a0      ;Sinon, passer aux
;                                     routines normales du GEM
          jmp      (a0)
test_2:   cmpi.w   #menu_bar,(a1)     ;Application lancée?
          bne.s    norm_getreg        ;Si non, ne rien faire
          movea.l  4(a0),a1           ;A1: Global-Array
          move.w   4(a1),apid         ;Identification de
;                                     l'application

```

```

testkbd:  bra.s      norm_getreg
          movea.l   saveadr,a0
          movea.l   12(a0),a1      ;INTOUT-ARRAY vers A1
          move.w    (a1),d0        ;Evènement intervenu
          andi.w    #1,d0          ;Keyboard ?
          beq      fin            ;si non, cela ne nous
;                                     intéresse pas
          move.w    10(a1),d0      ;Codes Scan et ASCII
          cmpi.w    #$2004,d0     ;<CONTROL>+D pour la date ?
          bne.s    no_date
          bsr      date           ;si oui, rechercher la date
no_date:  lea.l    commandes,a3   ;Rechercher le code de
;                                     la touche dans le tableau
search:   move.w    d0,last       ;et le sauvegarder
          tst.w    (a3)
          beq.s    fin            ;Si oui, redonner le code
;                                     'original'
;                                     de la touche
          cmp.w    (a3),d0        ;Trouvé dans le tableau ?
          beq.s    trouve        ;Si oui, sortir
          addq.l   #6,a3          ;Si non, augmenter le
;                                     pointeur et
;                                     réessayer
trouve:   bra.s    search
          move.l   a3,command
          move.w   #-1,action     ;Poser un flag pour le
;                                     traitement
message:  movea.l  12(a0),a1      ;Pointeur sur INTOUT-Array
          movea.l  12(a0),a1      ;Pointeur sur INTOUT-Array
          tst.b   2(a3)          ;Numéro de fonction
          beq.s   menu           ;Zéro, enregistrer l'option
;                                     du menu
formule:  movea.l  12(a0),a2      ;Intout-Array
          move.w   #1,(a2)       ;1 signifie: l'utilisateur
;                                     a appuyé
;                                     sur une touche
;                                     désignée ici
          move.w   4(a3),10(a2)  ;Si non, consigner la touche
menu:     bra.s    next
          movea.l  16(a0),a2      ;Addrin-Array
          movea.l  (a2),a2       ;Message-buffer de
;                                     l'application
          move.w   #16,(a1)      ;Nouvelle annoncée
;                                     'officiellement'
          move.w   #10,(a2)      ;10 signifie: on a cliqué
;                                     sur le titre du menu
          move.b   4(a3),7(a2)   ;Menu title index
          move.b   5(a3),9(a2)   ;Menu entry index
next:     addq.l   #6,a3         ;Saisie suivante
          move.w   last,d0       ;Code touche précédent
          move.l   a3,command    ;Consigner la commande
;                                     suivante

```

```

        cmp.w    (a3),d0          ;encore la même touche ?
        beq.s   fin              ;alors tout recommencer,
;                                     svp
        clr.w    action          ;sinon effacer le flag
fin:    movem.l  d2/a0-a3,save   ;revenir au registre
        rte                    ;retour du contrôle sur PRG
;charger le fichier
;"TOUCHES.PAR"

readfile: move.w    #-1,valid    ;Vrai, signe suivant
;                                     valable
        move.w   #2,-(sp)       ;Ouvrir le fichier pour le
;                                     lire
        move.l   #filename,-(sp)
        move.w   #open,-(sp)
        trap     #gemdos
        addq.l   #8,sp
        tst.w    d0              ;Une erreur ?
        bmi     error2          ;si oui, affichage d'un
;                                     message
        move.w   d0,handle
        lea.l    com_loaded,a3
        bsr     readbyte        ;Octet après D0
        move.b   d0,(a3)+      ;L'archiver comme une
;                                     commande
bigloop:  tst.b    d0            ;Fin du fichier ?
        beq.s   finish          ;Si oui, interrompre la
;                                     lecture
        moveq.l  #5,d1          ;Si non, lire les 6 (ou 5)
;octets suivants

inner:    bsr     readbyte
        move.b   d0,(a3)+
        addq.l   #1,d6
        dbra    d1,inner
        bra.s   bigloop
finish:   move.w   handle,-(sp)  ;Commandes suivantes
        move.w   #close,-(sp)  ;Fermer le fichier
        trap     #gemdos
        addq.l   #4,sp
        rts
readbyte: bsr     readchar      ;Prendre un signe du
;                                     fichier
        cmpi.b   #'0',d0        ;HEX-Digit correct ?
        blt.s   nohex
        cmpi.b   #'9',d0
        ble.s   hex
        cmpi.b   #'A',d0
        blt.s   nohex
        cmpi.b   #'F',d0
        bgt.s   nohex

```

```

hex:      tst.w      valid      ;Reprendre le signe ?
          beq.s      readbyte   ;Non, donc l'ignorer
          bsr.s      to_bin     ;Le transformer en nbr
;                                     binaire
          move.b     d0,d2      ;Consigner High-Nibble
          asl.b      #4,d2      ;et * 16
          bsr        readchar   ;Rechercher le low-digit
          bsr.s      to_bin
          add.b      d0,d2      ;l'additionner ici
          move.w     d2,d0
          rts
nohex:    cmpi.b     #';',d0     ;Début du commentaire ?
          beq.s      com
          cmpi.b     #13,d0     ;Fin de la ligne ?
          bne.s      readbyte   ;Non
          move.w     #-1,valid   ;Sinon Valid = Vrai
          bra.s      readbyte
com:      clr.w      valid      ;Commentaire, donc Valid =
;                                     Faux
          bra.s      readbyte
to_bin:   subi.b     #'0',d0     ;Transcoder ASCII --> BIN
          cmpi.b     #9,d0
          ble.s      label
          subq.b     #7,d0
label:    rts
readchar: pea      buffer      ;Lire un signe
          move.l     #1,-(sp)
          move.w     handle,-(sp)
          move.w     #read,-(sp)
          trap       #gemdos
          adda.l     #12,sp
          move.b     buffer,d0
          rts
error:    pea      err         ;Interrompre car annonce
;                                     d'erreur
_err:     move.w     #print,-(sp)
          trap       #gemdos
          addq.l     #6,sp
          move.w     #conin,-(sp)   ;Attendre action d'une
;                                     touche
          trap       #gemdos
          addq.l     #2,sp
          clr.w      -(sp)        ;Interrompre, ne pas
;                                     conserver en
;                                     résident
error2:   trap      #gemdos
          pea      err2         ;cf ci-dessus, mais pour
          ;                                     le disque
          bra.s     _err
date:     movem.l   d0/d1/a0,-(sp) ;Date à insérer dans

```

```

;
                                les
move.w    #get_date,-(sp)      ;commandes
trap      #gemdos
addq.l    #2,sp
clr.l     d1                    ;Effacer tous les bits
move.w    d0,d1
andi.w    #%11111,d1           ;Masquer le jour
divu.w    #10,d1                ;Former 2 chiffres
addi.w    #'0',d1
move.b    d1,commandes+5
swap.w    d1
addi.w    #'0',d1
move.b    d1,commandes+11
clr.l     d1
lsr.w     #5,d0                 ;'Démasquer' le bit du jour
move.w    d0,d1                 ;Traiter le mois
andi.w    #%1111,d1
divu.w    #10,d1                ;Former 2 chiffres
addi.w    #'0',d1

move.b    d1,commandes+23
swap.w    d1
addi.w    #'0',d1
move.b    d1,commandes+29
ext.l     d0
lsr.w     #4,d0                 ;ret additionner finalement
addi.w    #80,d0                 ;l'année 1980
cmpi.w    #100,d0
blt.s     date_ok               ;Pour après l'an 2000....
subi.w    #100,d0
date_ok:  divu.w    #10,d0        ;Former 2 chiffres
addi.w    #'0',d0
move.b    d0,commandes+41
swap.w    d0
addi.w    #'0',d0
move.b    d0,commandes+47
movem.l   (sp)+,d0/d1/a0
rts

.DATA
init_err: .DC.w 0
action:   .DC.w 0 ;Pas de commande au début
apid:     .DC.w $4321 ;Ap_Id, n'importe qu'elle
;         valeur
filename: .DC.b 'TOUCHES.PAR',0
err:      .DC.b 'plus de slot VBL libre pour '
          .DC.b 'Keyboard-utility -> TOUCHE',0
err2:     .DC.b 'je ne peux pas lire TOUCHES.PAR -> TOUCHE',0

commandes:
          .DC.b 32,4,1,0,11,0 ;Commande fixe: date

```

```

        .DC.b 32,4,1,0,11,1           ;Jour
        .DC.b 32,4,1,0,$34,'.'       ; Le
;                                         sous-programme,date
        .DC.b 32,4,1,0,11,0         ;Mois   place ici
;                                         éventuellmt
        .DC.b 32,4,1,0,11,0         ;Mois   la date
;                                         actuelle
        .DC.b 32,4,1,0,$34,'.'       ;
        .DC.b 32,4,1,0,11,8         ;Année
        .DC.b 32,4,1,0,11,8         ;Année
        .EVEN

        .BSS

com_loaded:
        .DS.b 2000                   ;2000 octets ou plus...
;                                     A partir d'ici
;                                     commencent les
;                                     commandes suivantes

last:    .DS.w 1   ;Dernière touche actionnée
handle:  .DS.w 1   ;GEMDOS File-Handle
valid:   .DS.w 1   ;Booléen: le signe du fichier
;                                     est -il valable, correct ?
buffer:  .DS.w 1   ;Mémoire tampon pour les signes
;                                     du
;                                     fichier
saveadr: .DS.l 1   ;Pointeur sur les blocs de
;                                     paramètres AES
old_vector:
        .DS.l 1   ;Pointeur vers le vrai GEM
command: .DS.l 1   ;Commande suivante à
;exécuter
save:    .DS.l 5   ;Mémoire tampon pour le registre

        .END

```

Chapitre 5

Trucs et astuces : généralités

L'objet de ce chapitre est de vous donner de petits utilitaires que vous pourrez insérer dans vos propres programmes ainsi que divers trucs qui vous faciliteront le travail quotidien sur votre ordinateur.

5.1. Faire une copie d'écran

Comme personne ne l'ignore, on déclenche une recopie de l'écran en appuyant simultanément sur les touches <ALTERNATE>+<HELP>. Mais comment faire la même chose au milieu d'un programme quelconque, par exemple lorsqu'il s'agit d'imprimer un graphique ?

Il y a pour ce faire plusieurs solutions. La plus courte consiste à utiliser une commande Poke:

```
Sdpoke 1262,0
```

qui s'écrit en assembleur:

```
clr.w $4ee
```

Ces deux commandes ont exactement le même effet. Notre ordinateur possède une cellule-mémoire nommée `_dumpflag`. Cette cellule-mémoire a normalement la valeur `-1(=$ffff)` et est automatiquement augmentée de un lorsque l'utilisateur appuie sur les touches <ALTERNATE>+<HELP>. Une des fonctions du système d'exploitation se charge de contrôler à intervalle régulier l'état de cette

cellule-mémoire. Lorsque l'état est égal à zéro, le programme courant est interrompu, et le contenu de l'écran est envoyé vers l'imprimante. Nous pouvons naturellement simuler cette remise à zéro grâce à la commande indiquée ci-dessus. Cette commande poke est tout à fait légale, car `_dumpflag` fait partie des adresses-système indiquées par la firme Atari.

La deuxième possibilité réside dans une routine XBIOS. Elle porte le nom `Scrdmp` (abréviation de Screen-Dump) et le numéro 20. On l'appelle de la façon suivante:

```
move.w #20, (-sp)
trap #14
addq.l #2, sp
```

Ce qui donne en langage C, avec l'inclusion du fichier "OSBIND.H"

```
Scrdmp();
```

Pourquoi faire simple quand on peut faire compliqué? il existe encore une autre possibilité de provoquer une copie d'écran, par le biais d'une fonction VDI qui ne fait rien de plus que d'appeler la fonction XBIOS ci-dessus `Scrdmp`. Sans doute n'a-t-elle été implémentée que pour rendre le VDI compatible avec la version IBM. En langage C, on peut écrire:

```
v_hardcopy (handle);
```

5.2. L'affichage des caractères ASCII

ayant un code inférieur à 32

L'Atari ST dispose d'un jeu de caractères de 256 signes. Les programmes sous TOS ne peuvent utiliser les codes ASCII inférieurs à 32 car ils servent de codes de commandes. Par exemple, le code 10 provoque un saut de ligne, 7 déclenche la petite clochette etc. Mais comment faire pour afficher à l'écran les signes attribués à ces codes?

C'est le rôle attribué à la fonction BIOS nommée `Bconout`: cette fonction permet d'envoyer un signe particulier vers un périphérique de sortie (imprimante, interface sériel, écran, interface-MIDI, processeur du clavier etc). Les appareils

indiqués entre parenthèses sont numérotés de 0 à 4. Mais le simple transfert du signe vers l'appareil numéro 2 (écran) suffit pour entraîner l'interprétation du signe en question qui devient une commande. Il existe cependant un appareil numéro 5, qu'on appelle aussi le canal ASCII. Ce numéro 5 interpelle lui aussi l'écran, mais sans interpréter les signes de commande, ce qui nous permet d'afficher les 32 premiers signes:

```

move.w #Signe,-(sp) ; signe concerné
move.w #5,-(sp) ; numéro d'appareil ou de canal
move.w #3,-(sp) ; numéro de la fonction Bconout
trap #13 ; BIOS
addq.l #6,sp ; Correction stack

```

En langage C, on écrira:

```
Bconout (5,Signe);
```

Cette fonction existe aussi en Basic:

```
Bios (3,5,Signe)
```

5.3. Déterminer la position du pointeur de la souris

Connaissez-vous le logiciel graphique STAD ? Il se charge, après que vous ayez cliqué sur une option d'un menu, de placer le pointeur de la souris sur la surface dessinée. Ceci est facile à réaliser en principe car les coordonnées de la flèche de la souris se trouvent constamment à deux endroits de la mémoire vive. Pour l'ancien TOS en ROM il s'agit de \$26e0 (pour les coordonnées x) et de \$26e2 (pour y); dans le nouveau Blitter-TOS, ces valeurs se trouvent à l'adresse \$2740 (pour x) et \$2742 (pour y). Pour que notre programme puisse fonctionner avec l'une ou l'autre des versions du TOS, il faut qu'il sache laquelle se trouve dans l'ordinateur :

```

'Placer la flèche de la souris dans le coin supérieur gauche de
l'écran
'
If Dpeek(&hfc001e)=&hc46 ! Date = 6.2.88 ?
  Dpoke &h26e0,0 ! si oui, il s'agit de l'ancien TOS
  Dpoke &h26e2,0
Else
  Dpoke &h2740,0 ! si non, il s'agit du Blitter-TOS

```

```

Dpoke &h2472,0
Endif
,
Hidem          ! effacer la flèche
Showm         ! la faire réapparaître

```

Dans l'ancien TOS, l'adresse \$fc001e contient la référence codée de la date de fabrication (06.02.1986 = \$0c46). Cette adresse permet de contrôler facilement le type de système d'exploitation tournant sur l'ordinateur. Précisons encore que ces deux commandes Dpoke permettent certes de modifier les coordonnées de la flèche de la souris de façon interne, mais que l'image de la flèche n'est pas encore déplacée pour autant sur l'écran. Elle ne prendra sa nouvelle position que lorsqu'elle sera de nouveau déplacée. Nous contournons cette difficulté en désactivant et réactivant consécutivement et très rapidement le curseur.

5.4. Activer et désactiver le pointeur de la souris

Il est parfois nécessaire de supprimer temporairement l'affichage de la flèche de la souris, par exemple lorsqu'on se sert d'un logiciel d'édition de textes ou lorsqu'on veut vider totalement l'écran. Voici les commandes utilisées en Basic-GFA:

```

Hidem      !désactive le pointeur de la souris
Showm     !réactive le pointeur de la souris

```

En langage C, on se sert de deux fonctions VDI:

```

v_hide_c (handle);          /*désactive le pointeur de la souris*/
v_show_c (handle, reset);  /*réactive le pointeur de la souris*/

```

reset est un flag prenant la valeur 0 ou 1. Lorsque reset = 0, la flèche de la souris s'affiche dans tous les cas. Si par contre reset = 1, le pointeur ne redevient visible que si le nombre des appels de v_show_c est exactement équivalent à celui des appels de v_hide_c, qui a rendu la flèche invisible. Il est donc intéressant de disposer d'un sous-programme permettant de désactiver le pointeur de la souris et de le réafficher à la fin: ainsi le pointeur ne réapparaîtra automatiquement que s'il était déjà présent au moment où l'on est passé dans le sous-programme. Enfin, si vous voulez programmer en Assembleur, vous disposez de deux routines simples à l'intérieur des routines graphiques Line-A. On les appelle de la façon suivante:

dc.w	\$a00a	;désactive le pointeur de la souris
dc.w	\$a009	;réactive le pointeur de la souris

Ces commandes permettent facilement de désactiver la souris dans les programmes TOS dont les noms de fichiers se terminent par .PRG. Il ne s'agit pas réellement de commandes-machines mais d'un type particulier d'appel-système (LINE A). Nous ne pouvons vous donner de plus amples informations dans les limites de ce livre; si cela vous intéresse, veuillez vous reporter à des ouvrages plus spécialisés.

5.5. Une souris à la demande

Vous connaissez certainement les logiciels WordPlus ou FirstWord, dans lesquels une simple touche permet de désactiver la flèche de la souris. Cette flèche réapparaît automatiquement lorsqu'on touche à la souris. Il est nécessaire de supprimer l'affichage de la flèche car le caractère du clavier pourrait s'inscrire au-dessus du dessin de la flèche. Lorsque l'utilisateur bougera ensuite la souris, le caractère sera partiellement ou totalement effacé, puisque le déplacement de la flèche provoque le réaffichage de la première couche de l'arrière-fonds. Comme le caractère aura été écrit sur le dessin de la flèche, il ne figure pas sur l'arrière-fonds. Les autres logiciels procèdent de la même façon mais la flèche de la souris n'y redevient visible qu'après l'inscription du caractère.

Mais comment s'y prend WordPlus pour réafficher la flèche lorsqu'on bouge la souris? Il faut pour comprendre cela posséder quelques connaissances sur la commande `evnt_multi` de l'AES. Cette commande surveille l'irruption d'un ou plusieurs événements précisés à l'avance, comme par exemple l'action d'appuyer sur une touche ou le fait de cliquer sur une option d'un menu. Il peut s'agir aussi de l'apparition ou de la disparition de la flèche de la souris dans un carré défini. Beaucoup de nos lecteurs connaissent déjà cette possibilité sans pourtant avoir une idée exacte de son utilité. En effet, lorsqu'il suffit d'appuyer sur une touche pour faire disparaître la flèche de la souris, ceci revient à lui donner une position correspondant à un minuscule carré (hauteur et largeur = un pixel). Il suffit donc de la déplacer pour la voir réapparaître. En attendant, tous les autres événements, comme par exemple la frappe au clavier, sont traités sans que l'on se soucie de la flèche de la souris.

5.6. Régler la vitesse d'exécution de la souris

A peine se sont-ils habitués à ce merveilleux petit animal qu'est la souris, que les utilisateurs se mettent déjà à ronchonner. L'un voudrait pouvoir positionner plus exactement cette flèche, l'autre voudrait au contraire pouvoir balayer tout l'écran par un simple mouvement du poignet. Il est possible de satisfaire tout le monde.

Vous n'ignorez pas que la souris est gérée par le processeur du clavier. Lorsqu'un changement d'état se produit pour l'une des touches du clavier ou pour la souris, le processeur principal en est averti. Dans un premier temps, le registre A0 pointe sur un groupe de données qui décrit en détail l'évènement en question et, dans un deuxième temps, un vecteur permet de sauter jusqu'à la routine capable de traiter ce groupe de données.

L'un de ces vecteurs, celui qui accompagne la fonction XBIOS kbdvbase (numéro 34), se charge exclusivement de gérer les données concernant la souris. Nous avons besoin de ce vecteur pour manipuler quelque peu les données. Comment faire pour accélérer d'un facteur deux la vitesse d'exécution de la souris? C'est très simple: lorsque l'ordinateur reçoit un groupe d'instructions concernant la souris, il passe par le vecteur ci-dessus pour aller à la routine concernée. Pour multiplier par deux l'ampleur du déplacement, il suffit de transmettre deux fois ce groupe d'instructions à la routine chargée des déplacements de la souris. C'est le rôle des deux commandes JSR(a1) de notre programme. Il faut bien sûr sauvegarder le registre A1 dont nous avons besoin dans notre routine (ceci vaut d'ailleurs pour tous les autres registres). Voilà le petit truc tout simple qui permet de doubler l'ampleur des mouvements de la souris!

```

;
; Accélérateur de la souris      SPEED.S
;      MP      03-05-88
;
gemdos      = 1
xbios       = 14
keep        = $31
kbdvbase    = 34
      .TEXT
      movea.l 4(sp),a0                ;place-mémoire
      move.l  #$100,d6
      add.l  12(a0),d6
      add.l  20(a0),d6
      add.l  28(a0),d6
      move.w  #kbdvbase,-(sp)        ;il y a ici plusieurs
;                                     adresses possibles

```

```

trap      #xbios
addq.l   #2,sp
movea.l  d0,a0
move.l   16(a0),old      ;routine de la souris
move.l   #new,16(a0)
clr.w    -(sp)           ;programme résident
move.l   d6,-(sp)
move.w   #keep,-(sp)
trap     #gemdos
new:     move.l   a1,-(sp)      ;sauvegarde du registre
        movea.l  old,a1       ;routine originale...
        jsr     (a1)          ;appelée deux fois...
        jsr     (a1)          ;consécutivement
        movea.l  (sp)+,a1     ;retour au registre
        rts                ;puis quitter
        .BSS
        old:    .DS.l 1
        .END

```

On peut aussi faire l'inverse: diminuer l'ampleur des mouvements de la souris. Naturellement, l'ordinateur ne peut en aucun cas n'exécuter qu'à moitié un groupe d'instructions. Il peut par contre n'exécuter qu'un mouvement sur deux. Ceci engendre un petit problème: lorsque vous appuyez sur une des touches de la souris, l'ordinateur utilise le même vecteur que pour les déplacements. On doit cependant continuer à tenir compte de toutes les actions des touches de la souris, c'est pourquoi il faut d'abord contrôler s'il y a eu un simple clic ou un déplacement. Les mouvements relatifs qui ont lieu depuis le dernier traitement se trouvent dans les 2ème et 3ème octets du groupe d'instructions. Si rien n'y est modifié, les instructions sont transmises telles quelles dans tous les cas. Nous transformons l'état du bit numéro 0 en un flag: s'il a toujours la valeur zéro après sa transformation, c'est qu'il n'y a pas de mouvement, sinon c'est qu'il y a bien déplacement de la souris. C'est ainsi que nous procédons pour que le groupe d'instructions ne soit traité qu'une fois sur deux. Tout le reste du programme est semblable au programme d'accélération donné ci-dessus.

```

;
;Ralentisseur de la souris   SLOW.S
;   MP 03-05-88
;
gemdos   = 1
xbios    = 14
keep     = $31
kbdvbase = 34
        .TEXT
        movea.l  4(sp),a0      ;place-mémoire
        move.l   #$100,d6
        add.l    12(a0),d6

```

```

    add.l    20(a0),d6
    add.l    28(a0),d6
    move.w   #kbdvbase,-(sp)      ;il y a ici plusieurs
;                                adresses possibles
    trap     #xbios
    addq.l   #2,sp
    movea.l  d0,a0
    move.l   16(a0),jump+2        ;remplacer la routine de la
    move.l   #new,16(a0)         ;souris par sa nouvelle
                                routine
    clr.w    -(sp)                ;programme résident
    move.l   d6,-(sp)
    move.w   #keep,-(sp)
    trap     #gemdos
new:    tst.b   1(a0)              ;évènement ?
        bne.s  slow              ;si oui, ralentir
        tst.b  2(a0)             ;pas de mouvement, simple clic ?
        beq.s  jump              ;si oui, toujours exécuter
slow:   bchg  #0,flag            ;n'exécuter la routine originale
        beq.s  fin               ;qu'une fois sur deux
jump:   jmp   $12345678         ;l'adresse est déterminée
;                                à chaque lancement du programme
fin:    rts
        .BSS
flag:   .DS.b 1
        .END

```

On multiplie l'effet de ces deux programmes en les faisant appeler plusieurs fois consécutivement par le GEM. Il y a bien une limite à tout cela, car le traitement d'un groupe d'instructions nécessite tout de même un certain délai (environ une milliseconde), et il ne faut pas le dépasser sous peine de voir se produire quelques effets assez spectaculaires... Encore une remarque: les routines originales de traitement des instructions concernant la souris ne se chargent pas du déplacement de la flèche sur l'écran, elles ne font que modifier ses coordonnées absolues. Le dessin de la flèche se fait à l'aide d'une autre routine-interrupt pendant un VBL.

Vous êtes parmi les heureux possesseurs d'un moniteur Atari noir et blanc? Le programme ci-dessous ne vous sera d'aucune utilité. Il concerne en effet les moniteurs couleurs, sur lesquels se produit un phénomène bizarre en résolution moyenne: la flèche de la souris se déplace deux fois plus vite dans la direction verticale que dans l'horizontale. C'est assez agaçant, mais nous allons pouvoir résoudre ce petit problème grâce aux connaissances que nous venons d'accumuler.

L'astuce consiste à ne laisser le mouvement vertical se produire qu'une fois sur deux. Il suffit pour cela de modifier son état à chaque appel de la routine, donc de la remettre à zéro. C'est tout simple et cela suffit à régler le problème.

```

;
;Déplacement proportionnel  PROP.S
;de la souris (définition moyenne)
;  MP 04-05-88
;
gemdos  = 1
xbios   = 14
keep    = $31
kbdvbase = 34
        .TEXT
movea.l 4(sp),a0           ;place-mémoire
move.l  #$100,d6
add.l   12(a0),d6
add.l   20(a0),d6
add.l   28(a0),d6
move.w  #kbdvbase,-(sp)    ;il y a ici plusieurs
                           ;adresses possibles

trap    #xbios
addq.l  #2,sp
movea.l d0,a0
move.l  16(a0),jump+2 ;remplacer la routine de la
move.l  #new,16(a0)  ;souris par sa nouvelle routine
clr.w   -(sp)        ;programme résident
move.l  d6,-(sp)
move.w  #keep,-(sp)

new:    bchg  #0,flag      ;n'exécuter qu'une fois sur deux
        beq.s jump       ;le déplacement vertical
        clr.b 2(a0)      ;vider le compteur Y
jump:   jmp   $12345678  ;l'adresse est déterminée
;                           à chaque lancement du programme

        .BSS
flag:   .DS.b 1
        .END

```

5.7. Un nouveau pointeur pour la souris

Ne disposer que d'un seul symbole, la flèche, comme pointeur des mouvements de la souris n'est pas seulement monotone: cela peut aussi s'avérer très peu pratique, par exemple lorsqu'il s'agit, en création graphique, de placer très précisément des points sur l'écran. Il serait bien plus utile dans ce cas de disposer d'une croix très fine. Nous pouvons facilement résoudre ce problème, en utilisant la commande Basic-GFA Defmouse, qui correspond dans la plupart des langages compilés à la fonction GEM graf_mouse.

Il faut fondamentalement distinguer les pointeurs standards de la souris et les pointeurs que l'on peut dessiner soi-même. Les pointeurs standards sont ceux dont la forme est définie dans le GEM et que vous pouvez toujours utiliser. Vous connaissez déjà la flèche et certainement l'abeille qui apparaît lors des manipulations de disquettes. Voici la liste des symboles standards offerts par le GEM pour la souris:

```
0 flèche
1 curseur de texte (croix)
2 abeille
3 main avec l'index pointé
4 main à plat
5 croix très fine
6 croix plus épaisse
7 croix évidée
```

Vous pouvez sélectionner un de ces symboles à l'aide de la commande Basic-GFA

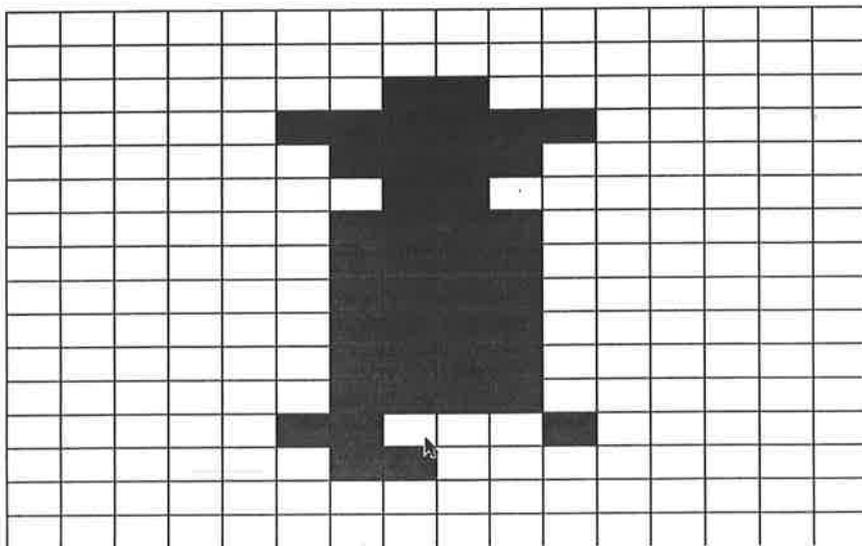
```
Defmouse Numéro
```

Dans l'AES, vous écrivez :

```
graf_mouse (numéro, 0L)
```

'Numéro' désigne ici le numéro du symbole désiré et choisi dans la liste ci-dessus. Le 'L' qui suit le zéro signifie que le paramètre à transmettre prend la forme d'un mot-long, qui n'a pas encore ici de signification.

Il y a cependant des situations dans lesquelles les huit symboles standards ne suffisent pas. Dans un logiciel de création graphique, un crayon pourrait avantageusement remplacer une croix; dans un logiciel de jeu, on voudrait disposer d'un objet extraordinaire (avion, silhouette humaine etc). Un programmeur peut dessiner son propre symbole. Voyons d'abord quelles sont les données utilisées pour le dessin du pointeur de la souris:



Voici l'image que nous voudrions avoir comme symbole du pointeur de la souris. Elle s'inscrit dans une grille monochrome de 16 x 16 points. Il nous faudra faire tenir le contenu de cette grille dans 16 mots. Chaque mot 'décrit' le contenu d'une ligne de la grille, et les 16 colonnes verticales par ligne se retrouveront dans chacun des 16 bits de chaque ligne. Un bit positif symbolise un point visible; le bit le plus élevé d'une ligne (bit 15) représente le point le plus à gauche. Cela peut vous paraître bien compliqué: rassurez-vous, nous allons immédiatement vous donner un petit programme transformant ce type de grille en 16 mots.

Il nous faut, en plus de la forme du pointeur, ce que l'on appelle un masque. En général, ce dernier ressemble à la forme du pointeur, mais son épaisseur est supérieure d'un pixel à celle de la forme. De plus, le contenu défini par le masque est noirci entièrement. Si tel n'était pas le cas, le pointeur ne ressortirait plus sur le fond lorsqu'il serait amené à traverser une surface totalement noire sur l'écran. C'est pourquoi, avant que le pointeur ne soit affiché sur l'écran, le système efface un petit carré de l'écran qui est remplacé par le masque en question (bit positif = effacer le point). Si le masque est un peu plus large que le contour du pointeur, celui-ci sera entouré d'un contour en blanc lorsqu'il traversera une surface noire sur l'écran, ce qui facilite le repérage du curseur.

Il nous faut encore désigner un point central. En effet, lorsqu'un programme recherche la position en cours de la souris, le GEM lui transmet les coordonnées. Nous venons de voir que la grille de dessin du pointeur comprend $16 * 16 = 256$ points. Les deux coordonnées transmises par le GEM désigneront le point central,

celui qui paraît le plus logique: le point de la pointe dans le cas de la flèche, le point de croisement des deux traits dans le cas de la croix etc. Nous devons indiquer à l'ordinateur le point central de notre grille, en précisant ses coordonnées relatives dans le coin en haut à gauche de notre grille.

Il nous faut enfin préciser le numéro de la teinte de notre pointeur, ainsi que celle du fond du masque. Cela nous donne 37 mots en tout:

Offset	Signification
0	coordonnées-x du point central (0-15)
2	coordonnées-y du point central (0-15)
4	réservé, doit se trouver sur un
6	index des couleurs pour le masque (habituellement 0)
8	index des couleurs pour le pointeur (habituellement 1)
10	16 mots pour le masque
42	16 mots pour le pointeur

(offset en octets, toutes les valeurs sous forme de mots)

Les mots numéros 4, 6 et 8 sont à vrai dire toujours semblables. Le calcul du masque et du pointeur est par contre assez pénible et entraîne souvent des fautes d'inattention. Laissons donc à l'ordinateur le soin d'exécuter ce travail à l'aide du programme suivant:

```
'
' Editeur de la souris pour Basic GFA MOUSEDIT.LST
' MP 19-06-88
'
OPTION BASE 0
DIM donnee%(1,16)
souris%=0
masque%=1
xmax%=640+320*(XBIOS(4)=0)
ymax%=200-200*(XBIOS(4)=2)
largeur%=xmax% DIV 16
hauteur%=ymax% DIV 16
'
FOR i%=0 TO 15
  donnee%(souris%,i%)=0
NEXT i%
'
GOSUB dessin
'
ALERT 1,"Editeur de souris Pointeur | t. droite =
fin",1," Suite ",dummy%
GOSUB .traitement(souris%,4)
```

```

'
FOR i%=0 TO 15
  donnee%(masque%,i%)=donnee%(souris%,i%)
NEXT i%
'
ALERT 1,"Maintenant Masque! | t. droite =
fin",1," Suite ",dummy%
GOSUB traitement(masque%,2)
'
ALERT 1,
"Ne plus cliquer | que le point central",1," Envoi ",dummy%
REPEAT
  MOUSE x%,y%,k%
UNTIL k%>0
'
actionx%=x% DIV largeur%
actiony%=y% DIV hauteur%
'
CLS
coordonnées de la souris:"
FOR i%=0 TO 15
  PRINT USING "#####",donnee%(souris%,i%);
  IF i%=7
    PRINT
  ENDIF
NEXT i%
PRINT
PRINT
'
PRINT "Masque:"
FOR i%=0 TO 15
  PRINT USING "#####",donnee%(souris%,i%);
  IF i%=7
    PRINT
  ENDIF
NEXT i%
PRINT
PRINT
'
PRINT "Masque:"
FOR i%=0 TO 15
  PRINT USING "#####",donnee%(masque%,i%);
  IF i%=7
    PRINT
  ENDIF
NEXT i%
PRINT
PRINT
PRINT "point central:"
PRINT SPC(5);"(";actionx%;",";actiony%;")"
'

```

```

END
'
PROCEDURE traitement (was%, type%)
  oldxp%=-1
  oldyp%=-1
  '
  REPEAT
    REPEAT
      MOUSE x%, y%, k%
      IF k%=0
        oldxp%=-1
      ENDIF
    UNTIL k%0
    IF k%=1
      xp%=x% DIV largeur%
      yp%=y% DIV hauteur%
      IF xp% AND yp% AND (xp%oldxp% OR yp%oldyp%)
        IF donnee%(was%, yp%) AND 2^(15-xp%)
          donnee%(was%, yp%)=donnee%(was%, yp%) AND
65535-2^(15-xp%)
          DEFFILL 1, 0, 1
          PBOX xp%*largeur%, yp%*hauteur%, xp%*largeur%+largeur%,
yp%*hauteur%+hauteur%
        ELSE
          donnee%(was%, yp%)=donnee%(was%, yp%) OR 2^(15-xp%)
          DEFFILL 1, 2, type%
          PBOX xp%*largeur%, yp%*hauteur%, xp%*largeur%+largeur%,
yp%*hauteur%+hauteur%
        ENDIF
      oldxp%=xp%
      oldyp%=yp%
    ENDIF
  UNTIL k%=2
RETURN
'
PROCEDURE dessin
  DEFFILL 1, 0, 1
  FOR ligne%=0 TO 15
    FOR colonne%=0 TO 15
      PBOX
colonne%*largeur%, ligne%*hauteur%, colonne%*largeur%+largeur%,
ligne%*hauteur%+hauteur%
    NEXT colonne%
  NEXT ligne%
RETURN
'
PROCEDURE testmouse
ALERT 1, "Test de la souris | t. droite = fin", 1, " Encore ", dummy%
mouse$=""

```

```

mouse$=mouse$+MKI$(actionx%)+MKI$(actiony%)+MKI$(1)+MKI$(0)+MKI$(1)

FOR i%=0 TO 1
  FOR j%=0 TO 15
    mouse$=mouse$+MKI$(donnee%(i%,j%))
  NEXT j%
NEXT i%
'
DEFMOUSE mouse$
SHOWM
REPEAT
  MOUSE dum%,dum%,k%
UNTIL k%=2
RETURN
NEXT colonne%
NEXT ligne%
RETURN
'
PROCEDURE testmouse
ALERT 1,"Test de la souris | t. droite = fin",1," Encore ",dummy%
mouse$=""
mouse$=mouse$+MKI$(actionx%)+MKI$(actiony%)+MKI$(1)+MKI$(0)+MKI$(1)

FOR i%=0 TO 1
  FOR j%=0 TO 15
    mouse$=mouse$+MKI$(donnee%(i%,j%))
  NEXT j%
NEXT i%
'
DEFMOUSE mouse$
SHOWM
REPEAT
  MOUSE dum%,dum%,k%
UNTIL k%=2
RETURN

```

Mode d'emploi : le programme édite d'abord la forme du 'pointeur puis son masque. La touche gauche de la souris sert à placer ou à effacer un point, la touche droite sert à terminer une séquence de travail. Lorsque la forme est dessinée, elle est recopiée dans le masque, si bien qu'il vous suffit ensuite de tracer une ligne autour du contour. Vous devriez aussi noircir complètement le restant du masque. Le programme crée les 16 mots pour la forme et le masque en nommant les coordonnées relatives du point central.

Ce chapitre n'est pas vraiment le lieu pour expliquer le calcul des valeurs de vérité, mais nous allons tout de même en parler un peu pour être complet, en nous servant de l'exemple suivant:

```
Xmax%=640+320*(Xbios(4)=0)
```

Il s'agit de déterminer la largeur de l'écran en pixels; Xbios(4) indique sa résolution actuelle (0=basse, 1=moyenne, 2=haute). Il est assez étonnant de voir une expression comprenant deux fois le signe égal. En Basic, ce signe peut prendre deux significations: il sert d'opérateur d'attribution d'une valeur (A=0) mais aussi d'opérateur de comparaison (If A=0...). Dans notre exemple, il s'agit visiblement d'attribuer une valeur à la variable Xmax%, la valeur résultant de la somme de la constante 640 avec le résultat de 320*(Xbios(4)=0). Le deuxième terme paraît absurde, car son facteur (Xbios(4)=0) est une comparaison et non un nombre. Nous devons préciser que l'ordinateur peut vraiment calculer une comparaison de ce type. Le résultat en sera une valeur de vérité: vrai ou faux. Comme on ne peut guère se servir de ces termes dans un ordinateur, on les représente par des valeurs numériques: 0=faux et -1=vrai. Pour l'ordinateur, 3=2 est équivalent à zéro, puisque 3 n'est pas égal à 2. Par contre 3>2 fournira le résultat -1 puisqu'il s'agit d'une comparaison juste.

Revenons à notre exemple: (Xbios(4)=0) prend la valeur -1 pour une résolution basse, puisque la fonction XBIOS y prend la valeur 0. Dans les résolutions moyenne et haute, ce terme prend la valeur 0, car la comparaison s'avère fausse. Dans ces deux derniers cas, on pourrait écrire:

```
Xmax%=640+320*0      (640)
```

alors que pour la résolution basse, on obtient:

```
Xmax%=640+320*(-1)  (=640-320=320)
```

Il s'agit là justement de la résolution actuelle de l'écran pour les lignes horizontales. Ce n'est pas très simple à comprendre, mais c'est bref, rapide et élégant. Nous utilisons plusieurs fois cette possibilité dans notre programme.

Nous possédons maintenant une foule de chiffres, sans encore pour autant disposer d'un nouveau pointeur de souris. Cela va changer grâce à une des variantes de la commande Defmouse:

```
Defmouse Souris$
```

Ceci nous permet de ne plus transmettre un nombre mais un string contenant les 37 mots décrivant le pointeur de la souris. Pour reproduire cela sous forme de lignes de programme, nous allons recourir à une boucle For. La fonction Mki\$(Mot)

convertit un mot (donc un nombre) en deux signes (donc un string), que l'on peut écrire ainsi :

```
Chr$(HighByte(Mot)) + Chr$(LowByte(Mot)) .
```

Nous vous donnons comme exemple le programme ci-dessous, qui contient un petit programme graphique. La touche gauche de la souris vous permet de placer des points et la touche droite d'interrompre le programme. Le curseur utilisé est le symbole Atari:

```
'
' Nouveau pointeur de souris MOUSENEW.LST
' MP 19-06-88
'
souris$="" ! String des coordonnées de la souris
'
FOR i%=0 TO 36
  READ donnee%
  souris$=souris$+MKI$(donnee%) ! Transformer les données en
                                string
NEXT i%
'
DEFMOUSE souris$
SHOWM
'
DO ! Mini-painter
  REPEAT
    MOUSE x%,y%,k%
    UNTIL k%>0
    EXIT IF k% AND 2
    PLOT x%,y%
  LOOP
'
' Point central
DATA 7,7
DATA 1
DATA 0,1
'
! le curseur (1)
DATA 4064,4064,4064,4064,4064,4064,4064,4064
DATA 8176,8176,16376,16376,32764,64446,64446,62366
'
DATA 0,1344,1344,1344,1344,1344,1344,1344
DATA 3424,2336,2336,6448,4368,12568,24844,0
'
```

Attention: après l'apparition d'une boîte d'erreur, de dialogue ou de sélection de fichiers, c'est la flèche qui redevient le pointeur de la souris, et

qui le reste. C'est pourquoi vous devez, après une commande Alert, réinstaller le pointeur de la souris par un nouveau recours à Defmouse.

Voici la fonction AES équivalente (par exemple pour la programmation en C):

```
graf_mouse(255, coordonnées de la souris
```

Le nombre 255 remplace le numéro d'un pointeur standard de la souris et montre que le paramètre 'coordonnées' pointe sur un tableau contenant les 37 mots.

5.8. Contrôle de l'état de la manette de jeu

en Basic-GFA 2.xx

Lorsqu'on écrit un programme de jeu, on veut le plus souvent pouvoir utiliser la manette de jeu (joystick). Comme le Basic GFA ne comporte (jusqu'à sa version 2.02) pas de commande directe pour la manette de jeu, on en est réduit à créer soi-même des routines. On dispose pour cela de deux adresses qui contrôlent l'état de la manette de jeu et du bouton de tir. Il s'agit des adresses 3593 (manette de jeu) et 3582 (bouton de tir). Seul d'ailleurs le bit 0 peut nous intéresser dans l'adresse 3582. En lisant les adresses, nous voyons que les mouvements sont affectés des valeurs suivantes:

```
1  haut
2  bas
4  gauche
8  droite

1  bouton de tir
```

Les valeurs des déplacements en diagonal résultent de l'addition des deux directions voulues, par exemple: 5 (soit 4+1) pour en haut à gauche.

Il faut ajouter 128 au résultat de cette somme pour pouvoir appuyer sur le bouton de tir au cours d'un mouvement. Nous vous donnons ci-après un exemple en Basic GFA, vous ne devriez avoir aucun mal à le traduire en d'autres langages.

```
'Contrôle de la manette de jeu
',
Do
  Direction=Peek(3593)
  Feu=Peek(3582) And 1
  If Direction=1
    Print "en haut"
  Endif
  If Direction=2
    Print "en bas"
  Endif
  If Direction=4
    Print "à gauche"
  Endif
  If Direction=8
    Print "à droite"
  Endif
  If Direction=9
    Print "en haut à droite"
  Endif
  If Direction=5
    Print "en haut à gauche"
  Endif
  If Direction=10
    Print "en bas à droite"
  Endif
  If Direction=6
    Print "en bas à gauche"
  Endif
  If Feu=1
    Print "bouton de tir"
  Endif
Loop
```

5.9. Contrôle de l'état de la manette de jeu

en Assembleur

Notre système d'exploitation est capable de gérer la plupart des périphériques à l'aide de fonctions très simples. Tous, sauf un: la manette de jeu. Pour pouvoir travailler avec une (ou même plusieurs) manette(s), nous devons écrire les routines nous-mêmes.

Il faut d'abord savoir que, contrairement à l'écran ou l'imprimante, la manette est contrôlée par un deuxième processeur, qui gère principalement le clavier mais aussi d'autres périphériques comme la souris et la manette de jeu. Cette combinaison de deux processeurs (le processeur principal 68000, qui fait tourner vos programmes, et le processeur du clavier) complique un peu l'initialisation de la manette mais simplifie considérablement son contrôle.

Il faut d'abord que le processeur principal puisse communiquer avec son petit collègue. Techniquement, cela se produit par le biais d'une interface série, mais ce détail a peu d'importance pour nous. Nous allons plutôt nous occuper d'une routine XBIOS permettant de transmettre des ordres au processeur du clavier: `Ikbdws`, qui signifie 'Intelligent keyboard write string'. On l'appelle de la façon suivante:

```

move.l #string, -(sp)
move.w #longueur - 1, -(sp)
move.w #25, -(sp) ; numéro de la fonction
trap #14
addq.l #8, sp

```

Le processeur du clavier est capable de traiter lui-même toute une série de commandes: pour en savoir plus, reportez-vous aux manuels Atari. Nous ne nous servons que des trois commandes suivantes:

\$14 Active le contrôle joystick pour deux manettes. A chaque modification de l'état d'une des manettes, le processeur du clavier transmet un groupe d'informations au processeur principal:

1 octet header, qui précise la manette concernée par la modification
 \$fe = joystick 0 (interface souris ou joystick)
 \$ff = joystick 1 (manette de jeu seulement)

1 octet de position de la manette 0 (voir ci-dessous)

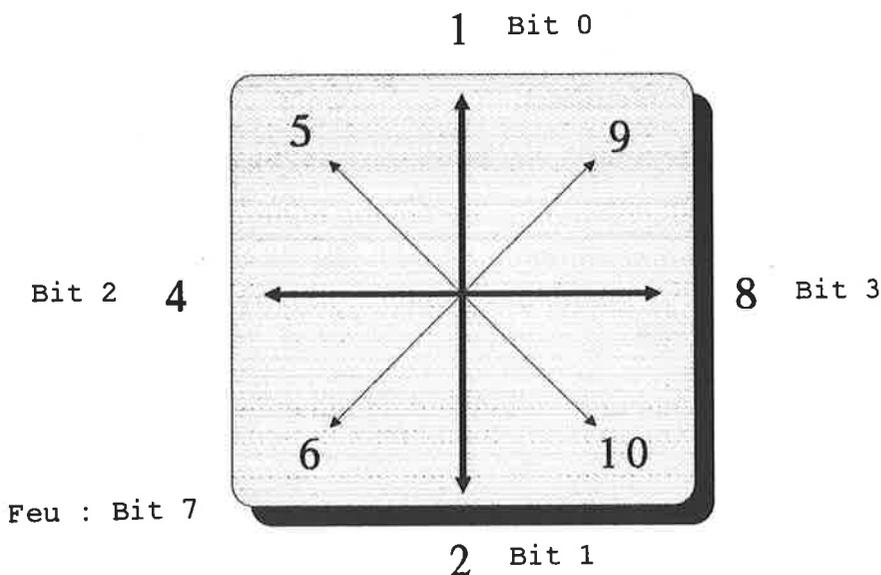
1 octet de position de la manette 1

Attention: cette commande désactive le contrôle de la souris.

\$15 met fin au mode contrôle activé par \$14

\$8 réactive le contrôle de la souris.

La position du joystick est encodée de la façon suivante :



Il ne nous reste plus qu'à détailler le groupe d'informations que le processeur du clavier envoie au processeur principal et qui contient la position de la manette. Le joystick n'est pas le seul instrument envoyant de tels groupes d'informations: la souris, le clavier etc. fonctionnent aussi de cette manière. C'est pourquoi le processeur principal trie d'abord les paquets en catégories réparties selon leur origine. Chaque catégorie est ensuite soumise à des routines qui lui sont particulières. Les adresses de ces routines se trouvent dans un tableau, dont vous recherchez l'adresse à l'aide de la fonction XBIOS kbdvbase (abréviation de: keyboard vector base).

Nous ne nous intéressons pour le moment qu'au vecteur chargé du traitement des données en provenance de la manette de jeu: c'est le 7ème vecteur du tableau, qui est donc appelé par un offset de 24 octets ($7*4-4$). Nous devons orienter ce vecteur vers notre routine de manette de jeu. Mais qu'est-ce au juste qu'une routine de manette de jeu?

La routine joystick est une interface entre le groupe d'informations arrivant du processeur du clavier et le programme ou le logiciel nécessitant la position du joystick. Cette routine ne se charge pas des mouvements d'une figurine de jeu quelconque; elle se borne à transmettre au processeur principal les mouvements du pointeur de la manette, en laissant au programme principal le soin de prendre en charge la représentation sous forme de figurine.

Encore une précision importante: lorsque le système d'exploitation reçoit un groupe d'informations et appelle notre routine, le registre A0 est affecté d'un pointeur vers ce groupe. La routine doit se terminer par 'RTS' et être aussi brève que possible; Atari n'autorise que des routines entraînant un temps de traitement d'une milliseconde au maximum.

Après toutes ces explications, vous devriez sans peine comprendre le programme suivant:

```

;
; Contrôle de la manette en Assembleur  JOYSTICK.S
;   MP 09-06-88
;
gemdos    = 1
bios      = 13
xbios     = 14
print     = 9
bconout   = 3
kbdvbase  = 34
.TEXT
        move.w    #kbdvbase,-(sp)      ;Rechercher le pointeur
                                        sur manette

        trap      #xbios
        addp.l    #2,sp
        movea.l   d0,a5                ;Retour:pointeur vers un
                                        tableau

        lea.l     24(a5),a5            ;7ème pointeur : joystick
        move.l    (a5),oldvec          ;sauver ancien contenu
                                        (pointe vers RTS)

        move.l    #joystick,(a5)      ;puis installer la
                                        routine

        pea      onstr
        clr.w    -(sp)
        move.w   #25,-(sp)
        trap     #xbios
        addq.l   #8,sp
loop:    move.b   position,d0           ;position actuelle du
                                        joystick

        btst     #7,d0                 ;bouton tir utilisé ?
        bne.s    exit                  ;si oui, terminer ici le
                                        programme

        andi.w   #%1111,d0            ;concerne seulement bits
                                        0 à 3

        asl.w    #2,d0                 ;multiplié par 4
        lea.l    tableau,a0           ;comme index pour
                                        l'affichage

        move.l   0(a0,d0.w),-(sp)
        move.w   #print,-(sp)

```

```

trap      #gemdos
addq.l   #6, sp
bra.s    loop
exit:    pea      offstr          ;ne plus déclarer le
                                   joystick

        move.w   #1, -(sp)
        move.w   #25, -(sp)
        trap     #xbios
        addq.l   #8, sp
        move.l   oldvec, (a5)    ;rétablir routine de
                                   souris après RTS

        clr.w    -(sp)          ;terminer le programme
        trap     #gemdos
joystick: cmpi.b  #$ff, (a0)     ;joystick 1 déplacé ?
        bne.s   finish
        move.b   2(a0), position ;consigner la nouvelle
                                   position

finish:   rts
        .DATA
tableau:  .DC.l  j0, j1, j2, 0, j4, j5 ;adresses des strings
        .DC.l  j6, 0, j8, j9, ja    ;(0= 'non pilotable')
position: .DC.b  0

onstr:   .DC.b  $14 ;$14 = mode automatique
offstr:  .DC.b  $15, 8 ;$15 = sortir et $8 rétablir souris
j0:      .DC.b  13, 'pas appuyé ' , 0
j1:      .DC.b  13, 'en haut ' , 0
j2:      .DC.b  13, 'en bas ' , 0
j4:      .DC.b  13, 'à gauche ' , 0
j5:      .DC.b  13, 'en haut à gauche ' , 0
j6:      .DC.b  13, 'en bas à gauche ' , 0
j8:      .DC.b  13, 'à droite ' , 0
j9:      .DC.b  13, 'en haut à droite ' , 0
ja:      .DC.b  13, 'en bas à droite ' , 0
        .BSS

oldvec:  .DS.l  1
        .END

```

5.10. Freezer et reset par une simple touche

Vous êtes assis devant votre ordinateur: le joystick en main, vous chassez un extra-terrestre à travers tout l'écran, les bits sifflent à vos oreilles, la bataille fait rage; encore deux minutes, et vous aurez gagné le jeu par un score encore jamais atteint.

Ah! vous étiez bien près de gagner! mais voilà que votre téléphone sonne. Deux possibilités: soit votre logiciel comprend l'option pause vous permettant de reprendre le jeu avec une énergie nouvelle, là où vous étiez avant votre conversation téléphonique, soit votre jeu n'est pas doté d'une possibilité d'interruption-pause, vous jouez de malchance et n'atteindrez pas votre high-score historique.

Le programme ci-dessous va vous permettre d'interrompre vos logiciels de jeux qui ne sont pas dotés d'une option pause. Le logiciel en question doit satisfaire à deux conditions: vous devez pouvoir le lancer par un double-clic à partir du bureau GEM et il ne doit pas désactiver le VBL-Queue. Vous trouverez dans le chapitre consacré aux programmes résidents les explications sur le VBL-Queue. Pour l'instant, sachez que si vous pouvez, dans votre logiciel, faire une copie-écran par <ALTERNATE>+<HELP>, le programme FREEZER devrait fonctionner correctement.

Votre dictionnaire vous apprend que 'to freeze' signifie 'geler'. Notre programme FREEZER va donc vous permettre de stopper toutes les procédures en cours, comme l'eau qui s'arrête de couler lorsqu'elle gèle (en faisant abstraction des glaciers). Nous nommons FREEZER un programme qui attend une certaine combinaison de touches pour stopper les processus en cours et les reprendre après avoir reçu une autre combinaison de touches. Naturellement, l'ordinateur n'est pas entièrement inactif pendant le temps d'attente, puisqu'il faut bien qu'il réagisse à la combinaison de touches lui ordonnant de reprendre le traitement là où il fut interrompu.

En prime, notre programme vous offre la possibilité de déclencher un 'reset' de l'ordinateur par une simple combinaison de touches. Ceci réjouira tout particulièrement les possesseurs de Méga-ST, ordinateur dont le bouton de reset est difficilement accessible. L'objectif paraît simple, passons à sa réalisation. Il faut tout d'abord que notre programme reste résident dans la mémoire de travail, afin de pouvoir guetter la combinaison de touches lui ordonnant de reprendre le traitement interrompu. Ce contrôle d'évènement doit se faire à intervalles réguliers. Nous avons choisi d'utiliser le VBL-Interrupt, car il est plus simple à programmer et largement suffisant pour atteindre notre but. Voici qui résout le problème de l'initialisation du programme. Avant de revenir au bureau GEM, notre programme se limite à indiquer les touches utilisables. L'utilisateur pourrait oublier leur signification et confondre la combinaison provoquant la pause 'Freezer' avec celle qui déclenche un 'Reset', ce qui vraisemblablement provoquerait quelques explosions colériques de moyennes à grandes amplitudes... C'est pourquoi notre programme va attendre quelques secondes, pour que vous ayez le temps de bien lire les indications affichées à l'écran.

Tournons-nous maintenant vers l'utilisation des touches du clavier: nous avons choisi comme touches de commande <ALTERNATE> <CONTROL> ainsi que les deux touches <SHIFT>. La fonction BIOS numéro 11 kbshift permet de surveiller le statut de ces touches. Mais ce n'est pas simple d'appeler le système d'exploitation lors des 'interrupt', ça ne marche pas à tous les coups. Il n'est pas indispensable pour nous d'appeler cette fonction, puisqu'il existe dans la mémoire une place réservée pour la sauvegarde de l'état de ces touches spéciales. Le problème étant que l'adresse de cette place-mémoire n'est pas la même dans l'ancien et dans le nouveau TOS. Avant le 6.2.1986, cette valeur se trouve à l'adresse \$e1b, alors que le Blitter-TOS se sert de l'adresse \$e61. C'est pourquoi notre programme recherche en tout premier lieu la date du système d'exploitation (qui n'est pas la date que vous avez portée dans le tableau de contrôle) afin d'identifier la version du TOS et de savoir s'il faut prendre l'adresse \$e1b ou \$e61. D'où l'utilisation de 'dispatcher'. La date du système se trouve dans l'une des premières adresses de la ROM.

Il ne nous reste plus qu'à préciser comment nous nous y prenons pour provoquer un 'reset' ou pour 'geler' l'action d'un logiciel. Pour déclencher le premier processus, il suffit de parvenir à l'adresse \$fc0000. C'est là que débute la ROM, c'est là aussi que se trouve la première commande-machine exécutée après un 'reset' ou lors de l'allumage de l'ordinateur. Il n'est pas non plus bien compliqué de geler l'action d'un programme en cours, il nous suffit d'attendre jusqu'à ce que l'utilisateur appuie sur la touche servant à relancer le programme. Le système d'exploitation ignore les autres interrupts-VBL qui se produisent durant ce délai d'attente (il y a jusqu'à 70 retours d'image par seconde).

```

;
; Arrêt d'un programme ou reset  FREEZER.S
;   MP   06-06-88
;
gemdos      = 1
print      = 9
xbios      = 14
keep       = $31
superexec  = 38
_vblqueue  = $456
nvbls     = $454
kbshift    = $e1b
shift_blt  = $e61
date       = $fc001e

.TEXT
movea.l    4(sp),a0           ;calculer la place-mémoire
move.l    #$100,d6
add.l     12(a0),d6

```

```

        add.l    20(a0),d6
        add.l    28(a0),d6
        pea     title                ;afficher le texte
        move.w   #print,-(sp)
        trap    #gemdos
        addq.l   #6,sp
        moveq.l  #15,d0              ;délai (environ 2 secondes)
loop1:   moveq.l  #-1,d1
loop2:   dbra    d1,loop2
        dbra    d0,loop1
        cmpi.w   #$c46,date          ;ancien TOS ou Blitter-TOS ?
        beq.s   old_tos
        move.w   #shift_blt,shift+2 ;Blitter-TOS -> Patch
                                des adr
        move.w   #shift_blt,wait+2  ;esses de statut des
                                touches
old_tos: pea    init                ;initialisation dans
                                superviseur
        move.w   #superexec,-(sp)
        trap    #xbios
        addq.l   #6,sp
        clr.w    -(sp)              ;garder le programme en
                                résident
        move.l   d6,-(sp)
        move.w   #keep,-(sp)
        trap    #gemdos
init:    move.w   nvbls,d0           ;Nombre de routines VBL
        lsl.w   #2,d0              ;multiplié par 4
        movea.l  _vblqueue,a0       ;Liste des adresses start
        moveq.l  #4,d1              ;rechercher dans la 2ème saisie
search:  tst.l    0(a0,d1.w)          ;si l'espace est libre,
                                présence ici d'un zéro
        beq.s   found
        addq.w   #4,d1              ;sinon, ausculter la saisie
                                ;suivante
        cmp.w   d0,d1              ;tous auscultés ?
        bne.s   search
                                ;si oui, bombes
found:   move.l   #vbl,0(a0,d1.w)   ;vecteur vers les routines VBL
        rts
vbl:    move.w   d0,-(sp)           ;sauver le registre
shift:  move.b   kbshift,d0        ;statut des touches spéciales
        andi.b   #15,d0            ;Alt, Ctrl et 2 x shift
                                ;à masquer
        cmpi.b   #15,d0            ;Toutes appuyées ?
        bne.s   no_reset
        jmp     $fc0               ;déclencher le reset
no_reset: cmpi.b  #7,d0            ;Seulement Ctrl et 2xShift ?
        bne.s   end_vbl           ;déclencher la pause 'freezer'
wait:   move.b   kbshift,d0
        andi.b   #8,d0            ;Alternate appuyé ?

```

```

                beq.s    wait          ;si pas encore, attendre
end_vbl:      move.w    (sp)+,d0      ;retour au registre
                rts
                .DATA
title:        .DC.b    27,'E',27,'P'
                .DC.b    '<< FREEZER >> ---> Control + les 2 Shift ',13,10

                .DC.b    ' << FIN >>    ---> Alternate ',13,10,10
                .DC.b    ' << RESET >>  ---> Control + Alternate + les 2
Shift ',0

                .END

```

5.11. Pour épargner votre écran

Si vous vous servez intensivement de votre ordinateur, que ce soit pour saisir des textes longs, pour programmer etc, il vous est certainement déjà arrivé de devoir vous interrompre pour aller prendre un repas et vous avez laissé allumée votre configuration. Cela ne nuit pas à l'ordinateur.

Par contre, ces images fixes représentent un grand danger pour votre écran. Les électrons qui composent l'image viennent frapper la face interne du tube cathodique toujours aux mêmes endroits. Ceci endommage la couche d'enduit qui garde ensuite la marque de l'image. Celle-ci restera visible sur l'écran même après avoir éteint le moniteur. Pour remédier à cela, vous pouvez à chaque pause soit baisser l'intensité lumineuse de l'écran jusqu'à ce qu'il soit noir, soit utiliser le programme ci-dessous.

```

////////////////////////////////////
; Prenez soin de votre écran          ;;          SAVER.PRG          ;
////////////////////////////////////
illegal    = $10

                movea.l   4(sp),a0          ; calculer la
                                                place-mémoire nécessaire
move.l     #$100,d6          ;
add.l     12(a0),d6         ;
add.l     20(a0),d6         ;
add.l     28(a0),d6         ;
pea       init              ; initialisation dans le
                                                superviseur
move.w    #38,-(sp)         ;

```



```

        rts                ;
;::::::::::::::::::::::::::::::::::::;
; nouvelle routine interrupt        ;
; pour le clavier                  ;
;::::::::::::::::::::::::::::::::::::;
new_ir:                ;
        move.w    #1000,compteur    ; deux minutes de délai
(secondes ; 71)
        bsr      vbl2

        .DC.w $4ef9                ; code conditionnel pour Jmp
spr_adr:                ;
        .DC.l 0                    ; place pour l'adresse dans
                                la routine originale
;::::::::::::::::::::::::::::::::::::;
;
        .DATA
compteur: .DC.w 1000            ;
vector:  .DC.l 0                ;

```

Il calcule d'abord la place-mémoire nécessaire et installe en mode superviseur deux routines interrupt, d'une part une routine-VBL dans VBL-Queue et d'autre part une nouvelle routine interrupt pour le clavier et l'interface MIDI. Vous trouverez de plus amples explications au sujet de VBL-Queue dans le chapitre consacré aux programmes résidents.

Dans l'Atari, on trouve sous \$118 le vecteur interrupt des deux composants ACIA qui gèrent le clavier et l'interface MIDI. Nous introduisons ce vecteur dans la variable spr_adr, que nous gardons disponible dans le programme en la dotant d'une valeur longue. Cette variable est précédée d'une pseudo-valeur \$4EF9 qui remplace l'opcode pour la commande Jmp. En bref, ceci nous permet d'introduire ainsi un Jmp dans la routine interrupt normale. Cette programmation permet de procéder à la modification sans utiliser de registre, ce qui nous évite d'avoir à les sauver.

Après avoir précisé ce vecteur, nous pouvons enregistrer notre routine. Celle-ci interviendra lors de toute action de la souris, des touches de la souris, de celles du clavier et de tous signaux en provenance de l'interface MIDI. Cette routine place le compteur sur la valeur permettant d'introduire un délai d'attente.

Dans la routine VBL1 (la routine VBL ainsi installée), la valeur du compteur décroît 71, 60 ou 50 fois par seconde selon la fréquence-écran. Une fois que la valeur zéro est atteinte, le programme passe sur la synchronisation externe, et comme celle-ci fait défaut, l'écran s'éteint. Après quoi s'installe une routine VBL2, qui attend jusqu'à ce qu'elle reçoive un signal du clavier, de la souris ou de

l'interface MIDI, ce qui relance le compteur par le biais du nouvel interrupt du clavier et de l'interface MIDI. Le moniteur repasse alors sur la synchronisation interne, ce qui remet l'écran dans son état habituel.

Voici enfin la description détaillée de la placé-mémoire gérant la synchronisation. Seuls les deux bits inférieurs sont utilisés dans cet octet:

\$FF8220A

Bit	Signification
0	0 = synchro. interne 1 = synchro. externe
1	0 = 60 Hz 1 = 50 hz

Pendant que l'écran est éteint, l'ordinateur peut continuer à travailler, si bien que vous pouvez par exemple compiler de longs programmes, sans avoir constamment à bouger la souris.

5.12. Le véritable multitasking sur l'Atari ST

Vous êtes-vous déjà laissé entraîner à discuter avec le possesseur d'un Amiga sur les avantages et inconvénients de cet ordinateur par rapport à l'Atari ST? Ce dut être une rude discussion, se terminant sûrement par un match nul. A côté de ses aptitudes incontestables pour la création graphique et sonore, l'Amiga possède une qualité très enviée par le propriétaire d'un Atari: le multitasking ou multi-tâches en français. Ce terme fait référence à la possibilité de faire tourner simultanément plusieurs programmes, ou plus exactement de faire en sorte que l'utilisateur croit qu'il fait tourner plusieurs programmes en même temps. L'Atari de fabrication standard ne peut faire cela, même si ses accessoires permettent parfois de frôler le travail en multitâche.

Convenons que si l'Amiga en est capable, cela ne doit rien avoir d'inaccessible pour l'Atari. En effet, cette aptitude au fonctionnement multitâche dépend beaucoup moins du matériel que du système d'exploitation de l'ordinateur. Nous allons donc essayer de faire tourner deux programmes en même temps sur notre Atari.

Attention : pour comprendre le programme ci-dessous, vous devez avoir quelques connaissances de base en matière de programmation du ST en langage machine ainsi que sur le MFP 68901, qui est présenté dans le chapitre sur les programmes résidents (tout au moins ses timer, la seule chose qui nous intéresse ici). Un système d'exploitation multitâche est un des domaines les plus complexes mais sans doute les plus intéressants de la programmation système. Même si vous n'y connaissez encore rien en langage Assembleur, vous pouvez maintenant affirmer avec force que votre Atari fait aussi du multi-tâches, et faire aux sceptiques une démonstration avec le programme MULTITSK.PRГ (qui se trouve sur la disquette jointe à ce livre).

Passons aux choses intéressantes, avec une première précision: il va de soi que le traitement d'une petite routine résidente lors d'un interrupt (par exemple VBL-Queue) ne constitue pas pour nous un traitement multitâche, ce serait trop simpliste! Notre objectif consiste à faire que deux programmes soient traités exactement à égalité par l'ordinateur, et que chacun des programmes dispose de la moitié du temps de traitement de l'ordinateur. Nous allons utiliser une bascule qui se charge à intervalles réguliers d'interrompre un premier programme (après avoir sauvegardé ses registres et son compteur) pour passer à un deuxième programme, pour lequel il devra rechercher les registres telles qu'ils étaient au moment de son interruption pour ensuite le relancer. Cette 'bascule' s'appelle un dispatcher.

Notre programme se compose des éléments suivants: un dispatcher, deux programmes de démonstration et une initialisation permettant de lancer le tout. En voici d'abord le texte complet :

```

;
; Mini-multitâche : MULTITSK.S
; 15-02-88  MP
;
gemdos      = 1
setblock    = $4a
super       = $20
xbtimer     = 31
xbios       = 14
jdisint     = 26
conin       = 1
ret         = 13
print       = 9
             movea.l   sp,a5                ;calculer la
                                             place-mémoire nécessaire
             movea.l   4(a5),a5
             move.l    12(a5),d6
             add.l     20(a5),d6

```

```

add.l    28(a5),d6
addi.l   #$1100,d6
move.l   d5,d0                                ;adresse start pour
                                              basepage
add.l    d6,d0                                ;+ place-mémoire
                                              nécessaire
andi.l   #-2,d0                               ;créer l'adresse et
                                              l'utiliser
movea.l  d0,sp                                ;comme stackpointer
move.l   d6,-(sp)
move.l   d5,-(sp)
move.w   d6,-(sp)
move.w   #setblock,-(sp)
trap     #gemdos
adda.l   #12,sp
.DC.w   $a00a                                ; effacer le pointeur de
                                              souris
pea      prompt
move.w   #print,-(sp)                        ; afficher message de
                                              départ
trap     #gemdos
addq.l   #6,sp
clr.l    -(sp)                                ; passer en superviseur
move.w   #super,-(sp)
trap     #gemdos
addq.l   #6,sp
move.l   d0,oldssp
move.w   #$2700,sr                            ; verrouiller l'interrupt
pea      dispatcher                          ; lancer le timer A
move.w   #192,-(sp)                          ; 200 Hz
move.w   #5,-(sp)
clr.w    -(sp)
move.w   #xbtimer,-(sp)
trap     #xbios
adda.l   #12,sp
move.l   #stack2,register2+60                ; stackpointer
move.l   #job2,pcounter2                    ; initialiser l'adresse
                                              de
move.w   #$2300,status2                     ; départ et statut pour
                                              Job 2
move.w   #1,jobno                            ; commençons par job 1
movea.l  #stack1,sp
pea      job1                                ; adresse de départ sur
                                              le stack
move.w   #$2300,-(sp)                        ; autoriser de nouveau
                                              superviseur/interrupts
rte
finish:  move.w   #13,-(sp)                   ; verrouiller timer A
move.w   #jdisint,-(sp)
trap     #xbios
addq.l   #4,sp

```

```

    pea    dispatcher      ; effacer timer
    clr.w  -(sp)
    clr.w  -(sp)
    clr.w  -(sp)
    move.w #xbtimer, -(sp)
    trap   #xbios
    adda.l #12, sp
    move.l oldssp, -(sp)      ; mode user
    move.w #super, -(sp)
    trap   #gemdos
    addq.l #6, sp
    .DC.w $a0 ; réafficher pointeur souris
    pea    crsroff          ; désactiver le curseur
    move.w #print, -(sp)    ; séquence VT-52: ESC f
    trap   #gemdos
    addq.l #6, sp
    clr.w  -(sp)           ; fin
    trap   #gemdos
; le dispatcher se charge de passer d'un programme à l'autre
dispatcher:
    cmpi.w #1, jobno       ; quel est le job en
                           ; cours
                           ; de traitement
    bne.s  dis2            ; le deuxième ?
    move.w (sp), status    ; non, le premier
    move.l 2(sp), pcounter
    movem.l d0-d7/a0-a7, register
    movem.l register2, d0-d7/a0-a7
    move.w status2, (sp)
    move.l pcounter2, 2(sp)
    move.w #2, jobno
    bclr   #5, $fffffa0f
    rte
dis2:    move.w (sp), status2 ; interruption du 2ème
                           ; job
    move.l 2(sp), pcounter2
    movem.l d0-d7/a0-a7, register2
    movem.l register, d0-d7/a0-a7
    move.w status, (sp)
    move.l pcounter, 2(sp)
    move.w #1, jobno
    bclr   #5, $fffffa0f
    rte
; Viennent ensuite deux programmes de
; démonstration qui vont être activés à
; tour de rôle par le dispatcher
job1:    movea.l $44e, a0
         adda.l #32000, a0
         move.w #7999, d0
loop:    eori.l #1, -(a0)
         move.w #30, d1

```

```

lp2:      dbra      d1,lp2                ; boucle d'attente
          dbra      d0,loop
          bra.s     job1
job2:     move.w    #conin,-(sp)          ; conin avec écho
          trap      #gmdos
          addq.l    #2,sp
          cmpi.w    #ret,d0              ; touche <RETURN> ?
          beq       finish                ; si oui, terminer
          bra.s     job2
          .DATA
prompt:   .DC.b    27,'E',27,'e',10,10
          .DC.b    '                      MULTI-TASKING sur votre ST'
          .DC.b    13,10,' programme de démonstration',13,10,10,10
          .DC.b    ' pendant que le premier Job dessine des lignes'
          .DC.b    ' et les efface,',13,10
          .DC.b    ' le deuxième traite ce qui arrive depuis'
          .DC.b    ' le clavier.',13,10,10
          .DC.b    ' les 2 programmes tournent à égalité!'
          .DC.b    13,10,10,10
          .DC.b    ' votre nom s.v.p.:      ==> ',0
crsroff:  .DC.b    27,'f',0
          .BSS

oldssp:   .DS.l    1
register:  .DS.l    16 ; on procède ici à la sauvegarde
pcounter: .DS.l    1 ; de tous les registres, du compteur
status:   .DS.w    1 ; et du registre des statuts;
register2:
          .DS.l    16 ; cf ci-dessus, pour le 2ème job
pcounter2:
          .DS.l    1
status2:  .DS.w    1
jobno:    .DS.w    1 ; programme en cours
          .DS.w    1000 ; 2 programmes donc 2
                                stacks

stack1:   .DS.w    1000
stack2:
          .END

```

Le premier problème consiste à appeler le dispatcher: nous nous servons du timer A du MFP. Nous avons sélectionné une fréquence de 200 Hz, ce qui signifie que le dispatcher va passer 200 fois par seconde d'un programme à l'autre. Cela se fait à une telle vitesse que le processus n'est pas perceptible pour l'utilisateur, sans cependant excéder les capacités de l'ordinateur - songez que ces passages d'un programme à l'autre consomment aussi du temps-calcul.

Chacun des deux programmes en cours nécessite une place-mémoire suffisante pour y enregistrer tous les registres du processeur, y compris le registre des états et le compteur (PC). Le dispatcher se borne à enregistrer les données en cours dans

l'une des places-mémoires et à préparer le processeur à passer dans la deuxième. Le registre des états et le compteur du programme interrompu se trouvent dans le stack. Il nous suffit de surimprimer les deux valeurs avec celles du programme à activer, si bien que le dispatcher repasse dans l'autre programme lorsqu'il reçoit la commande RTE.

Vous avez maintenant compris le principe de fonctionnement: nous pouvons tester notre système à l'aide de deux programmes de démonstration. C'est là malheureusement qu'intervient le problème principal: le système d'exploitation TOS ne peut absolument pas traiter des appels de fonctions provenant de deux logiciels à la fois. Ce qui revient à dire que le système d'exploitation ne peut s'occuper que d'un seul programme à la fois. C'est pourquoi dans notre programme de démonstration, seule la boucle d'attente de saisie permet l'accès au système d'exploitation; pendant ce temps, l'autre programme dessine quelques lignes en travaillant directement dans la mémoire de l'écran et donc sans recourir à des fonctions du système d'exploitation.

Il est donc clair que nous n'avons à ce moment qu'un dispatcher, sans encore disposer d'un système d'exploitation capable d'en utiliser toutes les ressources. La description d'un tel système dépasserait nettement les limites de cet ouvrage. Mais, même s'il ne lui donne pas accès aux fonctions du système d'exploitation, notre exemple de programme offre au programmeur averti en Assembleur un vaste choix de nouvelles possibilités qu'il mettra un certain temps à épuiser. Personne ne vous oblige à faire intervenir simultanément deux programmes totalement indépendants l'un de l'autre comme c'est le cas dans notre exemple. Prenons l'exemple d'un traitement de texte ou d'un éditeur dans lequel l'un des "job" surveille le clavier pour insérer dans le texte les caractères saisis pendant qu'un deuxième se charge de l'affichage à l'écran. Autre exemple: un programme de création graphique qui pourrait simultanément imprimer une image tout en en traitant une autre, ce qui rendrait à l'utilisateur la disponibilité de son ordinateur juste après avoir lancé l'impression.

5.13. Utilitaire d'impression

Le programme qui va suivre dépasse un peu les limites de cet ouvrage à cause de sa longueur. Mais il vous montre très bien comment programmer certaines choses en Assembleur, et il sera un outil très utile même pour un non-programmeur.

Son but peut se résumer en quatre mots : impression de fichiers ASCII. Vous allez nous dire qu'il est sans doute exagéré d'écrire 500 lignes en Assembleur pour un

travail réalisable tout bêtement par un double-clic au niveau du bureau GEM ("seule la visualisation ou l'impression de ce document est possible : voir - imprimer - annuler"). Votre objection n'est pas injustifiée, mais avez-vous vraiment déjà imprimé de cette façon des listings de programmes? Quant à nous, il nous est apparu dès la première impression qu'il manquait quelque chose. Même certaines machines à écrire à traitement de texte peuvent numérotter les lignes automatiquement dans une impression page par page, on doit donc pouvoir facilement le faire avec un Atari, et ne parlons pas de l'ajout d'une ligne d'en tête à chaque page... Naturellement, n'importe quel traitement de texte vous permettra une sortie impression avec en-tête pour chaque page, mais il vous manquera toujours la numérotation des lignes.

On a fait là des économies de bouts de chandelles au niveau software. Le programme qui suit remplira tous ces objectifs, tout en étant adaptable à n'importe quelle imprimante; vous pourrez même remplacer à l'impression un 'a' par un 'o' si bon vous semble. Vous n'avez même pas besoin d'un Assembleur pour fixer les paramètres de l'imprimante, car ils se trouvent dans un fichier-paramètres que l'on peut écrire avec n'importe quel traitement de texte ou même un simple éditeur.

Nous allons d'abord donner un mode d'emploi pour nos lecteurs peu intéressés par la programmation; l'explication du programme viendra ensuite.

Le programme peut être lancé soit depuis le bureau GEM, soit en le plaçant dans le dossier AUTO. L'écran doit se trouver en résolution haute ou moyenne. Le fichier PRINT.PAR est indispensable dès le lancement du programme, car il contient toutes les informations nécessaires pour le pilotage de l'imprimante. Comme nous l'avons dit, vous pouvez créer ce fichier à l'aide de n'importe quel traitement de texte (sauvegarde sous forme ASCII). Les chiffres doivent être saisis en hexadécimal sur deux rangs, les lettres A à F écrites en majuscules. Pour ne pas allonger exagérément le programme, nous avons renoncé au dépistage d'erreur. Les commentaires explicatifs sont introduits par un point-virgule. Vous avez un exemple de fichier de paramètres pour imprimante Epson (et compatible) sur la disquette jointe à ce livre (PRINT.PAR).

Dans ce fichier, le premier nombre indique le nombre de lignes à imprimer par page. On ajoute 3 lignes à ce nombre pour l'en-tête; ce nombre ne vous concerne que si vous voulez faire des impressions page par page. Le deuxième nombre indique le nombre de caractères par ligne. Dans les impressions page par page, il permet de centrer l'en-tête suivie d'une ligne remplie de tirets. Les lignes suivantes sont construites de façon à donner en première position le code en provenance du fichier suivi des codes envoyés à l'imprimante. Si la ligne ne comprend qu'un seul nombre, cela signifie que l'impression de ce code a été refoulée. Les trois

premières lignes jouent un rôle particulier: la ligne numéro deux contient un string envoyé à l'imprimante avant le démarrage de l'impression, la ligne numéro trois un string signifiant que le processus est terminé. La ligne numéro quatre reçoit le code de commande de votre imprimante pour une avancée de page. Le tableau doit obligatoirement se terminer par 00.

Vous lancez le programme en appuyant simultanément sur <ALTERNATE>+<SHIFT>+<HELP>. Vous devez vous trouver dans une application GEM, à un endroit où le programme en cours attend une réponse de votre part. Il ne s'agit que d'une apparente contrainte, car primo il n'existe quasiment plus de programme ne faisant appel qu'au TOS et secundo n'importe quel logiciel attend toujours à un moment donné un de vos ordres.

Après avoir appelé PRINT, vous constatez que la ligne 25 de votre écran fonctionne comme une ligne d'édition. Entrez d'abord le nom du fichier que vous voulez imprimer. S'il s'avère introuvable, vous recevrez un message d'erreur. Il y aura problème si vous faites apparaître le message 'Insérez la disquette B dans l'unité A', car ce message ne s'affichera pas, et le programme se dirigera de toute façon vers la disquette se trouvant dans le lecteur actuel.

L'ordinateur vous demande ensuite si l'impression doit se faire page par page, c'est-à-dire s'il doit provoquer un saut de page après le nombre de lignes que vous aurez précisé dans le fichier des paramètres. Si vous retenez l'option impression page par page, le programme vous réclame une en-tête qui apparaîtra centrée en haut de chaque page sur la première ligne. Dernière question: désirez-vous que les lignes soient numérotées?

L'impression commence et vous pouvez aller tranquillement vous asseoir dans votre fauteuil. Notons cependant que ce programme peut aussi parfaitement fonctionner avec un spooler qui vous rendra immédiatement la disponibilité de votre ordinateur. Nous n'avons pas inséré de spooler dans ce programme, premièrement parce qu'il est déjà assez long comme cela, et deuxièmement parce que nous pensons que nos lecteurs ont déjà leur spooler préféré sous la main.

Venons-en à la description du programme lui-même. Vous trouverez dans le chapitre sur les programmes résidents tous les secrets de l'installation d'un VBL-Queue et nous n'y reviendrons donc pas ici. Les choses deviennent plus intéressantes avec le label vbl. La variable du système d'exploitation `_dumpflag` pretourne la valeur 0 lorsque l'utilisateur appuie sur les touches <ALTERNATE>+<HELP> pour déclencher une copie-écran. Si cette valeur est différente de 0, nous pouvons tout de suite arrêter les frais. Sinon, nous demandons, au moyen de la fonction BIOS Kbshift, si la touche <SHIFT> gauche a été appuyée.

Si ce n'est pas le cas, l'utilisateur ne voulait qu'une copie-écran simple; si cette touche est appuyée, c'est par contre à nous de jouer.

Quelques mots d'explication sur le 'critical error handler'. Il s'agit d'une commande faisant apparaître sur l'écran un message d'erreur, par exemple lorsque la disquette est défectueuse ou que l'imprimante ne réagit pas. Si ce message apparaît durant un interrupt-VBL, il a de fâcheux effets sur votre ordinateur, et vous ne vous en tirez qu'en appuyant sur la touche 'reset'. C'est pourquoi nous remplaçons la routine originale par un RTS, ce qui permet d'ignorer ce message. Ceci a une conséquence assez criticable, la disparition du message 'insérez la disquette B dans l'unité A...'; il faut faire avec, il n'y a pas d'autre choix possible.

Nous avons déjà présenté par ailleurs la boucle d'attente de saisie INPUT.S, qui nous sert ici encore de routine de saisie. Elle sert à la saisie du nom de fichier et éventuellement à celle de l'en-tête. C'est là qu'intervient la lecture du fichier. Nous utilisons la fonction GEMDOS Ffirst pour déterminer la taille-mémoire nécessaire pour la routine de lecture. Le reste est facile à comprendre, nous n'allons plus jeter qu'un coup d'oeil sur le traitement du fichier de paramètres.

Le sous-programme 'readfile' charge le fichier PRINT.PAR. Les données précisant le nombre de lignes par page et de caractères par ligne sont mémorisées à part; tous les autres codes sont repris dans le format donné ci-dessus. La seule différence réside dans le fait que le nombre de caractères de commande vient s'insérer entre chaque code et son signe de commande. Lorsque notre programme doit sortir un signe, il appelle le sous-programme sap (search and print) qui recherche dans le tableau le signe transmis dans D0. Ce dernier une fois trouvé, l'ordinateur donne les codes de commande adéquats; s'il ne le trouve pas, il transmet l'octet D0 lui-même.

Bien sûr, ce programme n'est pas parfait. Mais c'est une base solide qui vous permettra de développer votre propre programme d'imprimante. Nous vous donnons encore quelques conseils: ce programme a bien meilleure allure en tant qu'accessoire doté d'une boîte de dialogue. Il faudrait de plus qu'il puisse numéroter les pages et faire figurer dans l'en-tête le nom du fichier. Il serait intéressant de pouvoir faire ressortir les mots clés du langage utilisé dans un listing de programmation. Vous voyez que vous pouvez encore faire jouer vos talents !

```
;
;      Print-utility      PRINT.S
;      MP 26-03-87
;
gemdos = 1
```

```

bios      = 13
xbios     = 14
keep      = $31
setdta    = $1a
open      = $3d
close     = $3e
sfirst    = $4e
print     = 9
l1ist     = 5
read      = $3f
supexec   = 38
kbshift   = 11
conin     = 7
conout    = 2
_dumpflag = $4ee
nvbls     = $454
_vblqueue = $456
etv_critic = $404

        .TEXT

        movea.l    4(sp),a0          ; calculer place-mémoire
                                   ; nécessaire

        move.l     #$100,d6
        add.l      12(a0),d6
        add.l      20(a0),d6
        add.l      28(a0),d6      ; utilisé plus loin

        bsr        readfile         ; charger le fichier des
                                   ; paramètres

        pea        init_vbl         ; initialisation en
        move.w     #supexec,-(sp)   ; mode superviseur
        trap       #xbios
        addq.l     #6,sp

        clr.w      -(sp)
        move.l     d6,-(sp)        ; garder le programme en
                                   ; résident

        move.w     #keep,-(sp)
        trap       #gemdos

init_vbl: move.w   nvbls,d0
        lsl.l     #2,d0            ; processus que vous
                                   ; connaissez

        movea.l    _vblqueue,a0    ; déjà bien
        moveq.l    #4,d1           ; laissez libre le

premier enregist.
vbl_loop: tst.l   0(a0,d1.w)
        beq.s     vbl_found
        addq.l    #4,d1

```

```

        cmp.w    d0,d1
        bne.s   vbl_loop

message !
        bra     error                ; plus rien de libre ?
vbl_found:
        move.l  #vbl,0(a0,d1.w)
quit:   rts

vbl:   tst.w    _dumpflag            ; ALT + HELP ?
        bmi.s   quit

        move.w  #-1,-(sp)            ; si oui, rechercher le
                                        statut des touches spéciales

        move.w  #kbshift,-(sp)
        trap    #bios
        addq.l  #4,sp
        andi.w  #2,d0                ; seule la touche
                                        <SHIFT> gauche
                                        est importante pour
                                        nous ici

        beq.s   quit                ; si elle n'a pas été
                                        appuyée...

        move.l  etv_critic,crt       ; Sauvegarder le
                                        critical error hdl

        move.l  #critic,etv_critic  ; sous une routine
                                        propre RTS

        subi.w  #1,_dumpflag        ; pas de copie-écran !

        movea.l $44e,a0              ; début de l'utilisation
                                        de l'écran

        lea.l   30720(a0),a0         ; début de l'utilisation
                                        de ligne 25

        lea.l   scr_save,a1          ; mémoire-intermédiaire

        move.w  #319,d0              ; 320 mots longs = 1

ligne de texte
save_loop:
        move.l  (a0),(a1)+           ; sauvegarder contenu de
                                        mémoire vive

        clr.l   (a0)+                ; puis la vider

        dbra   d0,save_loop
        pea   strr
        move.w #print,-(sp)
        trap  #gemdos
        addq.l #6,sp
        move.w #40,-(sp)             ; boucle de saisie du
                                        nom de fichier

        pea   string
        move.w #24,-(sp)
        move.w #27,-(sp)

```

```

bsr      rdstr
adda.l   #10,sp

pea      dta                ; créer
                        disc-transfer-address

move.w   #setdta,-(sp)
trap     #gemdos
addq.l   #6,sp

clr.w    -(sp)             ; attribut-fichier pour
                        search-first

pea      string
move.w   #sfirst,-(sp)
trap     #gemdos
addq.l   #8,sp
tst.w    d0                ; le fichier existe ?
bmi      not_found
move.l   taille,d6        ; taille en octets
subl.l   #1,d6            ; -1 pour la boucle
clr.w    -(sp)            ; lecture seule (open)
pea      string
move.w   #open,-(sp)
trap     #gemdos
addq.l   #8,sp
move.w   d0,handle

pea      pages_text       ; demande si impression
                        page par page

move.w   #print,-(sp)
trap     #gemdos
addq.l   #6,sp
bsr      oui_non          ; appel de la routine
move.w   d5,pageparpage  ; sauvegarder le résultat
beq.s    non_s

pea      title_text       ; nécessaire pour la
                        saisie de l'en-tête

move.w   #print,-(sp)
trap     #gemdos
addq.l   #6,sp

move.w   #40,-(sp)        ; boucle de saisie pour
                        l'en-tête

;
pea      title
move.w   #24,-(sp)
move.w   #30,-(sp)
bsr      rdstr
adda.l   #10,sp

non_s:   pea      ask_ln_text ; demande si
                        numérotation des lignes

```

```

    move.w    #print,-(sp)
    trap     #gemdos
    addq.l   #6,sp
    bsr     oui_non           ; résultat conservé dans
                             d5
    pea     printing
    move.w   #print,-(sp)
    trap     #gemdos
    addq.l   #6,sp

    moveq.l  #2,d0           ; String-Init du fichier
                             paramètres
    bsr     sap             ; à rechercher et
                             afficher
    moveq.l  #1,d7           ; Premier numéro de ligne
    clr.w    ligne_page

new_page:
    tst.w    pageparpage    ; En-tête ?
    beq.s   read_lp        ; si non, poursuivre
    cmpi.w  #1,d7          ; si oui: première page ?
    beq.s   no_ff          ; pas d'avancée de page
    moveq.l #4,d0          ; rechercher et afficher
    bsr     sap             ; code du form-feed

no_ff:     bsr     print_title ; imprimer en-tête
read_lp:   move.w  ligne_page,d0 ; nouvelle page ?
    cmp.w   lpp,d0
    bne.s   noff
    clr.w   ligne_page     ; remettre le compteur à
                             zéro

noff:     bra.s   new_page   ; et un saut de page
    tst.w   d5             ; numérotation de ligne ?
    beq.s   no_ln
    bsr     output_ln     ; si oui, afficher
                             numéro de ligne

no_ln:    pea     load_byte  ; charger un signe
    move.l  #1,-(sp)
    move.w  handle,-(sp)
    move.w  #read,-(sp)
    trap    #gemdos
    adda.l  #12,sp
    tst.l   d0             ; erreur ?
    bmi    read_error

    move.b  load_byte,d0   ; prendre un signe dans
                             le buffer
    bsr     sap           ; et l'afficher
    cmpi.b  #10,load_byte ; Fin de ligne ?
    bne.s   n_eol
    addq.w  #1,d7         ; si oui, augmenter
                             numéro de ligne

    addq.w  #1,ligne_page

```

```

        subi.l   #1,d6           ; décrémente le compteur
        bpl     read_lp        ; long-ersatz pour DBRA
n_eol:  bra.s    quit_prnt
        subi.l   #1,d6
        bpl     no_ln         ; si non, ne rien
                                afficher
quit_prnt:
        moveq.l #13,d0         ; retour-chariot
        bsr     sap
        moveq.l #10,d0        ; et line-feed
        bsr     sap
        moveq.l #3,d0         ; afficher le string exit
        bsr     sap
quit_vbl:
        move.w  handle,-(sp)   ; fermer fichier
        move.w  #close,-(sp)
        trap    #gemdos
        addq.l  #4,sp

        movea.l $44e,a0       ; reprendre la dernière
                                ligne
        lea.l   30720(a0),a0
        lea.l   scr_save,a1
        move.w  #319,d0
get_loop: move.l  (a1)+,(a0)+
        dbra   d0,get_loop
        move.l  crt,etv_critic ; critical error handle
                                normal
critic:  rts                  ; et not_found pour
                                terminer
not_found:
        pea    n_found_text   ; envoyer un message si
                                le fichier
err_ff:  move.w  #print,-(sp)  ; n'existe pas
        trap    #gemdos
        addq.l  #6,sp
key:    move.w  #conin,-(sp)   ; attendre l'action
                                d'une touche
        trap    #gemdos
        addq.l  #2,sp
        bra.s  quit_vbl
prt_error:
        pea    n_ready_text   ; imprimante pas encore
                                prête
        bra.s  err_ff
read_error:
        pea    flop_text      ; erreur de disquette
        bra.s  err_ff
output_ln:
        lea.l  ligne,a5       ; afficher numéro de
                                ligne

```

```

        move.b    #' ',5(a5)          ; initialiser string
                                           d'affichage
        move.b    #' ',6(a5)
        move.b    #' ',7(a5)
        move.b    #' ',8(a5)
        move.w    d7,d1              ; d1 comme variable
                                           auxiliaire
ln_loop: moveq.l  #4,d0              ; variable de boucle
        move.w    d1,d2
        andi.l    #$0000ffff,d2     ; réduire évtlmt à la
                                           taille d'un mot
        divu.w    #10,d2            ; division par la base
                                           du système
;                                     numérique choisi
        move.w    d2,d1              ; résultat entier
        swap.w    d2                 ; reste dans un autre mot
        addi.b    #'0',d2           ; former un chiffre et
                                           l'inscrire
        move.b    d2,0(a5,d0.w)     ; dans un string
        dbra     d0,ln_loop
        clr.w     d0                 ; effacer les zéros de
                                           tête
del_0_lp: cmpi.b  #'0',0(a5,d0.w)   ; le chiffre est un zéro ?
        bne.s     quit_del_0        ; si non, le nombre
                                           commence bien ici
        move.b    #' ',0(a5,d0.w)   ; si oui, effacer
        addq.w    #1,d0              ; et passer au suivant
        bra.s     del_0_lp
quit_del_0:
        clr.w     d4                 ; afficher numéro de
                                           ligne (string)
prt_ln_lp:
        move.b    0(a5,d4.w),d0
        bsr      sap
        addq.w    #1,d4
        cmpi.w    #9,d4              ; tous les signes
                                           affichés ?
        bne.s     prt_ln_lp
        rts                          ; si oui, nous avons
                                           terminé
; routine de recherche-affichage, signe dans d0
sap:    lea.l     codesimprim,a3     ; search and print
                                           character
s_loop: tst.b     (a3)                ; Zéro ?
        beq.s     quit_search        ; si oui, fin du tableau
        cmp.b     (a3),d0            ; l'octet à afficher se
                                           trouve dans d0
        beq.s     s_found            ; trouvé ?
        move.b    1(a3),d1          ; si non, sauter au
                                           suivant

```

```

    andi.w    #$ff,d1          ; élargir dl à 16 bit
    lea.l    2(a3,d1.w),a3
    bra.s    s_loop          ; et poursuivre la
                             recherche
quit_search:
    andi.w    #$ff,d0          ; si pas trouvé,
                             afficher tel quel
    move.w    d0,-(sp)
    move.w    #l1ist,-(sp)
    trap     #gemdos
    addq.l    #4,sp
    tst.l     d0              ; erreur d'impression ?
    beq      prt_error       ; la signaler et
                             interrompre
empty:      rts
s_found:    move.b    1(a3),d2 ; porter le nombre de
                             signes
    andi.w    #$ff,d2        ; à 16 bits
;           ; (sans signe placé
;           ; devant)
    subi.w    #1,d2          ; -1 pour dbra
    bmi.s    empty          ; pas de signe ? refouler
    addq.l    #2,a3          ; pointeur sur le
                             premier code de
prt_lp:     move.b    (a3)+,d0 ; commande puis imprimer
    andi.w    #$ff,d0
    move.w    d0,-(sp)
    move.w    #l1ist,-(sp)
    trap     #gemdos
    addq.l    #4,sp
    tst.l     d0              ; erreur ?
    beq      prt_error
    dbra     d2,prt_lp
    rts
;
oui_non:    clr.w     d5          ; pré-réglage : non
line_no:    move.w    #conin,-(sp) ; guetteur de touche
    trap     #gemdos
    addq.l    #2,sp
    cmpi.w    #'Z',d0          ; en majuscule ?
    ble.s    gross
    subi.w    #'a'-'A',d0      ; si non, nous en
                             produisons une
gross:      cmpi.w    #'N',d0
    beq.s    end_oui_non
    cmpi.w    #'O',d0
    bne.s    line_no
    moveq.l   #1,d5          ; si 'O' numéro de ligne
end_oui_non:
    rts          ; résultat dans d5
                (booléen)

```

```

; Affichage centré de l'en-tête
print_title:
    move.w    cpl,d3                ; lecture de la largeur
                                   ; de la ligne
    clr.w    d0                    ; rechercher la taille
                                   ; du string du titre
    lea.l    title,a1
1_lp:      tst.b    0(a1,d0.w)      ; fin du string ?
    beq.s    end_found
    addq.l   #1,d0                 ; si non, signe suivant
    bra.s    l_lp
end_found:
    sub.w    d0,d3                ; diviser par deux
    lsr.w    #1,d3                ; la différence
12_lp:     moveq.l #',',d0        ; et afficher des
                                   ; espaces vides
    bsr     sap
    dbra    d3,12_lp
13_lp:     move.b  (a1)+,d0        ; afficher le titre
    beq.s   14_lp                ; fin ? alors sortir de
                                   ; là
    bsr     sap                  ; sinon, afficher et
                                   ; passer
14_lp:     bra.s   13_lp          ; au signe suivant
    moveq.l #13,d0                ; ligne suivante...
    bsr     sap
    moveq.l #10,d0
    bsr     sap
    move.w  cpl,d3
    subi.w  #2,d3
15_lp:     moveq.l #'-',d0
    bsr     sap
    dbra    d3,15_lp
    moveq.l #13,d0                ; encore un saut de
                                   ; ligne...
    bsr     sap
    moveq.l #10,d0
    bsr     sap
    moveq.l #10,d0                ; on ne conserve pas d0
    bsr     sap
    rts
; Chargement du fichier de paramètres "PRINT.PAR"
readfile:  move.w  #-1,valid       ; Vrai, signe suivant
                                   ; valable
    move.w  #2,-(sp)              ; ouvrir le fichier pour
                                   ; le lire
    move.l  #filename,-(sp)
    move.w  #open,-(sp)
    trap   #gemdos
    addq.l  #8,sp
    tst.w  d0                     ; une erreur ?

```

```

    bmi      par_error
    move.w   d0,handle
    bsr      readbyte      ; chercher nombre de
                           lignes par page

    clr.b    lpp
    move.b   d0,lpp+1
    bsr      readbyte      ; nombre de caractères
                           par ligne

    clr.b    cpl
    move.b   d0,cpl+1
    lea.l    codesimprim,a3
    bsr      readbyte      ; octet suivant d0
bigloop:   move.b   d0,(a3) ; à mettre dans notre
                           liste
    addq.l   #1,d6         ; 1 oct. place-mémoire
                           pour longueur
    tst.b    d0            ; fin du fichier ?
    beq.s    finish        ; si oui, interrompre la
                           lecture
    clr.w    d3            ; initialiser compteur
                           des codes de
innerloop:
    clr.w    cr            ; commande: pas de fin
                           de ligne
    bsr      readbyte      ; prendre l'octet
    tst.w    cr            ; fin de ligne avant cet
                           octet ?

    beq.s    no_eol
    move.b   d3,1(a3)      ; si oui, sauvegarder la
                           longueur
;                               du string de commande
no_eol:   lea.l   2(a3,d3.w),a3 ; Saisie suivante s.v.p.
    bra.s    bigloop
    move.b   d0,2(a3,d3.w) ; archiver code de
                           commande
    addq.l   #1,d3         ; augmenter index des
                           codes de com.

    bra.s    innerloop
finish:   move.w   handle,-(sp) ; fermer le fichier
    move.w   #close,-(sp)
    trap     #gemdos
    addq.l   #4,sp
readbyte: bsr      readchar ; lire un signe du
                           fichier
HEX-digit? cmpi.b   #'0',d0 ; signe conforme en
    blt.s    nohex
    cmpi.b   #'9',d0
    ble.s    hex
    cmpi.b   #'A',d0

```

```

        blt.s      nohex
        cmpi.b    #'F',d0
        bgt.s     nohex
hex:    tst.w      valid                ; accepter le signe ?
        beq.s     readbyte           ; si non, l'ignorer
        bsr.s     to_bin             ; transformer en binaire
        move.b    d0,d2              ; high-nibble
        asl.b     #4,d2              ; et * 16
        bsr      readchar           ; lire low-digit
        bsr.s     to_bin
        add.b     d0,d2              ; additionner
        move.w    d2,d0
        addq.l    #1,d6              ; réserver 1 octet-mémoire de plus

nohex:  rts
        cmpi.b    #';',d0            ; début de commentaire ?
        beq.s     com
        cmpi.b    #13,d0            ; fin de ligne ?
        bne.s     readbyte           ; Non
        move.w    #-1,valid         ; Sinon Valid:= true
        move.w    #-1,cr
com:    bra.s     readbyte
        clr.w     valid              ; si commentaire, valid
                                         := faux

to_bin: bra.s     readbyte
        subi.b    #'0',d0           ; transformation ASCII
                                         --> BIN

        cmpi.b    #9,d0
        ble.s     label
        subq.b    #7,d0
label:  rts
readchar: pea    buffer              ; lire un signe
        move.l    #1,-(sp)
        move.w    handle,-(sp)
        move.w    #read,-(sp)
        trap      #gemdos
        adda.l    #12,sp
        move.b    buffer,d0
error:  pea      vbl_text             ; plus de saisie VBL
                                         libre

err_cont: move.w  #print,-(sp)
        trap      #gemdos
        addq.l    #6,sp
        move.w    #conin,-(sp)      ; attendre action d'une
                                         touche

        trap      #gemdos
        addq.l    #2,sp
        clr.w     -(sp)             ; fin du programme
                                         (Pterm)
        trap      #gemdos           ; (pas en résident)

```

```

par_error:
    pea      par_text      ; le fichier de
                        paramètres manque
    bra.s    err_cont
;
    .PATH 'H:\'
    .INCLUDE 'INPUT.IS'

    .DATA
    .EVEN

filename: .DC.b 'PRINT.PAR',0
strr:     .DC.b 27,'Y',56,32,27,'p Print-Utility ',27
          .DC.b 'q Fichier : ',0
n_found_text:
    .DC.b 27,'Y',56,48,27,'K Fichier inexistant '
    .DC.b '!!!',0
title_text:
    .DC.b 27,'Y',56,48,27,'K En-tête:',0
n_ready_text:
    .DC.b 27,'Y',56,48,27,'K Votre imprimante a des problèmes '
    .DC.b '(Off Line ?)',0
ask_ln_text:
    .DC.b 27,'Y',56,48,27,'K Désirez-vous numérotter les lignes ?'
    .DC.b '(O/N)',0
pages_text:
    .DC.b 27,'Y',56,48,27,'K Désirez-vous imprimer page par page ?'
    .DC.b '(O/N)',0
flop_text:
    .DC.b 27,'Y',56,48,27,'K Votre disquette a des problèmes',0
printing:
    .DC.b 27,'Y',56,48,27,'K Patientez, impression...',0
vbl_text: .DC.b 27,'E',10,' je ne parviens pas à entrer dans '
          .DC.b ' VBL-Queue - TOUCHE',0
par_text:
    .DC.b 27,'E',10,'il manque le fichier paramètres PRINT.PAR! ',0
    .BSS
    .EVEN

handle:   .DS.w 1 ; GEMDOS file handle
valid:    .DS.w 1 ; booléen: les signes du fichier
;         sont-ils corrects ?
cr:       .DS.w 1 ; booléen: le dernier signe est-il
;         un retour-charriot ?
buffer:   .DS.w 1 ; mémoire intermédiaire pour les
;         signes tirés du fichier
load_byte:
    .DS.b 1
string:   .DS.b 41
title:    .DS.b 41
ligne:    .DS.b 9 ; préparation des numéros

```

```

;                                de ligne

cpl:      .DS.w 1  ; caractères par ligne
lpp:      .DS.w 1  ; lignes par page
pageparpage:
.DS.w 1
ligne_page:
        .DS.w 1  ; compte les lignes sur la page en cours
crt:      .DS.l 1  ; critical error handler
scr_save: .DS.b 1280                ; place pour une ligne
                                         de texte
                                         (1280 x 25 = 32000)
;
dta:      .DS.b 21  ; on inscrit ici diverses données
attribut: .DS.b 1   ; concernant le fichier à
horloge:  .DS.b 2   ; imprimer
datum:    .DS.b 2
taille:   .DS.b 4   ; important: taille en octets
name:     .DS.b 14
codesimprim:
        .DS.b 7000
; on charge ici les codes de commande
; pour l'imprimante
        .END

```

5.14. Pour recevoir des messages d'erreur

à la place des bombes

Lorsqu'un programmeur habitué à utiliser un langage évolué prend contact pour la première fois avec l'Assembleur, il éprouve une gêne majeure en constatant l'absence des messages d'erreur courants. En effet, en Assembleur, un programme peut être syntaxiquement correct tout en contenant des fautes de logique qui n'apparaîtront qu'en faisant tourner le programme. Par exemple, le programme peut tenter d'accéder à des variables-systèmes sans avoir auparavant activé le superviseur. Il peut aussi arriver que le processeur tente de traiter une chaîne comme s'il s'agissait d'un programme, ce qui n'aurait aucun sens.

Le programmeur en Assembleur ne recevra que des bombes pour lui signaler ses erreurs. Vous avez peut-être remarqué que le nombre de bombes peut varier.

Pour comprendre le rapport qui existe entre le nombre de bombes et l'erreur ainsi signalée, vous devez savoir qu'il existe au début de la mémoire principale une liste de 256 vecteurs correspondant à des adresses. Ces vecteurs sont numérotés de 0 à

255. Une partie de ces vecteurs sert au traitement des erreurs. Par exemple, lorsque le processeur rencontre un opcode inconnu ou une commande illégale, il "sait" qu'il doit interrompre le programme en cours pour passer à l'adresse se trouvant dans le vecteur numéro 4. En résumé: chaque type d'erreur est relié directement à un numéro de vecteur.

Quant au contenu de ces vecteurs (plus précisément: quant aux routines sur lesquelles ils sont pointés), il dépend entièrement du système d'exploitation. Dans l'Atari, tous les vecteurs chargés du traitement des erreurs sont dirigés vers l'affichage de bombes. C'est là qu'est décidé le nombre de bombes à afficher selon le type d'erreur commise. Il est mis fin au programme en cours par la fonction GEMDOS Term (numéro 0).

Examinons de plus près les erreurs les plus fréquentes en précisant le numéro du vecteur concerné:

Vecteur 2 **erreur de bus**
 Vous avez tenté d'accéder à une adresse qui n'existe pas (=il n'y a ni mémoire, ni périphérique) ou à laquelle vous ne pouvez accéder qu'en mode superviseur (par exemple, les variables-systèmes).

Vecteur 3 **erreur d'adresse**
 Vous avez tenté d'accéder à une adresse impaire par une opération de mot ou de mot long.

Vecteur 4 **commande illégale**
 Le processeur rencontre un opcode inconnu; on peut aussi provoquer cette erreur à l'aide de la commande "illegal".

Vecteur 5 **division par zéro**

Vecteur 8 **atteinte à un privilège**
 Vous avez tenté de faire exécuter une commande machine qui n'est tolérée qu'en mode superviseur (exemple: rte). Cette erreur n'apparaît pas lorsque vous accédez à des domaines protégés de la mémoire, car on utilise alors le vecteur 2, erreur de bus.

Ces quelques indications vous permettent au moins de vous faire une idée sur l'origine de l'erreur et de la localiser plus facilement. Il serait tout de même plus commode de recevoir, à la place des bombes, des messages en clair. C'est le but du programme ci-dessous.

```

;
; Messages d'erreur à la place des bombes ANTIBOMB.S
;      MP      09-06-88
;
gemdos      = 1
xbios      = 14
print      = 9
keep       = $31
superexec  = 38

        .TEXT
movea.l    4(sp),a0          ; place-mémoire
                                nécessaire

move.l     #$100,d6
add.l     12(a0),d6
add.l     20(a0),d6
add.l     28(a0),d6
pea       init              ; initialisation dans le
                                superviseur

move.w     #superexec,-(sp)
trap      #xbios
addq.l    #6,sp
clr.w     -(sp)             ; garder le programme en
                                mémoire

move.l     d6,-(sp)
move.w     #keep,-(sp)
trap      #gemdos
init:     lea.l    erreur,a1      ; début du traitement
                                des erreurs

adda.l    #$2000000,a1        ; vecteur dans l'octet
                                le plus élevé

lea.l     $8,a0              ; erreur de bus: début
moveq.l   #6,d0              ; installer 7 vecteurs

init_loop:
move.l    a1,(a0)+           ; écrire l'adresse dans
                                le vecteur

adda.l    #$1000000,a1       ; augmenter de 1 le
                                vecteur

dbra     d0,init_loop
rts

erreur:   bsr      simule      ; pas de sous-programme,
                                positionne
                                ; PC sur le stack
nop
simule:   move.l   (sp)+,$3c4   ; PC y compris numéro de
                                vecteur

clr.l    d0
move.b   $3c4,d0              ; numéro de vecteur
subi.w   #2,d0                ; moins 2

```

```

        asl.w      #2,d0                ; multiplié par 4 =
                                        ; offset pour
                                        ; le pointeur du string
        lea.l      tableau,a0
        move.l     0(a0,d0.w),-(sp)    ; pointeur sur message
                                        ; d'erreur
        move.w     #print,-(sp)
        trap      #gemdos
        addq.l     #6,sp
        moveq.l    #30,d0              ; boucle d'attente
wait1:  moveq.l    #-1,d1
wait2:  dbra      d1,wait2
        dbra      d0,wait1
        move.w     #1,-(sp)            ; annonce d'erreur
        clr.l     -(sp)
        trap      #gemdos

        .DATA
        .EVEN

tableau: .DC.l bus,adresse,illegl,division
bus:     .DC.b 27,'l erreur Bus - adresse fausse ou '
        .DC.b 'protégée',0
adresse: .DC.b 27,'l erreur Adresse - Seules sont autorisées '
        .DC.b ' les adresses paires',0
illegl:  .DC.b 27,'l erreur Illégale - code inconnu',0
division: .DC.b 27,'l Division par zéro',0
check:   .DC.b 27,'l erreur - CHK - débordement du'
        .DC.b 'registre',0
vtrap:   .DC.b 27,'l erreur - TRAPV - TRAP on overflow',0
privileg: .DC.b 27,'l erreur de privilège - cette commande '
        .DC.b 'réclame le superviseur',0
        .END

```

Il est évident que le programme doit être résident, en mémoire centrale. L'initialisation se limite à orienter les vecteurs d'erreurs sur une routine qui remplace l'affichage des bombes. Nous verrons plus loin pourquoi le numéro de vecteur se place dans l'octet le plus élevé de chaque vecteur. Les vecteurs 6 et 7 ne sont pas vraiment des 'erreurs' mais ils se trouvent entre les vecteurs que nous avons retenus et sont donc traités en même temps que les autres pour éviter de compliquer le programme.

Lorsqu'une erreur survient, le processeur passe automatiquement à la routine d'erreur. Le programme doit rechercher de quel vecteur provient l'appel. Nous nous servons pour cela d'une petite astuce: nous avons déjà signalé ci-dessus que dans chaque vecteur, l'octet le plus élevé contient le numéro de vecteur. Par exemple, lorsque survient un code illégal (vecteur numéro 4) et que le traitement

de cette erreur doit commencer à l'adresse \$12345, le vecteur de l'instruction "illegal" prend la valeur \$04012345, qui est un nombre hexadécimal (huit rangs). Cette valeur est inscrite dans le compteur. Mais comme ce compteur du processeur ne possède que 24 bits vers l'extérieur (interne: 32 bits), il ne tient pas compte du 4, il fait comme si ce chiffre n'existait pas. Pour accéder à ce 4, nous devrions pouvoir accéder directement au compteur, ce qui est malheureusement impossible, mais si nous appelons un sous-programme, le compteur est sauvegardé dans le stack. Nous simulons donc l'appel d'un sous-programme dans lequel nous nous bornons à appeler un mot long se trouvant dans le stack, qui n'est rien d'autre que la valeur du compteur que nous venons de sauvegarder, et qui contient le numéro de vecteur dans son octet le plus élevé. Une astuce raffinée, non? Pour rester fair-play, nous devons avouer que l'idée n'est pas de nous: c'est l'astuce utilisée dans la routine originale du système d'exploitation des Atari qui appelle les bombes.

Le reste ne pose plus de problème. Nous préparons un string pour chaque numéro de vecteur. Ces strings commencent tous par la séquence ESC 1 du VT-52 qui efface la ligne du curseur actuel et positionne le curseur à l'extrême gauche. Après l'affichage des strings, le programme attend encore environ 4 secondes pour vous laisser le temps de bien lire le texte du message. Comme d'habitude, ce programme prend fin grâce à la fonction GEMDOS Term (numéro 0). Aucun document ne mentionne le retour d'un code d'erreur lié à cette fonction, mais comme la routine d'origine affichant des bombes se sert de ce procédé, nous faisons de même.

5.15. Pour passer un moniteur couleur de 50 à 60 Herz

Vous avez certainement remarqué que votre moniteur couleur ou votre poste de télévision scintillent bien plus que votre écran monochrome. Ce phénomène provient de la fréquence de retour de l'image qui est plus basse, puisqu'elle est de 50 contre 71 Hz: un écran couleur produit 50 images par seconde, pendant qu'un moniteur monochrome en produit 71. En Amérique, le courant électrique usuel est généralement distribué à 60 Hz, si bien que votre Atari est tout à fait capable d'utiliser une fréquence de 60 Hz pour son écran. Cette fréquence est d'ailleurs supportée par la plupart des moniteurs ou des téléviseurs vendus en Europe. Pour savoir si votre écran supporte cette fréquence, lancez le programme ci-dessous: si l'image se met à défiler sans arrêt, c'est que votre moniteur ne peut la synchroniser.

La représentation de l'image, que ce soit en couleur ou en noir et blanc, se fait dans l'Atari grâce au "shifter". Dans l'un de ses registres se trouve un bit qui sert à passer de 50 à 60 Hz:

```
Adresse:$FF820A (16744970) (Octet)
```

Sous cette adresse se trouvent les deux bits qui nous concernent. Le bit 0 est utilisé pour la sauvegarde d'écran et sert à activer au choix la synchronisation externe ou interne. Le deuxième, le bit 1, nous intéresse ici plus particulièrement:

```
Bit 0  0 = synchronisation interne
        1 = synchronisation externe
Bit 1  0 = 60 Hz
        1 = 50 Hz
```

Nous allons écrire notre programme de façon à ce que la fréquence change à chaque démarrage. Nous pouvons écrire:

```
Basic-GFA: Spoke &HFF820A, (Peek(&HFF820A) Xor 2)

Langage C: char *pointeur;
           long save_sp;
           save_sp=gemdos(0x20,0L);
           pointeur = 0xFF820A;
           *pointeur = *pointeur ^ 2;
           gemdos(0x20,save_sp);

Assembleur: clr.l    -(sp)
            move.w   #$20,-(sp)
            trap     #1
            addq.l   #6,sp
            bchg     #1,$FF820A
            move.l   D0,-(sp)
            move.w   #$20,-(sp)
            trap     #1
            addq.l   #6,sp
```

Vous constatez que le processeur doit travailler en mode superviseur pour pouvoir accéder au registre du "shifter". L'accent circonflexe utilisé en langage C ne signifie pas une élévation de puissance (qui n'existe pas en C): il symbolise un "ou exclusif".

```
*****
*   passage de 50 à 60 Hz   *
*****
            clr.l    -(sp)           * mode superviseur
            move.w   #$20,-(sp)
            trap     #1

            addq.l   #6,sp
```

```
bchg    #1,$FF820A    * basculer le bit de 50 à 60 Hz
move.l  D0,-(sp)      * mode user
move.w  #$20,-(sp)
trap    #1

addq.l  #6,sp
clr.w   -(sp)        * fin du programme
trap    #1
```

Chapitre 6

Trucs et astuces concernant le GEM

La caractéristique la plus spectaculaire des ordinateurs équipés d'un processeur 68000 est bien l'usage de fenêtres, boîtes de dialogue et du bureau GEM. Par contre, si l'utilisateur veut se servir lui-même de ces éléments dans ses programmes (surtout s'il s'agit d'un débutant) il constate qu'il lui faut utiliser un nombre impressionnant de commandes et de paramètres. C'est pour faciliter cet apprentissage que nous avons écrit ce chapitre, illustré de nombreux exemples de programmation.

6.1. Les jeux de caractères du GDOS et du GEM

Connaissez-vous le GDOS? A quoi sert donc le fichier Assign.sys? Ces deux questions sont étroitement liées, et nous allons donc les expliquer ensemble.

Tout commença au temps de l'élaboration du GEM que vous connaissez bien, le "Graphics Environment Manager". Ce système ne fut pas conçu à l'origine pour l'Atari, mais plutôt pour les PC et compatibles. Le but original était de créer une collection de routines permettant une programmation unifiée surtout au niveau graphique. C'est-à-dire qu'un programme tournant sous GEM devait sans aucune modification pouvoir tourner sur n'importe quel autre matériel, même muni d'une carte graphique différente.

C'est la raison pour laquelle le système graphique du GEM est divisé en deux parties: l'une dont les routines sont indépendantes du type de matériel utilisé et l'autre dont les routines dépendent de l'appareil utilisé. Les premières sont appelées depuis un logiciel, et constituent le VDI (Virtual Device Interface): pour exécuter leur travail graphique à proprement dit, elles sont cependant obligées de

recourir aux fonctions dépendantes du matériel adaptées à un ordinateur précis. Dans la pratique, cela signifie qu'un programme peut tourner sur deux ordinateurs différents grâce à ces routines VDI (indépendantes du type de matériel) qui sont unifiées, et qui se chargent d'appeler les routines propres à tel ou tel matériel. Ces routines universelles s'appellent d'ailleurs GIOS (= Graphics Input/Output System).

Il existe donc pour toute carte graphique, ainsi que pour les divers périphériques (imprimante etc), un GIOS particulier (nous sommes toujours en plein domaine PC !) se chargeant d'exploiter les possibilités offertes. La liaison entre ce VDI et le GIOS est assurée par le GDOS, le Graphics Device Operating System. Le GDOS permet de recharger depuis une disquette les drivers nécessaires dans le GIOS.

La situation était sensiblement différente lorsqu'on pensa à implémenter le GEM dans l'Atari. Comme le ST est le seul 68000 offrant le GEM et comme il ne dispose que de trois modes graphiques très semblables, on pouvait s'épargner les drivers GIOS conçus pour diverses cartes graphiques. Si bien que l'on put tout aussi bien se passer du GDOS, car là où les drivers n'existent pas, il est impossible d'en charger depuis une mémoire de masse.

L'ensemble GDOS/GIOS n'est rien d'autre que le programme GDOS.PRG mentionné ci-dessus (il existe aussi une version nommée GDOS2.PRG, dans laquelle on a supprimé une petite erreur de programmation) et flanqué des drivers comme FX80.SYS ou META.SYS. Quel est le rôle de ce GDOS? Il faut d'abord savoir que le GDOS ne peut être lancé qu'en le plaçant dans le dossier AUTO d'une disquette. Le programme reste résident après un reset. Pour fonctionner, il doit lire un fichier se trouvant dans le répertoire principal: ASSIGN.SYS (qui ne doit pas figurer dans le dossier AUTO). Ce fichier contient les noms des drivers disponibles dans GIOS (*.SYS; ASSIGN.SYS n'est pas en lui-même un driver) ainsi que les noms des jeux de caractères mis à la disposition des divers périphériques (*.FNT). Le GDOS n'entre en action que lorsqu'un logiciel, par exemple EASY-DRAW, interpelle un driver ou un nouveau jeu de caractères. Le fichier nécessaire, dont le nom doit figurer dans ASSIGN.SYS, est alors chargé automatiquement.

Après vous avoir asséné ces longues considérations théoriques assez arides, nous allons tenter de nous faire pardonner en vous disant tout de suite pourquoi nous avons besoin de ce GDOS, que vous devez bien posséder quelque part dans vos collections de programmes. Relisez bien le titre de ce paragraphe, vous verrez qu'il y était question d'un jeu de caractères... En effet, nous voulons embellir nos programmes en utilisant de nouveaux jeux de caractères.

Attention, pour les expérimentations que nous vous proposons ci-après, vous devez absolument disposer des fichiers suivants: le programme GDOS.PRG ou GDOS2.PRG, un fichier ASSIGN.SYS, les jeux de caractères IBMHSS10.FNT, IBMHSS14.FNT, IBMHSS18.FNT et IBMHSS36.FNT. Commencez par formater une disquette pour y recopier très exactement ces fichiers, en plaçant le GDOS.PRG (ou GDOS2.PRG) dans un dossier AUTO. Recopiez alors sur cette disquette le programme DEMOFONT.PRG que vous trouvez sur la disquette jointe à ce livre et dont nous vous réécrivons le texte ci-dessous. Réinitialisez votre configuration à l'aide de la nouvelle disquette confectionnée. Après un délai très court, vous devez recevoir la ligne "GEMVDI installed", qui confirme la bonne installation du GDOS. Ensuite, et seulement ensuite, vous pouvez lancer le programme DEMOFONT.PRG. Votre écran devrait se trouver en résolution haute ou moyenne.

Voici le texte de ce programme. Quelques unes des fonctions utilisées vous seront peut-être inconnues, nous vous les expliquons juste après.

```

/*****
/* Démonstration de fontes      DEMOFONT.C      */
/* MP 19-02-88      (seulement avec GDOS) */
/*****

int contrl[12],
  intin [128]
  ptsin[128],
  intout[128],
  ptsout[128],
  handle,
  dummy,
  work_in[12],
  work_out[57],
  pxyarray[10],
  points[]=          /* tailles des strings */
  {10, 14, 18, 24, 36, 72},
  y[] =             /* coordonnées pour l'affichage */
  {15, 30, 55, 85, 130, 199}; /* des textes */

main ()
{
  int chargees;
  register int i;

  appl_init ();
  for(i = 0; i < 10; work_in[i++] = 1);
  work_in[10] = 2;
  handle = graf_handle (&dummy, &dummy, &dummy, &dummy);
  v_opnvwk (work_in, &handle, work_out);

```

```

v_clrwk (handle);
appl_exit (); /* vider l'écran */
chargees=vst_load_fonts (handle, 1); /* charger les fontes */
printf("fontes disponibles=%d\n",chargees);
appl_exit ();
vst_font (handle, 2); /* et les activer */
vst_effects (handle, 0); /* pas d'effets texte */
vst_alignment (handle, 1, 0, &dummy, &dummy); /* affichage
                                                    centré */

for (i=0; i<6; i++)
{
  vst_point (handle, points[i], &dummy, &dummy, &dummy, &dummy);
  v_gtext (handle, 320, y[i], "Démo de fontes");
}
gemdos (0x1); /* attendre action d'une touche */
vst_unload_fonts (handle, 1); /* désactiver les fontes */
v_clsvwk (handle);
appl_exit ();
}

```

Venons-en aux explications concernant les fonctions VDI nous permettant de travailler avec les nouveaux jeux de caractères. Ce sont vraiment des fonctions, qui renvoient effectivement une valeur, dont nous ne tenons cependant pas compte, comme vous le constatez en lisant le texte du programme. En effet, nous y appelons directement les routines sans les faire précéder d'un signe "=". Ceci empêche certes le dépistage d'éventuelles erreurs, mais ce programme doit surtout servir d'outil de démonstration. Si vous voulez disposer d'une description approfondie des fonctions, reportez-vous à un des nombreux ouvrages consacrés au GEM.

Peut-être pourrez-vous vous satisfaire des quelques explications ci-dessous:

```
vst_load_fonts (handle, select);
```

Cette fonction sert à charger les jeux de caractères consignés dans le fichier ASSIGN.SYS avant l'initialisation de l'ordinateur, et les met à la disposition du périphérique concerné (habituellement, il s'agit de l'écran) sous l'identificateur handle. "select" est réservé pour les versions futures du GEM et doit se trouver sur 1.

```
vst_font (handle, font);
```

Ceci permet d'activer les jeux de caractères chargés par la fonction vst_load_fonts. "font" indique les numéros du jeu de caractères (NdT: en imprimerie, on parle de la 'fonture') désiré. La fonte du système d'exploitation, qui est toujours présente

dans l'ordinateur, porte le numéro 1, la fonte chargée juste après porte le numéro 2 etc. Attention: plusieurs fontes ne se distinguant que par la taille (plus ou moins grande) de caractères de même type ne constituent en fait qu'un seul jeu de caractères, même si sur votre disquette elles se trouvent dans des fichiers différents. La taille des caractères se règle à l'aide de la fonction

```
vst_point (handle, point, &d, &d, &d, &d);
```

"point" représente la taille d'un carré contenant le caractère, taille mesurée en points. Le point est ici une unité de mesure empruntée à l'imprimerie; un point = 1/72 de pouce soit environ 1/28 mm. L'ordinateur choisit automatiquement la meilleure taille standard chargée (dans notre exemple 10, 14, 18, 36 points) pour agrandir à la taille désirée les caractères d'une des fontes. Attention: on ne peut pas donner à 'point' n'importe quelle valeur, car on risquerait d'obtenir des résultats absurdes. On attribue certes une valeur à "&d", mais ceci ne nous intéresse pas ici (d'où le "d" comme "dummy").

```
vst_effects (handle, effect);
```

Cette fonction permet d'engendrer certains effets de présentation des caractères:

0	normal
1	gras
2	maigre
4	italique
8	souligné
16	contours (outlined)
32	spéciaux

On peut combiner ces effets, il suffit d'additionner les valeurs concernées et de transmettre la somme par "effect".

```
vst_alignment (handle, hor, ver, &d, &d);
```

L'affichage d'un texte sur un écran graphique pose toujours un problème bien particulier: lorsqu'on indique la position du début de l'affichage sous la forme de coordonnées, on ne sait jamais précisément quel est le point du string qui doit prendre la position en question: s'agit-il du point gauche en bas ou du point gauche en haut, ou même d'un point situé en plein milieu de la chaîne? Il faut désigner un point... de repère dans la chaîne, dont on précise ensuite les coordonnées exactes. C'est là le rôle de la fonction ci-dessus, dans laquelle "hor" indique la place dans le champ horizontal:

0	l'abscisse x désigne le début de la chaîne
1	l'abscisse x désigne le milieu de la chaîne
2	l'abscisse x désigne la fin de la chaîne

Nous donnons à "hor" la valeur 1, afin que tous les strings, en résolution basse ou haute, s'affichent au centre de l'écran si on leur donne une abscisse x de 320. Le paramètre "var" vous offre le choix entre six lignes de références différentes:

0	l'ordonnée y désigne le bord inférieur du caractère (sans la boucle évtl)
1	l'ordonnée y désigne le bord supérieur d'une minuscule
2	l'ordonnée y désigne le bord supérieur d'une majuscule
3	l'ordonnée y désigne le bord inférieur du cadre entourant un caractère
4	l'ordonnée y désigne le bord inférieur d'une boucle sous la ligne (g,y...)
5	l'ordonnée y désigne le bord supérieur du cadre entourant un caractère

v_gtext (handle, x, y, string);

Cette fonction sert enfin à afficher effectivement un texte, en tenant compte de toutes les options sélectionnées auparavant.

Que faisons-nous maintenant de toutes ces fontes? Vous pouvez d'abord vous en servir pour les intitulés dans vos programmes ou pour l'affichage courant dans vos fenêtres; les grands caractères font toujours leur effet dans les programmes graphiques, et si vous vous procurez l'interface GENLOCK, vous pourrez créer de beaux effets vidéo etc. etc. etc...

6.2. Pour confectionner votre propre bureau GEM

Lorsqu'on fait soi-même ses propres programmes, on a toujours tendance à les comparer avec ceux des professionnels. On constate alors souvent que ces programmes possèdent leur propre bureau, leur propre Desktop. Pourquoi ne pas tenter nous aussi de doter nos programmes de leur propre bureau? Le bureau a l'avantage d'être géré automatiquement par le GEM. Par exemple, si on représente sur un desktop des touches de fonction recouvertes dans le cours du programme par une fenêtre ou une boîte de dialogue, le GEM se charge du réaffichage automatique de ces touches une fois les fenêtres ou les boîtes refermées si vous avez pris soin de sauvegarder votre nouveau bureau. Avant d'en venir à la création

elle-même, nous voudrions d'abord vous donner quelques explications sur la façon de créer un bureau.

De par sa structure, le bureau n'est rien d'autre qu'un 'arbre d'objets graphiques' se composant d'un fond gris (en résolution haute) et de différents pictogrammes (icônes, touches de fonction etc). Ces objets graphiques sont intégrés dans le fond gris de type G-BOX. Comme il est impossible de donner dans RSC la taille exacte de ce fond, il faut par la suite l'adapter aux exigences du programme en cours. C'est là que des pointeurs se chargent de désigner les coordonnées correctes pour sa largeur et sa hauteur, une fois chargé le fichier ressource (Ressource-file) et précisée l'adresse de l'arbre concerné. Comme le bureau est une structure OBJECT définie dans le fichier header GEMDEFS.H du compilateur, nous pouvons écrire les coordonnées de la façon suivante (desktop contient l'adresse de l'objet):

```
desktop->ob_width=640
desktop->ob_height=400
```

Comme les autres objets sont des éléments de cet 'arbre d'objets' sauvegardé en tant que nouveau bureau GEM, le screen-manager va recevoir les informations nécessaires pour redessiner automatiquement ces objets graphiques lorsqu'ils auront été recouverts par une fenêtre ou une boîte de dialogue.

Venons en maintenant à la description détaillée de la création de notre nouveau bureau GEM. Après avoir créé un fichier ressource adéquat (vous trouverez un exemple de fichier ressource sur la disquette jointe à ce livre), il faut commencer par le charger à partir du programme. Puis on recherche l'adresse des objets qui doit être transmise à la routine `wind_set`, et qui est indispensable pour pouvoir sauvegarder un nouveau bureau GEM. On sauvegarde le nouveau bureau en appelant cette fonction avec le paramètre `WF_NEWDESK`. Il ne nous reste plus qu'à dessiner le nouveau bureau grâce à `objc_draw` pour disposer de notre propre bureau GEM géré automatiquement par le GEM.

Nous vous donnerons ensuite un petit programme pour vous montrer comment il faut procéder lors de la programmation. Vous trouverez sur la disquette jointe à ce livre le fichier header nécessaire ainsi que le fichier ressource.

Dans notre exemple, nous créons un bureau GEM avec deux icônes, des touches de fonction et une boîte de dialogue comme boîte d'informations. En appuyant sur la touche <F10>, vous ferez apparaître une boîte de dialogue recouvrant une partie du bureau. En cliquant sur la case "OK", vous ferez automatiquement réapparaître la partie cachée du bureau après avoir refermé la boîte de dialogue. Vous constatez

ainsi que le screen-manager a bien pris en charge la gestion du nouveau bureau. Pour mettre fin à ce programme, appuyez sur la touche < F1>. DESKTOP.C :

```
#include< stdio.h>
#include< gemdefs.h>
#include< osbind.h>
#include< obdefs.h>
#include< gembind.h>
#include"desktop.h"                /* charger le fichier header */

#define FALSE 0
#define TRUE -1
int vdi_handle;                    /* VDI-handle */
int contrl[11],                    /* VDI-Parameterarray */
    intin[256],
    intout[45],
    ptsin[256],
    ptsout[12];
OBJECT *desktop;

main()
{
    GRECT box;
    int taste;
    int annuler=FALSE;

    open_aes();
    rsrc_gaddr(R_TREE,FORM1,&desktop); /* recherche de
                                        l'adresse */
    desktop->ob_width=640;            /* coordonnées de la
                                        largeur */
    desktop->ob_height=400;           /* et de la hauteur à
                                        modifier */
    wind_set(0,WF_NEWDESK,desktop,0,0); /* dessiner et déclarer
                                        le */
                                        /* nouveau bureau */
    objc_draw(desktop,ROOT,MAX_DEPTH,desktop->ob_x,desktop->ob_y,
              desktop->ob_width,desktop->ob_height);
    do
    {
        taste=(evnt_keybd()/0x100); /* attendre action d'une
                                        touche */
        switch(taste)
        {
            case 0X44:                /* touche F10 appuyée */
                open_dialog();        /* ouvrir boîte de
                                        dialogue */
                annuler=FALSE;
                break;
        }
    }
}
```

```

        case 0x3B:                /* touche F1 appuyée */
            annuler=TRUE;        /* terminer le
                                programme */

            break;
        default:
            annuler=FALSE;
            break;
    }
}while(annuler!=TRUE);
aes_exit();                      /* sortir proprement du progr. */
}

int open_aes()
{
    extern int gl_apid;          /* AES Application-ID */
    int work_in[11],work_out[57]; /* Arrays pour ouvrir */
    register int loop;          /* Index des boucles */
    int dummy;                  /* Dummy-returnvariable */

    appl_init();                /* ouvrir AES */
    if(gl_apid==-1) exit(1);     /* si !AES Init ->exit */
    if(!rsrc_load("desktop.rsc")) exit(1); /* charger
                                        resource-file;
                                        sinon ->exit */

    vdi_handle=graf_handle(&dummy,&dummy,&dummy,&dummy);
                                /* rechercher handle des périphériques */
    for(loop=0;loop< 10;loop++)
        work_in[loop]=1; /* initialiser Work-array pour pouvoir
                            ouvrir 'virtual work' */

    work_in[10]=2;
        v_opnvwk(work_in,&vdi_handle,work_out);/* ouvre 'virtual
                                                work station */

    if(!vdi_handle) exit(1);
        graf_mouse(ARROW,0L); /* souris sous forme de flèche */
}

int aes_exit()
{
    v_clswnk(vdi_handle);        /* fermer virtual work station */
    rsrc_free();                 /* libère la place de resource */
    appl_exit();                 /* referme AES */
    exit(0);                     /* retourne au desktop */
}

int open_dialog()
{
    GRECT box;                   /* surface écran utilisateur */
    OBJECT *tree;                /* objets pour dialog */
    int button;                  /* index des boutons choisis */
}

```

```

    rsrc_gaddr(R_TREE,FORM2,&tree); /* adresse des dialogues */
    form_center(tree,&box.g_x,&box.g_y,&box.g_w,&box.g_h);
/* centrage de la boîte de dialogue sur l'écran et */
/* retour valeurs de surface utilisée */
    box_draw(tree,box.g_x,box.g_y,box.g_w,box.g_h);

    button=form_do(tree,0); /* attend sélection de 'exit' */
    tree[button].ob_state &= SELECTED; /* détruit le flag
/* 'sélectionné' dans l'arbre */
    box_undraw(box.g_x,box.g_y,box.g_w,box.g_h);
}

box_draw(tree,x,y,w,h) /* ouvrir la boîte de dialogue */
long tree;
int x,y,w,h;
{
    form_dial(FMD_START,0,0,0,0,x,y,w,h);
    form_dial(FMD_GROW,0,0,0,0,x,y,w,h);
    objc_draw(tree,ROOT,MAX_DEPTH,x,y,w,h);
}

box_undraw(x,y,w,h) /* fermer la boîte de dialogue */
int x,y,w,h;
{
    form_dial(FMD_SHRINK,0,0,0,0,x,y,w,h);
    form_dial(FMD_FINISH,0,0,0,0,x,y,w,h);
}

```

6.3. Un accessoire de recherche automatique

de fichier dans le disque dur

Ce programme doit sa création à l'amertume du possesseur de disque dur obligé de rechercher longuement dans son disque dur si bien rangé un ou plusieurs programme(s) portant la même extension. Cette recherche s'avère longue et fastidieuse. Admettons par exemple que nous ayons besoin de constituer un dossier contenant tous les fichiers portant l'extension .BAS. Il faudra ouvrir et refermer tous les dossiers et sous-dossiers un par un, voir s'ils contiennent un fichier à extension .BAS, qu'il faudra éventuellement recopier dans le dossier prévu à cet effet. La recherche s'avère être la plus longue partie du travail, c'est pourquoi nous décidons d'écrire un programme (sinon, pourquoi posséder un ordinateur?). Comme nous aimerions pouvoir procéder à une telle recherche de fichier même

lorsque nous nous trouvons dans un logiciel (traitement de texte, création graphique etc), nous avons fait un programme ayant le statut d'accessoire.

Ce programme doit nous permettre de rechercher n'importe quelle chaîne de caractères, y compris en prévoyant l'usage du masque ? et de la troncature *, dans l'ensemble du disque dur. Le résultat doit s'afficher dans une fenêtre et doit même pouvoir sortir vers une imprimante. Nous avons tenté dans la programmation de réduire à son minimum le temps de recherche: indépendamment du nombre de réponses correctes trouvées, notre programme met environ 15 secondes pour lire les noms contenus dans 100 dossiers de 400 fichiers répartis sur trois partitions du disque dur.

Comme ce programme doit servir d'accessoire et qu'il recourt à une fenêtre, nous en profiterons pour vous expliquer, après le texte du programme, le principe de fonctionnement d'un programme-accessoire ainsi que la programmation d'une fenêtre simple. Nous allons d'abord décrire le programme sans nous étendre sur ces deux points, car trop d'explications nuiraient à la clarté de notre exposé, surtout qu'elles nous détourneraient de l'explication des routines principales utilisées dans le programme ci-après.

Avant de passer à la description du programme, voici quelques explications concernant l'utilisation que nous avons faite des routines du système d'exploitation.

Drvmap()

BIOS Numéro 10

Appel: long Drvmap()

Le résultat de cette fonction donne un vecteur indiquant les unités de disque connectées. Le numéro de bit correspond à l'unité de disque, par exemple le bit 0 correspond à l'unité A etc.

Dsetdrv()

GEMDOS Numéro 0x0E

Appel: long Dsetdrv(drv)
 int drv;

Dsetdrv permet d'activer l'unité de disque mentionnée sous drv; on observe la règle suivante: A = 0, B = 1 etc.

Dsetpath()**GEMDOS Numéro 0x3B**

Appel: int Dsetpath(chemin)
 char *chemin;

Dsetpath permet de placer le répertoire actuel sur le nom indiqué sous 'chemin'. La valeur de la fonction est 0 en cas de succès, sinon vous recevez un numéro négatif d'erreur.

Fsetdta()**GEMDOS Numéro 0x1A**

Appel: void Fsetdta(buffer)
 char *buffer;

La fonction Fsetdta sert à déterminer l'adresse DTA (Disc transfer adress). Les informations obtenues par Ffirst et Fnext sont déposées dans ce buffer. C'est normalement le TOS qui se charge de préinstaller ce buffer DTA qui a une taille de 44 octets. Si l'on désire reprendre les informations contenues dans ce buffer pour les traiter dans un autre programme, on utilise la fonction fsetdta qui permet de transformer un espace-mémoire délimité en un buffer DTA. Dans le programme ci-dessous, nous nous sommes servis de DTAREC, dont voici la structure:

```
typedef struct dtarec
{
    char reserved[21];
    char attribut;
    int time;
    int date;
    long size;
    char name [14];
} DTAREC;
```

Les 21 premiers octets sont réservés pour le système. On trouve ensuite l'attribut du fichier qui résulte de l'utilisation ou de la combinaison des possibilités suivantes:

```
0x00  accès normal au fichier (lecture et écriture)
0x01  accès normal mais protection contre l'écriture
0x02  enregistrement dissimulé
0x04  enregistrement système dissimulé
0x08  nom de la disquette
0x10  sous-répertoire
0x20  fichier saisi puis refermé (statut archivé)
```

Les deux informations suivantes indiquent l'heure et la date système du fichier. La taille du fichier se trouve ensuite sous 'size' sous forme de mot long. On trouve enfin sous 'name' le nom du fichier suivi de son extension se terminant par un octet nul (format C-string).

Fsfirst() **GEMDOS Numéro 0x4E**

Appel: int Fsfirst(nom,attribut)
 char *nom;
 int attribut;

Fsfirst sert à rechercher dans le répertoire actuel la première mention du fichier indiqué sous 'name'. On peut se servir du masque ? et de la troncature *. La recherche peut être limitée en recourant à la caractéristique saisie sous 'attribut'. Vous trouvez sous 'fstda' la répartition des valeurs servant à l'attribut. Si la recherche débouche sur un résultat positif, la fonction prend la valeur 0; elle prend la valeur -33 lorsqu'il n'y a plus de fichier (le répertoire n'est pas vide mais on n'y trouve plus de fichier répondant au nom recherché); elle prend enfin la valeur -49 s'il n'y a plus de fichier, c'est-à-dire lorsque le répertoire est vide.

Fsnext() **GEMDOS Numéro 0x4F**

Appel: int Fsnext()

Fsnext permet de rechercher les autres noms recherchés d'après les critères fournis pour Fsfirst.

La fonction prend la valeur 0 si la recherche s'avère positive, sinon elle retourne un numéro d'erreur négatif.

Les autres fonctions AES et VDI utilisées dans ce programme seront explicitées après le listing, en même temps que la programmation d'une fenêtre.

Venons-en maintenant à la description du programme lui-même.

Après avoir déclaré le programme grâce à appl_init et lui avoir conféré un numéro d'identification, on initialise le champ des saisies VDI et on ouvre le 'virtual screen workstation' (à l'aide de open_vwork()).

La fonction menu_init() permet ensuite de créer le texte nécessaire dans les menus pour la sélection de cet accessoire.

Tout ceci étant fait, le programme s'engage dans une boucle sans fin (fonction `main_loop()`): en effet, nous voulons que ce programme soit un accessoire, et cette boucle sans fin constitue une des caractéristiques essentielles distinguant les accessoires des programmes ordinaires.

La fonction `event_mesag` sert à rendre notre programme sélectionnable par un simple clic sur l'option appropriée dans un menu déroulant. Dès que se produit ce clic, la fonction `dialog()` provoque sur l'écran l'ouverture d'une boîte de dialogue dont nous avons défini la structure (voir `desk_tree[]` au début du listing).

Nous avons maintenant centré la boîte de dialogue, réservé la surface nécessaire sur l'écran, dessiné un rectangle élastique et enfin la boîte elle-même; la fonction `form_do` permet alors de traiter les informations entrées par l'utilisateur.

On entre le nom à rechercher: la saisie est validée par un `< RETURN >`. La boîte de dialogue disparaît, le string se transforme en lettres majuscules, le programme contrôle l'exactitude de la saisie. Si tout est en ordre, il saute à la fonction `event()`, sinon il réaffiche la boîte de dialogue pour permettre à l'utilisateur de rectifier son erreur. On ne peut pas entrer `*.*`, car ceci reviendrait à rechercher tous les noms, ce qui est absurde. Le programme refuse également la saisie d'une chaîne vide (simple appui sur la touche `< RETURN >`).

La fonction `event` va fournir la plus grosse part du travail. Elle gère l'ensemble du traitement des fenêtres, c'est-à-dire qu'elle contrôle les éléments sélectionnés par l'utilisateur dans la fenêtre et réagit en conséquence. Elle appelle ensuite la routine de recherche après avoir créé une fenêtre, grâce à la fonction `'fenêtre()'`.

Procédons par ordre: en premier lieu, la fonction `event()` se charge de créer une fenêtre. Pour qu'elle ne soit pas dérangée dans son travail, nous appelons la fonction `wind_update`, qui va refouler les saisies effectuées éventuellement par l'utilisateur (le paramètre doit se trouver sur 1 ou true).

La fonction `"fenêtre()"` engendre tout d'abord la ligne d'information qui viendra s'incruster dans la fenêtre: c'est là que figurera le nom à rechercher.

On doit ensuite délimiter l'espace de travail du desktop, puis réserver la place pour la fenêtre et placer le nom dans la ligne réservée à cet usage. Il faut ensuite déterminer la taille de l'ascenseur vertical, ce que l'on fait à l'aide de la formule suivante:

$\text{Taille} = 1000 * \text{nombre de lignes visibles} / \text{nombre total de lignes}$

Si nous avons par exemple un nombre total de 120 lignes pour 21 lignes visibles, on obtiendra:

```
Taille = 1000 * 21/120
```

Si le nombre total de lignes est inférieur à 21, on obtient une taille supérieure à 1000; pour éviter cela, on insère la ligne suivante

```
slize = (taille>1000) ? taille : 1000;
```

qui empêche la taille de l'ascenseur de dépasser la valeur 1000.

On détermine la taille du curseur de l'ascenseur et sa position, après quoi la fenêtre s'ouvre; son champ de travail défini est transféré dans un champ de coordonnées rectangulaire.

Ces paramètres sont repris plus loin pour les fonctions VDI qui effacent le contenu de la fenêtre et doivent donc en connaître les coordonnées. Ces routines VDI engendrent par ordre: primo l'index des couleurs de remplissage, secundo le type de remplissage et tertio le rectangle correspondant aux coordonnées mentionnées ci-dessus. On appelle ensuite la fonction `clear_wi()`.

La recherche commence alors au moyen de la fonction `root_search()`. Ce travail se décompose en deux parties et en deux routines. La routine `root_search()` commence la recherche dans le répertoire-racine, et identifie tous les dossiers et fichiers à ausculter. La recherche ultérieure dans ces dossiers constitue la deuxième étape du processus, prise en charge par la routine `next_search()`. Cette fonction recherche le string souhaité dans les différents niveaux hiérarchiques des dossiers.

Commençons par la routine `root_search()`. Le premier pas consiste à identifier les unités de disque connectées (`Drvmap`). Comme cette fonction renvoie un vecteur-bit (`drive`), il faut le transformer en un nombre utilisable. Après quoi le programme ausculte toutes les partitions du disque dur.

Il signale à chaque fois l'unité de disque en cours (`Dsetdrv`), son identificateur étant repris dans l'intitulé du chemin d'accès (`disquette`). Comme nous voulons pouvoir reprendre ces identifications plus tard, il nous faut créer le buffer DTA au moyen de `Fsetdta`. Il suffit d'ajouter le chemin actuel (`Dsetpath`) et la recherche peut commencer dans le répertoire-racine de l'unité de disque actuelle.

Comme premier critère de recherche, nous introduisons d'abord `**`, ce qui veut dire que nous procédons à la recherche de tous les titres. Nous identifions ainsi

tous les dossiers se trouvant dans le répertoire-racine. Nous relançons ensuite une recherche dans ce même répertoire, mais en nous limitant cette fois aux noms répondant au critère recherché. Nous sauvegardons les noms de fichiers trouvés, plus exactement leur chemin d'accès, dans le fichier prévu pour cela. La fonction `next_search` se charge ensuite de la recherche dans les divers sous-répertoires de chaque unité de disque. Cette fonction se répète jusqu'à ce qu'elle ait ausculté tous les dossiers. Le principe de fonctionnement de cette routine ressemble très fortement à celui de `root_search()`, à la seule différence près que cette routine `root_search` sélectionne l'unité de disque actuelle.

Une fois auscultés tous les répertoires de toutes les partitions du disque dur, le nombre des fichiers trouvés est transmis à un buffer texte. Nous avons pris pour cela le premier élément (élément 0) du Text-array, ce qui nous permet de ne pas compter la ligne d'information et de pouvoir ainsi représenter 21 lignes au lieu de 20.

La recherche prend alors fin, et la fonction `event()` reprend le contrôle des opérations. Après le calcul de la nouvelle taille du curseur vertical, ce dernier est positionné et les 21 lignes affichées dans la fenêtre. S'il y a plus de 21 lignes, le curseur vertical ainsi que les flèches évidées au-dessus et au-dessous de l'ascenseur permettent de parcourir le texte (qui se compose des chemins d'accès des fichiers trouvés). Les événements provoqués en actionnant soit la coulisse soit les flèches évidées sont traités par les routines `curseur()` et `flèche()`.

Lorsque l'on déplace le curseur, la fonction '`curseur()`' calcule la nouvelle position atteinte et replace le curseur au bon endroit. Elle transmet en même temps l'index de la première ligne du texte résultant de ce déplacement. Le contenu primitif de la fenêtre s'efface et le nouveau contenu vient s'incruster à l'aide de la fonction `show_text`.

Si l'on se sert de `WM_ARROWED`, on peut distinguer quatre effets possibles:

1: l'utilisateur a cliqué sur la partie grisée en haut de l'ascenseur.

Le texte revient une 'page' en arrière, en ôtant de 21 le chiffre d'index de la 1ère ligne (l'index ne peut être inférieur à zéro). La fonction calcule ensuite la nouvelle position du curseur et affiche le nouveau texte résultant de ce déplacement.

2: l'utilisateur a cliqué sur la partie grisée en bas de l'ascenseur.

Le texte avance d'une 'page'. La fonction additionne 22 au chiffre d'index de la 1ère ligne, après quoi tout se passe comme sous 1.

- 3: l'utilisateur a cliqué sur la flèche évidée en haut de l'ascenseur.

L'index de la 1ère ligne est décrémenté de 1. Tout se passe ensuite comme expliqué ci-dessus.

- 4: l'utilisateur a cliqué sur la flèche évidée en bas de l'ascenseur.

L'index de la 1ère ligne est incrémenté de 1. Tout se passe ensuite comme expliqué ci-dessus.

Si l'utilisateur clique dans la fenêtre sur la petite case de fermeture, le programme lui demande s'il veut d'abord imprimer la liste. Si tel est le cas, le programme saute jusqu'à la fonction `impression()`. Il commence par attendre que l'imprimante soit prête à recevoir des données. La fonction `Cprnout` se charge ensuite de l'impression ligne par ligne et lettre par lettre. Attention: le bit supérieur de chaque caractère à imprimer doit se trouver sur 0.

Si l'utilisateur ne souhaite pas d'impression, on lui demande encore s'il veut rechercher un autre type de fichier. Si tel est le cas, le programme recommence du début dans la routine `main_loop()`, dans la boucle intérieure `while`. Lorsque l'utilisateur sort du programme, il faut encore refermer la fenêtre puis l'effacer. Le programme est alors terminé, il attend dans sa boucle sans fin un nouvel appel en provenance du menu déroulant.

Voici le listing du programme:

```

/* Programme : CHERCHE.C */
/* recherche de fichiers sur les partitions de disque durs */

#include <stdio.h>
#include <osbind.h>
#include <gemdefs.h>
#include <gembind.h>
#include <obdefs.h>
#include <string.h>

#define WI_KIND (NAME|CLOSER|DNARROW|UPARROW|VSLIDE)
#define COL(x) 8*x-8
#define LI(y) 16*y+16
#define TLIGNES 21
#define DESK 0
#define TRUE -1

```

```
#define FALSE 0
#define EOS '\0'
#define NULL 0

int contrl [12],
    intin [128],
    ptsin [128],
    intout [128],
    ptsout [128],
    work_in [11],
    work_out[57],
    msgbuff [8],
    vdi_handle,
    phys_handle,
    appl_id,
    menu_id,
    wi_handle,
    x1,y1,w1,h1,
    x2,y2,w2,h2,
    top,
    array[4];

typedef struct dtarec
{
    char reserved[21];
    char attribut;
    int time;
    int date;
    long size;
    char name[14];
}DTAREC;

typedef struct ordrec
{
    char filepath[80];
}ORDNER;

typedef struct filerec
{
    char filename[80];
}FILENAME;

char string1[]="Entrez le nom recherché";
char string2[]="Nomfic";
char string3[]="Fichier : _____";
char string4[]="XXXXXXXXXXXXX";
char string5[]=" Fin ";

TEDINFO teds[]={
```

```

string2, string3, string4, 3, 6, 0, 0x1180, 0x0, -1, 12, 24);

OBJECT desk_tree[]={
-1, 1, 3, G_BOX, NONE, OUTLINED, 0x21100L, 150, 100, 300, 160,
2, -1, -1, G_STRING, NONE, NORMAL, string1, 40, 16, 200, 16,
3, -1, -1, G_FTEXT, EDITABLE, NORMAL, &teds[0], 50, 90, 192, 16,
0, -1, -1, G_BUTTON, SELECTABLE|DEFAULT|EXIT|LASTOB, NORMAL, string5, 120
, 130, 64, 16};

DTAREC dtabuffer;
int hindex,
    quantite;
char chemin[80],
    nomfichier[12];
ORDNER name[100];
FILENAME fichier[100];

root_search()
{
    int encore,
        result,
        help,
        ind_1,
        disk,
        drv,
        new,
        ii;
    long drive;
    char disquette[5],
        nbretotal[5];

    drive=Drvmap();
    for(disk=-1, ii=1; (drive+1)>ii; disk++, ii*=2)
        ;
    if(disk<2)
    {
        form_alert(1, "[3][Pour le disque dur !!][ OK ]");
        return;
    }
    quantite=0;
    for(drv=2; drv<=disk; ++drv)
    {
        new=TRUE;
        ind_1=0;
        Dsetdrv(drv);
        *disquette=drv+'A';
        strcpy((disquette+1), ":");
        strcat(disquette, "\\");
        strcpy(chemin, disquette);
    }
}

```

```

Fsetdta(&dtabuffer);
Dsetpath(chemin);
Fsfirst("*. *",16);
help=(int) (dtabuffer.attribut);

if((help==16) && ((strcmp(dtabuffer.name, ".") != NULL) && ((strcmp
(dtabuffer.name, "..") != NULL))
    {
        ++ind_1;
        strcpy((name+ind_1)->filepath, dtabuffer.name);
    }
    while(Fsnext()==0)
    {
        help=(int) (dtabuffer.attribut);
        if((help==16) && ((strcmp(dtabuffer.name, ".") != NULL) && ((strcmp
(dtabuffer.name, "..") != NULL))
            {
                ++ind_1;
                strcpy((name+ind_1)->filepath, dtabuffer.name);
            }
        }
        result=Fsfirst(nomfichier,16);
        if(result==0)
            fichier_chemin();
        while(Fsnext()==0)
            fichier_chemin();
        hindex=ind_1;
        do
        {
            encore=next_search(hindex, disquette, new);
            new=FALSE;
        }while(encore==TRUE);
    }
    sprintf(nbretotal, "%d", quantite);
    strcpy((&fichier[0])->filename, nomfichier);
    strcat((&fichier[0])->filename, " : " );
    strcat((&fichier[0])->filename, nbretotal);
    strcat((&fichier[0])->filename, " trouvé(s).");
}

next_search(indexs, disquette, new)
int indexs;
char *disquette;
int new;
{
    register int xx;
    int oflag,
        ind_2,
        result,
        help;
    static int loop=0;

```

```

static int ind_4;
char dossier[80];

if(new==TRUE)
    ind_4=loop=0;
ind_2=indexs;
++loop;
oflag=FALSE;
for(xx=ind_4+1;xx<=indexs;++xx)
{
    strcpy(chemin,disquette);
    strcpy(dossier,(name+xx)->filepath);
    if(loop==1)
        strcat(chemin,dossier);
    else
        strcpy(chemin,dossier);
    Dsetpath(chemin);
    result=Fsfirst("*.*",16);
    help=(int)(dtabuffer.attribut);
    if((help==16)&&((strcmp(dtabuffer.name,".")!=NULL)
        &&((strcmp(dtabuffer.name,"..")!=NULL)))
    {
        oflag=TRUE;
        ++ind_2;
        strcpy((name+ind_2)->filepath,chemin);
        strcat((name+ind_2)->filepath,"\\");
        strcat((name+ind_2)->filepath,dtabuffer.name);
    }
    while(Fsnext()==0)
    {
        help=(int)(dtabuffer.attribut);
        if((help==16)&&((strcmp(dtabuffer.name,".")!=NULL)
            &&((strcmp(dtabuffer.name,"..")!=NULL)))
        {
            oflag=TRUE;
            ++ind_2;
            strcpy((name+ind_2)->filepath,chemin);
            strcat((name+ind_2)->filepath,"\\");
            strcat((name+ind_2)->filepath,dtabuffer.name);
        }
    }
    result=Fsfirst(nomfichier,16);
    if(result==0)
        fichier_chemin();
    while(Fsnext()==0)
        fichier_chemin();
}
ind_4=indexs;
hindex=ind_2;
return(oflag);

```

```

}

fichier_chemin()
{
    ++quantite;
    strcpy((fichier+quantite)->filename, chemin);
    strcat((fichier+quantite)->filename, "\\");
    strcat((fichier+quantite)->filename, dtabuffer.name);
}

transforme(string1, string2)
char *string1,
    *string2;
{
    register int longueur;
    register int tt;
    longueur=strlen(string1);
    for(tt=1;tt<=longueur;++tt)
        *string2++=toupper(*string1++);
}

event ()
{
    int compteur,
        slize,
        taille,
        phrase;
    register int tt,
        zz;

    graf_mouse(M_OFF, 0L);
    wind_update(TRUE);
    fenetre();
    compteur=1;
    root_search();
    taille=1000*21/quantite;
    slize=(taille<1000) ? taille : 1000;
    wind_set(wi_handle, WF_VSLSIZE, slize, 0, 0, 0);
    wind_set(wi_handle, WF_VSLIDE, 0, 0, 0, 0);
    for(tt=0, zz=0; (zz<=TLIGNES) && (tt<=quantite); ++zz)
    {
        v_gtext(vdi_handle, (COL(1)+x2), (LI(zz)+y2),
            ((fichier+tt)->filename));
        ++tt;
    }
    wind_update(FALSE);
    graf_mouse(M_ON, 0L);
    top=0;
    do
    {
        evt_mesag(msgbuff);

```

```

graf_mouse(M_OFF,0L);
wind_update(TRUE);
switch(msgbuff[0])
{
    case WM_CLOSED:
        phrase=quest();
        compteur--;
        break;
    case WM_VSLID:
        new_top();
        break;
    case WM_ARROWED:
        fleche();
        break;
}
wind_update(FALSE);
graf_mouse(M_ON,0L);
}while(compteur>0);
return(phrase);
}

fleche()
{
switch(msgbuff[4])
{
    case 0:
        top-=21;
        if(top<0)
            top=0;
        set_vslid();
        break;
    case 1:
        top+=22;
        if(top>(quantite-21))
            top=(quantite-21);
        set_vslid();
        break;
    case 2:
        if(top>0)
        {
            --top;
            set_vslid();
        }
        break;
    case 3:
        if(top<(quantite-TLIGNES))
        {
            ++top;
            set_vslid();
        }
}
}

```

```

break;
}
}

set_vslid()
{
    int poscurseur;

    clear_wi();
    show_text();

    poscurseur=(int) (1000*((double)top/((double) (quantite-TLIGNES))));

    wind_set(wi_handle,WF_VSLIDE,poscurseur,0,0,0);
}

show_text()
{
    register int tt,
               zz;

    for(tt=top,zz=0;(zz<=TLIGNES)&&(tt<=quantite);++zz)
    {
        v_gtext(vdi_handle,(COL(1)+x2),(LI(zz)+y2),
                ((fichier+tt)->filename));
        ++tt;
    }
}

new_top()
{
    int posnew;
    long posact;

    posact=msgbuff[4];
    posnew=(int) (posact*(quantite-TLIGNES)/1000);
    wind_set(wi_handle,WF_VSLIDE,((int)posact),0,0,0);
    top=posnew;
    clear_wi();
    show_text();
}

fenetre()
{
    int slize,
        taille;
    char infos[80];

    strcpy(infos," Type de fichier : ");
    strcat(infos,nomfichier);
}

```

```

strcat (infos, " ");
wind_get (DESK, WF_WORKXYWH, &x1, &y1, &w1, &h1);
wi_handle=wind_create (WI_KIND, 0, 0, 640, 400);
wind_set (wi_handle, WF_NAME, infos, 0, 0);
taille=1000*21/quantite;
sliste=(taille<1000) ? taille : 1000;
wind_set (wi_handle, WF_VSLSIZE, sliste, 0, 0, 0);
wind_set (wi_handle, WF_VSLIDE, 0, 0, 0, 0);
form_dial (FMD_GROW, 320, 200, 20, 20, x1, y1, w1, h1);
wind_open (wi_handle, x1, y1, w1, h1);
wind_get (wi_handle, WF_WORKXYWH, &x2, &y2, &w2, &h2);
array[0]=x2;
array[1]=y2;
array[2]=(x2+(w2-1));
array[3]=(y2+(h2-1));
clear_wi ();
}

clear_wi ()
{
vsf_color (vdi_handle, 0);
vsf_interior (vdi_handle, 1);
v_bar (vdi_handle, array);
}

close_window ()
{
clear_wi ();
form_dial (FMD_SHRINK, 320, 200, 20, 20, x1, y1, w1, h1);
wind_close (wi_handle);
wind_delete (wi_handle);
graf_mbox (20, 20, 320, 200, 50, 16);
}

dialog ()
{
GRECT box;
char buffer[12];
register int xx;

for (xx=0; xx<12; ++xx)
nomfichier[xx]=EOS;

form_center (desk_tree, &box.g_x, &box.g_y, &box.g_w, &box.g_h);
graf_mbox (20, 20, 50, 16, 320, 200);

form_dial
(FMD_START, 320, 200, 20, 20, box.g_x, box.g_y, box.g_w, box.g_h);

```

```

form_dial
(FMD_GROW, 320, 200, 20, 20, box.g_x, box.g_y, box.g_w, box.g_h)

objc_draw
(desk_tree, 0, MAX_DEPTH, box.g_x, box.g_y, box.g_w, box.g_h);
form_do(desk_tree, 0);
desk_tree[3].ob_state &= SELECTED;
form_dial
(FMD_SHRINK, 320, 200, 20, 20, box.g_x, box.g_y, box.g_w, box.g_h);
form_dial
(FMD_FINISH, 320, 200, 20, 20, box.g_x, box.g_y, box.g_w, box.g_h);
strcpy(buffer, (&teds[0])->te_ptext);
transforme(buffer, nomfichier);

if(((strcmp(nomfichier, "*. *")==0) ||
(strcmp(nomfichier, "")==0))
return(FALSE);
else
return(TRUE);
}

quest()
{
int freturn;

freturn=form_alert
(2, "[1][Sortie de liste|sur imprimante ?][ Oui | Non ]");
switch(freturn)
{
case 1:
ausdruck();
break;
case 2:
break;
}
freturn=form_alert
(2, "[3][Chercher autre chose ?][ Oui | Non ]");
if(freturn==2)
return(FALSE);
else
return(TRUE);
}

ausdruck()
{
char buchst[80];
register int xx,
YY;

int error,
letter;

```

```

while ((Cprnos())==0)
    form_alert(1,"[3][PB Imprimante !!!][ OK ]");
for(xx=0;xx<=quantite;+xx)
{
    strcpy(buchst,(fichier+xx)->filename);
    yy=0;
    Cprnout(13);
    Cprnout(10);
    while (*(buchst+yy)!=EOS)
    {
        letter=(int)buchst[yy];
        letter=(letter&255);
        error=Cprnout(letter);
        if(error!=-1)
        {
            form_alert(1,"[3][Erreur d'impression...][ OK ]");
            goto ENDE;
        }
        yy++;
    }
    Cprnout(10);
    Cprnout(10);
    ENDE;
}

main_loop()
{
    int flag1,
        flag2;

    while(TRUE)
    {
        graf_mouse(0,0L);
        evnt_mesag(msgbuff);
        if((msgbuff[0]==AC_OPEN) && (msgbuff[4]==menu_id))
        {
            do
            {
                wind_update(TRUE);
                flag1=dialog();
                wind_update(FALSE);
                if(flag1==FALSE)
                    form_alert(1,"[3][Donnez un autre nom, s.v.p][ OK
]");
            }
            else
            {
                flag2=event();
                clear_wi();
                close_window();
            }
        }
    }
}

```

```

    }
        }while((flag1==TRUE)&&(flag2==TRUE));
    }
}

menu_init()
{
    menu_id=menu_register(appl_id," Inspecteur");
    if(menu_id==-1)
    {
        form_alert(1,"[1][Plus de place pour le menu ACC][ OK ]");
        appl_exit();
    }
}

open_vwork()
{
    register int i;
    int dummy;

    phys_handle=graf_handle(&dummy,&dummy,&dummy,&dummy);
    vdi_handle=phys_handle;
    for(i=0;i<10;work_in[i++]=1);
    work_in[10]=2;
    v_opnvwk(work_in,&vdi_handle,work_out);
}

main()
{
    appl_id=appl_init();
    open_vwork();
    menu_init();
    main_loop();
}

```

6.4. Programmation simple des fenêtres

Comme nous vous l'avons promis dans les commentaires relatifs au programme précédent, nous allons maintenant vous fournir quelques explications concernant la manipulation et la programmation des fenêtres. Nous allons vous expliquer les commandes les plus importantes pour la gestion des fenêtres et vous donner quelques petits programmes d'exemples.

Comme nous l'avons précisé dans l'intitulé de ce paragraphe, nous allons nous limiter aux éléments les plus simples de la programmation des fenêtres, mais ils devraient vous suffire pour comprendre le principe de base de cette programmation.

Assez de discours. Les fenêtres furent, à l'époque de la mise sur le marché des ST, la caractéristique la plus spectaculaire de cette nouvelle génération d'ordinateurs. Dans un traitement de texte, la fenêtre permet de feuilleter rapidement le texte ou de le faire défiler (scrolling) horizontalement ou verticalement. On peut même ouvrir simultanément plusieurs fenêtres, et traiter ainsi plusieurs textes de front. Naturellement, cet avantage comporte aussi des inconvénients: la programmation de ce type de fenêtre est considérablement plus difficile par rapport à la programmation d'un affichage simple à l'écran dans les anciens traitements de texte. Heureusement, le système d'exploitation comprend de nombreuses fonctions facilitant sensiblement le travail. Avant d'en venir à la présentation des éléments, nous allons vous expliquer les commandes servant à programmer les diverses possibilités offertes par les fenêtres.

Nous allons donc aborder ce qu'on appelle 'event-library', bibliothèque d'évènements. Les commandes qui s'y trouvent répertoriées permettent dans une certaine mesure de travailler en multitâche, c'est-à-dire de réaliser simultanément plusieurs opérations différentes. L'ordinateur peut par exemple contrôler les mouvements du pointeur de la souris tout en surveillant le clavier (pour voir si l'utilisateur y effectue une saisie) et en faisant réagir les options de la barre des menus qui doivent se 'dérouler' si l'utilisateur les effleure avec la souris. Ceci revient donc bien à contrôler simultanément un certain nombre d'évènements qui peuvent se produire. Lorsqu'un de ces évènements survient effectivement, le système active la routine correspondante qui va procéder au traitement adéquat pendant que l'ordinateur recommencera à contrôler si d'autres évènements surviennent. Comme on ne peut pas déterminer à l'avance combien de temps va durer une routine, on parle de 'multitasking' limité (routine se dit 'task' en anglais, d'où le mot 'multitasking' = faire tourner en même temps plusieurs routines, et plus exactement leur attribuer à chacune l'une après l'autre un laps de temps déterminé).

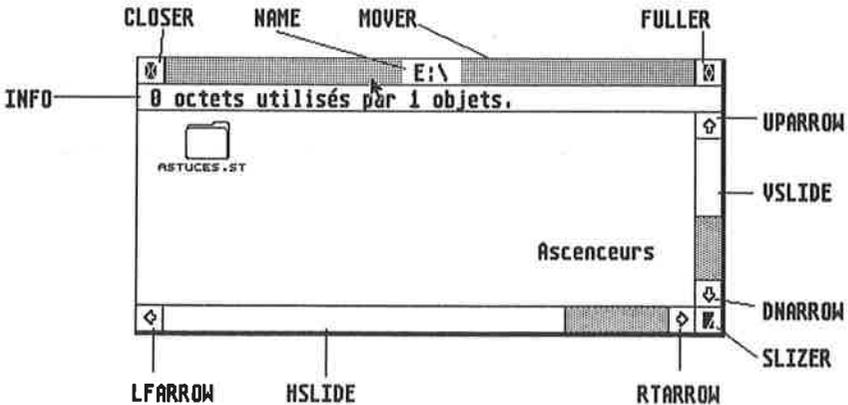
Pour faciliter le travail du programmeur, la bibliothèque d'évènements contient de nombreuses commandes spéciales correspondant à la gestion des évènements les plus fréquents.

Voici la liste des commandes:

1. evnt_keybd()
2. evnt_button()
3. evnt_mouse()
4. evnt_mesag()
5. evnt_timer()
6. evnt_multi()
7. evnt_dclick()

1. wind_create()
2. wind_open()
3. wind_close()
4. wind_delete()
5. wind_get()
6. wind_set()
7. wind_find()
8. wind_calc()
9. wind_update()

Voici maintenant l'image d'une fenêtre, avec les commandes permettant de modifier ses divers composants dans un programme.



Vous devez indiquer dans la variable 'wi_crkind' de la fonction 'wind_create()' les composants que vous désirez utiliser pour construire votre fenêtre. Chaque composant est représenté par un bit. Le composant correspondant sera retenu si son bit correspondant est positif. Voici la répartition de ces bits:

0x0001	- NAME	(ligne de titre)
0x0002	- CLOSER	(case de fermeture)
0x0004	- FULLER	(case plein écran)
0x0008	- MOVER	(barre de déplacement)
0x0010	- INFO	(ligne d'information)
0x0020	- SIZER	(case de contrôle de taille)
0x0040	- UPARROW	(flèche de défilement vers le haut)
0x0080	- DNARROW	(flèche de défilement vers le bas)
0x0100	- VSLIDE	(ascenseur vertical)
0x0200	- LFARROW	(flèche de défilement vers la gauche)
0x0400	- RTARROW	(flèche de défilement vers la droite)
0x0800	- HSLIDE	(ascenseur horizontal)

Vous retrouvez ces définitions dans les fichiers.H. Pour fixer la valeur des bits, il vous suffit de relier les constantes par l'opérateur OU. Par exemple:

```
wi_crreturn = wind_create (NAME|CLOSER|MOVER|INFO,wi_crwx,...)
```

Cet exemple vous permettrait de créer une fenêtre dotée des composants suivants: ligne de titre, case de fermeture, barre de déplacement et ligne d'informations. Les barres de défilement sont d'abord traitées comme les flèches de défilement: la différenciation entre ces deux composants ne se fait qu'ultérieurement.

Voici maintenant, étape par étape, la démarche à suivre pour voir une fenêtre apparaître sur votre écran.

Nous devons d'abord savoir quelle sera la taille maximale de notre fenêtre. Comme le bureau GEM représente déjà une fenêtre gérée directement et de façon autonome par l'AES, nous allons préciser la 'surface de travail' utilisée par notre fenêtre à l'aide de la fonction 'wind_get'; nous visons à obtenir la taille maximale possible pour une fenêtre créée par l'utilisateur.

C'est pourquoi nous écrivons cette fonction comme suit:

```
wind_get (0, 4, &x0max, &y0max, &w0max, &h0max);
```

Le zéro représente le window-handle du bureau GEM qui ne varie pas. Le paramètre 4 nous permet d'obtenir la taille de la surface de travail de la fenêtre

choisie par son handle: il s'agit ici de la fenêtre du bureau GEM. Les coordonnées obtenues sont sauvegardées dans les variables x0max à h0max; x0max et y0max contiennent les coordonnées du coin supérieur gauche de la fenêtre, tandis que w0max et h0max contiennent respectivement la largeur et la hauteur de la fenêtre. Il nous faut ensuite calculer la surface de travail de la fenêtre, car les coordonnées ci-dessus sont celles d'une surface comprenant la surface de travail plus les composants situés dans l'encadrement de la fenêtre. Nous utilisons pour cela la fonction 'wind_calc'. Pour pouvoir remplir son rôle, cette routine doit connaître les composants attribués à la fenêtre. Si nous désirons par exemple une fenêtre dotée d'une ligne de titre, d'une ligne d'informations, d'une case de fermeture, des deux flèches de défilement (vers le bas et le haut) ainsi que d'un ascenseur vertical, nous devons écrire:

```
wind_calc(1, NAME | INFO | CLOSER | DNARROW | UPARROW | VSLIDE,
          x0max, y0max, w0max, h0max, &x1, &y1, &w1, &h1)
```

Le paramètre 1 demande à 'wind_calc' de calculer la taille de la surface de travail dans la fenêtre. Le résultat est transmis aux variables x1 à h1.

Nous n'oublions pas qu'il nous faut calculer non seulement la surface de travail maximale mais aussi la surface d'écran qui sera ensuite effectivement utilisée. Cette dernière peut fort bien être plus restreinte mais en aucun cas plus grande que la première. Pour cela, nous faisons encore appel à la fonction wind_calc, mais en entrant un 0 (et non plus un 1) dans le premier paramètre. Nous recevons en retour les coordonnées de la surface occupée par la fenêtre, y compris son cadre.

```
wind_calc(0, NAME | INFO | CLOSER | DNARROW | UPARROW | VSLIDE,
          x1, y1, w1, h1, &x1max, &y1max, &w1max, &h1max)
```

Munis de ces coordonnées, nous pouvons appeler la fonction wind_create, qui sert à réserver la place nécessaire à la fenêtre. Si tout se passe bien, nous devons recevoir un numéro d'identification, appelé 'window-handle', qui nous servira plus tard à appeler la fenêtre. Comme le GEM-AES ne peut gérer que 8 fenêtres simultanément, ce window-handle identifie bien chaque fenêtre individuellement. S'il n'y a plus de fenêtre disponible, on reçoit une valeur négative.

On écrit donc:

```
handle=
wind_create(NAME | INFO | CLOSER | DNARROW | UPARROW | VSLIDE, x1max, 1max, w1max, h1max)
```

Avant de pouvoir ouvrir la fenêtre à l'aide de `wind_open`, il nous reste encore à communiquer à l'AES, sous forme de strings, les adresses du nom de la fenêtre et de la ligne d'informations. On se sert pour cela de la fonction `wind_set`. Comme nous n'indiquons que les adresses sous forme de strings, nous devons déclarer ces variables comme étant 'static'. On écrira:

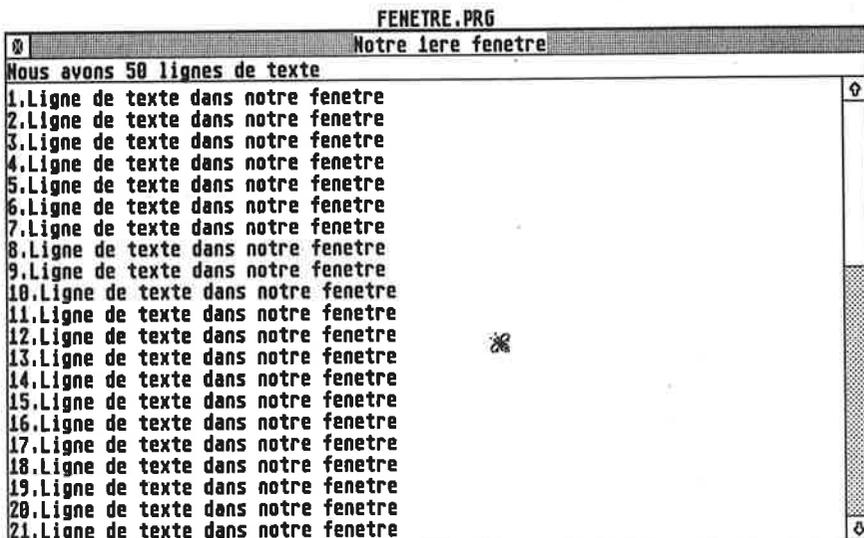
```
static char titre[] = "nom de la fenêtre";
static char d'information[] = "information fenêtre";
wind_set(handle,2,titre,0,0);
wind_set(handle,3,information,0,0);
```

Nous pouvons enfin ouvrir notre fenêtre à l'aide de `wind_open`:

```
wind_open(handle,xlmax,ylmax,wlmax,hlmax,);
```

Cette ligne de programme permet d'ouvrir la fenêtre dans son plus grand format sur l'écran. Si vous désirez en ouvrir une plus petite, divisez les variables `xlmax` à `hlmax` par une valeur vous permettant d'obtenir la taille que vous souhaitez et remplacez les valeurs maximales par les nouvelles valeurs ainsi obtenues.

Si vous provoquez maintenant l'affichage de la fenêtre engendrée par la programmation ci-dessus, vous voyez qu'elle est encore démunie de sa surface de travail, comme le montre l'image suivante:



Pour dégager une surface de travail, vous devez auparavant la vider de son fond grisé à l'aide des trois routines VDI que nous allons expliquer.

vsf_color()**VDI Numéro 25**

Appel: `set_color = vsf_color(handle,color_index)`
 `int set_color;`
 `int vsf_color();`
 `int handle;`
 `int color_index;`

Cette fonction sert à préciser la teinte de remplissage; les couleurs sont sélectionnées par `vs_color()`. Pour les moniteurs monochromes, vous n'avez le choix qu'entre 0=noir et 1=blanc. 'Color_index' vous donne l'index des couleurs; votre sélection sera mémorisée par 'set_color'. 'Handle' désigne ici le VDI-handle.

vsf_interior()**VDI Numéro 23**

Appel: `set_interior = vsf_interior(handle,stil)`
 `int set_interior;`
 `int vsf_interior();`
 `int handle;`
 `int stil;`

Cette fonction permet de choisir le type de remplissage parmi les cinq possibilités suivantes:

- | | |
|---|---|
| 0 | la surface reste blanche |
| 1 | la surface est remplie par la teinte de remplissage |
| 2 | la surface est remplie par le motif créé sous <code>vsf_style()</code> |
| 3 | la surface est remplie par le motif rayé créé sous <code>vsf_stule()</code> |
| 4 | la surface est remplie par un motif que vous avez vous-même créé. |

v_bar()**VDI Numéro 11,1**

Appel: `v_bar(handle,pxyarray);`
 `int v_bar();`
 `int handle;`
 `int pxyarray[4];`

Cette fonction permet de dessiner des surfaces remplies selon un motif, en tenant compte des attributs retenus auparavant. La fonction pxyarray doit contenir les coordonnées suivantes:

```
pxyarray[0] - abscisse x du coin supérieur gauche
pxyarray[1] - ordonnée y du coin supérieur gauche
pxyarray[2] - abscisse x du coin inférieur droit
pxyarray[3] - ordonnée y du coin inférieur droit
```

Avant d'entrer ici les coordonnées de la surface de travail (x1 à h1), nous devons encore préciser les coordonnées du point correspondant au coin inférieur droit. Ce calcul s'effectue lors de la saisie qui prend l'aspect suivant:

```
pxyarray[0] = x1;
pxyarray[1] = y1;
pxyarray[2] = (x1+w1);
pxyarray[3] = (y1=h1);
```

Une fois terminée l'initialisation de pxyarray, nous faisons intervenir l'une après l'autre les trois routines VDI mentionnées ci-dessus, le contenu de la fenêtre est ensuite effacé.

```
vsf_color(vdi_handle,0; /**/
vsf_interior(vdi_handle,1); /**/
v_bar(vdi_handle,pxyarray); /**/
```

Et voilà! nous disposons enfin d'une fenêtre ressemblant à une belle feuille blanche, sur laquelle nous allons pouvoir écrire.

Voici un résumé qui vous donnera une vue d'ensemble de toutes les étapes parcourues jusqu'ici:

1. wind_get(0,4,&x0max,&y0max,&w0max,&h0max);
2. wind_calc(1,NAME|INFO|CLOSER|DNARROW|UPARROW|VSLIDE,x0max,y0max,w0max,h0max,&x1,&y1,&w1,&h1)
3. wind_calc(0,NAME|INFO|CLOSER|DNARROW|UPARROW|VSLIDE,x1,y1,w1,h1,&xlmax,&ylmax,&wlmax,&hlmax)
4. handle=wind_create(NAME|INFO|CLOSER|DNARROW|UPARROW|VSLIDE,xlmax,ylmax,wlmax,hlmax);
5. wind_set(handle,2,ligne de titre,0,0);
6. wind_set(handle,3,ligne d'information,0,0);
7. wind_open(handle,xlmax,ylmax,wlmax,hlmax);
8. pxyarray[0] = x1;
 pxyarray[1] = y1;

```

    pxyarray[2] = (x1+w1);
    pxyarray[3] = (y1+h1);
9. vsf_color(vdi_handle,0);
10.vsf_interior(vdi_handle,1);
11.v_bar(vdi_handle,pxyarray);

```

Naturellement, il faut avant toute chose déclarer et définir les variables ainsi que les strings utilisés pour le nom et la ligne d'informations de la fenêtre.

Nous mémorisons, dans un buffer d'une structure array, le texte que nous voulons inscrire dans la fenêtre. Pour plus de simplicité, le texte se compose d'une seule et même phrase précédée par un numéro courant: cela nous a paru suffisant pour une démonstration.

Pour inscrire ce texte dans la fenêtre, nous utilisons une autre fonction VDI: `v_gtext`.

`v_gtext()`

VDI-Numéro 8

Appel: `v_gtext(vdi_handle,x,y,string);`
`int v_gtext();`
`int vdi_handle;`
`int x;`
`int y;`
`char string[n];`

Cette fonction affiche le texte se trouvant sous `string[]` en le positionnant sur l'écran graphique selon les coordonnées `x/y`.

La fenêtre est maintenant ouverte, et le texte `y` est bien inscrit. L'étape suivante consiste à calculer la taille et la position de l'ascenseur vertical. Sa taille sera calculée selon la formule suivante:

```
Taille=1000* Quantité des lignes visibles/nombre total de lignes
```

Dans notre exemple, la quantité de lignes visibles s'élève à 20 et le nombre total de lignes est de 50; ce qui nous donne la formule:

```
taille = 1000 * 20/50
```

Nous obtenons la valeur 400, qui est reprise par 'wind_set':

```
wind_set (wi_handle, WF_VSLSIZE, taille, 0, 0, 0);
```

La taille de l'ascenseur devra varier en même temps que le nombre de lignes. Sa position est calculée de la façon suivante:

```
place = 1000 * (1ère position dans la fenêtre - 1ère position
du texte) / (nombre total des lignes de texte - quantité de
lignes visibles)
```

Cette valeur sera re-calculée à chaque déplacement de l'ascenseur et reprise sous 'wind_set':

```
wind_set (wi_handle, WF_VSLSIZE, place, 0, 0, 0);
```

Les travaux préliminaires sont terminés, une fonction 'evnt_multi' ou 'evnt_mesag' va prendre en charge le contrôle de la fenêtre et la gestion des demandes de l'utilisateur. Dans notre exemple, nous avons retenu la fonction 'evnt_mesag', qui s'avère suffisante pour notre démonstration. Cette fonction, comme nous l'avons expliqué plus haut, guette une action quelconque de l'utilisateur dans la fenêtre. Cette boucle se répète jusqu'à ce que l'utilisateur clique sur la case de fermeture. Lorsque l'utilisateur clique sur un des composants de la fenêtre, le programme identifie l'action dont il s'agit en allant dans le message-buffer numéro 0 (msgbuff[0]) qui contient une constante pouvant prendre les significations suivantes:

```
#define MN_SELECTED 10 /*clic sur une option d'un menu*/
#define WM_REDRAW 20 /*redessiner la fenêtre*/
#define WM_TOPPED 21 /*activer une fenêtre*/
#define WM_CLOSED 22 /*clic sur la case de fermeture*/
#define WM_FULLED 23 /*clic sur le 'FULLER', le motif de
remplissage*/
#define WM_ARROWED 24 /*clic sur la flèche de défilement
vers le haut*/
#define WM_HSLID 25 /*déplacement de l'ascenseur
horizontal*/
#define WM_VSLID 26 /*déplacement de l'ascenseur
vertical*/
#define WM_SIZED 27 /*modification de la taille de la
fenêtre*/
#define WM_MOVED 28 /*déplacement de toute la fenêtre*/
#define WM_NEWTOP 29 /*fenêtre ré-activée*/
#define AC_OPEN 40 /*clic sur un menu d'accessoires*/
#define AC_CLOSE 41 /*fermeture de l'accessoire*/
```

Ces définitions se trouvent dans un fichier-header du compilateur. Les constantes que nous avons utilisées vont maintenant être contrôlées par une structure 'switch()-case:' pour ensuite se diriger vers les routines adéquates.

Par exemple, après un déplacement de l'ascenseur vertical (WM_VSLIDE), notre programme va calculer la nouvelle position de son curseur puis l'y envoyer. Il va ensuite rechercher la nouvelle première ligne (pour cette formule, voir le listing de new_top()), vider la fenêtre et repositionner tout le texte.

Si l'utilisateur par contre se sert d'une des flèches de défilement (ou de la barre grisée) WM_ARROW, une autre routine nommée flèche() va se renseigner pour savoir sur quelle flèche ou sur quelle barre grisée a été exécuté le clic. Cette information se trouve dans le message-buffer numéro 4 (msgbuff[4]).

Si on clique sur une des barres grisées, le texte avance ou recule d'une page entière. Il suffit pour cela d'ajouter ou de retirer le nombre voulu à l'index de la première ligne. En cas de clic sur une des flèches, cet index n'augmente ou ne diminue que de 1. A chaque fois, il faut recalculer la position du curseur de l'ascenseur, qui n'a certes pas été directement mis en cause mais qu'il faut bien amener sur sa nouvelle position, puisqu'il doit fournir à l'utilisateur une idée approximative de l'endroit où il se trouve dans son texte. Ceci se fait grâce à wind_set, comme nous l'avons vu plus haut.

Si un clic se produit sur la case de fermeture, le programme quitte la boucle evnt_mesag, referme la fenêtre par wind_close, l'efface par wind_delete; il referme l'écran virtuel par v_clsawk: en dernier lieu, appl_exit vous fait sortir du programme.

Voilà, nous venons de parcourir les étapes les plus importantes pour créer une fenêtre, y inscrire un texte et gérer ses divers composants. Comme nous vous l'avons dit, nous n'avons pas utilisé toutes les possibilités offertes par l'AES pour la gestion des fenêtres, car cela aurait largement dépassé les limites de cet ouvrage. Notre but est atteint si vous avez maintenant perdu toute appréhension et que vous vous mettez à utiliser ces fenêtres dans vos propres programmes. Pour aller plus loin dans ce domaine, essayez de vous procurer un manuel adéquat.

Voici enfin le texte complet de ce programme que nous avons déjà tant commenté:

```
#include<stdio.h>                FENETRE.C
#include<osbind.h>
#include<gemdefs.h>
#include<gembind.h>
#include<obdefs.h>
```

```

#define WI_KIND (NAME|CLOSER|INFO|DNARROW|UPARROW|VSLIDE)
/*composants de la fenetre*/
#define SP(x) 8*x-8 /*'X' indique la colonne souhaitee*/
#define LG(y) 16*y+16 /*'Y' indique la ligne souhaitee*/
#define GLIGNES 20 /*nombre total de lignes dans la
fenetre*/
#define DESK 0 /*window-handle pour le desktop*/
#define TRUE -1
#define FALSE 0
#define NULL 0L

int contrl[12], /*GEM-Arrays pour VDI et AES*/
    intin[128],
    ptsin[128],
    intout[128],
    ptsout[128],
    work_in[11],
    work_out[57],
    msgbuff[8],
    vdi_handle, /*VDI-Handle*/
    phys_handle, /*workstation-handle*/
    appl_id, /*identification de l'application*/
    wi_handle, /*window-handle*/
    xl,y1,w1,h1, /*surface de travail du desktop*/
    x2,y2,w2,h2, /*surface de travail maximale dans
la fenetre*/
    x3,y3,w3,h3, /*surface de travail dans la fenetre*/
    pxyarray[4], /*champ de coordonnees rectangulaire*/
    top, /*1ere ligne visible du texte*/
    quantite; /*nombre total de lignes de texte*/

typedef struct textrec /*installation du buffer-texte*/
{
    char lignetexte[40]; /*longueur de la ligne*/
}TEXT;

TEXT textarray[50]; /*nombre de lignes*/

main()
{
    appl_id=appl_init(); /*declarer le programme*/
    open_vwork(); /*initialisations*/
    text_init();
    event(); /*traitement des evenements*/
    close_window(); /*fermer la fenetre*/
    v_clsvwk(vdi_handle); /*fermer le 'virtual screen
workstation'*/
    appl_exit(); /*sortir du programme*/
}

```

```

open_vwork()
{
    register int i;          /*index courant*/
    int dummy;
    /*indiquer le workstation-handle*/
    phys_handle=graf_handle(&dummy,&dummy,&dummy,&dummy);
    vdi_handle=phys_handle;
    for(i=0;i<10;work_in[i++]=1); /*initialiser le champ*/
    work_in[10]=2;
    v_opnvwk(work_in,&vdi_handle,work_out); /*ouvrir le 'virtual
                                           screen station*/
}
text_init()                /*initialisation du text-array*/
{
    register int ii;
    char nbre[2];

    quantite=50;
    for(ii=0;ii<=49;+ii)
    {
        sprintf(nbre,"%d",ii+1);
        strcpy(((textarray+ii)->lignetexte),nbre);
        strcat(((textarray+ii)->lignetexte),
               ".Ligne de texte dans notre fenetre");
    }
}
event()                    /*gestion de la fenetre*/
{
    int slize,              /*taille relative*/
        taille;           /*taille absolue*/
    register int tt,       /*index courant*/
                zz;

    graf_mouse(M_OFF,0L);  /*deconnecter la souris*/
    wind_update(TRUE);     /*plus d'action possible pour
l'utilisateur*/
    fenetre();             /*creer la fenetre*/
    taille=1000*20/quantite; /*calculer la taille absolue*/
    slize=(taille<1000) ? taille : 1000; /*calcul de la taille
                                           relative*/
    wind_set(wi_handle,WF_VSLSIZE,slize,0,0,0); /*determiner la
                                           taille du curseur vertical*/
    wind_set(wi_handle,WF_VSLIDE,0,0,0,0); /*determiner la
                                           position du curseur vertical */
    for(tt=0;(zz<=GLIGNES)&&(tt<=quantite);++zz) /*affichage des
                                           résultats */
    {
        v_gtext(vdi_handle,(SP(1)+x2),(LG(zz)+y2),
        ((textarray+tt)->lignetexte));
        ++tt;
    }
}

```

```

}
wind_update(FALSE);      /*redonner la main à l'utilisateur*/
graf_mouse(M_ON,0L);     /*reactiver le pointeur de la
                           souris*/

top=0;
do
{
    evt_mesag(msgbuff);
    graf_mouse(M_OFF,0L);
    wind_update(TRUE);
switch(msgbuff[0])      /*où s'est produit le clic?*/
{
    case WM_VSLID:      /*clic sur l'ascenseur*/
        new_top();      /*calculer la nouvelle lere ligne
                           visible*/
        break;          /*puis reorganiser le texte en
                           consequence*/
    case WM_ARROWED:    /*clic sur la fleche de defilement*/
        fleche();       /*calculer la nouvelle position du
                           curseur*/
        break;          /*puis reorganiser le texte en
                           consequence*/
    default:
        break;
}
    wind_update(FALSE);
    graf_mouse(M_ON,0L);
}while(msgbuff[0] != WM_CLOSED); /*jusqu'à ce qu'un clic se
    produise sur la case de fermeture*/
}
fenetre()
{
    static char name[]="Notre lere fenetre",      /*titre de la
                                                    fenetre*/
                info[]="Nous avons 50 lignes de texte"; /*String
                                                    pour la ligne d'information*/

    wind_get(DESK,4,&x1,&y1,&w1,&h1); /*indiquer la surface de
                                     travail du desktop*/
    wind_calc(1,WI_KIND,x1,y1,w1,h1,&x2,&y2,&w2,&h2); /*taille
    maxi de la surface de travail dans la fenetre*/
    wind_calc(0,WI_KIND,x2,y2,w2,h2,&x3,&y3,&w3,&h3); /*surface
    totale de travail dans la fenetre*/
    wi_handle=wind_create(WI_KIND,x3,y3,w3,h3); /*reserver de la
    place pour la fenetre*/
    wind_set(wi_handle,2,name,0,0);      /*ligne de titre*/
    wind_set(wi_handle,3,info,0,0);      /*ligne d'information*/
    wind_open(wi_handle,x3,y3,w3,h3);   /*ouvrir la fenetre*/
    pxyarray[0]=x2;                      /*reprendre les coordonnees dans le*/
    pxyarray[1]=y2;                      /*champ rectangulaire defini*/
}

```

```

pxyarray[2]=(x2+(w2-1));
pxyarray[3]=(y2+(h2-1));
clear_window();          /*vider le contenu de la fenetre*/
}

new_top()
{
    int  posnew;          /*nouvel index*/
    long posact;         /*position actuelle du curseur*/

    posact=msgbuff[4];
    posnew=(int) (posact*(quantite-GLIGNES)/1000);
/*calculer le nouvel index */
wind_set(wi_handle,WF_VSLIDE,((int)posact),0,0,0); /*determiner
la position actuelle du curseur */
    top=posnew;          /*sauver l'index de ligne*/
    clear_window();      /*vider le contenu de la fenetre*/
    show_text();         /*insérer le texte*/
}

fleche()
{
    switch(msgbuff[4])   /*où s'est produit le clic?*/
    {
        case 0:          /*clic sur la barre de deplacement
                           du haut*/
            top-=20;      /*nouvel index de la lere ligne
                           visible*/

            if(top<0)
                top=0;
            set_vslid();  /*repositionner le curseur*/
            break;

        case 1:          /*clic sur la barre du bas*/
            top+=21;      /*nouvel index de la lere ligne
                           visible*/

            if(top>(quantite-20))
                top=(quantite-20);
            set_vslid();  /*repositionner le curseur*/
            break;

        case 2:          /*clic sur la fleche superieure*/
            if(top>0)     /*debut de texte pas encore atteint*/
            {
                --top;    /*nouvel index lere ligne visible*/
                set_vslid(); /*repositionner le curseur*/
            }
            break;

        case 3:          /*clic sur la fleche inferieure*/
            if(top<(quantite-GLIGNES)) /*fin de texte pas encore
            atteinte*/
            {
                ++top;    /*nouvel index de lere ligne visible*/
            }
        }
    }
}

```

```

        set_vslid(); /*repositionner le curseur*/
    }
    break;
}
}
set_vslid()
{
    int poscurseur;

    clear_window(); /*nouvelle position du curseur*/
    show_text(); /*vider le contenu de la fenetre*/
    /*inserer le texte dans la fenetre*/
    /*calculer la nouvelle position du
    curseur*/

    poscurseur=(int) (1000*((double)top/((double) (quantite-GLIGNES))));
    wind_set(wi_handle,WF_VSLIDE,poscurseur,0,0,0); /*nouvelle
    position du curseur*/
}
show_text()
{
    register int tt, /*index courant*/
    zz;

    for(tt=top,zz=0;(zz<=GLIGNES)&&(tt<=quantite);++zz)
        /* afficher les nouvelles lignes visibles à chaque fois*/
        {
            v_gtext(vdi_handle,
            (SP(1)+x2),(LG(zz)+y2),((textarray+tt)->lignetexte));
            +tt;
        }
}
clear_window()
{
    vsf_color(vdi_handle,0); /*selection de la couleur sur
    'blanc'*/
    vsf_interior(vdi_handle,1); /*type de remplissage*/
    v_bar(vdi_handle,pxyarray); /*remplissage du rectangle*/
}
close_window()
{
    wind_close(wi_handle); /*fermer la fenetre*/
    wind_delete(wi_handle); /*effacer la fenetre*/
}

```

6.5. Confection d'un accessoire

Un accessoire est en fait un programme un peu particulier, différent des programmes 'normaux'. En effet, la première caractéristique d'un accessoire consiste à porter un nom se terminant par l'extension .ACC. Les programmes munis de cette extension sont chargés automatiquement lors de l'allumage ou de la réinitialisation de l'ordinateur. Ce chargement peut se faire soit depuis la disquette-système soit depuis la partition C d'un disque dur; le système d'exploitation se charge de mémoriser le programme-accessoire dans un buffer prévu à cet effet puis de le lancer. Le GEM accepte au maximum 6 accessoires, accessibles par le menu déroulant intitulé 'bureau' tout en haut à gauche de l'écran. Deuxième grande différence entre un programme-accessoire et un programme normal: l'accessoire, après son chargement et sa mise en fonction, s'oriente vers une boucle sans fin où il attend un appel éventuel de la part de l'utilisateur, appel auquel il devra répondre. Cette attente est programmée à l'aide de la fonction 'evnt_multi' ou 'evnt_mesag', si bien que tout se déroule en quelque sorte en arrière-plan, sans faire obstacle au bon fonctionnement d'un programme ordinaire. Les accessoires sont généralement des programmes-auxiliaires, comme par exemple un disque RAM, un programme de recherche de fichier, un agenda etc... que l'on peut appeler depuis n'importe quel autre programme ou logiciel disposant d'une barre de menus. Une fois appelé, l'accessoire remplit son rôle pour ensuite reprendre sa position d'attente et permettre à l'utilisateur de reprendre son travail là où il l'avait laissé.

En résumé, les caractéristiques essentielles des accessoires sont les suivantes:

- l'extension .ACC à la fin de son nom
- leur chargement et lancement automatiques lors de l'initialisation de l'ordinateur
- la possibilité de les appeler depuis un menu
- ils sont résidents en mémoire, ce qui veut dire qu'ils ne peuvent pas être écrasés par d'autres programmes (ils ont leur propre buffer)
- ils ne prennent jamais fin car ils se composent d'une boucle sans fin
- ils ne doivent pas contenir de menu déroulant.

Voici les étapes les plus importantes pour programmer un accessoire:

- l'accessoire doit être déclaré auprès de l'AES par 'appl_init'
- si nécessaire, il faut préciser le workstation-handle au moyen de 'graf_handle'

- si nécessaire, il faudra ouvrir une unité virtuelle à l'aide de 'v_opnvwk'
- il faut inscrire son nom servant à le sélectionner dans le menu déroulant à l'aide de 'menu_register' et sauvegarder dans une variable le numéro d'identification ainsi obtenu.
- la dernière étape consiste à passer dans une boucle sans fin

Pour mieux illustrer tout cela, nous avons écrit un exemple de programme. Cet accessoire se borne à faire apparaître un message d'alarme qui disparaît lorsqu'on clique sur la touche OK, si bien que l'accessoire attend un nouvel appel.

Nous espérons vous avoir fourni maintenant assez d'explications au sujet de la programmation d'un accessoire.

```

#include <stdio.h>                DEMOACC.C
#include <osbind.h>
#include <gemdefs.h>
#include <gembind.h>
#include <obdefs.h>

#define FALSE 0
#define TRUE  -1

int  contrl[12],                /*GEM-Arrays pour VDI et AES*/
     intin[128],
     ptsin[128],
     intout[128],
     ptsout[128],
     work_in[11],
     work_out[57],
     msgbuff[8],
     vdi_handle,                /*Handle du VDI*/
     phys_handle,              /*handle du workstation*/
     appl_id,                   /*identification de l'application*/
     menu_id;

main()
{
    appl_id=appl_init();/*déclarer le programme*/
    open_vwork();       /*initialisations*/
    menu_init();        /*déclarer l'accessoire*/
    main_loop();        /*boucle sans fin*/
}

open_vwork()
{
    register int i;        /*index courant*/
    int dummy;            /*valeur dummy*/
    /**/
    phys_handle=graf_handle(&dummy,&dummy,&dummy,&dummy);

```

```

vdi_handle=phys_handle;
for(i=0;i<10;work_in[i++]=1); /*initialiser le champ*/
work_in[10]=2;
v_opnvwk(work_in,&vdi_handle,work_out); /*ouvrir le 'virtual
screen workstation*/
}
menu_init()
{
    menu_id=menu_register(appl_id," HELLO..."); /*déclarer le
nom de l'accessoire dans le menu*/
    if(menu_id==-1) /*erreur*/
    {
        form_alert
(1,"[1][plus de place dans la barre des menus][ OK ]");
        appl_exit(); /*sortir du programme*/
    }
}
main_loop()
{
    int flag1, /*condition d'interruption 1*/
        flag2; /*condition d'interruption 2*/

while(TRUE) /*boucle sans fin*/
    {
        graf_mouse(0,0L);
        evt_mesag(msgbuff); /*guetteur d'évènements*/
        if((msgbuff[0]==AC_OPEN)&&(msgbuff[4]==menu_id))
/*l'accessoire a été sélectionné */
        {
            /*ici se trouve le programme principal et son appel*/

            form_alert(1,"[3][ HELLO... | pour sortir appuyez |
<RETURN> ][ OK ]");
        }
    }
}

```

Chapitre 7

Trucs et astuces de programmation

Ce chapitre devrait vous faciliter le travail quotidien en vous donnant quelques routines bien utiles dans la programmation et quelques connaissances plus approfondies en matière d'informatique.

7.1 Le clavier

Il n'existe quasiment pas de programme qui puisse se passer de la saisie par le clavier. Pour bien programmer, nous allons vous indiquer quelques possibilités se rapportant à la saisie de données ou au paramétrage des touches.

Nous l'avons déjà souvent dit dans cet ouvrage: le clavier de l'Atari ST est doté de son propre processeur, c'est pourquoi on l'appelle 'clavier intelligent'. Ce processeur se charge non seulement de la gestion du clavier mais aussi de celle des signaux provenant de la souris ou du joystick (manette de jeu) ainsi que de l'horloge en temps réel. Toutes ces données sont transmises au processeur principal par le biais d'une interface série après intervention d'un interrupt. Le système d'exploitation de l'Atari ST traite les informations reçues, et les met ainsi à la disposition de l'utilisateur directement ou indirectement par des variables-système.

7.1.1 Les touches spéciales

Il s'agit ici des touches Shift, Caps-lock, Control, Alternate, Clr/home et Insert. C'est la fonction BIOS(11,modus) du système d'exploitation qui permet de savoir si une ou plusieurs touches ont été actionnées. En Basic-GFA, l'appel de cette fonction s'écrit:

```
Son.tast%=Bios(11,-1)
```

En langage C, on écrit:

```
char valeur;
valeur = Kbshift(-1);
```

ou encore:

```
char valeur;
valeur = bios(11,-1);
```

et en Assembleur:

```
move.w #-1,-(sp)      [ou move.w #$FFFF,-(sp)]
move.w #11,-(sp)
trap #13
addq.l #4,sp
move.b D0,valeur
```

Comme c'est le plus souvent le cas dans les routines du système d'exploitation, la valeur -1 sert à demander l'état actuel alors que les autres valeurs sont reprises comme étant des valeurs nouvelles. Il est donc possible de simuler l'appui sur des touches spéciales.

Le résultat s'incarne dans une valeur dans laquelle chaque Bit possède sa propre signification:

Bit	Signification
0	touche shift droite
1	touche shift gauche
2	touche control
3	touche alternate
4	touche caps-lock

5	touche clr/home
6	touche insert

Les autres bits sont dépourvus de signification. Cette façon d'appeler n'est pas forcément la plus rapide; de toute façon, elle n'est pas utilisable dans une routine-interrupt. En regardant de plus près ce qui se passe après cet appel, on constate que la valeur est transmise au registre des retours D0 à partir d'une variable-système non enregistrée.

On peut tout aussi bien prendre cette valeur directement là où elle se trouve en mémoire, comme le font de nombreux programmes. C'est le cas par exemple dans l'éditeur du Basic-GFA Version 2.0, qui s'interrompt lorsqu'on actionne simultanément les trois touches <CONTROL><SHIFT><ALTERNATE>.

Malheureusement, l'ancien TOS n'a pas la même adresse que le nouveau Blitter-TOS, si bien qu'il vous appartient de fixer vous-même la valeur adéquate selon votre ordinateur. Nous avons déjà montré comment on s'y prend pour solutionner rapidement ce petit problème (voir le programme "freezer"). Voici un bref rappel des adresses:

Ancien TOS du 06.02.1986	\$E1B (3611)
Blitter TOS du 22.04.1987	\$E61 (3681)

Les adresses impaires vous indiquent que cette variable doit être lue comme un octet (sinon il se produira une erreur d'adresse). En Basic-GFA, on écrit:

```
valeur%=Peek (&HE1B)
```

en assembleur:

```
move.b $E1B, valeur.
```

En langage C, vous devrez recourir à un pointeur char que vous placez sur la place concernée de la mémoire:

```
char *pointeur,  
      valeur;  
pointeur=0xE1B;  
valeur=*pointeur;
```

Lorsque vous utilisez la place-mémoire, ayez toujours présent à l'esprit le fait qu'elle peut se voir attribuer une autre place en cas de nouvelle version du TOS.

7.1.2 Les touches de fonction et les touches fléchées

Le système d'exploitation distingue les touches ordinaires des touches spéciales. Les touches ordinaires sont dotées d'une valeur ASCII; vous trouverez une table de ces valeurs à la fin de ce volume. Votre Atari ST possède plus de touches que nécessaire pour le standard ASCII. Vous disposez par exemple de 10 touches de fonction.

Il faut d'abord savoir que l'ordinateur ne distingue pas les touches selon leur valeur ASCII mais selon leur code Scan. Le mot 'scan' signifie à peu près explorer (examiner, repérer) et désigne le processus qui se déclenche dès que l'utilisateur a appuyé sur une touche. Les touches sont rangées dans une sorte de matrice dont les colonnes sont régulièrement balayées. La valeur propre à chaque signe est transmise du clavier à l'ordinateur. On reçoit en retour un mot-long grâce aux routines du système d'exploitation chargées des touches du clavier.

Dans le mot suivant se trouve la valeur ASCII; si elle n'existe pas, on trouve la valeur 0. En tout cas, la première valeur représente toujours le code Scan du caractère, si bien que le système d'exploitation peut toujours tourner grâce à cette valeur. On peut même ainsi distinguer les touches des chiffres du bloc numérique de celles formant la ligne supérieure du clavier. Voici un exemple en Basic-GFA:

```

valeur%=Gemdos(&H7)
scan%=(valeur% div 65536) and 255

```

pour écrire le même processus en langage C:

```

long valeur;
valeur=Crawcin()
valeur/=65536;
valeur&=255;

```

et en Assembleur:

```

move.w    #7, -(sp)
trap      #1
addq.l    #2, sp
asr.l     #8, D0

```

asr.l	#8, D0
and.l	#\$FF, D0
move.w	D0, valeur

Vous trouverez, dans les annexes, les valeurs propres à chaque caractère dans le tableau des codes Scan.

Nous n'oublions pas de vous signaler aussi une autre possibilité offerte par le Basic-GFA pour vous servir de toutes les touches sans appeler le système d'exploitation: la fonction Inkey\$. Normalement, cette fonction vous donne un string d'un caractère avec la lettre ou le chiffre correspondant; s'il n'y a pas de code ASCII, elle vous donne un string de deux caractères, le premier étant un octet nul et le deuxième correspondant au code Scan:

```
Do
  Signe$=Inkey$
  Exit if Signe$<>" "
Loop
'
If asc(Signe$)=0
  Scanc%=ASC(Right$(Signe$,1))
```

7.1.3 Pour modifier le jeu des touches

Le système d'exploitation de l'Atari ST vous offre la possibilité de modifier la répartition des caractères sur le clavier. Si vous en avez envie, vous pouvez ainsi vous confectionner un clavier espagnol ou allemand, ou même un clavier rangé par ordre alphabétique. On peut aller plus loin encore, et replacer les capuchons des touches à la nouvelle position désirée: il suffit de tirer dessus prudemment et de les réenfoncer à leur nouvelle position.

Nous allons d'abord examiner les principes théoriques de cette réalisation. L'Atari ST distingue entre trois claviers: le clavier de saisie ordinaire, le clavier lié à la touche SHIFT, et le clavier lié à la touche CAPSLOCK. Le premier contient par exemple toutes les lettres minuscules et certains signes de ponctuation, le deuxième contient les majuscules et les chiffres, et le troisième toutes les majuscules en conservant les autres touches sur leur position ordinaire. Vous trouverez la répartition exacte des jeux de caractères dans "Le grand livre de l'Atari ST" chez le même éditeur. Un jeu de caractères se compose fondamentalement de 128 octets alignés dans l'ordre des codes scan avec les valeurs ASCII correspondantes. Il est donc possible d'attribuer un caractère quelconque aux touches de fonction.

Pour procéder à ces modifications, le système d'exploitation vous offre la fonction Xbios(16,normal,shift,capslock). La fonction délivre un pointeur orienté vers une table de vecteurs contenant les adresses des trois jeux de caractères. Pour conserver l'un ou l'autre des jeux de caractères standards, il vous suffit d'attribuer la valeur -1.

Voici ce que vous devez écrire en Basic-GFA:

```

Normal$=""
Shift$=""
Caps$=""
'
For I%=1to128
  Read Touche%
  Normal$=Normal$+Chr$(Touche%)
Next I%
'
For I%=1to128
  Read Touche%
  Shift$=Shift$+Chr$(Touche%)
Next I%
'
For I%=1to128
  Read Touche%
  Caps$=Caps$+Chr$(Touche%)
Next I%
'
Adr%=Xbios(16,L:Varptr(Normal$),L:Varptr(Shift$),L:Varptr(Caps$))
Data ... (128 Valeurs pour Normal)
Data ... (128 Valeurs pour Shift)
Data ... (128 Valeurs pour Caps)

en langage C:

char normal[128],
      shift[128],
      caps[128];
char *z_nor,
      *z_sh,
      *z_ca;
long adr;
(il faut d'abord remplir les champs!!!)
z_nor = normal;
z_sh = shift;
z_ca = caps;
adr = Xbios(16,z_nor,z_sh,z_ca);

et en Assembleur:

pea    caps

```

```

pea      shift
pea      normal
move.w  #16,-(sp)
trap    #14
add.l   #14,sp
.data
normal: .dc.b ... 128 valeurs pour normal
shift:  .dc.b ... 128 valeurs pour shift
caps:   .dc.b ... 128 valeurs pour capslock

```

7.1.4 Une mémoire-tampon pour le clavier

Après vous avoir donné toutes ces précisions sur les touches, nous allons maintenant approfondir les principes fondamentaux de leur traitement par l'ordinateur. Vous avez certainement déjà remarqué que votre ST mémorise tous les appuis que vous effectuez sur les touches, même si l'exécution de l'action ainsi engendrée n'intervient qu'après un certain délai. Essayez par exemple de maintenir enfoncée assez longtemps la barre des espaces, puis cliquez sur un nom de fichier pour l'afficher à l'écran: vous verrez que l'affichage va défiler sans arrêt pendant un certain temps, à supposer d'ailleurs que votre fichier soit assez long!

L'Atari ne va chercher une touche que lorsque le programme en cours le réclame par le biais d'une routine du système d'exploitation. Toutes les touches actionnées avant l'intervention de cette routine sont mémorisées dans un buffer spécial, de telle sorte que leur action n'est pas perdue. Dans l'Atari, ce buffer compte 256 octets et peut contenir 64 caractères.

Commençons donc par rechercher l'adresse de ce buffer, en nous aidant de la fonction Xbios(14,device) du système d'exploitation. Le paramètre device vous indique qu'il existe d'autres buffers du même type pour d'autres périphériques de saisie:

```

device    périphérique de saisie
-----
0         RS232
1         Clavier
2         MIDI

```

Les principes fondamentaux que nous allons vous présenter en nous appuyant sur l'exemple du clavier, s'appliquent aussi aux deux autres périphériques.

Après avoir appelé cette fonction, nous recevons un pointeur dirigé vers le champ d'informations, qui contient les informations suivantes, énumérées dans l'ordre:

Adresse du buffer	(long)	(\$COE)	(\$C84)
Taille du buffer	(word)	(\$100)	(\$100)
Head Index	(word)	(????)	(????)
Tail Index	(word)	(????)	(????)
Low water mark	(word)	(\$40)	(\$40)
High water mark	(word)	(\$C0)	(\$C0)

Nous recevons bien plus d'informations que nous ne le désirions, car le buffer du clavier est structuré comme un buffer circulaire. Ce terme signifie que l'écriture reprend au début du buffer lorsque celui-ci a été vidé une première fois. C'est ce que vous indiquent les deux variables 'head' (tête) et 'tail' (queue). Ceci vous explique aussi pourquoi vous ne recevez pas le contenu du buffer, puisqu'il change quasiment à chaque pression sur une touche. La première colonne de valeurs entre parenthèses se rapportent à l'ancien TOS et la deuxième au nouveau blitter-TOS.

Le processus se déroule de la façon suivante: lorsque le buffer est vide, les deux variables 'head' et 'tail' ont la même valeur. Lorsque vous appuyez sur une touche, le système inscrit un mot-long à partir de l'adresse du buffer plus 'head', augmenté de quatre. Le processus est semblable lors de la lecture d'un enregistrement à partir du buffer: le système lit un mot-long à partir de l'adresse du buffer plus 'tail', augmenté de quatre. Il n'y a que deux cas particuliers: primo, lorsque le buffer est vide et que 'head' et 'tail' ont la même valeur, si bien que le système n'accepte pas les touches; secondo lorsque le buffer est plein, 'head' devrait alors être placé sur 'tail' (la tête mord la queue). Le système d'exploitation ne tient pas compte de la touche appuyée, il s'écroule.

Les deux dernières variables, 'low water mark' et 'high water mark', ne sont pas utilisées pour la gestion du clavier: elles servent à contrôler le transfert de données par l'interface RS232, de la même manière qu'un indicateur de niveau dans un dispositif de remplissage automatique de cuves. Le transfert se met en route lorsqu'on est en-dessous de 'low water mark' et s'arrête lorsqu'on dépasse 'high water mark' aussi longtemps qu'il le faut pour que le système ait le temps de traiter assez de caractères pour pouvoir repasser en-dessous de 'low water mark'.

Nous allons maintenant nous intéresser à la structure que prend, dans le buffer, l'enregistrement correspondant à une touche. Nous savons déjà que ces enregistrements prennent la forme d'un mot-long. Dans la partie supérieure se trouve à nouveau le code Scan, et dans la partie inférieure le code ASCII de la touche qui vient d'être appuyée.

Voilà qui nous offre une multitudes de possibilités! En premier lieu, nous pouvons porter la taille de ce buffer à 32 K-octets, et donc lui donner assez de place pour contenir 8000 caractères. Il suffit de réserver un espace suffisant en mémoire, puis de transmettre au pointeur (\$DB0 ou *C76) un champ d'informations adéquat:

```

move.l #tbuf_new,$DB0      (Mega ST: move.l #tbuf_new,*C76)
.data
tbuf_new: .dc.l t_buf
          .dc.w $7D00
          .dc.w $0
          .dc.w $0
          .dc.w $40
          .dc.w $C0
.bss
t_buf    .dc.l 8000

```

On peut aussi avoir intérêt, par exemple dans une routine interrupt, à lire les caractères directement dans le buffer du clavier. Nous connaissons déjà le chemin: il s'agit de lire une valeur longue à partir de l'adresse du buffer plus 'tail', dans le cas où 'tail' est différent de 'head'; on augmente 'tail' pour ensuite contrôler si l'on a pas déjà atteint la fin du buffer. Si tel est le cas, il faut mettre 'tail' sur 0. Le fait d'augmenter 'tail' ne suffit pas toujours à communiquer au système d'exploitation le caractère lu; c'est pourquoi il est recommandé d'effacer ensuite la valeur longue du buffer.

Ces explications vous permettent naturellement de vider le buffer du clavier. Il est par exemple intéressant de mémoriser les appuis sur les touches dans un buffer lorsqu'on travaille avec un traitement de texte ou un éditeur, car il arrive fréquemment qu'on laisse le doigt trop longtemps sur la touche BACKSPACE ou DELETE et que l'on détruit ainsi plus de données qu'on ne le désirait. Il suffit pour cela de créer un petit buffer ou de réécraser l'ensemble du buffer en le remplissant de 0 pour ensuite effacer 'head' et 'tail'.

Encore un petit truc: dans la variable-système, à l'adresse \$484 (identique sur le Méga ST), le bit 3 vous permet (lorsqu'il est positif) d'obtenir le statut des touches spéciales dont nous venons de parler. Il suffit ensuite d'appeler la touche pour obtenir non seulement son code ASCII et Scan mais aussi son statut de touche spéciale dans les bits 24 à 31. Attention: si vous voulez modifier les bits, votre processeur doit se trouver en mode superviseur, sinon vous recevrez des bombes:

```

clr.l    -(sp)      *pour passer en mode superviseur
move.w   #$20,-(sp)
trap     #1
addq.l   #6,sp

```

```

bset      #3,$484    *mettre le bit en positif
move.l   D0,-(sp)   *repasser en mode usuel
move.w   #$20,-(sp)
trap     #1
addq.l   #6,sp

```

7.2 Les fichiers batch

L'Atari possède un bureau graphique qui simplifie considérablement le travail courant. On peut par exemple facilement recopier ou formater les disquettes, afficher les répertoires etc. Dans les autres ordinateurs domestiques (et même dans les ordinateurs sous MS-DOS) tout ceci doit se faire en entrant des ordres au clavier. Malgré tous ses avantages, l'environnement graphique a aussi des inconvénients: il limite les possibilités d'action de l'utilisateur, il lui arrive même de compliquer les choses, comme par exemple lorsqu'il s'agit de compiler un programme. Certes, on peut entrer les paramètres à l'aide des boîtes de dialogue dans les programmes portant l'extension .TTP (TOS Takes Parameter), mais le processus de compilation de plusieurs programmes (passes) consécutivement doit se faire 'à la main'.

C'est précisément là qu'intervient notre programme-batch. Son rôle consiste à reprendre des informations dans un fichier en ASCII, lancer un programme et transmettre les paramètres nécessaires. Vous trouverez toutes les explications un peu plus loin, au paragraphe 'chargement de programmes'.

Quelle est la composition du fichier ASCII? Remarquons tout d'abord que chaque ligne du fichier ASCII lance un programme et transmet les paramètres présents. Ce fichier-batch permet donc de lancer plusieurs programmes à la suite l'un de l'autre. Voici la syntaxe d'une ligne:

```
Nom du programme (sans extension .PRG) paramètre paramètre ...
```

Chaque paramètre est séparé du précédent par un espace blanc: on peut écrire autant de lignes qu'on le désire l'une au-dessous de l'autre. Nous vous indiquons encore un moyen pour garantir une utilisation plus universelle du fichier batch. Par exemple, pour ne pas être limité à un seul nom de programme durant le processus de compilation, vous avez la possibilité de n'entrer certains paramètres qu'après avoir appelé le fichier batch. Dans le fichier ASCII, ces paramètres seront signalés par % ou par un numéro de 1 à ... Voici par exemple le fichier batch contenu dans les fournitures Atari standards pour le compilateur du langage C:

```

rm %1.0
rm %1.TOS
cp68 %1.c %1.i
c068 %1.i %1.1 %1.2 %1:3 -f
rm %1.i
cl68 %1.1 %1.2 %1.s
rm %1.1
rm %1.2
écrit3 %1
as68 -l -u %1.s
fas tlink %1.tos=%1,bible
rm %1.s
wait

```

Une petite ombre au tableau vient un peu gâcher votre plaisir: le programme-batch livré avec l'Atari ne peut traiter les programmes faisant appel au GEM. Si vous tentez l'expérience, soit vous plantez le système, soit vous renoncez à vous servir de la souris. On peut le regretter, d'autant plus qu'il serait bien pratique de disposer d'un petit fichier pour lancer le Basic-GFA suivi automatiquement de son compilateur (en version 2.xx).

7.3 Les programmes-résidents et le vbl-queue

Pour expliquer ce qu'est un programme résident, il faut commencer par expliquer ce qu'est un programme non-résident. Une application ordinaire, par exemple un traitement de texte ou un compilateur, n'est chargée dans la mémoire vive que lorsqu'elle tourne: elle se trouve en principe dans une mémoire de masse (disquette ou disque dur) et il faut la charger à chaque fois qu'on en a besoin. Mais lorsque vous devez utiliser un programme qu'il faut souvent appeler, son rechargement fréquent depuis une mémoire de masse risque de freiner considérablement l'ordinateur. C'est pourquoi il faut les rendre 'résidents' dans la mémoire de travail: il suffit alors de les lancer en cas de besoin, car ils sont déjà chargés.

Dans ce chapitre, nous allons vous initier à différentes techniques qui vous permettront ensuite d'écrire vous-même des programmes résidents. Les accessoires, qui sont en fait une catégorie de programme-résident, ont déjà fait l'objet d'un chapitre particulier, et nous n'y reviendrons pas. A quoi peuvent bien vous servir des programmes résidents? N'ayez crainte! vous allez vous en rendre compte par vous-même!

Lorsqu'on installe des programmes-résidents, il faut particulièrement faire attention à deux choses:

- comment mon programme va-t-il se charger dans la mémoire vive?
- qui va appeler mon programme, et pour quoi faire?

L'Atari vous offre deux réponses à la première question, chacune ayant ses avantages et ses inconvénients. Voilà d'abord la fonction GEMDOS 'keep process': elle met fin au programme en cours, mais sans l'effacer. A elle seule, elle peut déjà suffire pour faire tout ce que nous nous proposons de réaliser. Le seul inconvénient étant que les programmes restant résidents prennent de la place dans la mémoire vive, place qui n'est plus disponible pour les autres programmes que vous voudriez faire tourner. A la fin de ce chapitre, nous vous expliquerons comment contourner cette difficulté grâce à une deuxième possibilité.

Pour l'instant, regardez de plus près le mini-programme ci-dessous:

```
move.w  #$31,-(sp)    ; numéro de la fonction pour 'keep process'  
trap    #1             ; Appel du GEMDOS.
```

Ces deux lignes sont déjà à elles-seules un vrai programme-résident, totalement dépourvu de sens, mais un vrai programme-résident tout de même: il ne fait rien d'autre que se terminer sans arrêt. Comme il utilise pour cela le numéro de fonction \$31, le système d'exploitation ne l'efface pas et le laisse dans la RAM jusqu'à ce qu'il en soit vidé par un reset. Il est clair qu'il manque des éléments pour lui donner un sens valable.

Pour être plus exact, il manque même deux éléments. Premièrement, il manque le programme à exécuter après le chargement (et qui prend fin par l'appel de 'keep process'; cette partie s'appelle l'initialisation); deuxièmement il manque aussi la routine qui doit figurer dans le système. L'initialisation ne se fait qu'une fois lors du chargement du programme, alors qu'on doit pouvoir lancer le deuxième élément aussi souvent que nécessaire: c'est cette deuxième partie seulement que nous désignons sous le terme de 'routine résidente', bien que l'initialisation reste aussi dans la mémoire vive.

En élargissant notre mini-programme à la lumière de toutes ces explications, nous obtenons la structure suivante:

```

Initialisation...
-
-
move.w  #$31,-(sp)      ; keep process
trap    #1

Programme résident...
-
-

```

L'initialisation nous amène tout de suite à la deuxième question: quand allons-nous faire tourner notre programme-résident? Précisons notre question: notre programme-résident sera lancé pendant qu'un autre programme, que nous appelons programme-principal, est en train de tourner. Comment dire à l'ordinateur où se trouve le programme-résident, comment il doit l'appeler, et surtout quand il doit le faire? Pour répondre à ces questions, nous allons recourir à une notion nouvelle (que les professionnels connaissent certainement déjà): le traitement d'exception, que nous appelons tout simplement 'exception'. Ce mot décrit bien ce dont nous parlons: est 'exceptionnel' tout ce qui sort de l'ordinaire. Notre ordinateur fonctionne de façon ordinaire lorsqu'il fait tourner un programme-principal, par exemple un logiciel de traitement de texte. Toute autre procédure représente une exception, le mot n'ayant rien de négatif. Une erreur commise dans un programme provoque déjà l'irruption automatique d'une procédure d'exception, signalée à l'utilisateur par l'apparition de bombes. Il y a aussi des exceptions plus utiles, comme par exemple le système d'exploitation.

Nous allons utiliser à notre profit ce système en définissant comme 'exception' le traitement de notre propre programme-résident (personne ne peut nous l'interdire). Ces exceptions peuvent être de nature très diverse: il peut d'abord s'agir de signaler des erreurs dans un programme; il peut s'agir de procédures appelées à intervalles réguliers ou enfin de procédures appelées par une commande-machine.

Pour bien clarifier ce dont il est question, précisons qu'on appelle 'interrupt' ce qui déclenche un traitement d'exception. La procédure d'exception s'appelle 'exception'. En termes mathématiques, on peut dire que l'interrupt est la condition nécessaire pour provoquer une exception. La formulation n'est pas très correcte, mais elle rend bien ce que nous voulons dire. Pour tous nos lecteurs qui s'intéressent de près au langage-machine, nous allons détailler ce qui se passe lors d'une procédure d'exception. Le processeur passe en mode superviseur pour pouvoir contrôler l'ensemble du système. Après quoi, le compteur du programme (un pointeur désignant la commande qui devait normalement être exécutée) ainsi

que le registre des états sont enregistrés dans la pile superviseur. Un des 256 vecteurs d'exception permet ensuite de passer à la routine que l'interrupt doit traiter. Le numéro de vecteur affecté à l'exception est multiplié par quatre et interprété ensuite comme adresse sur le vecteur. Le numéro dépend du type d'interrupt demandé.

7.3.1 Le retour-image (VBL)

Nous venons de parler d'intervalles réguliers, ce qui présuppose l'existence d'une sorte de métronome activant le programme-résident à chaque fois qu'il le faut. Ce métronome se nomme une 'horloge', un 'timer'. Ce n'est pas obligatoirement une véritable horloge, car le moniteur peut le plus souvent remplacer un timer. La puce-vidéo se trouvant dans l'ordinateur engendre une interruption à chaque retour de ligne et à chaque retour de l'image par le rayon des électrons. Selon le type de votre moniteur (couleur ou monochrome), ce retour de l'image se produit 50 ou 71 fois par seconde; le retour de ligne se produit toutes les 35 à 64 microsecondes. Nous étudierons plus loin les autres sources possibles d'interruption.

Le programmeur décide de la cadence à laquelle il veut que soit appelé son programme-résident. Dans la plupart des cas, il suffira d'utiliser le retour d'image comme horloge. C'est ce retour d'image que nous appellons le VBL (vertical blank). Il est d'ailleurs prudent de ne pas appeler trop souvent le programme-résident, car on risquerait de ne plus laisser assez de temps au programme-principal pour exécuter le traitement dont il est chargé. La pratique montre que le VBL suffit amplement dans la grande majorité des cas: il ne consomme pas beaucoup de temps-calcul et sa cadence est déjà assez élevée pour ne plus être perceptible par l'utilisateur.

En tant que programmeurs, nous ne sommes d'ailleurs pas les seuls à avoir recours ainsi au VBL: l'Atari ST s'en sert aussi pour exécuter toute une série d'actions, dont l'une va retenir plus particulièrement notre attention: le VBL-Queue. 'Queue' fait allusion ici à l'expression 'faire la queue' ou 'attendre son tour'. En effet, il y a dans le VBL-Queue une suite de programmes qui attendent chacun à leur tour d'être appelé automatiquement par chaque VBL (d'où le nom de VBL-Queue).

Examinons tout cela de plus près: notre ordinateur dispose d'un tableau qui peut contenir jusqu'à 8 adresses de début de routines à exécuter à chaque VBL. Voilà qui est très pratique pour le programmeur: il lui suffit d'enregistrer l'adresse de son programme sous une adresse libre dans le VBL-Queue, et le tour est joué! Il ne nous reste plus qu'à préciser l'endroit où se trouve cette liste et comment on s'y inscrit.

Il existe pour cela deux variables-système: la première s'appelle 'nvbls' et se charge de vous communiquer le nombre d'adresses qu'il est possible d'enregistrer dans le VBL-Queue; la deuxième s'appelle '_vblqueue': elle pointe sur une liste qui comprend des mots longs 'nvbls' et contient les adresses des routines à exécuter. Nous devons donc examiner la liste sur laquelle pointe '_vblqueue' pour voir où il y a encore une adresse libre. L'adresse est libre si elle contient un 0. Si nous ne trouvons pas d'adresse libre, nous devons interrompre notre recherche (nous verrons plus loin comment nous y prendre dans ce cas). Attention: vous ne pouvez accéder aux variables-système qu'en mode superviseur.

Nous vous indiquons ci-dessous la façon dont commence le plus souvent un programme-résident. Vous pouvez utiliser ce début de programme comme séquence d'initialisation dans vos propres programmes: il vous suffira ensuite d'y 'accrocher' les routines voulues, qui seront appelées par le VBL et doivent se terminer par RTS.

```

gemdos      =      1
bios        =      13
xbios       =      14
keep        =     $31
superexec   =      38
_vblqueue   =     $456           ;pointeur vers une liste des
                                ;routines VBL
nvbls       =     $454           ;nombre de routines VBL
                                ;possibles
fréquence   =      35           ;valeur Init pour le
                                ;compteur, voir la
                                ;description

text

move.l 4(sp),a0           ;réserver la place mémoire
                           ;nécessaire

move.l #$100,d6
add.l 12(a0),d6
add.l 20(a0),d6
add.l 28(a0),d6

pea    init              ;initialisation en
                           ;superviseur

move.w #superexec,-(sp)
trap   #xbios
addq.l #6,sp

clr.w  -(sp)             ;mettre le programme en
                           ;résident

move.l d6,-(sp)
move.w #keep,-(sp)

```

```

init      trap    #gemdos
          move.w  nvbls,d0          ;nombre des routines VBL
          lsl.w  #2,d0             ;multiplié par 4
          move.l  _vblqueue,a0     ;liste des adresses de
                                   départ
search    clr.w   d1
          tst.l   (a0,d1)          ;si libre, vous trouvez
                                   ici un zéro
          beq.s  found
          addq.w #4,d1             ;sinon, scruter la
                                   prochaine adresse
          cmp.w  d0,d1             ;toutes les adresses
                                   scrutées?
          bne.s  search
          illegal                ;oui, donc menace de
                                   bombes
found     move.l  #vbl,(a0,d1)     ;vecteur vers les
                                   routines VBL
          clr.w  counter           ;mettre le compteur à 0
rts
vbl       subi.w #1,counter
          bpl.s  fin
          ; votre routine suit ici
end       rts
          end

```

Vous devriez maintenant, à l'aide de ces quelques lignes, installer vos propres routines sans aucun problème. Notez que les routines VBL tournent toujours en mode superviseur. Dans vos programmes, vous pouvez modifier tous les registres du 68000. Il est par contre plus prudent de ne pas faire intervenir le système d'exploitation, car ceci mènerait presque sûrement à de violentes interférences avec les logiciels en cours, et vous savez que tout cela risque fort de se terminer par quelques bombes sur votre écran... Si vous êtes vraiment obligé d'attendre l'action d'une touche quelconque, vous devez interpellier directement les adresses correspondantes et courir le risque de provoquer des incompatibilités. Tant que vous êtes un programmeur-débutant, essayez d'éviter ce genre de chose. Nous vous recommandons de vous procurer un listing du système en ROM que vous aurez tout le loisir d'étudier sérieusement (par exemple: Le livre du développeur).

A titre d'exemple pratique, nous vous donnons un petit programme d'animation de la palette des couleurs. Ceux qui utilisent des programmes graphiques savent

certainement ce que cela veut dire: une partie des 16 registres des couleurs va prendre la teinte rouge à intervalles réguliers. On n'obtient pas vraiment une image animée (qui bouge), mais ceci suffit déjà pour susciter de très beaux effets - pensez par exemple à la cascade de NeoChrome.

Comme nous voilà devenus exigeants, nous n'allons pas en rester à la rotation régulière des registres des couleurs: nous allons créer un certain nombre de palettes qui interviendront chacune à leur tour. Inconvénient: notre système demande plus de travail. Avantage: nous ne sommes plus limités à 16 couleurs. Même en résolution moyenne, nous pourrions utiliser les 512 teintes possibles, pas simultanément certes, mais c'est déjà un beau résultat. On pourra même réaliser des fondus-enchaînés très doux sans que la résolution n'en souffre.

Le retour d'image est ici le meilleur timer possible: en effet, si on modifie le registre des couleurs lorsque le rayon des électrons est en pleine action, l'image se met à trembler. Que se passe-t-il par contre si nous n'activons pas une nouvelle palette de couleurs à chaque VBL? Pour parer à ces deux risques, nous introduisons un compteur placé sur une certaine valeur lors de l'initialisation du programme. Ce compteur va se 'décrémenter' (c'est-à-dire reculer de une unité) à chaque VBL. Tant qu'il reste supérieur ou égal à zéro, aucune nouvelle palette n'est activée. Une nouvelle palette ne sera chargée que lorsque le compteur aura atteint la valeur -1: il reprendra ensuite sa valeur d'origine.

Naturellement, il faut dans ce programme un tableau contenant les palettes de couleurs souhaitées. Cette liste se termine par -1. Une palette de couleurs se compose de 16 mots. En écriture hexadécimale, cela donne \$ORGB: R, G et B représentent chacun la proportion utilisée en R = Rouge, G = Green (vert) et B = Bleu, et peuvent prendre des valeurs comprises entre 0 et 7. Ceci nous donne bien un maximum de 512 couleurs possibles. Dans le cas d'un moniteur monochrome SM124, seul le bit 0 de la première couleur peut être pris en compte. Lorsqu'il est positif, l'écran s'affiche en inversion-vidéo; ceci limite les possibilités d'animation de l'image. Le programme suivant permet tout de même d'obtenir un bel effet sur un écran SM124.

```

;
; Animation de la palette des couleurs : DEMOVBL.S
; MP 18-01-88
;
gemdos = 1
bios = 13
xbios = 14
keep = $31

```

```

superexec      =      38
_vblqueue     =    $456                ; pointeur sur liste des
                                        routines VBL
nvbls         =    $454                ; nbre de routines VBL
                                        possibles
farbregister   =    $ff8240            ; registre des couleurs
wie_ofst      =      35                ; valeur init. du compteur

                                        section text

                                        move.l  4(sp),a0                ; Réserve place
                                        ; mémoire
                                        move.l  #$100,d6
                                        add.l   12(a0),d6
                                        add.l   20(a0),d6
                                        add.l   28(a0),d6

                                        pea     init                ; Superviseur mode
                                        move.w  #superexec,-(sp)
                                        trap    #xbios
                                        addq.l  #6,sp

clr.w  -(sp)                ; Programme résident
                                        move.l  d6,-(sp)
                                        move.w  #keep,-(sp)
                                        trap    #gemdos

init:
                                        move.w  nvbls,d0                ; nombre de routines VBL
                                        lsl.w  #2,d0                ; * 4
                                        move.l  _vblqueue,a0            ; Liste des adresses de
                                        ; début

search:
                                        clr.w  d1
                                        tst.l  (a0,d1)                ; Libre ? alors 0
                                        beq.s  found
                                        addq.w #4,d1                ; Sinon scruter l'adresse
                                        ; suivante
                                        cmp.w  d0,d1                ;
                                        bne.s  search
                                        illegal                ; Plus de place -> BOMBES

found:
                                        move.l  #vbl,(a0,d1)            ; vecteur vers les
                                        ; routines VBL
                                        lea    palette,a0            ; initialiser le pointeur
                                        move.l  a0,act_palette
                                        clr.w  counter                ; compteur à 0
                                        rts

vbl:
                                        subi.w #1,counter                ;

```

```

        bpl.s   ende           ;
        move.w  #wie_ofst,counter ;

        move.l  act_palette,a0 ; Palette suivante
        tst.w   (a0)           ; -1 ?
        bpl.s   label         ; Non
        lea    palette,a0      ; si oui, reprendre au
                                début

label:   move.l  a0,act_palette
        moveq  #7,d0           ; recopier 8 mots longs
        lea    farbregister,a1
vbl_loop: move.l  (a0)+,(a1)+
        dbra   d0,vbl_loop
        move.l  a0,act_palette ; et noter l'état actuel
ende:    rts

        section data
palette: dc.w   1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        dc.w   0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        dc.w   -1

        section bss
act_palette: ds.l  1           ;
counter:     ds.w  1           ;

        end

```

Les possesseurs d'un moniteur-couleur devront entrer des valeurs adéquates dans la palette. Lorsque vous aurez vu le programme se répéter un certain nombre de fois, vous aspirerez certainement à y mettre fin: il faut pour cela effacer la mention de ce programme dans le VBL-Queue (ce qui revient à mettre un zéro). C'est pourquoi vous devez, lors de l'initialisation, conserver une trace de l'adresse sous laquelle vous avez inscrit votre routine. C'est là que vous re-saisirez un zéro (mot long!) lorsque vous aurez assez admiré votre programme. Pour éviter tout malentendu, sachez que vous pouvez faire usage de ce VBL-Queue dans les programmes non-résidents, mais il vous faudra absolument conserver quelque part la mention de l'adresse sous laquelle vous travaillez pour pouvoir la remettre à zéro. Sinon, vous risquez d'avoir une adresse désignant un programme qui n'existe plus.

Il ne s'agissait ici que de l'une des multiples utilisations possibles du VBL-Queue, que nous retrouverons encore souvent par la suite. Pour tous nos lecteurs qui désirent travailler en professionnels, nous allons tout de même dès à présent expliquer comment il faut s'y prendre lorsque les huit adresses sont occupées dans le VBL-Queue. Il est tout à fait légitime de modifier les variables-système. Nous pouvons donc réserver une place en mémoire pour des (en principe 9) mots longs

(nvbls + 1) et y recopier les adresses déjà existantes dans une nouvelle liste. Après avoir entré notre dernière adresse, nous pointons '_vblqueue' sur la nouvelle liste et nous augmentons 'nvbls' de une unité. Il faut absolument respecter l'ordre indiqué; si vous intervertissez l'ordre d'exécution des deux dernières étapes, vous pourriez voir se déclencher un VBL (juste après avoir augmenté de 1 le 'nvbls'), mais comme l'ancienne liste ne comprend que 8 adresses, la neuvième étant indéfinie, ça va sentir très fort le roussi et certainement nous amener quelques bombes sur l'écran. Un mot encore au sujet des programmes que vous voulez lancer automatiquement. Si l'un de vos programmes situés dans le dossier AUTO utilise le VBL-Queue, il faut éviter qu'il se trouve en première position dans la file d'attente. En effet, pour des raisons internes, la première adresse est toujours réécrasée après le chargement des programmes figurant dans le dossier AUTO. Pour plus de sécurité, n'utilisez que les adresses libres au-delà de la deuxième position dans la liste, et tout ira bien.

7.3.2 Le retour de ligne (HBL)

Le retour de ligne représente la deuxième source d'interrupt: nous l'étudierons pour que notre étude soit absolument complète, tout en sachant que nous éviterons autant que possible de nous en servir. Il sert surtout dans les programmes de création graphique, et vous trouverez de nombreux exemples dans les ouvrages consacrés à ce sujet.

Le premier problème vient de ce que ce retour de ligne se fait à trois fréquences très différentes selon le type de moniteur utilisé. Dans le cas d'un moniteur monochrome, 400 lignes multipliées par 71 images par seconde, cela nous donne 28400 lignes par seconde ou un HBL toutes les 35 microsecondes. Avec un moniteur couleur, nous avons 200 lignes pour des fréquences de 50 ou 60 Hz. Cela nous donne 10000 lignes/seconde pour 50 Hz ou 12000 pour 60 Hz. Ces valeurs théoriques sont cependant toujours inférieures à la réalité, car nous ne comptons pas le bord de l'image, qui comprend environ 100 lignes pour un moniteur couleur.

Les valeurs exactes n'ont d'ailleurs pas beaucoup d'intérêt; retenez simplement que la répétition du HBL se produit à une cadence inimaginable. Prenons l'exemple des 35 microsecondes: un cycle complet prend 0,125 microseconde dans notre ordinateur, et il faut un minimum de 4 cycles pour exécuter une commande en assembleur. Le 68000 ne peut rien faire en-dessous de ces valeurs minimales. Entre deux HBL, il ne reste donc que le temps d'exécuter 70 commandes rapides en assembleur: le programme-principal se trouve alors très ralenti, voire même stoppé. Vous savez maintenant pourquoi on n'utilise pas le HBL dans les programmes courants: cela consomme trop de temps.

A côté de la création graphique, voilà bien le seul domaine dans lequel on peut trouver une utilité au HBL: perdre du temps. Vous voilà déjà en train de penser que nous avons perdu la tête. Notre idée n'est pas si bizarre. En ralentissant l'affichage, vous vous accordez du temps pour lire soit un message soit un menu; dans un logiciel de création graphique, on obtiendra un effet tout à fait spécial; dans un programme de recherche des erreurs, le ralentissement vous permet de suivre l'affichage en toute tranquillité. Savez-vous par exemple comment on s'y prend pour faire qu'une option de menu inutilisable à un moment donné apparaisse en caractères grisés? Essayez, vous serez bien surpris! Saviez-vous qu'un menu déroulant se ré-enroule de bas en haut et non l'inverse? Même si vous trouvez cela peu intéressant, il n'est pas indifférent de savoir si c'est le curseur horizontal ou vertical qui va se dessiner en premier dans les ascenseurs d'une fenêtre GEM.

Avec le VBL, tout était relativement simple, il suffisait de nous inscrire dans une liste. Vous comprenez tout de suite qu'il n'existe pas de liste HBL. Nous allons utiliser ici un des vecteurs-exceptions dont nous avons déjà parlé. Au début de sa mémoire principale, le ST possède une liste de 256 vecteurs. Un vecteur n'est rien d'autre que l'adresse de début d'une routine quelconque. Le vecteur 28 par exemple désigne la routine lancée à chaque VBL et chargée, entre autres choses, de traiter le VBL-Queue. Vous allez dire que l'ordinateur pourrait fort bien se débrouiller tout seul et sauter directement à la bonne adresse sans l'aide de vecteur. Vous avez raison, mais imaginez un peu ce qui se passerait si on produisait une nouvelle version du système d'exploitation dans laquelle la routine VBL ne serait plus du tout à la même adresse: il vous faudrait carrément un nouveau processeur, puisque l'ancien ne saurait pas où se trouve l'adresse adéquate.

C'est pourquoi il existe des vecteurs réinitialisés juste après l'allumage de l'ordinateur. Dans le VBL, l'ordinateur n'a qu'à vérifier l'adresse se trouvant au vecteur 28 pour s'y rendre. Cette technique nous permet d'ailleurs d'enclencher des routines différentes des routines d'origine: il nous suffit d'écrire, en mode superviseur (!), l'adresse sous laquelle se trouve le vecteur capable de lancer notre routine. Nous devons éventuellement conserver une trace de l'ancienne valeur afin de pouvoir la restaurer lorsque nous en avons terminé. Notons que l'adresse d'un vecteur est le numéro de ce vecteur multiplié par 4.

Pour le HBL, nous utilisons le vecteur numéro 26, qui se trouve à l'adresse \$68. Nous y inscrivons l'endroit où se trouve notre routine HBL. Malheureusement, il se trouve que la plupart du temps l'interrupt HBL est désactivé; le système l'ignore purement et simplement. Les commutateurs chargés de le réactiver se trouvent directement dans le processeur, très exactement dans le registre des états. Nous n'allons pas nous étendre plus longuement sur ce registre: limitons nous à l'examen des bits 8 à 10. Ces bits servent à confectionner ce qu'on appelle le masque d'interrupt, qui peut prendre les valeurs 0 à 7. A quoi cela sert-il? Notre processeur

identifie 7 degrés différents de priorité dans les interrupts (IPL = interrupt priority level). Lorsque survient un interrupt, il n'est exécuté que si son IPL est plus grand ou égal au masque d'interrupt dans le registre des états du 68000. Cela sert à désactiver les interrupts peu importants comme précisément le HBL.

L'Atari ST ne se sert que de 3 des 7 degrés possibles. Le HBL a reçu l'IPL 2, le VBL travaille avec un IPL de niveau 4. Sous IPL 6, on rencontre un composant tout à fait particulier, le MFP, sur lequel nous reviendrons plus tard. Nous avons dit plus haut que le système ignore purement et simplement le HBL: ceci signifie en clair que son masque d'interrupt se trouve sur 3 ou même au-dessus. Pour autoriser le HBL, il nous suffirait de placer le masque sur 2 (en binaire: 010). Nous disons bien 'suffirait', car cet appel manque précisément dans notre programme de démonstration.

Avant de vous perturber un peu plus, étudiez donc le programme-résident ci-après:

```

;DEMOHBL.S
;HBL - démonstration
; 12-02-88 MP
;

gemdos      = 1
keep       = $31
setvector  = 5
bios       = 13

section text

move.l     4(sp),a0           ;place-mémoire nécessaire
move.l     #$100,d6
add.l     12(a0),d6
add.l     20(a0),d6
add.l     28(a0),d6

pea       hbl                ;initialiser le vecteur 26
move.w    #26,-(sp)          ;(IPL 2 = Interrupt HBL)
move.w    #setvector,-(sp)
trap      #bios
addq.l    #8,sp

clr.w     -(sp)
move.l    d6,-(sp)           ;octets utilisés
move.w    #keep,-(sp)       ;laisser le programme
                                résident
trap      #gemdos

```


A quoi sert ce programme? En principe, il ne fait que diriger le vecteur HBL (n 26) vers notre propre routine HBL. Le programme reste résident en mémoire centrale. Notre routine HBL se compose de 31 NOP (No Operation, n'exécuter aucune opération) et d'un RTE qui termine obligatoirement toute procédure d'exception (d'où son nom RTE = Return From Exception). Toutes ensemble, ces commandes nécessitent 148 cycles, ce qui est déjà beaucoup pour un moniteur monochrome. Si vous possédez un moniteur couleur, vous pouvez tranquillement ajouter quelques NOP.

Vous avez certainement remarqué que nous n'avons pas eu besoin de mettre le masque d'interrupt sur 2. Cela n'aurait servi à rien, car le bureau GEM le remet automatiquement sur 3 à chaque fin de programme, ce qui nous aurait barré la route. Heureusement, il tolère la présence d'une routine HBL lors du lancement du programme suivant, si bien que vous ne remarquerez un ralentissement dans votre travail que si vous chargez une nouvelle application. Le temps de chargement d'un programme peut aussi augmenter considérablement: évitez de vous précipiter sur le bouton reset. Si le programme que vous voulez lancer refuse obstinément de se laisser charger, tentez l'opération avec un disque RAM: les routines de disquette sont très exigeantes sur les délais.

Les auteurs du système d'exploitation ont utilisé un petit truc pour que le HBL ne reste actif que s'il est utile après le lancement d'un programme: la routine HBL standard a pour unique tâche de placer le masque d'interrupt sur 3, si bien que ce grand consommateur de temps se désactive lui-même s'il n'est pas remplacé par une nouvelle routine comme par exemple les NOP ci-dessus.

Récapitulons:

- la présence d'une routine HBL ne signifie pas automatiquement qu'elle ait été appelée par un HBL;
- la présence d'un HBL n'est tolérée que si un programme est lancé depuis le bureau GEM ou si le programmeur abaisse le masque d'interrupt jusqu'à une valeur inférieure ou égale à 2;
- la routine HBL standard sert à placer sur 3 le masque d'interrupt;
- une routine HBL doit être extrêmement rapide et se terminer par RTE.

En tenant bien compte de toutes ces caractéristiques, vous ne rencontrerez aucun problème dans la manipulation du HBL.

7.3.3 L'horloge du MFP

Nous n'avons pas encore utilisé une horloge réelle comme source d'interrupt, et c'est précisément ce que nous allons faire maintenant. Le timer en question se trouve dans le MFP 68901 (MFP = multi fonctional peripheral = périphérique multi fonctions). Ce composant gère toutes les interruptions de notre ordinateur qui sont engendrés par des périphériques et se charge aussi d'autres tâches. Le MFP est si complexe que sa description prendrait tout un livre. Nous allons donc nous limiter à l'essentiel: le timer.

Pour être plus exacts, nous devrions tout de suite dire 'les' timers, car le MFP en a quatre, nommés A, B, C et D. Seul le timer A est totalement libre pour l'usage du programmeur, si bien que nous n'avons qu'une horloge à notre disposition - mais attention, quelle horloge! Notre système d'exploitation possède heureusement quelques routines bien efficaces pour programmer le timer, mais ceci ne nous dispense pas totalement de mettre la main à la pâte pour améliorer un peu les choses.

La pulsation de l'horloge du MFP se produit au 2.457.600ième de seconde. On dit aussi que l'horloge est réglée sur 2.4576 MHz. A chaque fois que ce délai est écoulé, l'horloge envoie une impulsion à un 'répartisseur' qui de son côté parcourt en ordre décroissant un compteur situé dans le MFP. Lorsque le compteur a atteint le zéro, une exception se déclenche, ce qui signifie une intervention de notre programme-résident. Nous ne pouvons donner à ce compteur qu'une valeur comprise entre 1 et 256, ce qui n'est pas beaucoup compte tenu de la rapidité des impulsions - nous obtenons au moins 9600 interrupts par seconde. C'est pourquoi le répartisseur est chargé de ne pas décompter toutes les impulsions: il diminuera le compteur d'une unité en ne tenant compte par exemple que d'une impulsion sur cent. Lorsque le compteur atteint zéro, il déclenche un interrupt; naturellement, nous devons indiquer auparavant à l'ordinateur où se trouve le programme qu'il doit maintenant exécuter.

C'est précisément le rôle de la fonction XBTIMER (numéro 31) du XBIOS. On lui transmet les paramètres suivants: le numéro du timer (de 0 à 3 pour les timers de A à D), l'adresse de la routine d'exception que l'interrupt doit traiter, ainsi que les registres de contrôle et de données. Le registre de contrôle détermine le nombre d'impulsions que le répartisseur doit compter avant de se décider à décrémenter le compteur d'une unité. Le tableau ci-dessous vous donne les valeurs exactes de ce calcul:

Registre de controle MFP	
Registre de controle	Configuration Timer
0	Stoppé
1	4
2	10
3	16
4	50
5	64
6	100
7	200

Le registre de contrôle peut encore prendre d'autres valeurs, mais elles ne servent pas à la mesure du temps: pour plus de détails, reportez-vous à un manuel spécialisé.

Dans vos programmes, vous allez écrire:

```

move.l #adresse_de_début,-(sp)
move.w #registre_des_données,-(sp)
move.w #registre_de_contrôle,-(sp)
move.w #numéro_timer,-(sp) ; 0 pour le timer A
move.w #31,-(sp) ; numéro de la fonction XBTIMER
trap #14
add.l #12,sp

```

Le registre des données, que nous devons transmettre également, est la valeur chargée dans le timer au début de la procédure et à chaque fois que le timer atteint zéro. Voici un exemple: admettons que le registre de contrôle ait la valeur 5 et celui des données la valeur 192. Le répartisseur va donc diviser les 2,4576 MHz par 64, ce qui nous donnera une décrémentation du compteur à 38,4kHz. La dernière opération consiste à diviser cette valeur par le nombre 192 du registre des données, le tout nous donne une fréquence d'interruption de 200Hz exactement. Faites attention de ne pas confondre le compteur avec le registre des données, car seul le compteur va réellement se décrémentation. Le registre des données est conservé tel qu'il est: il est recopié dans le timer lorsque le compteur a atteint zéro.

Il y a plusieurs façons de stopper ce genre de timer. Pour l'arrêter définitivement, utilisez la même fonction que pour le lancer, mais en plaçant un zéro dans le registre de contrôle (voir tableau ci-dessus). Pour le mettre hors service quelques temps, il suffit de mettre sur 7 le masque d'interrupt, le MFP ayant le 'level' 6. Mais

attention: en procédant de cette manière, vous désactivez tous les interrupts, et même pour longtemps. Il est en fait plus adroit dans ce cas de ne déconnecter que le timer A, en utilisant dans le XBIOS la fonction 'Jdisint' (disable interrupt); vous écrivez:

```

move.w #numéro,-(sp) ; numéro de l'exception
move.w #26,-(sp) ; numéro de fonction
trap #14
addq.l #4,sp

```

Le seul paramètre est ici 'numéro', qui doit être (avec le timer A) sur 13 (les autres valeurs concernent des interrupts qui ne nous intéressent pas ici). Pour réactiver le timer MFP, il existe une fonction dont l'appel est identique à celui de 'Jdisint' y compris en ce qui concerne le numéro de fonction. Il s'agit de 'Jenabint' (enable interrupt, numéro de fonction 27).

Vous savez maintenant activer le timer, lui dire où il doit intervenir et à quelle fréquence. Il ne nous reste plus qu'à étudier le traitement de l'interrupt en lui-même: l'exception. Vous devez respecter quatre règles:

- le programme doit sauvegarder tous les registres nécessaires et les restaurer à la fin de la procédure d'exception;
- le programme ne doit jamais faire usage de TRAP pour appeler une fonction du système d'exploitation;
- le programme doit sortir en refermant le MFP;
- le programme doit se terminer par un RTE.

Voici un exemple remplissant toutes ces conditions:

```

movem.l d0-d7/a0-a6,-(sp) ;sauvegarder les registres
-
-
-
- ; figure ici le texte de
; votre programme
-
movem.l (sp)+,d0-d7,a0-a6 ; restaurer les registres
bclr #5,$fffffa0f ; interrupt en service
rte

```

En principe, vous devez avoir tout compris, sauf peut-être l'avant dernière ligne. La commande 'bclr' permet justement de refermer le MFP. Vous pouvez considérer que le registre figurant sous '\$ffffa0f' (registre de l'interrupt en service) est une partie du masque d'interrupt du MFP (lequel compte jusqu'à 16 IPL). Aussi longtemps que le bit 5 de ce registre est positif, toutes les sources d'interrupt soumises au timer A sont désactivées. Vous devez effacer ce bit pour rétablir les 13 autres sources d'interrupt. Le tout fonctionne comme le masque d'interrupt du processeur.

Sachez encore que le compteur du MFP n'a que 8 bits de largeur (ce qui correspond en informatique à l'âge de la pierre!) On ne peut y inscrire que des valeurs comprises entre 0 et 255. Dire qu'on peut initialiser le compteur avec des valeurs entre 1 et 256 revient à dire que la valeur 256 est équivalente à la valeur 0. Ce zéro sera décrémenté ($0 - 1 = 255$) jusqu'à ce qu'il revienne à zéro. Nous pouvons donc engendrer des fréquences d'interrupt comprises entre 614,4 kHz (registre des contrôles sur 1 et registre des données sur 1) et 48 Hz (registre des contrôles sur 7 et registre des données sur 256 = 0).

Vous pouvez maintenant tenter d'installer vous-même une petite routine rapide. Nous vous donnons, à titre d'exemple, un programme qui provoque l'inversion vidéo de l'écran - un octet à chaque exception ('eari.b #fff,xx'). Vous devez naturellement sauvegarder les variables des registres de votre programme à chaque fois que vous quittez la routine. Pour obtenir l'adresse de départ de l'écran, utilisez tout simplement '_v_bas_ad' (\$44E, Video BASis ADress). Vous trouvez une utilisation encore plus intéressante du timer dans le chapitre consacré au multitasking.

7.3.4 La commande Trap

C'est bien une des premières commandes que l'on enseigne aux débutants en assembleur. Elle ouvre la porte vers le système d'exploitation, qui est une gigantesque collection de fonctions. Le mot 'trap' signifie 'piège', en tant que verbe: 'tendre un piège'. Vous savez que l'Atari distingue le mode 'user' (user mode) et le mode 'superviseur' (supervisor mode). Comment s'y prendre pour appeler le système d'exploitation qui tourne en mode superviseur, dans un logiciel qui, lui, tourne en mode user? Les moyens ordinaires ne suffisent plus. Vous vous souvenez peut-être de ce que nous venons de voir ci-dessus: lorsqu'on déclenche un interrupt, le processeur passe automatiquement en mode superviseur avant d'appeler la routine concernée. Il nous faut donc un interrupt spécial pour appeler le système d'exploitation: cet interrupt se chargera, lorsque nous voudrions passer dans le système d'exploitation, de sauter jusqu'à l'adresse de la routine-système.

Le hardware ne peut pas savoir quand nous voulons passer dans le système d'exploitation, c'est pourquoi nous avons un 'software-interrupt': il s'agit précisément de la commande 'Trap'. Tout se passe comme si nous tendions un piège à l'ordinateur, piège qui le fait tomber en mode superviseur.

Examinons de plus près ce que fait le processeur lorsqu'il reçoit une commande 'Trap':

- le compteur du programme en cours, ainsi que ses registres d'états, sont sauvegardés dans la pile superviseur;
- le processeur passe en mode superviseur;
- le contenu d'un vecteur est chargé dans le compteur de programme.

Ce sont exactement les mêmes étapes que celles qui sont parcourues lors d'un hardware-interrupt. Un vecteur déterminé est attribué à chaque hardware-interrupt, alors qu'à chaque commande 'Trap' correspondent, non pas un seul, mais 16 vecteurs! C'est pourquoi la commande 'Trap' est toujours suivie d'un nombre compris entre 0 et 15 qui désigne le vecteur voulu. Un 'Trap #n' passe par le vecteur $32 + n$ qui se trouve à l'adresse $\$80 + 4n$.

Vous devriez continuer à lire tout ce chapitre, même si vous pensez que tout ceci ne vous intéresse guère, vu que vous pouvez utiliser le système d'exploitation sans connaître toutes ces subtilités. Car tout ce que vous faites revient à utiliser des software-interrupts. Nous avons jusqu'ici surtout parlé des réactions déclenchées par une interruption. Il nous reste à répondre à la question suivante: si la commande 'Trap' est en fait une interruption, puis-je, comme avec les hardware-interrupt, créer mes propres interruptions?

Evidemment, la réponse est oui. Dans le paragraphe consacré au HBL, nous avons déjà montré comment il faut s'y prendre pour amener les vecteurs (et le 'Trap' passe bien par des vecteurs) à désigner nos propres routines. A quoi cela peut-il servir? Tous les programmes écrits proprement appellent le système d'exploitation par le biais des différentes commandes 'Trap'. Comme nous sommes maintenant des programmeurs chevronnés, il nous est très facile de capturer des appels du système d'exploitation pour les examiner et les remplacer par nos propres routines. Pour illustrer notre propos, voici un exemple concernant la sortie sur écran.

Pour vous donner un exemple concret, voici un programme dépourvu de signification: il se limite à remplacer le signe %, , par les mots 'p.100'. Ces programmes dépourvus d'utilité sont souvent les meilleurs exemples qui soient!

```

;
; Commande TRAP : POURCENT.S
; remplace le signe '%' par 'p. 100'
; MP 16-02-88
;

bios          = 13
gemdos        = 1
setexception  = 5
keep          = $31
conout        = 3
console       = 2

section text

move.l  sp,a5          ; Réserve place
                        ; mémoire

move.l  4(a5),a5
move.l  12(a5),d6
add.l   20(a5),d6
add.l   28(a5),d6
add.l   #$100,d6

pea     new_bios      ; modifier le vecteur
                        ; TRAP #13
move.w  #45,-(sp)     ; (Numéro 45)
move.w  #setexception,-(sp)
trap    #bios
addq.l  #8,sp
move.l  d0,old_vector ; sauver l'ancien vecteur

clr.w   -(sp)         ; Programme résident
move.l  d6,-(sp)
move.w  #keep,-(sp)
trap    #gemdos

new_bios: move.l  sp,a0          ; Pointeur pile
                        ; Superviseur
btst    #5,(sp)          ; Un appel mode
                        ; superviseur ?
bne.s   label           ; Oui
move.l  usp,a0          ; sinon Point. pile User
subq.l  #6,a0           ; 6 octets plus bas, voir
label:  cmpi.w  #conout,6(a0) ; le numéro de fonction ?
bne.s   call_bios       ;
cmpi.w  #console,8(a0) ;
bne.s   call_bios       ;
cmpi.w  #'%',10(a0)     ; est-ce '%' ?
bne.s   call_bios       ; Non

clr.w   d5

```

```

loop:      lea    pourcent,a0      ; mettre le pointeur ->
                                ; chaine
            move.b (a0,d5),d2      ;
            andi.w #$ff,d2        ;
            move.w d2,-(sp)        ;
            move.w #console,-(sp) ;
            move.w #conout,-(sp)   ; imprimer

            pea    return          ;
            move.w sr,-(sp)        ;
            move.w d5,save         ;

call_bios: move.l old_vector,a1    ; aller à l'adresse du
                                ; bios
            jmp    (a1)            ;

return:    addq.l #6,sp            ; correction de la pile
            move.w save,d5        ;
            addq.w #1,d5          ;
            cmpi.w #7,d5          ;
            bne.s  loop           ;
            rte                    ;

            section data
pourcent:  dc.b 'p. 100'

            section bss

old_vector: ds.l 1
save:      ds.w 1

end

```

7.3.5 La RAM de l'écran

Au début du chapitre sur les programmes résidents, nous avons promis de trahir un petit secret, au moment où nous disions que tous les programmes résidents consomment beaucoup de place-mémoire. Il n'existe pas de programme qui ne consomme pas de place en mémoire, mais il existe par contre une place en mémoire qui n'est quasiment jamais utilisée, si bien qu'en l'occupant, on ne perd pas grand chose: la RAM-Vidéo. Vous allez dire que cette RAM contient l'image en cours, ce qui est très juste. Mais la taille de cette mémoire a été calculée généreusement et peut contenir 32 K-Octets. Les informations relatives à une image ne

représentent que 640 x 400 bits soit 32000 octets (1 K-Octet = 1024 octets). Si bien qu'il nous reste environ 768 octets à la fin de la RAM, et ces octets sont généralement inoccupés. On pourrait fort bien y sauvegarder la routine d'initialisation d'un programme résident: Il faut pour cela que la routine reste accessible, par exemple au moyen d'un adressage PC-relatif. Le programme ne se termine plus par 'keep process' mais par une fonction 'term' toute simple (numéro 0).

Ceci peut s'avérer intéressant, mais comporte certains inconvénients:

- on risque de charger plusieurs programmes résidents dans ce petit coin mémoire, si bien que différents vecteurs pointeront sur la même adresse: situation très chaotique! Imaginez un peu ce qui se passerait si vous chargiez votre disque RAM là où il y a déjà le spooler d'imprimante!
- il arrive que les logiciels mal programmés vident l'écran en remplissant la RAM-Vidéo avec des 0. Si ce processus s'étend non seulement aux 32000 octets de l'image mais aussi à tous les octets de la mémoire, votre vecteur sera pointé sur le vide... (nous concédons qu'il s'agit là d'une hypothèse très pessimiste)
- rien ne vous garantit que la place en question soit effectivement libre dans la mémoire vidéo; la RAM Vidéo peut fort bien commencer à l'adresse \$f8300 (dans les mémoires 1 Méga-octet), mais il resterait encore assez de place.

En raison de tous ces inconvénients, il paraît peu raisonnable de vouloir à tout prix économiser quelques octets de la mémoire vive. Une programmation 'propre', qui n'utilise que des moyens autorisés et classiques, se rentabilise toujours à un moment donné. Nous vous recommandons de vous satisfaire de la fonction 'keep' dans le GEMDOS à chaque fois que faire se peut.

7.3.6 L'appel du système d'exploitation

à partir des programmes résidents

Nous venons de dire qu'un programme résident ne peut pas appeler des fonctions du système d'exploitation. Vous trouverez cependant dans ce livre quelques programmes dans lesquels nous utilisons ce type de routine: nous vous devons bien quelques explications.

Fondamentalement, on ne doit jamais appeler le système d'exploitation avant que le traitement de la fonction en cours ne soit terminé. C'est très simple lorsqu'on utilise des 'traps', car le programme principal n'est pas encore entré dans le système d'exploitation et nous pouvons alors utiliser tous les routines qui nous intéressent.

Il en va tout autrement dans un programme résident (VBL, Timer ...) appelé périodiquement. Nous ne pouvons pas savoir à un moment donné si l'ordinateur est en train de traiter le programme principal ou s'il en a terminé avec la fonction-système en cours de traitement. Dans ce cas, l'ordinateur observe la règle suivante: il ne peut appeler des fonctions GEMDOS que si le programme principal est un programme GEM et s'il se trouve à un endroit où il attend une réaction quelconque de l'utilisateur (p.e. la réponse à une boîte de dialogue ou à un message). En principe, il peut toujours appeler des fonctions BIOS et XBIOS, mais il peut en résulter des problèmes lorsque deux programmes se disputent la même fonction ou le même périphérique, par exemple l'imprimante ou l'unité de disquette. Dans ce cas, l'expérimentation concrète passe avant l'étude abstraite d'un listing de programme. On devrait absolument éviter de se servir alors de l'AES et du VDI.

7.4 L'heure à la seconde près

Dans le sous-système du clavier, l'Atari ST est équipé d'une horloge véritable, qui détermine l'heure et la date système. Dans le GEMDOS comme dans le XBIOS, il existe des fonctions servant à régler ou à lire l'heure et la date contenues dans le processeur du clavier. Malheureusement, ces fonctions ne vous indiqueront l'heure qu'à deux secondes près, ce qui s'avère tout à fait insuffisant pour l'horloge de l'écran. Auriez-vous envie d'acheter une montre ne vous indiquant que les secondes paires?

L'horloge dépendante du processeur du clavier compte bel et bien seconde par seconde: pour avoir l'heure exacte, il va nous falloir dialoguer directement avec ce processeur sans passer par les fonctions système. Il existe en effet une commande l'obligeant à transmettre l'heure et la date au processeur principal, commande qui passe par la routine 'Ikbdws' (intelligent keyboard write string) du XBIOS. Les octets envoyés directement depuis le processeur du clavier doivent par contre être traités par une routine-interrupt. On peut trouver le vecteur pointant sur cette routine à l'aide de la fonction 'kbdvbase' (keyboard vector base). Normalement, ce vecteur est pointé sur la routine originale du XBIOS qui reprend l'indication de la date et de l'heure demandée par la fonction. En ce qui nous

concerne, nous allons purement et simplement diriger ce vecteur sur un programme de notre composition.

La commande servant à demander l'heure s'écrit \$1C; le processeur du clavier renvoie alors 6 octets:

```
1 octet pour l'année (par exemple: 89)
1 octet pour le mois
1 octet pour le jour
1 octet pour l'heure
1 octet pour les minutes
1 octet pour les secondes
```

Tous ces nombres sont fournis en format BCD, abréviation de 'binary-coded decimal code', ce qui signifie que le nombre décimal à deux rangs est enregistré dans un octet de façon à ce que les dizaines se trouvent dans le 'nibble' supérieur (= dans les 4 bits supérieurs) et les unités dans le 'nibble' inférieur. En pratique, cela veut dire tout simplement qu'un nombre décimal sous sa forme BCD se présente sous la même forme que dans le système hexadécimal. Le décimal 24 s'écrit donc \$24 en format BCD.

Dans un premier temps, notre sous-programme 'get_time' oriente sur notre routine le vecteur de traitement du groupe des données relatives au calendrier. Après quoi il demande l'heure grâce à 'ikbdws'. Le programme principal entre alors dans une boucle d'attente qui prend fin lorsqu'il reçoit l'heure et la date (dans l'interrupt, nous utilisons le flag 'arrivé'). La routine interrupt ne fait rien d'autre que lire l'heure, les minutes et les secondes (l'adresse de départ de ce groupe de données se trouve dans A0). C'est le programme principal qui se charge de convertir en nombres binaires les nombres en format BCD.

```

;
; L'heure à la seconde près : HORLOGE.S
; MP 26-06-88
;

xbios = 14
ikbdws = 25
kbdvbase = 34
clockvec = 20

section text

get_time: move.w #kbdvbase, -(sp) ; prendre le vecteur
trap #xbios ; routine d'horloge
addq.l #2, sp
```

```

        move.l  d0,a5
        lea    clockvec(a5),a5
        move.l  (a5),a6
        move.l  #new_clock,(a5)

        clr.w  arrive

        pea    command          ; demander l'heure
        clr.w  -(sp)
        move.w  #ikbdws,-(sp)
        trap   #xbios
        addq.l  #8,sp

wait:   tst.w   arrive          ; attendre que l'heure soit
        beq.s  wait          donnée

        move.l  a6,(a5)      ; retour à la routine
                                originale

        move.b  heures,d0    ; Format BCD vers binaire
        bsr    bcdbin        ;
        move.b  dl,heures
        move.b  minutes,d0
        bsr    bcdbin
        move.b  dl,minutes

        move.b  secondes,d0
        bsr    bcdbin
        move.b  dl,secondes

        rts                  ; Fin du ss-programme

bcdbin: clr.w   dl            ; d0[BCD] --> dl[BIN]
        move.b  d0,d1
        lsr.b   #4,d1        ;
        mulu   #10,d1        ;
        andi.b  #$f,d0       ;
        add.b   d0,d1        ;
        rts

new_clock: move.b  3(a0),heures ; lire le groupe
        move.b  4(a0),minutes
        move.b  5(a0),secondes

        addq.w  #1,arrive    ; flag pour le programme
                                principal

        rts

        section data

```

```

command:      dc.b      $1c                ; Commande : demander
                                                    1'heure

                section bss

even
arrive:       ds.w      1
heures:       ds.b      1
minutes:      ds.b      1
secondes:     ds.b      1

                end

```

7.5 Movem

Attention: ce paragraphe s'adresse aux programmeurs avertis en assembleur. Les autres lecteurs peuvent le sauter, car soit ils n'y comprendront rien, soit ils ne pourront de toute façon rien faire avec les indications qui s'y trouvent.

Il est souvent nécessaire de recopier ou de vider de gros espaces-mémoire, par exemple la mémoire d'écran. La vitesse d'un disque RAM dépend en grande partie de la rapidité de la routine de recopiage.

La solution la plus simple consiste à recopier l'adresse de début, l'adresse de fin et la taille du passage à recopier dans deux adresses et un registre, puis d'en faire une boucle:

```

                lea      source,a0
                lea      cible,a1
                move.w   taille/4-1,d0
loop:          move.l   (a0)+,(a1)+
                dbra    d0,loop

```

La commande `move.l` consomme 20 cycles (sur l'Atari, un cycle = 0,125 microsecondes): il faut lire trois mots (le code de la commande, puis les deux mots de la source) et en écrire deux (dans la cible). Le tout fait 5 mots à manipuler soit 4 cycles par mot. La commande 'Dbra' porte cela à 10 cycles. Un tiers des 30 cycles consommés pour un parcours de la boucle sert donc à la gestion de cette boucle. On peut ici faire des économies en n'écrivant pas dans une boucle les commandes 'move.l' utilisées, mais en les écrivant l'une à la suite de l'autre. Il

nous suffit de connaître à l'avance la taille du passage à recopier pour programmer un nombre suffisant de commandes 'move'.

Il existe pourtant un moyen encore plus rapide, car le processeur doit toujours rechercher l'opcode de la commande 'move' propre à chaque mot-long, ce qui consomme 1/5 des cycles. Pour éviter cela, on peut utiliser la commande 'movem' (move multiple registers) qui peut, en une seule commande, copier jusqu'à 16 mots-longs de la mémoire dans les registres. Habituellement, cette commande sert à sauvegarder les registres dans des sous-programmes pour ensuite les recharger. Mais nous pouvons aussi en faire un autre usage.

Nos données vont devoir faire un petit détour et passer par les registres du processeur: il y en a 16, mais pour des raisons de sécurité, nous ne toucherons pas (au moins en mode superviseur) au stackpointer (A7). Calculons combien de cycles l'ordinateur va consommer pour recopier 15 mots-longs, d'un côté à l'aide de commandes 'move' normales et de l'autre côté à l'aide de MOVEM (sans la boucle DBcc):

```

MOVE.M:
20 cycles * 15 mots-longs = 300 cycles = 20 cycles par mot long

MOVEM.L:
2 commandes * (16 cycles + 8 cycles * 15 mots-longs) = 272
cycles =
18,1333333 cycles par mot-long

```

L'adressage choisi dans le deuxième cas pour la commande MOVEM est du type long (#####.L). Si vous réussissez à trouver un espace-mémoire suffisant pour 15 mots-longs dans les 32 premiers K-Octets, vous pouvez utiliser le type d'adressage court (####.W). Ceci vous fera économiser encore 8 cycles et la consommation sera alors ramenée à 17,6 cycles par mot-long. La ligne de calcul sous MOVEM.L commence par '2 commandes' puisqu'il faut une commande pour prendre les données dans la mémoire et les amener dans les registres puis une autre commande pour les amener dans l'espace-mémoire cible.

Encore un détail: avant le recopiage, il faut sauvegarder tous les registres utilisés puis les restaurer; mais le temps-calcul ainsi consommé est insignifiant lorsqu'on manipule de gros espaces dans la mémoire.

7.6 Rechargement de programmes

Lorsque des programmes ou des groupes de programmes deviennent très encombrants, il est évident qu'il faut les diviser en unités plus petites qu'on appelle des modules. Ces modules peuvent alors s'appeler et se charger l'un l'autre à partir d'une mémoire de masse, ce qui laisse plus de place libre dans la mémoire vive. Citons pour exemple le 'shell', la partie utilisateur d'un compilateur, qui va lancer, tour à tour et selon les besoins, soit le linker soit le compilateur. Nous allons vous expliquer comment cela fonctionne.

Nous allons tout d'abord distinguer entre deux types de chargement. La première possibilité consiste à charger un programme B en l'appelant depuis un programme A: A reste dans la mémoire et reprend son traitement lorsque B prend fin. La deuxième possibilité consiste à charger et lancer B lorsque A a pris fin et a disparu de la mémoire vive: on ne peut plus reprendre le traitement sous A même après être sorti de B. Il est intéressant de noter que la première possibilité passe par le GEMDOS et la deuxième par l'AES.

Commençons par la deuxième possibilité: la fonction AES utilisée s'appelle 'Shel Write'.

Une fois cette fonction appelée, il ne se passe tout d'abord rien. Il faut attendre que le programme en cours se termine; le programme spécifié est alors chargé et lancé sans qu'on revienne au bureau GEM. L'ordinateur reviendra au bureau GEM dès que ce programme aura pris fin. S'il s'agit de charger un programme GEM, votre écran se vide et se remplit d'un fond gris durant le délai nécessaire au chargement; vous verrez apparaître le nom de fichier du programme en cours de chargement sur la ligne supérieure de l'écran, exactement comme si vous aviez appelé ce programme par un double-clic à partir du desktop.

Nous allons vous montrer l'utilisation de cette fonction à l'aide de trois petits programmes d'exemple. Un loader vous permettra de charger deux programmes, l'un sous GEM, l'autre sous TOS. Le nom de fichier du loader sera transmis comme une ligne de commande, si bien que le programme appelé pourra contrôler s'il a été appelé depuis le bureau GEM ou depuis le loader (le bureau GEM n'envoie pas de ligne de commande).

Vous vous étonnerez peut-être de voir le programme TOS commencer par 'appl_init()': c'est tout à fait légal, et cela nous permet de revenir au loader par 'shel_write'.

```

/*****
/*      SHEL_WRITE - Programme de démonstration      */
/*      Loader                LOADER.C                */
/*                16-02-88      MP                    */
/*****/

#define TOS 0
#define GEM 1

main ()
{
    appl_init ();

    switch (form_alert (3, "[2][SHEL-WRITE -
Démo|Charger][GEM|TOS|Quit]"))
    {
        case 1: shel_write (1, GEM, 1, "LOADED.PRG", "LOADER.PRG");
                break;
        case 2: shel_write (1, TOS, 1, "TOSLDD.TOS", "LOADER.PRG");
                }

    appl_exit ();
}

/*****
/*      SHEL_WRITE - Programme de démonstration      */
/*      Program to be loaded                LOADED.C  */
/*                16-02-88      MP                    */
/*****/

char    command [80];
char    parameter [80];

main ()
{
    appl_init ();

    form_alert (1, "[3][Ce programme peut|être rechargé][La
Classe...]");
    shel_read (command, parameter);

    if (*parameter)
    {
        shel_write (1, 1, 1, parameter, "");
    }

    appl_exit ();
}

```

```

/*****
/*      SHEL_WRITE - Programme de démonstration      */
/*      TOS - Program to be loaded      TOSLDD.C      */
/*      16-02-88      MP      */
/*****

char    command [80];
char    parameter [80];

quit ()
{
    appl_init ();

    shel_read (command, parameter);
    if (*parameter)
    {
        shel_write (1, 1, 1, parameter, ".");
    }

    appl_exit ();
}

main ()
{
    printf ("On peut aussi charger des programmes TOS !
(Touche)\n");
    gemdos (7); /* Direct Conin without Echo */

    quit ();
}

```

Venons-en à la deuxième possibilité de chargement des programmes, la fonction GEMDOS nommée 'Pexec' (numéro \$4b). En langage C, on écrit:

```

long Pexec (mode, filename, command, environment)

int mode;
char *filename;
char *command;
char *environment;

En assembleur:

pea    environment
pea    command
pea    filename
move.w #mode, -(sp)
move.w #$4b, -(sp)

```

```
trap      #1
add.l    16, sp
```

'mode' peut prendre une des valeurs suivantes: 0, 3, 4 ou 5 et décrit la sous-fonction souhaitée. Nous n'en retenons que trois:

```
Mode = 0
```

Le programme est chargé et lancé. 'filename' représente le nom de fichier du programme, 'command' représente la ligne de commande (un string vide en cas d'absence) et 'environment' l'environnement du programme à charger: si vous ignorez ce que cela veut dire, consolez-vous: nous ignorons aussi la signification de ce paramètre et nous laissons un string vide. Lorsque le programme prend fin, D0 contient un code-return qui peut être précisé à l'aide de la fonction GEMDOS Term(\$4c) depuis le programme à charger.

```
Mode = 3
```

Il ne se produit que le chargement du programme. Les paramètres sont les mêmes que ceux mentionnés ci-dessus. D0 contient l'adresse du Basepage du programme chargé, que vous allez utiliser pour la 4ème option:

```
Mode = 4
```

Ce mode lance le programme chargé par mode 3. Le seul paramètre transmis est l'adresse du basepage, communiquée lors du chargement. D0 contient aussi un code-return.

Si vous rencontrez des problèmes en vous servant de cette fonction GEMDOS, cela vient probablement du fait que vous avez oublié, en début de programme, de libérer toute la place-mémoire non utilisée (Setblock).

A titre d'exemple, nous allons vous donner un petit programme en Basic-GFA. Tous les propriétaires de disque dur savent qu'il est prudent de placer les têtes de lecture en position 'parking' avant de débrancher l'appareil. Ceci supprime le risque de voir les têtes s'écraser sur une piste de directory. C'est le but du programme SHIP.PRГ qui se trouve sur la disquette d'initialisation du disque dur.

Malheureusement, on ne peut lancer ce programme depuis le disque dur lui-même, puisque celui-ci sera réactivé pour actualiser le directory dans la fenêtre. Notre petit programme sert à lancer le programme SHIP.PRГ pour ensuite tomber dans

une boucle sans fin; il vous suffit alors de débrancher l'ordinateur. Copiez d'abord les deux programmes, SHIP.PRG et HD_SHIP.PRG dans un dossier sur votre disque dur et lancez HD_SHIP.PRG lorsque vous en avez besoin.

```

'
' Programm: HD_SHIP.BAS
'
' Ce programme lance SHIP.PRG depuis la meme partition de disque
dur
' et rentre dans une boucle sans fin...
'
Reserve 1000
'
If Exist ("SHIP.PRG") <>0                                !-Si programme est
                                                         présent
Mode%=0                                                  ! charger et lancer
String1$="SHIP.PRG"+Chr$(0)                             ! Nom du programme
String2$=Chr$(0)                                         ! pas de ligne de
                                                         commandes
String3$=Chr$(0)                                         ! pas d'environnement
Ptr1%=Varptr(String1$)                                   !
Ptr2%=Varptr(String2$)                                   !
Ptr3%=Varptr(String3$)                                   !
'
Void Gemdos (&H4B,Mode%,L:Ptr1%,L:Ptr2%,L:Ptr3%)
'
Print At (10,10);"SHIP exécuté...Eteignez votre ordinateur !!!"
'
Boucle:
Goto Boucle
Else
Alert 3," SHIP.PRG | non trouvé ! ",1,"Dommage",Dummy%
Endif

```

Reste à préciser quand choisir quel mode de chargement. L'utilisation de la fonction AES 'Shel_write' s'impose lorsque de gros programmes s'appellent l'un l'autre, ou lorsqu'on veut absolument retourner au bureau GEM après l'exécution du programme chargé. On utilisera par contre la fonction du GEMDOS ('Pexec') pour des 'shell', des petits programmes servant à charger des extraits d'un programme plus vaste dans lequel on revient une fois le traitement terminé.

7.7 Les sous-programmes en assembleur

Que peut-on bien dire sur un sujet aussi simple que l'écriture de sous-programmes, surtout en assembleur? On les appelle par 'jsr' et on les referme par 'rts', ce qui s'avère largement suffisant dans les petites routines. Lorsque les programmes deviennent plus importants, on peut toujours procéder de cette manière, mais c'est assez dangereux: le risque est grand en effet de modifier les registres d'un des nombreux sous-programmes servant au programme principal. Tout langage évolué un tant soit peu intelligent offre des possibilités beaucoup plus sûres pour appeler sans risque un sous-programme. C'est justement ce que nous allons maintenant étudier.

7.7.1 Paramètres et variables locales

Comme dans la plupart des langages évolués, nous appelons 'sous-programmes' les procédures et fonctions. Nous allons maintenant ajouter trois concepts, souvent utilisés eux-aussi dans les langages évolués: les paramètres, les variables locales, les récursions. Vous trouverez quelques exemples concrets dans ce chapitre, très utiles pour les programmeurs se servant de l'assembleur et qui pourront constituer le fondement de votre bibliothèque en assembleur.

Nous allons commencer par vous expliciter deux commandes-machines que vous avez certainement déjà rencontrées sans trop comprendre leur utilité: les commandes 'link' et 'unlk'. Leur description pose peu de problème, mais ne prend son sens que si vous comprenez à quoi elles peuvent vous servir.

```
link:   link registre des adresses,#valeur constante
```

On commence par sauvegarder dans le stack le registre des adresses, après quoi on recopie la valeur actuelle (donc celle qui vient d'être modifiée) du stackpointer dans le registre des adresses qui vient d'être sauvegardé. On additionne enfin la valeur constante au stackpointer. Nous vous expliquerons plus loin ce à quoi tout cela peut bien servir.

```
unlk:  unlk registre des adresses
```

Cette commande sert à revenir d'une commande link: le registre des adresses est recopié dans le stackpointer; le premier mot-long du stack actuel est ensuite recopié dans le registre des adresses.

Pour l'instant, contentez-vous de retenir que ces commandes existent et qu'elles vont nous servir.

Examinons une petite procédure écrite en Pascal:

```
PROCEDURE testproc (x1, x2 : INTEGER);
  VAR local : INTEGER;
  BEGIN
    local :=x1 + x2;
  END;
```

L'appel de cette procédure s'écrit:

```
testproc (1,2);
```

Cette procédure ne prend aucun sens dans un programme, car elle exécute des opérations dont le résultat n'est plus accessible une fois la routine terminée; elle suffira bien pour notre exemple. Qu'y a-t-il alors de si remarquable dans ce bout de programme? Premièrement, nous transmettons deux paramètres et deuxièmement cette procédure nécessite une variable locale.

Habituellement, le programmeur en assembleur transmet les paramètres dans deux registres de données; ce type de registre peut aussi contenir les variables locales. Cette façon de procéder serait très satisfaisante dans notre exemple, car c'est la manière la plus rapide de transmettre des paramètres. Mais plus le nombre de paramètres ou de variables va augmenter, plus il nous faudra de registres. Comme nous ne disposons 'que' de 15 registres (8 registres de données et 7 d'adresses), nous allons vite atteindre les limites de notre système.

Nous aurions d'ailleurs un autre problème: le manque de clarté dans nos programmes. Normalement, on se sert des registres au niveau du programme principal qui se charge d'appeler les sous-programmes. Il nous faudrait donc constamment faire attention à ne pas utiliser dans un sous-programme un registre déjà utilisé dans le programme principal. Le tout deviendrait encore plus compliqué lorsqu'il s'agirait d'appeler un sous-programme à partir d'un autre sous-programme: aucun programmeur ne pourrait conserver une vue claire de son programme en utilisant de telles méthodes d'écriture.

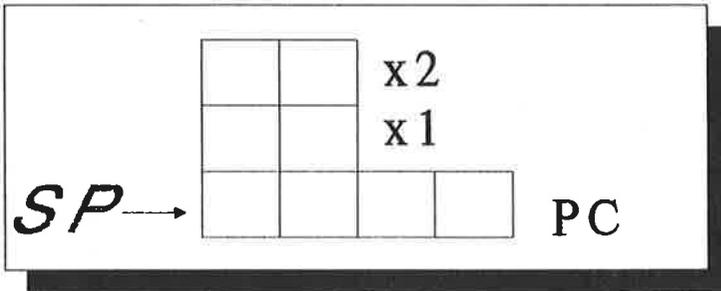
Nous devons donc trouver une autre façon de procéder pour transmettre nos paramètres et amener nos variables locales. Rappelons-nous comment nous faisons au niveau du système d'exploitation: nous plaçons les paramètres dans la pile. Pour formuler en assembleur notre exemple ci-dessus en Pascal, nous aurions écrit:

```

move.w #2,-(sp) ; x2
move.w #1,-(sp) ; x1
jsr testproc ; appel de sous-programme
addq.l #4,sp ; correction du stackpointer

```

Attention: contrairement à ce qui se fait en Pascal ou en C, il faut ici inscrire les paramètres de la droite vers la gauche, c'est-à-dire dans notre exemple d'abord x2 puis x1. Voici l'état de la pile après cette action:



A l'aide du stackpointer, nous avons accès aux deux paramètres, x1 est 4(sp) et x2 6(sp). Il nous reste à régler le problème des variables locales. Nous pourrions leur réserver encore 2 octets sur le stack, en retirant 2 du stackpointer actuel (le stack augmente de haut en bas). Nous pourrions alors accéder aux variables par (sp) et aux paramètres par 6(sp) et 8(sp). Si nous voulons sauver d'autres registres sur le stack, ces valeurs vont se décaler; ceci s'applique également lorsque, en enrichissant un programme, vous constatez qu'il vous faudrait plus de place-mémoire pour une variable locale. Il vous faudra dans ce cas recalculer les accès à vos paramètres.

Voici la solution à ce problème: juste au début du sous-programme, le stackpointer est recopié dans un registre d'adresses qui est vite sauvegardé dans le stack, car le programme principal pourrait aussi en avoir besoin. Nous dégageons ensuite la place nécessaire dans le stack pour nos variables locales et sauvons si nécessaire d'autres registres. Ceci nous permet de ne plus dépendre du stackpointer. Nous avons maintenant accès, par le biais du registre des adresses, aussi bien aux paramètres qu'aux variables locales. Les paramètres sont accessibles par un 'offset' positif et les variables locales par un 'offset' négatif. Comme ces trois actions (sauvegarde d'un registre d'adresses, recopiage du stackpointer dans ce registre d'adresses et modification du stackpointer dans ce registre pour faire de la place pour les variables locales) sont très souvent utilisées, on les a condensées

en une seule commande-machine: 'link'. Il vous suffit maintenant de relire la description de cette commande au début de ce paragraphe pour comprendre son utilité.

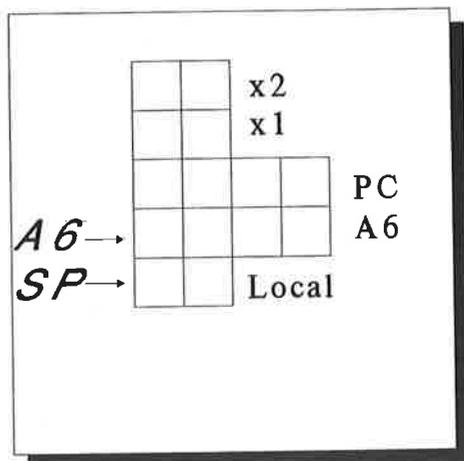
Voyons encore comment on écrirait en assembleur la procédure 'testproc' du Pascal:

```

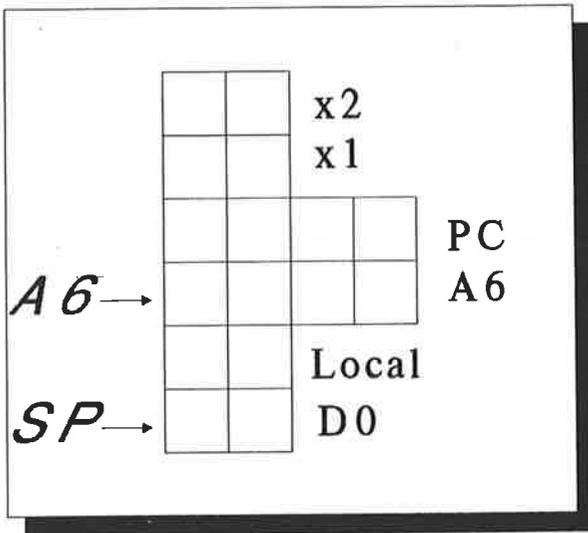
testproc: link    a6,#-2    ; 1 mot = 2 octets de la mémoire
           locale
           move.w  d0,-(sp) ; sauvegarde des registres
           move.w  #8(a6),d0 ; paramètre x1
           add.w   10(a6),d0 ; paramètre x2
           move.w  d0,-2(a6) ; offset négatif -> variable locale
           move.w  (sp)+,d0  ; rechercher les registres
           unlk   a6
           rts

```

Attention: la valeur constante de la commande 'link' est négative. On l'additionne à l'ancien stackpointer, si bien que sa valeur diminue: c'est précisément ce que nous voulons. Rappelez-vous que le stack augmente du haut vers le bas. Examinons le stack après l'exécution de la commande 'link':



Les paramètres se trouvent maintenant sous 8(a6) et 10(a6); les variables locales sous -2(a6). Après exécution de la deuxième commande qui sauvegarde le registre D0, le stack prend l'aspect suivant:



Vous voyez que les paramètres et les variables locales sont accessibles par le même 'offset' et le même registre d'adresses. Vous pouvez sauvegarder n'importe quel registre sans toucher à cette procédure. Par ailleurs, il est maintenant possible d'ajouter sans problème une deuxième variable locale: il suffira de transformer le -2 de la commande 'link' en -4 et la nouvelle variable se trouvera sous -4(a6). Ceci ne modifiera en rien les autres offset.

Cette façon de procéder offre un autre avantage: vous pourrez reprendre ces procédures et fonctions dans un langage compilé. Les routines en assembleur seront traitées soit à l'aide d'un inline-assembleur (par exemple Megamax C) soit à l'aide d'un linker. Les fonctions ne se distinguent des procédures que parce qu'elles donnent une valeur de retour dans le registre de données D0 avant de prendre fin.

Une dernière recommandation: nous vous conseillons de recopier, dans le registre du processeur, les paramètres et les variables locales que vous venez d'utiliser, avant de lancer un traitement complet et de ne les recharger qu'à la fin du traitement. Premièrement, le travail se fait bien plus vite avec des registres qu'en passant toujours par la mémoire principale (le stack est en fait lui aussi une partie de la mémoire principale) et deuxièmement la plupart des commandes-machines sont incapables d'exécuter deux accès-mémoire simultanément (exemple: on dispose de 'ADD Dx,mémoire' ou de 'ADD mémoire,Dx' mais il n'existe pas de commande 'ADD mémoire.mémoire'). Il est néanmoins vivement conseillé de sauvegarder les registres concernés avant de les utiliser, afin de ne pas

endommager par erreur le programme principal. Notons que le registre D0 (valeurs de retour) n'est pas sauvegardé dans les fonctions, car cela représenterait une source d'erreur d'autant plus redoutable que les erreurs les plus simples sont généralement les plus difficiles à détecter.

Vous en savez maintenant assez pour que nous puissions vous présenter quelques programmes concrets qui constituent d'excellents exemples et seront de surcroît des utilitaires appréciés par les programmeurs en assembleur.

Commençons par une routine de saisie de chaînes. Il vous est certainement déjà arrivé d'écrire un petit programme en assembleur et de constater qu'il vous manquait une possibilité de saisie. Le système d'exploitation de l'Atari comprend certes une boucle d'attente de saisie (une routine de saisie) mais cette fonction GEMDOS n'est confortable ni pour le programmeur ni pour l'utilisateur. C'est pourquoi nous vous donnons maintenant un petit programme que vous pourrez inclure, par le biais d'une instruction 'include', dans vos propres programmes. Nous utilisons nous-même souvent ce programme dans les exemples contenus dans ce livre, et vous le trouvez naturellement sur la disquette.

Que doit faire cette routine? Elle doit accepter une saisie à un endroit quelconque de l'écran, si bien que nous devons nous attendre à trouver un système de coordonnées dans la liste des paramètres. Il faut de plus poser une limite à la taille des strings. On doit pouvoir effacer des caractères à l'aide de <BACKSPACE> et vider le champ de saisie avec <ESCAPE>. Vous notez donc une grande ressemblance avec les champs 'edit' d'une boîte de dialogue.

Voici la structure de cette routine:

```

move.w #max_taille,-(sp)
pea    adresse_cible      ; adresse cible du string
move.w #ligne,-(sp)      ; coordonnées
move.w #colonne,-(sp)
jsr    rdstr              ; nom du sous-programme
add.l  #10,sp

```

Comme la chaîne doit se terminer par un octet nul, la cible doit prévoir une taille de 'max_taille + 1 octet' pour la chaîne.

Voici le programme lui-même:

```

;
; Boucle d'attente de saisie INPUT.S
; MP 10-12-87
;

rdstr:  link      a6,#0          ; Pas de RAM pour les
      movem.l   d0-d2/a3,-(sp) ; Sauver les registres

      movea.l   12(a6),a3      ; adr. cible pour la
      ;         saisie

      clr.w     d1             ; taille sur 0
      clr.b     (a3)          ; préparer '\0'

loop:   move.b   9(a6),Set_Cur+3 ; colonne du curseur
      addi.b   #32,Set_Cur+3    ;
      move.b   11(a6),Set_Cur+2 ; et la ligne
      addi.b   #32,Set_Cur+2

      pea     Set_Cur          ; Commande VT-52
      move.w  #9,-(sp)         ; PRINT LINE
      trap   #1
      addq.l  #6,sp

      pea     crsroff         ; désactiver le curseur
      move.w  #9,-(sp)
      trap   #1
      addq.l  #6,sp

      move.l   a3,-(sp)       ; Affichage de la chaîne
      ;         en cours
      move.w   #9,-(sp)       ; PRINT LINE
      trap   #1
      addq.l  #6,sp

      cmp.w   16(a6),d1       ; Comparer à la taille
      ;         max.
      beq.s   suite          ;

      move.w   16(a6),d2      ; taille max - taille
      ;         chaîne - 1

      sub.w   d1,d2           ;
      subq.w  #1,d2           ;
tirets: move.w  #'_','-(sp)    ;
move.w   #2,-(sp)            ;
      trap   #1
      addq.l  #4,sp

```

```

dbra      d2,tirets          ; décrémente le compteur
          add.b    d1,Set_Cur+3 ; curseur en fin de
                                chaîne
          pea      Set_Cur
          move.w   #9,-(sp)
          trap     #1
          addq.l   #6,sp

suite:    pea      crsrorn          ; réapparition du curseur
          move.w   #9,-(sp)
          trap     #1
          addq.l   #6,sp

          move.w   #7,-(sp)        ;
          trap     #1
          addq.l   #2,sp

          cmpi.w   #13,d0          ; RETURN
          beq.s    fin

          cmpi.w   #8,d0           ; Backspace
          beq.s    bs

          cmpi.w   #27,d0         ; Escape
          beq.s    clear

          cmpi.w   #32,d0         ; Touche 'normale' ?
          bmi      loop           ;
          cmpi.w   #127,d0        ;
          beq      loop           ;
          cmp.w    16(a6),d1       ;
          beq      loop           ;
          move.b   d0,0(a3,d1.w)   ;
          addq.w   #1,d1          ;
          clr.b    0(a3,d1.w)     ;
          bra      loop           ;

clear:    clr.b    (a3)           ;
          clr.w    d1             ;
          bra      loop           ;

bs:       tst.w    d1             ;
          beq      loop           ;
          subi.w   #1,d1          ;
          clr.b    0(a3,d1.w)     ;
          bra      loop           ;

fin:      pea      crsroff        ;
          move.w   #9,-(sp)

```

```

trap      #1
addq.l   #6, sp

movem.l  (sp)+, d0-d2/a3
unlk     a6
rts

section DATA

Set_Cur: DC.b 27, 'Y', 0, 0, 0           ; On placera ici les
                                         véritables coordonnées par la suite

crsrn:   DC.b 27, 'e', 0                 ; Séquences VT-52
crsroff: DC.b 27, 'f', 0

even

```

◆ Description du programme

Nous n'utilisons que très peu de variables locales, si bien que nous pouvons les conserver dans des registres; c'est pourquoi la constante de la commande 'link' sera 0 (= pas besoin de place dans le stack). Après avoir sauvegardé les registres dans le stack et initialisé les variables, nous entrons dans la boucle principale. On y crée d'abord une chaîne qui va placer le curseur dans sa position initiale à l'aide de la commande VT-52. Nous affichons ensuite à l'écran la partie du string déjà saisie (au début, naturellement, il n'y a encore rien). Si sa taille actuelle est inférieure à la taille maximale autorisée, nous ajoutons quelques tirets de soulignement (), qui indiquent à l'utilisateur (comme dans une boîte de dialogue) le nombre d'espaces qu'il peut encore utiliser pour l'écriture.

L'utilisateur peut maintenant intervenir activement et appuyer sur une touche. Les touches spéciales, <ESCAPE>, <BACKSPACE> et <RETURN> font l'objet d'un traitement particulier. Toutes les touches affichables (les touches non-affichables sont par exemple les touches de fonction ou Tab etc.) sont ajoutées au string actuel et affichées; le tout recommence ensuite à partir du début.

7.7.2 Entrée et sortie numérique

Comme exemple suivant, nous allons vous donner deux sous-programmes qui servent à entrer et afficher des chiffres. Ce qui n'est qu'une formalité pour les langages évolués devient une entreprise très complexe en assembleur, qui paraît bien plus redoutable encore que la saisie de string telle que nous venons de la décrire.

Le problème vient de ce que l'utilisateur ne peut entrer un nombre qu'en juxtaposant des chiffres, chaque chiffre étant représenté dans la mémoire par son code ASCII. Le processeur ne peut pas calculer en utilisant ces suites de chiffres, il réclame des nombres sous forme binaire. Voici un exemple plus concret: lorsque l'utilisateur entre le chiffre 12, l'ordinateur le traite d'abord comme une chaîne se composant de deux caractères, qui a en mémoire l'aspect suivant:

```
'1' = $31 = 49   '2' = $32 = 50,   fin du string = 0
```

Pour pouvoir faire des calculs avec ce nombre, il faut le transformer en nombre binaire:

```
$0C = 12
```

ce qui permet par exemple au processeur d'exécuter une addition avec ce nombre.

Pour nous résumer: le problème n'est pas tant d'entrer et sortir des suites de chiffres que de transformer ces suites de chiffres en nombres binaires et inversement. C'est justement l'objet du sous-programme que voici:

```

; Conversion ASCII/BINAIRE      CHIFFRES.S
;                               MP 28-05-88
;

; ASCII --> BINAIRE

bin:   link    a6,#-2           ; un mot variable locale
       movem.l d1/a0,-(sp)    ; sauvegarder le registre

       move.l  8(a6),a0       ; Adresse des chiffres
                               ASCII
       clr.l   d0             ; initialiser la variable
                               obtenue
       clr.l   d1             ; ainsi que la variable
                               ; auxiliaire

```

```

        clr.w    -2(a6)           ; remarque: le nombre est
                                ; positif
        cmpi.b  #'-',(a0)        ; y a-t-il un signe moins?
        bne     positif
        move.w  #-1,-2(a6)       ; consigner le négatif
        addq.l  #1,a0            ; et passer au caractère
                                ; suivant
positif:  tst.b    (a0)           ; fin du string?
        beq     finish_bin      ; si oui, interruption

        mulu   12(a6),d0         ; nombre = nombre * base
        move.b (a0)+,d1         ; chiffre suivant
        subi.b #'0',d1          ; ASCII --> BINAIRE
        cmpi.b #10,d1           ; supérieur à 9 (lettre)?
        blt    ok_bin
        subi.b #7,d1             ; si oui, égaliser
        cmpi.b #36,d1           ; une lettre minuscule?
        blt    ok_bin
ok_bin:  subi.b #32,d1           ; si oui, corriger
        add.l  d1,d0            ; ajouter le chiffre au
                                ; nombre déjà
        bra    positif          ; obtenu puis continuer

finish_bin:  tst.w    -2(a6)      ; nombre négatif?
        beq     not_négatif
        eori.l  #-1,d0           ; complémenté à deux
addq.l  #1,d0                    ; (d0 = -d0)

not_négatif:  movem.l (sp)+,d1/a0 ; retour des registres
        unlk   a6               ; sortir de link
        rts                    ; et fin

        ; BINAIRE --> ASCII

ascii:  link    a6,#0
        movem.l d0-d7/a0-a5,-(sp)

        move.l  14(a6),a0        ; adresse cible
        clr.w  -2(a6)           ; noter que le nombre est
                                ; positif
        move.l  8(a6),d1         ; nombre à afficher
        bpl    ascii_pos
        move.w  #-1,-2(a6)       ; négatif? -> poser le
                                ; flag
        eori.l  #-1,d1          ; et le rendre positif
        add.l  #1,d1            ; complémenté à deux

ascii_pos:  move.w  18(a6),d0     ; nombre de colonnes pour
                                ; affichage

```

```

                                clr.b   (a0,d0.w)      ; octet nul de fin de
                                string
ascii_loop:  subi.w  #1,d0          ; -1 pour dbra
                                divu   12(a6),d1      ; d1 = d1 / basis
                                swap   d1            ; reste dans le mot
                                                inférieur
                                add.b   #'0',d1       ; former les chiffres
                                cmpi.b #'9',d1       ; supérieur à 9?
                                ble    ascii_ok
                                addi.b #7,d1          ; si oui, en faire une
                                                lettre
ascii_ok:   move.b  d1,(a0,d0.w) ; écrire les chiffres en
                                                string
                                swap   d1            ; étendre le nombre
                                                'restant' à
                                andi.l  #$ffff,d1    ; un mot long
                                dbra   d0,ascii_loop
                                clr.w  d0            ; écraser le zéro en tête
                                                du nombre
ascii_del:  cmpi.b  #'0',(a0,d0.w) ; zéro?
                                bne    ascii_finish
                                move.b #' ',(a0,d0.w) ; si oui -> effacer
                                addq.w #1,d0
                                bra    ascii_del
ascii_finish:  tst.w  -2(a6)          ; nombre négatif?
                                beq    not_neg
                                move.b #'-',-1(a0,d0.w) ; si oui, '-' devant
                                                premier chiffre
not_neg:     movem.l (sp)+,d0-d7/a0-a5
                                unlk   a6
                                rts

```

Le sous-programme 'bin' transforme une suite de chiffres (saisis par exemple par l'utilisateur) en un nombre binaire et le retourne dans le registre D0; la routine 'ascii' se charge du processus inverse et s'utilise pour éditer des nombres. Cette fonction peut écrire les suites de chiffres en un string de longueur déterminée, si bien qu'à l'affichage, on obtiendra une colonne alignée sur les unités. Ces deux sous-programmes peuvent traiter des suites de chiffres de n'importe quel système numérique. Le plus grand nombre admis correspond à $65536 * \text{Base } -1$; le plus petit nombre possible est $-65536 * \text{Base } +1$. Une chaîne de chiffres ne peut comporter que les caractères 0 à 9, plus éventuellement les lettres et le signe moins; tout autre caractère est interdit, y compris l'espace vide. Comme ces routines ne servent que d'exemples, nous avons renoncé à y insérer un contrôle d'erreur.

Ces deux sous-programmes sont assez simples, c'est pourquoi nous nous limitons à une courte description:

'bin'

Tant qu'il y a des chiffres, le nombre déjà composé est multiplié par la base et le chiffre suivant est additionné.

On écrit:

```

move.w #base,-(sp) ; entre 2 et 36
pea string ; suite de chiffres en ASCII
(fin=octet nul)
jsr bin ; nombre d'après D0
addq.l #6,sp ; correction du stack

```

'ascii'

Tant que le nombre est différent de zéro, il est divisé par la base. On obtient un résultat et un reste, tous deux sous forme de nombre entier. Le reste est le chiffre qui va prendre place à côté des chiffres déjà obtenus. Le résultat de la division est considéré comme un nouveau nombre, et la procédure recommence.

On écrit:

```

move.w #rangs,-(sp) ; largeur d'affichage souhaitée
pea string ; cible, qui doit avoir 'rangs + 1
; octet'
; ou 'rangs + 2' de libre

move.w #base,-(sp)
move.l #nombre,-(sp)
jsr ascii
add.l #12,sp

```

Pour rendre un nombre négatif, on inverse tous ses bits (eor.l #-1,dx) puis on ajoute un.

Voici un programme-TOS, qui affiche sous forme hexadécimale les nombres décimaux saisis par l'utilisateur. Lors de son assemblage, ce programme nécessite la boucle d'attente de saisie INPUT.S

```

;
;Programme d'exemple pour NOMBRES.S
; MP 29-05-88
;

gemdos      = 1
print      = 9

          section text

boucle     move.w #10,-(sp)      ;saisie du string
          ;(longueur max.)
          pea      string      ;adresse cible
          move.w #5,-(sp)      ;ligne et
          move.w #10,-(sp)     ;colonne
          bsr      rdstr
          add.l #10,sp

          move.w #10,-(sp)     ;changer la saisie en
          ;nombre
          pea      string      ;(d'après D0)
          bsr      bin
          addq.l #6,sp
          tst.l d0             ;Nombre 0?
          beq.s exit           ;interruption

          move.w #10,-(sp)     ;former nombre
          ;hexadécimal
          pea      string
          move.w #16,-(sp)     ;base
          move.l d0,-(sp)     ;résultat de 'bin'
          bsr      ascii

add.l #12,sp

          pea      output      ;afficher le nombre
          move.w #print,-(sp)
          trap #gemdos
          addq.l #6,sp

          bra      boucle

exit:      clr.w -(sp)
          trap #gemdos

include 'chiffres.s'

```

```

                include 'input.s'

                section data
                even

output:         dc.b    27,'Y',39,42    ;placer le curseur
                section bss
                even
string         ds.b    11

                end

```

7.7.3 Exemples de récursion

On dit qu'un sous-programme est récursif lorsqu'il peut s'appeler lui-même; cela n'a de sens que si la répétition est soumise à une condition, faute de quoi la routine ne prendrait jamais fin. Beaucoup de programmeurs évitent la récursion qu'ils pensent être une fonction trop difficile pour eux. C'est un tort, car ce n'est pas bien difficile à comprendre. Les récursions permettent aussi d'écrire de beaux programmes, aussi bien au point de vue de l'aspect utilisation du programme qu'au point de vue du style.

L'exemple type d'une récursion est une fonction factorielle qui s'écrit $n!$ pour un nombre naturel n :

$$n! = 1 * 2 * 3 * \dots * n \quad \text{et } 0! = 1$$

Il existe aussi une définition récursive de cette factorielle; récursive signifie ici que la définition de la factorielle contient son concept même:

$$n! = n * (n-1)! \quad \text{et } 0! = 1$$

A première vue, cette définition paraît dépourvue de sens, puisqu'elle utilise le concept à définir, ce qui reviendrait par exemple à énoncer: la liberté consiste à être libre. C'est une chose qu'on peut bel et bien se permettre en mathématiques.

Pour calculer la faculté de 'n' en suivant notre définition de la récursion, nous écrivons:

```
n! = n * (n-1)!
n * (n-1) * [(n-1)-1]! = n * (n-1) * (n-2)!
n * (n-1) * (n-2) * (n-3)!
n * (n-1) * (n-2) * (n-3) * (n-4)! ...
```

Pour écrire tout cela, nous avons appliqué strictement la définition donnée ci-dessus, ce qui mènerait à une répétition sans fin si nous n'avions retenu une condition: $0! = 1$. Nous atteignons, à un moment donné, un point où $(n-k)$ égal zéro. Cela a une conséquence qui nous fait sortir de la définition, car nous avons déterminé $0!$ et nous pouvons le calculer.

Voici un exemple:

```
3! = 3 * 2!
    = 3 * 2 * 1!
    = 3 * 2 * 1 * 0!
    = 3 * 2 * 1 * 1
    = 6
```

Voici le listing d'un sous-programme chargé de calculer une factorielle récursive de façon:

```

;
; Calcul de la récursion          FACTO.S
;      MP      30-05-88
;

facul:      link      a6,#0          ; fixer la base du
;                                     paramètre
            tst.w     8(a6)         ; le paramètre est-il 0 ?
            bne.s    not_zero
            moveq    #1,d0          ; si oui, résultat = 1
            bra.s    finish_facul   ; puis fin
not_zero:   move.w   8(a6),-(sp)     ; mettre dans le stack le
            subi.w   #1,(sp)         ; paramètre -1
            bsr.s    facul           ; appeler la récursion
            addq.l   #2,sp           ; correction du stack

            mulu     8(a6),d0        ; D0 = paramètre *
;                                     ; (paramètre - 1)!

finish_facul:  unlk    a6
```

```

rts

                                end

```

Pour appeler cette fonction, on écrit:

```

move.w  #n, -(sp)
jsr     facul
addq.l  #2, sp

```

Le résultat est porté dans D0. Voici quelques explications: nous commençons par contrôler si le paramètre (que nous nommerons n) est nul. Si c'est le cas, le résultat est déjà acquis, nous pouvons clore le programme. Si ce n'est pas le cas, nous utilisons la même fonction pour calculer la faculté de n-1 et nous multiplions le résultat (qui se trouve dans D0) par n. Nous faisons comme si nous connaissions la faculté de n-1, ce qui n'est pas encore le cas, mais c'est justement la caractéristique même de la récursion.

En dernier lieu, nous allons vous donner un exemple amusant pour illustrer l'utilisation possible de variables locales dans une récursion. La particularité de cette récursion réside dans le fait que le code-programme de la récursion n'existe qu'en un seul exemplaire alors que chaque appel dégage la place nécessaire dans la pile pour les variables locales et le paramètre. Tout se passe comme s'il y avait plusieurs sous-programmes identiques avec des données différentes, mais c'est bien plus pratique.

Notre exemple prend l'aspect suivant:

```

proc_name: saisie d'un string 's'
            si string vide -> fin de la procédure
            sinon, appeler le sous-programme proc_name
appeler le string s à partir du pas 1
            fin de la procédure

```

Si l'utilisateur entre par exemple le mot 'Atari', il est mémorisé dans 's'. Comme 'Atari' n'est pas une chaîne vide, la procédure se répète, ce qui engendre une nouvelle variable 's', sans pour autant effacer l'ancienne. L'utilisateur peut alors entrer 'disquette'. Comme ce n'est pas non plus un string vide, la procédure 'proc_name' est appelée de nouveau. Si l'utilisateur appuie sur la touche <RETURN>, la condition est remplie pour une interruption: la procédure prend fin et la chaîne 's' est effacé. Nous nous retrouvons dans proc_name, mais une étape plus loin, là où 's' représente le mot 'disquette'. Ce mot s'affiche, la

procédure se termine et le mot 'disquette' disparaît. Il ne reste plus que l'ancien 's', représentant le mot 'Atari': ce mot s'affiche à son tour et le tout se termine.

```

;
; Démonstration pour les variables locales
STRINGS.S
;      MP      30-05-88
;

gemdos      = 1
conout      = 2
conin       = 7
print       = 9

section text

pea         titlestr      ; afficher le titre
move.w     #print, -(sp)
trap       #gemdos
addq.l     #6, sp

bsr        string        ; appeler le
                        sous-programme

move.w     #conin, -(sp) ; attendre la touche
trap       #gemdos
addq.l     #2, sp

clr.w     -(sp)          ; term
trap       #gemdos

string:    link          a6, #-20      ; réserver 20 octets

move.w     #19, -(sp)    ; appeler la routine de
                        saisie
pea        -20(a6)      ; adresse cible pour le
                        string

move.w     #10, -(sp)
move.w     #5, -(sp)
bsr        rdstr
add.l     #10, sp

tst.b     -20(a6)       ; string vide?
beq.s     vide

bsr.s     string        ; non, donc relancer
                        ; la routine

```

```

vide:      pea      crlf                ; avance de lignes
           move.w   #print,-(sp)
           trap     #gemdos
           addq.l   #6,sp

           pea      -20(a6)           ; afficher le mot saisi
           move.w   #print,-(sp)
           trap     #gemdos
           addq.l   #6,sp

           unlk    a6
           rts

           include 'input.s'

           section data

titlestr:  dc.b     27,'E',10,'programme: variables locales
           et'
           dc.b     'récursion en assembleur',13,10,10
           dc.b     'Entrez quelques mots. Fin = RETURN',13,10
           dc.b     'Le ST les affiche en ordre'
           dc.b     ' inverse',0

crlf:     dc.b     13,10,0

           end

```

Nous avons déjà expliqué en gros le mode de fonctionnement de ce programme. Nous utilisons des chaînes d'une taille maximale de 19 caractères, c'est pourquoi nous réservons 20 octets (19 + 1 nul) dans la pile par la commande 'link'. Ce programme nécessite aussi la routine INPUT.Q signalée au début de ce chapitre.

7.8 Le Basic-GFA et l'Assembleur

Nous allons vous donner deux exemples pour vous montrer comment utiliser vos propres routines-assembleur en Basic-GFA. Les programmes en question sont intéressants à titre d'exemple mais aussi dans la pratique avec tout langage de programmation, y compris l'assembleur. Il s'agit d'une méthode permettant de trouver des nombres aléatoires réellement aléatoires et d'un timer pouvant servir par exemple pour les 'benchmarks' dans vos programmes.

Commençons par la génération aléatoire de nombres. Nous avons souligné qu'il s'agissait de fournir des nombres aléatoires réellement aléatoires, ce qui n'était pas

du radotage de notre part. Un ordinateur est en effet bien incapable de trouver seul une suite de nombres choisis réellement au hasard. Il est et reste une machine, et les machines ne pensent pas, elles se programment. Il en va de même avec la génération aléatoire de nombres, qui résulte toujours en fait d'une formule préétablie.

La situation est donc la suivante: nous voulons obtenir des nombres tout à fait choisis au hasard, et l'ordinateur (comme nous venons de l'expliquer) est incapable de s'en charger. Nous avons besoin d'un intervenant absolument non-programmable, l'utilisateur assis devant son clavier fera fort bien l'affaire. Mais nous nous refusons à demander un nombre aléatoire à l'utilisateur, ce serait bien trop simple. Nous allons utiliser le temps de réflexion de l'utilisateur pendant lequel l'ordinateur attend un évènement quelconque.

Pour entrer quelques caractères, l'utilisateur a besoin de temps, et ce temps qui s'écoule représente un délai à chaque fois différent. Ce temps de réflexion apporte un élément absolument incalculable et donc réellement aléatoire. Nous allons installer un compteur décroissant, et c'est son état au moment où l'utilisateur appuie sur la touche voulue qui va nous fournir des nombres aléatoires. Inconvénient de ce système: si l'utilisateur a besoin de deux nombres aléatoires, le deuxième reste aléatoire au sens statistique, mais il est calculable par l'utilisateur.

Venons-en au programme: vous remarquerez dès l'abord que nous ne retenons pas d'espace-mémoire et que le programme se termine par un simple RTS. Nous ne pouvons donc pas l'appeler depuis le bureau GEM. Il initialise le timer du MFP (voir le chapitre sur les programmes résidents) et le règle sur environ 500 Hz, ce qui revient à dire que le compteur mentionné ci-dessus va se décrémenter 500 fois par seconde. Nous avons fixé les limites du compteur entre 0 et 95.

Notez bien que pour ce programme, l'assembleur utilise un adressage relatif PC, afin de pouvoir par la suite tourner dans n'importe quel espace-mémoire. Le programme doit être sauvegardé sans header.

Il faut bien que le fichier ainsi créé entre d'une façon ou d'une autre dans l'interpréteur Basic, rappelez-vous le titre de ce paragraphe. Le plus simple sera de traiter le fichier de programme engendré par l'assembleur dans un générateur-data, déjà présenté dans cet ouvrage (paragraphe 4.2). Ce générateur génère un programme-Basic qui contient les codes du programme assembleur dans des lignes Data. Ces lignes Data contiennent aussi un numéro de ligne et une somme de test (voir la description de ce générateur), si bien que nous devons effacer le premier et le dernier nombre de chaque ligne. Nous n'avons plus besoin

non plus des lignes précédant les datas, lignes qui servent à écrire un programme opérationnel. Il nous reste donc dans notre exemple 5 lignes à 16 nombres, soit 80 octets.

Nous devons maintenant faire défiler ces 80 octets dans la mémoire vive. Nous réservons un espace-mémoire de 80 octets dans lequel nous allons enregistrer les nombres octet par octet. Il vaut mieux ici utiliser la forme chaîne. Pour savoir comment on entre les octets un par un, reportez-vous au texte du programme lui-même.

La commande 'call', munie de l'adresse de début du programme (qui est la même que l'adresse du premier caractère de notre chaîne), lance tout le processus. Attention: à la fin du programme, il faut absolument penser à re-désactiver le timer. Si vous oubliez de le faire, vous allez effacer la variable 'Random\$' la prochaine fois que vous lancerez le programme, et cela fera très mauvais effet...

Il ne reste plus qu'à expliquer comment nous allons pouvoir lire les nombres aléatoires générés. Il est conseillé d'imprimer un listing assembleur avec l'adresse de chaque commande-machine. Nous pouvons par exemple y lire que la variable 'nombre' se trouve (dans notre programme) à l'octet \$44 après l'adresse de début du programme. Comme nous connaissons l'adresse du programme, nous additionnons encore \$44 et pouvons lire alors le nombre aléatoire à l'aide d'un Lpeek (Base+&h44). Vous trouverez dans le listing du programme la réponse à toutes vos autres questions:

```

;
; nombres aléatoires          RANDOM.S
; assembleur PC-relatif
;   MP      02-05-88
;
xbios      = 14
xbtimer    = 31

section text

lea       nombre, a0
move.l    _valr_initiale, (a0) ; randomize
pea      random                ; initialiser le timer A
move.w    #49, -(sp)           ; registre des données
                                   (décrémenté 49 fois)
move.w    #6, -(sp)            ; le répartisseur divise
                                   par 100
clr.w     -(sp)                ; Timer A
move.w    #xbtimer, -(sp)
trap     #xbios

```

```

        add.l    #12, sp

        rts

random:  move.l    a0, -(sp)          ; sauvegarder A0
        lea     nombre, a0
        subi.l  #1, (a0)            ; diminuer 500 fois par
                                   seconde
                                   ; le nombre
                                   ; aléatoire
        bpl.s   ok                  ; déjà fini?
        move.l  valr_initiale, (a0) ; recommencer

ok:      move.l   (sp)+, a0
        bclr   #5, $fffffa0f      ; délai de traitement...
        rte    ; puis sortir

        section data
valr_initiale: dc.l    96          ; nombres de 0 à ...
                                   ; (inclu)

        section bss
nombre:    ds.l    1

```

```

' Exemple de programme RANDOM                RND_DEMO.BAS
'      MP      19-05-88
'
Random$=Space$(80)
Base=Varptr(Random$)
'
For I%=0 To 79 Read X%
  Poke Base+I%,X%
Next I%
'
Call Base
'
Data 65,250,0,66,32,186,0,58,72,122,0,26,63,60,0,49
Data 63,60,0,6,66,103,63,60,0,31,78,78,223,252,0,0
Data 0,12,78,117,47,8,65,250,0,28,4,144,0,0,0,1
Data 106,4,32,186,0,12,32,95,8,184,0,5,250,15,78,115
Data 0,0,0,95,0,0,0,0,0,0,0,0,0,0,0,0
'
Print "Générateur de nombres aléatoires: appuyez sur une touche
(F=Fin)"
Print

'
Do
  A%=Inp(2)
  Exit If Upper$(Chr$(A%))="F"
  Print "nouveau nombre aléatoire: ";Lpeek(Base+&H44)

```

```

Loop
'
Void Xbios(31,0,0,0,L:0)           ! désactiver le timer
End

```

Ce procédé n'est certes pas très confortable, même s'il n'intervient que dans de petites routines-assembleur. Dans des programmes plus longs, on ne peut plus guère utiliser tout ce paquet de datas, adresses de début etc. C'est pourquoi il existe encore une autre possibilité d'appeler des programmes en assembleur depuis le Basic: la commande 'monitor'.

'Monitor' peut recevoir un paramètre (entre parenthèses) qui sera automatiquement recopié dans le registre D0 avant l'appel de la routine. Le Basic-GFA commet alors tout à fait volontairement une erreur en envoyant une commande-machine erronée, sanctionnée normalement par 4 bombes. Il nous faut donc un programme résident qui oriente vers notre routine le vecteur de l'instruction 'illégal' (numéro 4). Ce programme sera chargé avant le Basic-GFA et sera exécuté lorsque l'interpréteur Basic rencontrera une commande 'monitor'. On évite ainsi les 'poke' et le programme-Basic a meilleure allure. Il offre de plus la possibilité très intéressante de transmettre un paramètre.

Nous vous avons promis un timer à titre d'exemple. Nous avons retenu le 200ème de seconde comme unité de mesure, puisque c'est une des rares valeurs 'rondes' que le timer du MFP puisse traiter. Le programme qui doit intervenir commence au Label 'new_illegal'. On contrôle d'abord le paramètre transmis; nous avons de plus l'intention d'implémenter les fonctions suivantes:

Paramètre	Fonction
0	mettre le timer à zéro
1	(re)réglér le timer
2	arrêter le timer
>2	l'heure actuelle (en 200èmes de seconde) est enregistrée à l'adresse correspondant au paramètre.

Le texte du programme vous montre comment réaliser tout cela:

```

;
; mesurer le temps, pour les Benchmarks   TIME.S
;      MP      02-05-88
;
gemdos  = 1
bios    = 13
xbios   = 14

```

```

xbtimer      = 31
setexception = 5
keep         = $31

        section text

        move.l 4(a7),a0      ; calculer la
                               place-mémoire

        move.l #$100,d6
        add.l 12(a0),d6
        add.l 20(a0),d6
        add.l 28(a0),d6
        pea timer           ; initialiser timer A
        move.w #192,-(sp)   ; registre de données 192
        move.w #5,-(sp)    ; répartisseur divise par
                               ; 64 => 200Hz

        clr.w -(sp)        ; Timer A
        move.w #xbtimer,-(sp)
        trap #xbios
        add.l #12,sp

        pea new_illegal    ; vecteur de
                               l'instruction 'illegal'
        move.w #4,-(sp)    ; dirigé sur notre routine
        move.w #setexception,-(sp)
        trap #bios
        addq.l #8,sp
        clr.w -(sp)       ; interrompre, mais
                               laisser résident

move.l d6,-(sp)
        move.w #keep,-(sp)
        trap #gemos

timer:   tst.w flag        ; timer activé?
        beq.s ok          ; si non -> fin
        addi.l #1,heure   ; si oui -> laisser
                               l'horloge

ok:      bclr #5,$fffffa0f ; sortir du MFP
        rte              ; puis fin

new_illegal: addq.l #2,2(sp) ; 2(sp) représente le
                               compteur
                               ; sauvegardé; nous le
                               plaçons dans
                               ; la commande derrière
                               ILLEGAL

cmpi.l #2,d0             ; Paramètre supérieur à 2?
        bgt.s read_timer ; oui, donc vaut comme
adresse

```

```

        lsl.w    #2,d0          ; Opcode * 4 = offset
        lea    tableau,a0
        move.l  (a0,d0.w),a0   ; adresse de la
                                sous-routine
        jmp    (a0)

read_timer:  move.l  d0,a0      ; lire l'horloge après
                                (d0)
        move.l  heure,(a0)
        rte                                ; RTE provoque le retour
                                dans le
                                ; Basic-GFA
                                ; derrière la commande
                                ILLEGAL
                                ; (voir ci-dessus)

reset_timer: clr.l    heure     ; timer sur 0
        rte

start_timer: move.w   #1,flag   ; activer le timer
        rte

stop_timer:  clr.w    flag      ; arrêter le timer
        rte

        section data

heure:      dc.l      0
flag:       dc.w      0          ; 0 = timer désactivé

tableau:    dc.l      reset_timer ; adresse des routines
            dc.l      start_timer
            dc.l      stop_timer

        end

```

Et maintenant un programme en Basic pour illustrer nos explications. Il doit calculer le temps nécessaire pour parcourir une boucle vide FOR (10000 cycles) lorsque la variable courante est du type 'integer' ou 'real':

```

' Programme pour mesurer le temps          TIMEDEMO.BAS
'      MP      19-05-88
'
Monitor (0)                                ! Timer sur zéro
Monitor (1)                                ! Timer activé
'
For I%=1 To 10000

```

```
Next I%
'
Monitor (2)
Monitor (Varptr(A%))                ! Résultat après A%
'
Print " Délai pour une variable INTEGER : ";A%/200;" secondes"
'
'
Monitor (0)
Monitor (1)
'
For I=1 To 10000
Next I
'
Monitor (2)
Monitor (Varptr(A%))
'
Print " Délai pour une variable REAL : ";A%/200;" secondes    "
End
```

7.9 Simulation d'une roulette

Ce programme vous permettra de vous détendre un peu. Prenez donc votre capital de départ (n'ayez pas peur, nous vous avons généreusement octroyé 10.000 FF) et c'est parti!

Ce programme de simulation du jeu de la roulette va vous montrer comment, avec des moyens assez simples, vous pouvez obtenir des effets graphiques relativement riches. C'est assez facile en utilisant certaines commandes du Basic-GFA.

Le programme se divise en deux grandes parties: la partie graphique, qui se charge du dessin de la page de titre et de la table de roulette; la partie de calcul qui gère les nombres et leurs caractéristiques (pair/impair - rouge/noir etc) ainsi que vos gains... et vos pertes.

Dans la partie graphique, nous n'avons utilisé que quelques commandes et variables, dont la fonction est souvent évidente.

Après l'affichage de la page de titre, vous voyez s'afficher un bref rappel des règles du jeu de la roulette. Après avoir lu ces règles, vous êtes prié de faire votre première mise. Vous avez le choix entre un nombre et une série. Votre mise étant précisée, vous voyez apparaître une table de roulette, y compris la roulette elle-même, et les nombres s'affichent. Pour rendre la chose un peu plus

passionnante, nous avons créé un bruit de roulette en arrière-fond, simulant la vitesse décroissante de la petite boule (procédure 'nombre'). Si cela vous semble un peu trop long, vous pouvez modifier les boucles correspondantes.

Le résultat du tirage s'affiche sur la table, souligné en noir. On lance ensuite la routine chargée de l'exploitation du résultat qui va calculer la combinaison liée au nombre résultant du tirage. Elle va aussi calculer le taux de vos gains ou pertes. Toutes les possibilités du tirage étant calculées, le programme les compare à votre mise et en tire vos gains ou pertes. Vous voyez s'afficher graphiquement les caractéristiques de la mise que vous auriez dû faire pour gagner: 'couleur\$', 'si1\$' (paire ou impaire), 'si2\$' (1 à 18 et 19 à 36) et 'si3\$' (première, deuxième ou troisième douzaine). Vous voyez alors s'afficher le capital qui vous reste; vous pouvez risquer une nouvelle mise ou interrompre le jeu.

```
' *****
' *Simulation de roulette*          ROULETTE.LST
' *****
Dim Colonne(37)
Dim Ligne(37)
Xmilieu=320
Ymilieu=200
Capital=1000
Ruine=0
Cls
' *****
' * page de titre *
' *****
Deffill 1,2,13

Pbox 0,0,639,399
Graphmode 2
Deftext 1,8,0,26
Text 170,30,"Simulation de roulette"
Deftext 1,1,0,13
Text 210,390,"< appuyer sur une touche quelconque >"
Graphmode 1
Deffill 1,2,24
Pcircle Xmilieu,Ymilieu,155
Deffill 1,2,3
Pcircle Xmilieu,Ymilieu,140
Deffill 0,0,0
Pcircle Xmilieu,Ymilieu,85
Deffill 1,1,1
For T=1 To 1800 Step 100
  Pcircle Xmilieu,Ymilieu,85,2*T,2*T+90
Next T
Deffill 1,2,4
Pcircle Xmilieu,Ymilieu,60
```

```

Deffill 1,2,7
Pcircle 245,193,8
Defline 1,8,2,2
Colonnex=Xmilieu
Ligney=Ymilieu
Line Colonnex-40,Ligney,Colonnex+40,Ligney
Line Colonnex,Ligney-40,Colonnex,Ligney+40
Deffill 1,1,1
Pcircle Colonnex,Ligney,10
Deffill 1,0,0
Pcircle Colonnex,Ligney,5
Defline 1,1,0,0
Oq=Inp(2)
Cls
'
' *****
' * Simulation du bruit de la roulette *
' *****
For I=1 To 28
  Read Data1%
  Son1$=Son1$+Chr$(Data1%)
Next I
For I=1 To 28
  Read Data2%
  Son2$=Son2$+Chr$(Data2%)
Next I
For I=1 To 28
  Read Data3%
  Son3$=Son3$+Chr$(Data3%)
Next I
' *****
' * Coordonnees des nombres sur la table *
' *****
Index01=0
For Nbr=7 To 18
  For Col=30 To 36 Step 3
    Index01=Index01+1
    Colonne(Index01)=Col
    Ligne(Index01)=Nbr
  Next Col
Next Nbr
' *****
' * Coordonnees de 'zero' *
' *****
Nx=34
Ny=5
Nu=0
'
Gosub Explications
Cls
Do

```

```

Gosub Parier
Gosub Table
Gosub Refresh
Gosub Nombre
Gosub Affichage
Gosub Exploitation
Gosub Resultat
Ab=Inp(2)
If Ruine=1
    Gosub Again
Endif
Loop
'
'
Procedure Table
' *****
' * dessin de la table de roulette *
' *****
Cls
Deffill 1,1,1
Pbox 187,57,603,323
Graphmode 1
Deffill 1,2,13
Pbox 339,60,600,Xmilieu
Deffill 1,2,15
Pbox 230,60,299,294
Deffill 1,2,1
Pbox 230,60,299,90
Deffill 1,2,2
Pbox 190,60,230,Xmilieu
Pbox 299,60,339,Xmilieu
Line 230,Xmilieu,299,Xmilieu
Graphmode 2
Deftext 1,0,2700,13
Text 205,80,"P A S S E"
Text 205,165,"P A I R"
Text 205,240,"N O I R"
Deftext 1,0,900,13
Text 325,155,"M A N Q U E"
Text 325,220,"I M P ."
Text 325,310,"R O U G E"
Graphmode 1
Deftext 1,0,0,13
Text 231,310,"      "
Text 235,310,"1."
Text 258,310,"2."
Text 283,310,"3."
For I=1 To 36
    Print At (Colonne(I),Ligne(I));I
Next I

```

```

Print At (Nx, Ny) ; Nu
,
Graphmode 1
' *****
' * dessin de la roulette *
' *****
Deffill 1,2,24
Pcircle 520,184,55
Deffill 1,2,3
Pcircle 520,184,47
Deffill 0,0,0
Pcircle 520,184,26
Deffill 1,1,1
For T=1 To 1800 Step 100
  Pcircle 520,184,26,2*T,2*T+90
Next T
Deffill 1,2,4
Pcircle 520,184,17
Defline 1,4,2,2
Line 511,184,529,184
Line 520,175,520,193
Defline 1,1,0,0
Deffill 1,1,1
Pcircle 520,184,4
Deffill 0,0,0
Pcircle 520,184,1
Print At (1,5);"Votre mise :";
Print At (1,8);"Capital :";
Return
,
Procedure Nombre
' *****
' * tirage du nombre *
' *****
For I=1 To 1100           ! boucle d'attente
Next I
Flag=0
Deffill 0,0,0
Pcircle 520,184,12
For I=1 To 250           ! boucle de tirage
  Numero=Int(Rnd*37)
  If Numero<10
    Print At (65,12);Numero;" "
  Else
    Print At (65,12);Numero
  Endif
' *****
' * bruit de la roulette *
' *****
Pause 0
Void Xbios(32,L:Varptr(Son3$))   ! arrêter le bruit

```

```

' *****
' * Vitesse de la roulette *
' *****
If I<200                                ! Vitesse de rotation
                                        ! et defilement
                                        ! des nombres = Roulette
'
  For Ii=1 To 100
Pause 0
    Void Xbios(32,L:Varptr(Son1$))      ! bruit de la boule
  Next Ii
  Goto Marque
Endif
If I<300                                ! Roulette
  Pause 1
  Void Xbios(32,L:Varptr(Son2$))      ! bruitage de roulette
  For Ii=1 To 200                      ! ralentissement du
                                        ! bruit
  Next Ii
  Goto Marque
Endif
If I<400                                ! Roulette
  Pause 2
  Void Xbios(32,L:Varptr(Son2$))
  For Ii=1 To 300                      ! ralentissement du
                                        ! bruit
  Next Ii
  Goto Marque
Endif
If I<450                                ! Roulette
  Pause 3
  Void Xbios(32,L:Varptr(Son2$))
  If Flag=1
    Graphmode 2
    Deftext 1,16,0,26
    Text 200,30," Rien n'va plus!"
    Deftext 1,17,0,26
    Text 200,30," Rien n'va plus!"
    Graphmode 1
  Endif
  Flag=Flag+1
  For Ii=1 To 500                      ! ralentissement du
                                        ! bruit
  Next Ii
  Goto Marque
Endif
If I<490                                ! Roulette
  Pause 4
  Void Xbios(32,L:Varptr(Son2$))
  For Ii=1 To 800                      ! ralentissement du
                                        ! bruit

```

```

Next Ii
  Goto Marque
Endif
If I<500                                ! Roulette
  Pause 5
  Void Xbios(32,L:Varptr(Son2$))
  For Ii=1 To 1200                        ! ralentissement du
                                          bruit
    Next Ii
  Endif
  Marque:
  Next I
  Pause 0
  Void Xbios(32,L:Varptr(Son3$))
  Text 200,30,"                          "
Return
'
Procedure Refresh
' *****
' * renouveler le dessin de la table *
' *****
' afficher le resultat du tirage sur la table
If Numero=0                              ! tirage = '0'
  Print At (Nx,Ny);Nu                      ! affichage
Else                                       ! tirage > '0'
  Print At (Colonne (Numero) ,Ligne (Numero));Numero ! et
                                          affichage
Endif
Line 253,90,253,320
Line 276,90,276,320
Line 190,160,339,160
Line 190,224,339,224
Return
'
Procedure Affichage
' *****
' * Affichage du nombre tire *
' *****
' afficher le nombre tire
If Numero=0
  Print At (Nx,Ny);Chr$(27);"p";Numero
  Print At (Nx+2,Ny+2);Chr$(27);"q"
Else
  Print At (Colonne (Numero) ,Ligne (Numero));Chr$(27);"p";Numero
  Print At (Colonne (Numero+2) ,Ligne (Numero+2));Chr$(27);"q"
Endif
For Ii=1 To 6000                          ! boucle d'attente
Next Ii
Deffill 1,2,4                              ! puis redessiner la
'                                          ! roulette

```

```

Pcircle 520,184,17
Defline 1,4,2,2
Line 511,184,529,184
Line 520,175,520,193
Defline 1,1,0,0
Deffill 1,1,1
Pcircle 520,184,4
Deffill 0,0,0
Pcircle 520,184,1
Return
'
Procedure Explications
' *****
' * règles du jeu *
' *****
Graphmode 1
Deftext 1,5,0,26
Text Ymilieu,30,"R O U L E T T E"
Text 190,60,"....."
Deftext 1,0,0,13
Text 10,90," voici les règles du jeu:"
Text 10,110," vous pouvez miser sur un nombre entre 1 et 36 ou"
Text 10,130," serie, paire ou impaire, douzaine, manque, passe
ou"
Text 10,150," une ligne transversale."
Text 10,170," lorsque"
Text 10,170,"lorsque sort le zero, c'est la banque qui gagne."
Text 10,190,"TA = Transversale 1-2-3          D1 -> Serie
de 1 à 12"
Text 10,210,"TB = Transversale 4-5-6          D2 -> Serie
de 13 à 24"
Text 10,230,"TL = Transversale 34-35-36       D3 -> Serie
de 25 à 36"

Text 10,250,"PAIR -> nombres pairs"
Text 10,270,"IMPAIR -> nombres impairs"
Text 10,290,"MANQUE -> de 1 à 18"
Text 10,310,"PASSE -> de 19 à 36"
Text 10,330,"Rouge c.à.d. R -> chiffres rouges"
Text 10,350,"Noir c.à.d. N -> chiffres noirs"
Text 10,390,"< appuyez sur une touche >"
Q=Inp(2)
Return
'
Procedure Parier
' *****
' * Saisie de la mise et de l'enjeu *
' * (nombre ou serie) *
' *****
Debut:
Cls

```

```

Print "Vous avez: ";Capital;" FF comme capital de départ."
Input "Montant de votre mise ? ";Mise
If Mise>Capital
  Print "votre capital restant n'est plus suffisant !"
  Qq=Inp(2)
  Goto Debut
Endif
Input "Misez-vous sur un nombre ou une serie ? (n/s)";Wa$ If
Wa$="n"
  Input "Entrez le nombre sur lequel vous misez:";Nbre_parie
  If Nbre_parie>0 And Nbre_parie<37
    Pari$=Str$(Nbre_parie)
  Goto Marque2
Else
  Print "Nombre impossible !"
  Qq=Inp(2)
  Goto Debut
Endif
Else
  Deftext 1,0,0,13
  Text
50,120,"-----"
-----"
  Text 220,135,"Series selectionnables"
  Text 50,160,"TA = Transversale 1-2-3"
  Text 50,180,"TB = Transversale 4-5-6"
  Text 50,200,"TL = Transversale 34-35-36"
  Text 50,220,"PAIR/p ->nombres pairs"
  Text 50,240,"IMPAIR/i ->nombres impairs"
  Text 50,260,"MANQUE/ma ->serie de 1 à 18 d1 ->
  Serie de 1 à 12"
Text 50,280,"PASSE/pa ->serie de 19 à 36 d2 ->
  Serie de 13 à 24"
  Text 50,300,"Rouge/r ->nombres cases rouges d3 ->
  Serie de 25 à 36"
  Text 50,320,"Noir/n ->nombres sur cases noires"
  Input "Serie souhaitee (sous la forme de ses initiales) :
  ";Si$

  Endif
  Marque2:
Return
'
Procedure Exploitation
' *****
' * Calcul de vos pertes ou gains *
' *****
Gg=0 !pas encore gagne,
!calcul en cours
'
Taux=0
Flag02=0
' gagnant gg=1

```

```

If Wa$="s"                                !si choix de 'serie'

If Si$="p"                                  !si 'pair'
  Pari$="pair"
  For Nb=2 To 36 Step 2                    !tous les nombres
                                          pairs possibles

    If Numero=Nb
      Gg=1
      Taux=1
      Flag02=1
    Endif
  Next Nb
  If Flag02=0
Gg=0
    Goto Marquel
  Else
    Goto Marquel
  Endif
Endif
If Si$="i"                                  !si 'impair'
  Pari$="impair"
  For Nb=1 To 35 Step 2                    !tous les nombres
                                          impairs
                                          !possibles

    If Numero=Nb
      Gg=1
      Taux=1
      Flag02=1
    Endif
  Next Nb
  If Flag02=0
    Gg=0
    Goto Marquel
  Else
    Goto Marquel
  Endif
Endif
If Si$="ma"                                 !si 'manque'
  Pari$="manque"
  If Numero>0 And Numero<19                !tous les nombres
                                          entre 1 et 18

    Taux=1
    Gg=1
  Else
    Gg=0
  Endif
  Goto Marquel
Endif
If Si$="pa"                                 !si 'passe'
  Pari$="passe"

```

```
If Numero>18 And Numero<37                !tous les nombres de
                                           19 à 36
    Taux=1
    Gg=1
Else
    Gg=0
Endif
Goto Marque1
Endif
If Si$="d1" Or Si$="d2" Or Si$="d3"        !si 'douzaine'
    If Si$="d1"
        Pari$="1.douz"                    !tous les nombres de 1 à
                                           12
    Endif
    If Si$="d2"
Pari$="2.douz"                            !tous les nombres de 13 à
                                           24
    Else
        Pari$="3.douz"                    !tous les nombres de 25 à
                                           36
    Endif
    If Si$="d1" And Numero>0 And Numero<13
        Taux=2
        Gg=1
        Goto Marque1
    Endif
    If Si$="d2" And Numero>12 And Numero<25
        Taux=2
        Gg=1
        Goto Marque1
    Endif
    If Si$="d3" And Numero>24
        Taux=2
        Gg=1
        Goto Marque1
    Endif
Endif
If Si$="n"
    Pari$="Noir"                          !si 'noir'
    Gosub Couleur
    If Gg=1
        Taux=1
    Endif
    Goto Marque1
Endif
If Si$="r"
    Pari$="Rouge"                          !si 'rouge'
    Gosub Couleur
    If Gg=1
        Taux=1
    Endif
Endif
```

```

    Goto Marquel
  Endif
  If Si$>"t" And Si$<"tm"                !si 'transversale'
    Pari$="Transv."
    If Si$="ta" And Numero<4 Or Si$="tb" And Numero>3 And
Numero<7 Or Si$="tc" And Numero>6 And Numero<10
      Taux=12
      Gg=1
      Goto Marquel
    Endif
    If Si$="td" And Numero>9 And Numero>13 Or Si$="te" And
Numero>12 And Numero<16 Or Si$="tf" And Numero>15 And Numero<19
      Taux=12
      Gg=1
      Goto Marquel
    Endif
  If Si$="tg" And Numero>18 And Numero<22 Or Si$="th" And
Numero>21 And Numero<25 Or Si$="ti" And Numero>24 And Numero<28
    Taux=12
    Gg=1
    Goto Marquel
  Endif
  If Si$="tj" And Numero>27 And Numero<31 Or Si$="tk" And
Numero>30 And Numero<34 Or Si$="tl" And Numero>33
    Taux=12
    Gg=1
    Goto Marquel
  Endif
Endif
Else
  If Nbre_parie=Numero                    !si 'nombre'
    Taux=36
    Gg=1
  Else
    Gg=0
  Endif
Endif
Marquel:
If Numero=0                               !numero zero tire
  Gg=0
  Deftext 1,0,0,26
  Text 50,25,"L a b a n q u e   e n c a i s s e !"
Endif
If Gg=1                                    !gagne
  Gain=Mise*Taux                          !gain = mise * taux
  Capital=Capital+Gain
Else                                        !perdu
  Capital=Capital-Mise                    !capital restant =
                                          capital - mise
Endif

```

```
Return
'
Procedure Resultat
' *****
' * Sortir le resultat *
' *****
' determiner le string du champ de la mise
If Ruine=0                                     !pas de capital
  Flag01=0
  If Numero>0 And Numero<19
    Si1$="Manque"
  Else
    Si1$="Passe"
  Endif
  For Zq=2 To 36 Step 2
    If Numero=Zq
      Si2$="Pair"
      Flag01=1
  Endif
  Next Zq
  If Flag01=0
    Si2$="Impair"
  Endif
  If Numero>0 And Numero<13
    Si3$="1.douzaine"
  Endif
  If Numero>12 And Numero<25
    Si3$="2.douzaine"
  Endif
  If Numero>24
    Si3$="3.douzaine"
  Endif
  If Gg=1
    Deftext 1,1,0,13
    Text Ymilieu,50,"Vous avez gagné !!"
  Else
    Deftext 1,1,0,13
    Text Ymilieu,50,"Hélas, vous avez perdu !!"
  Endif
  If Couleur$="n"
    Coul$="Noir"
  Else
    Coul$="Rouge"
  Endif
  Print At (1,5);"Mise : ";Mise;"   "
  If Numero>0
    If Wa$="z"
      Print At (1,6);"Votre nombre : ";Nbre_parie
    Else
      Print At (1,6);"Votre choix : ";Pari$
    Endif
  Endif
```

```

Endif
Print At(11,8);"      "
Print At(1,8);"Capital restant : ";Capital
Graphmode 2
Deftext 1,1,0,13
If Numero>0
                                !afficher le resultat du
                                tirage
                                !sur la table de la
                                roulette
    Text 350,160,Coul$
    Text 350,180,Si1$
    Text 350,Ymilieu,Si2$
    Text 350,220,Si3$
Else
    Text 350,Ymilieu,"Z E R O "
Endif
Endif
If Capital<1
                                !vous n'avez plus de
                                capital
For I=1 To 50000
    Next I
    Ruine=1
    Cls
    Deftext 1,1,0,26
    Text 1,Ymilieu,"Helas vous êtes totalement ruiné!"
    Deftext 1,0,0,13
    Text 50,250,"Allez dormir, et revenez demain avec un nouveau
capital"
Endif
Return
'
ProcEDURE Couleur
' *****
' * Couleur du numero tire *
' *****
If Numero=1 Or Numero=3 Or Numero=7 Or Numero=9 Or Numero=12
Or Numero=14 Or Numero=16 Or Numero=18
    Couleur$="n"
    Goto Marque01
Endif
If Numero=19 Or Numero=3 Or Numero=23 Or Numero=25 Or
Numero=27 Or Numero=30 Or Numero=32 Or Numero=34 Or Numero=36
    Couleur$="n"
Else
    Couleur$="r"
Endif
Marque01:
If Si$=Couleur$
    Gg=1
Else

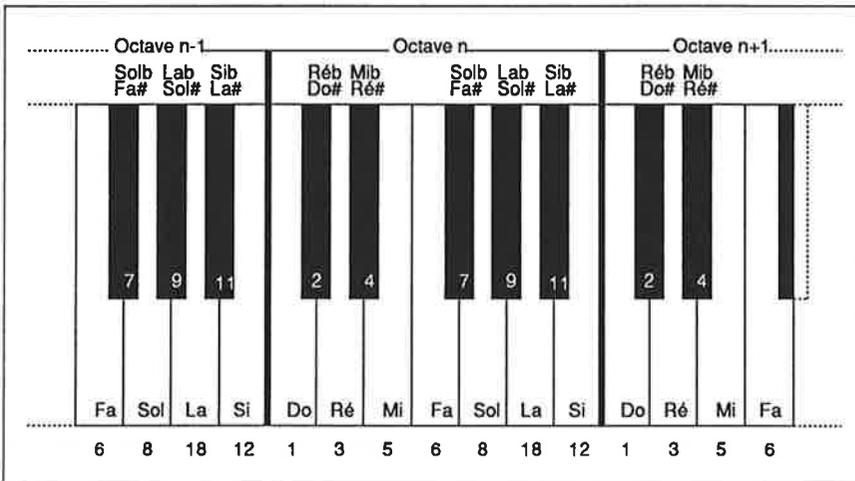
```


7.10.1 La commande sound en Basic GFA

C'est la solution la plus simple pour produire un son, et en voici la syntaxe:

```
Sound Numéro du registre, Volume sonore, Note, Octave, Durée
```

La puce dispose de trois registres sonores distincts (1 à 3), que l'on peut activer individuellement. Il est évident qu'en matière de polyphonie, ceci nous limite à trois voix. Le volume sonore va de 0 (inaudible) à 15 (fortissimo). Le son à émettre est caractérisé par deux paramètres: note et octave. 'Note' peut prendre une valeur comprise entre 1 (Ut) et 12 (Si), un octave entier étant numéroté demi-ton par demi-ton.



L'octave peut se trouver entre 1 (le plus grave) et 8 (le plus aigu). Le paramètre 'durée' sert à fixer le délai d'attente durant lequel le programme est arrêté. Le son n'est pas coupé automatiquement une fois ce délai écoulé; pour cela, il faut mettre sur zéro les paramètres 'volume', 'note' et 'octave'.

Pour engendrer un accord parfait de Do majeur, vous devez écrire:

```
Sound 1,15,1,4      ! on peut omettre 'durée' lorsqu'elle est
nulle
  Sound 2,15,5,4
  Sound 3,15,8,4,200
,
```

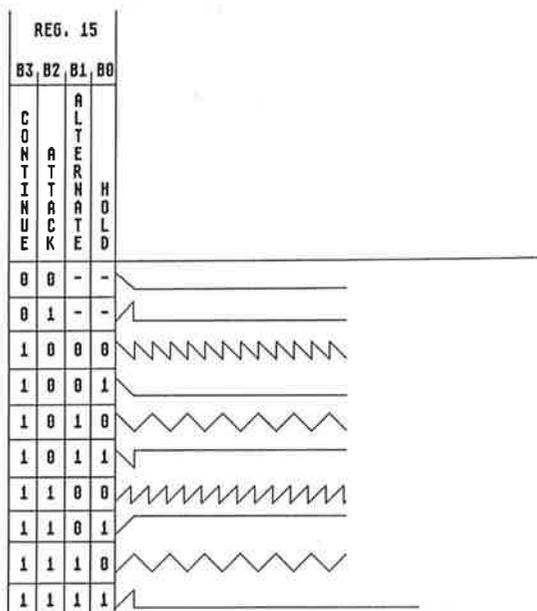
Sound 1,0,0,0	! désactiver le tout
Sound 2,0,0,0	
Sound 3,0,0,0	

7.10.2 Variez les sons en jouant sur la période de l'onde sonore

Les explications ci-dessus vous permettent d'engendrer toutes les notes d'un piano. Ceci s'avère souvent insuffisant: on peut par exemple créer des sons de blues ou élever très lentement un son en parcourant des intervalles inférieurs au demi-ton diatonique. Pour obtenir ces effets, vous laissez de côté les paramètres 'note' et 'octave' et vous attribuez seulement un nombre à la commande sound, nombre précédé d'un signe de numérotation (#). Ce nombre représente la 'période' du son: c'est le quotient arrondi de 125000 divisé par la fréquence en Herz du son souhaité. La constante 125000 vient de ce que les générateurs sonores sont échantillonnés à 1/8MHz soit 125000 Hz. Vous trouverez un peu plus loin un tableau des fréquences des sons les plus courants. La 'période' doit être comprise entre 0 (absence de son) et 4095.

7.10.3 Nouvelles possibilités par 'Wave'

Les sons produits à l'aide de la commande 'sound' peuvent se combiner pour engendrer de belles mélodies, mais ils sont trop 'plats': leurs vibrations sont trop régulières pour captiver longtemps l'attention d'un auditeur. Pour remédier à cela, la puce sonore a la possibilité de varier le volume sonore d'émission d'un même son selon ce qu'on appelle une courbe d'enveloppe, ou plus simplement l'enveloppe sonore. Ceci permet par exemple de faire commencer un son brutalement pour le laisser ensuite s'éteindre en résonnant longuement, en imitant ainsi une cloche. Vous avez le choix entre 9 enveloppes sonores:



La commande 'wave' vous permet de sélectionner une enveloppe pour un son déjà créé par sound:

Wave Registre de contrôle, Attribut, Forme de l'enveloppe, Durée de la période, Durée

Le registre de contrôle possède six bits et sert à activer ou désactiver les trois générateurs sonores (soundregister), en ajoutant à la demande un générateur de souffle sur l'un ou l'autre canal. Les bits 0 à 2 du registre servent à contrôler les canaux 1 à 3 (bit positif = canal activé) et les bits 3 à 5 à activer le générateur de souffle sur les canaux 1 à 3 (bit positif = souffle activé). Ainsi par exemple, la valeur 9 (qui s'écrit en binaire: 001001) signifie que le premier générateur est activé ainsi que son générateur de souffle. 'Attribut' désigne le générateur sonore auquel va s'appliquer l'enveloppe sonore choisie à l'aide de 'wave'. La répartition des bits est semblable à celle du registre de contrôle.

'Forme de l'enveloppe' et 'Durée de la période' servent à 'dessiner' l'enveloppe à proprement parler. Vous trouvez les numéros des formes d'enveloppe dans le tableau ci-dessus. 'Durée de la période' sert à fixer le temps que dure une période de l'enveloppe: on peut donc lui donner une forme 'ramassée' ou au contraire

'allongée'. Plus la valeur de cette 'durée de la période' est élevée, plus la courbe d'enveloppe est allongée. Les valeurs doivent être choisies entre 0 et 65535.

Tout cela paraît un peu sec et compliqué, mais regardez (et surtout écoutez) l'exemple suivant:

```

'
' Création sonore en Basic-GFA
'   MP 28-06-88
'
' 1. Commande de note et octave
'
For O%=1 To 8
  For N%=1 To 12
    Sound 1,15,N%,O%,2
  Next N%
Next O%
'
'
' 2. Commande concernant la durée de la période
'
For P%=10 To 1000
  Sound 1,15,#P%,1
Next P%
'
'
' 3. Commande Wave
'
Sound 1,15,1,4           ! Activer le son
Wave 1,1,14,15,200      ! Enveloppe: dents de scie isocelles;
                        ! durée:15
'
Wave 1,1,14,400,200     ! comme ci-dessus, mais durée plus longue
Wave 1,1,14,3000,200    ! durée encore plus longue
'
Wave 1,1,8,3000,200     ! même durée, mais dents de scie
                        ! descendantes
'
Wave 1,1,4000,200       ! chute linéaire, durée longue
'
'
' 4. Gamme de sons avec enveloppe à chute linéaire, comme des
cloches
'
For N%=1 To 13          ! 13 sons = gamme chromatique
  Sound 1,15,N% Mod 12,5+N% Div 12
  Wave 1,1,1,8000,20    ! Enveloppe, durée de l'enveloppe et
                        ! durée du son
Next N%

```

Sound 1,0,0,0,0	! désactiver le générateur sonore
-----------------	-----------------------------------

7.10.4 Sound en langage C et en assembleur

La programmation sonore en langage C et en assembleur paraît, au premier abord, bien plus compliquée qu'en Basic, car il faut plus de deux commandes pour produire un son. La puce sonore comporte 16 registres dans lesquels nous allons inscrire des données différentes. Avant de voir comment on accède à ces registres, nous allons expliquer leur rôle:

- 0,1 Durée de la période pour le canal A; le registre 0 contient les 8 bits inférieurs de la durée de la période, et le registre 1 les 4 (!) bits supérieurs.
- 2,3 Comme pour 0 et 1, mais concernant le canal B.
- 4,5 Comme pour 0 et 1, mais concernant le canal C.
- 6 Les 5 bits inférieurs servent à la fréquence du souffle; plus la valeur est petite, plus la fréquence est élevée.
- 7 Registre multi-fonctions
 - Bit 0 Activer/désactiver le canal A (bit positif/bit vide)
 - Bit 1 comme pour 0, mais pour le canal B
 - Bit 2 comme pour 0, mais pour le canal C
 - Bit 3 Souffle sur le canal A (bit positif=souffle activé)
 - Bit 4 comme pour 3, mais pour le canal B
 - Bit 5 comme pour 3, mais pour le canal C
 - Bit 6 ne concerne pas la génération de sons; doit se trouver sur 1
 - Bit 7 - comme pour 6
- 8 Volume sonore (bits 0 à 3) canal A
Lorsque le bit 4 est positif, le programme ne tient plus compte du volume sonore fixé, ce dernier étant alors déterminé par la courbe d'enveloppe actuelle.
- 9 Comme pour 8, mais pour le canal B.
- 10 Comme pour 8, mais pour le canal C.

- 11,12 Servent à fixer la durée (la rapidité) de l'enveloppe; plus la valeur est grande, plus la courbe se ralentit; le registre 11 contient le low-byte et le registre 12 le high-byte.
- 13 sert à choisir l'enveloppe; nous avons explicité les valeurs dans le tableau ci-dessus, dans le paragraphe concernant la commande 'wave'.
- 14 port A, n'a aucune signification pour nous.
- 15 comme pour 14, mais pour le port B.

Vous devriez tout d'abord vous laisser le temps de digérer toutes ces nouveautés, après quoi nous allons décrire l'accès aux registres. Officiellement, il faut passer par une routine XBIOS qui s'appelle 'Giaccess'; on écrit:

```
return = Giaccess (données, registre);
```

En assembleur:

```
move.w #registre, -(sp)
move.w #données, -(sp)
move.w #28, -(sp)          ; numéro de la fonction
trap #14
addq.l #6, sp              ; le résultat est accessible dans D0
```

Les données (8 bit) sont enregistrées dans le registre 'registre'. 'Registre' représente le numéro du registre de la puce sonore + 128, le 7ème bit étant positif. Lorsqu'il s'agit de lire un registre, on laisse tomber +128; le contenu actuel du registre se trouve soit sous 'return' ou sous 'D0'.

Le chemin 'non officiel' n'est intéressant qu'en assembleur. On écrit d'abord le numéro de registre souhaité sous l'adresse \$FFFF8800 (octet!). On peut alors soit lire le contenu actuel du registre sous cette même adresse soit écrire dans le registre en passant par l'adresse \$FFFF8802. Attention: vous devez penser à désactiver tous les interrupts lors de cette manoeuvre, pour éviter que ne se produise par exemple un clic du clavier engendré par un appui sur une touche entre le moment où vous choisissez le registre et le moment où vous y accédez.

Ce type de programmation sonore est assez pénible, mais très intéressante car elle vous permet de vous constituer votre propre bibliothèque d'objets sonores. Il ne

vous reste plus qu'à expérimenter: tout maestro doit faire sa dose quotidienne de gammes!

7.10.5 Musique en interrupt

Notre système d'exploitation TOS offre une possibilité particulièrement affriolante: il peut traiter entièrement automatiquement des morceaux de musique entiers durant un interrupt. Le traitement sonore s'écrit:

```
Dosound (commandes)
```

Ou encore:

```
pea      Commandes
move.w  #32,-(sp)    ; numéro de la fonction
trap    #14
addq.l  #6,sp
```

'Commandes' renvoie à la liste des commandes ci-dessous:

- \$00-\$0F** L'octet succédant à l'octet de commande est enregistré dans le registre déterminé par l'octet de commande.
- \$80** L'octet suivant est enregistré dans le registre temporaire.
- \$81** Trois octets de paramètres succèdent à la commande; le premier désigne le numéro d'un registre dans la puce sonore, dans lequel se trouve enregistré le contenu du registre temporaire (voir \$80); le deuxième octet (-128 à +127) est alors additionné au registre temporaire: le processus se répète au bout d'un cinquantième de seconde, il ne s'arrête qu'au moment où le registre temporaire a atteint la valeur finale qui se trouve dans le troisième octet-paramètre. Ceci permet par exemple de faire monter doucement un son ou d'en diminuer/augmenter lentement le volume sonore.
- \$82-\$FF** Lorsque cet octet prend la valeur 0, le traitement sonore est interrompu; autrement, ce paramètre représente une durée mesurée en 50ème de seconde.

Aussi pratique que paraisse cette routine à première vue, elle ne vous permet pas de programmer de grands morceaux de musique. Mais vous pouvez l'utiliser pour certains programmes musicaux: vous pouvez par exemple vous en servir pour déclencher un indicatif musical pendant que le programme principal est en train de se charger.

Nous vous donnons enfin le tableau des fréquences et des périodes sonores des 5 octaves de l'Atari, en vous précisant le low-byte et le high-byte pour les périodes:

Note	Fréquence	Période	High-byte	Low-byte

do				
do dièse = ré bémol				
ré				
ré dièse = mi bémol				
mi				
fa				
fa dièse = sol bémol				
sol				
sol dièse = la bémol				
la				
la dièse = si bémol				
si				
do				
do				
do dièse = ré bémol				
ré				
ré dièse = mi bémol				
mi				
fa				
fa dièse = sol bémol				
sol				
sol dièse = la bémol				
la				
la dièse = si bémol				
si				
do				
do				
do dièse = ré bémol				
ré				
ré dièse = mi bémol				
mi				
fa				
fa dièse = sol bémol				
sol				
sol dièse = la bémol				

```
la
la dièse = si bémol
si
do

do
do dièse = ré bémol
ré
ré dièse = mi bémol
mi
fa
fa dièse = sol bémol
sol
sol dièse = la bémol
la
la dièse = si bémol
si
do

do
do dièse = ré bémol
ré
ré dièse = mi bémol
mi
fa
fa dièse = sol bémol
sol
sol dièse = la bémol
la
la dièse = si bémol
si
do
```

7.11 Programmation graphique

En raison de sa haute résolution, l'Atari ST est pour ainsi dire prédestiné à servir en matière de programmation graphique. Ceci est perceptible au niveau de l'environnement graphique en utilisation ordinaire mais aussi et surtout au niveau des programmes de création graphique dont la vitesse est impressionnante. Les trucs que nous allons vous donner vous permettront de concevoir et réaliser vos propres graphiques encore plus rapidement et efficacement.

7.11.1 Affichage d'images en n'importe quel format

Tout utilisateur se servant intensément de son logiciel de création graphique se retrouve vite à la tête d'une véritable bibliothèque d'images, qui sont bien souvent enregistrées dans des résolutions différentes (haute, moyenne ou basse). Comme nous pensons cependant que la plupart des programmeurs travaillent avec des moniteurs monochromes, ils ont souvent des problèmes pour récupérer des images en provenance de moniteurs-couleurs. Nous avons donc écrit un programme permettant de convertir des images couleurs (sous deux résolutions) en images noir-et-blanc que l'on pourra de plus déplacer sur l'écran (ce qui n'a de sens qu'avec des images en format double-écran, comme par exemple celles que l'on peut créer sous PlusPaint) enfin sauvegarder le tout dans une mémoire de masse (disquette ou disque dur). On pourra aussi inverser l'image; ce programme permet de traiter aussi les images noir-et-blanc. C'est un accessoire, on peut donc l'appeler depuis un autre programme sans en sortir.

Voici comment se déroule ce programme:

on doit d'abord déterminer la résolution de l'image à récupérer, puis son format (Degas ou Screen). On sélectionne ensuite l'image désirée à l'aide d'une boîte de sélection d'objets, pour enfin la charger et donc la convertir. On peut alors déplacer l'image à l'aide des touches du curseur, l'inverser avec la touche `Clr-Home`, la sauvegarder avec la touche `Help`; la touche `Undo` vous sert à sortir du programme. Si vous possédez des images enregistrées dans un format différent de ceux prévus dans ce programme, complétez la procédure 'offset'. Pour le premier espace-mémoire destiné à recevoir une image, nous avons retenu une taille permettant de traiter une image double-écran: en la déplaçant sur l'écran à l'aide des touches fléchées, on pourra choisir d'en extraire un morceau que l'on pourra sauvegarder. L'image sauvegardée aura ainsi le format `Doodle` ou `Screen` et on pourra la reprendre pour d'autres usages après lui avoir fait subir le traitement décrit dans le paragraphe 'doodle' ci-après.

Pour mieux comprendre le fonctionnement de ce programme, reportez-vous aux explications fournies dans le texte:

```
#include<stdio.h>                SAVEIMG.C
#include<gemdefs.h>
#include<osbind.h>

#define TRUE -1
#define FALSE 0
```

```

#define EOS '\0'

main()
{
    int menu_id,
        appl_id,
        msg_buf[8];

    appl_id=appl_init(); /*Annoncer le programme*/

    if((menu_id=menu_register(appl_id," SAVEIMG")<0)
    /*initialiser le menu*/
    {
        form_alert
(1, "[1][plus de place dans | la barre des menus][OK]");
        /*si la barre des menus est complète, quitter*/
        appl_exit();
    }
    while(TRUE)
    {
        evnt_mesag(msg_buf); /*gchetteur d'événements*/
        if((msg_buf[0]==AC_OPEN) && (msg_buf[4]==menu_id)) /*clic
            sur l'option concernée dans le menu*/
        {
            traitement(); /*traiter les images*/
        }
    }
}

traitement()
{
    int fhandle,
        shandle,
offset,
    touche,
    invers,
    punct1,
    punct2,
    punct3,
    punct4,
    point1,
    point2,
    point3,
    point4,
    reso,
    *iptr1,
    *iptr2;

    static int flag=FALSE,
        imagenr=0;

```

```

register int ii,
           zz,
           xx,
           tt;

long phys_image,
     hvalr,
     format,
     plot1,
     plot2,
     plot3,
     plot4,
     *lptr2;

char chemin[80],
     nom_image[15],
     extender[4],
     saveimage[15],
     numero[5],
     *pointeur,
     *ram1,
     *ram2;

offset=0;
invers=FALSE;

if(flag==FALSE)
{
    chemin[0]=nom_image[0]=saveimage[0]=0;
    flag=TRUE;
}
reso=form_alert(0, "[1][Quelle est la résolution | de l'image
à lire][haute|moyenne|basse]");

format=form_alert
(0, "[1][Quel est son format ?][ Degas | Screen ]");
if(format==1) /*format Degas*/
    format=34;
else /*format Screen*/
    format=0;
strcpy(extender, "*.");
if(chercher_chemin(chemin, nom_image, extender)) /*trouver le
chemin d'accès
à l'image*/
{
    ram1=(char *)Malloc(65000L); /*réservé de la
                                place-mémoire*/
    hvalr=(long)ram1;
    ram1=(char *)(((hvalr/256)+1)*256);
    ram2=(char *)Malloc(33000L);
}

```

```

hvalr=(long)ram2;
ram2=(char *)(((hvalr/256)+1)*256); /*trouver une adresse
                                     qui soit un multiple de 256*/
if((ram1>0)&&(ram2>0)) /*assez de place-mémoire*/
{
    fhandle=Fopen(chemin,0); /*ouvrir le fichier*/
    if(fhandle>=0)
    {
        Fseek(format,fhandle,0); /* fixer une position de
                                   début adéquate*/
        Fread(fhandle,64000L,ram1); /*enregistrer
                                       l'image*/
        Fclose(fhandle); /*fermer le fichier*/
        phys_image=Physbase(); /*consigner l'adresse phys.
                                   de l'écran*/
        Vsync(); /*attendre l'interrupt-VBL
                   suivant*/
        Setscreen(-1L,ram2,-1); /* changer d'adresse
                                   physique
                                   de l'écran*/
        if(reso==1) /*résolution haute*/
        {
            iptr1=(int *)ram1;
            iptr2=(int *)ram2;
            for(ii=0;ii<=16000;++ii) /*copier l'image dans
                                       le secteur visible de l'écran*/
                *(iptr2+ii)=*(iptr1+ii);
        }
        else if(reso==2) /*résolution moyenne*/
        {
            iptr1=(int *)ram1;
            iptr2=(int *)ram2;
            tt=0;
            for(ii=0,zz=1;ii<=16000;ii+=2,zz+=2) /*convertir
                                                       et recopier mot pour mot les données de l'image*/
            {
                *(iptr2+tt)=*(iptr1+ii);
                *(iptr2+tt+40)=*(iptr1+zz);
                ++tt;
                if(tt%40==0)
                    tt+=40;
            }
            iptr1=(int *)ram1;
            iptr2=(int *)ram2;
            for(ii=0;ii<=16000;++ii) /* recopier l'image
                                       convertie*/
                *(iptr1+ii)=*(iptr2+ii);
        }
        else /*résolution basse*/
        {

```

```

iptr1=(int *)ram1;
lptr2=(long *)ram2;
tt=0;
for(ii=0;ii<=16000;ii+=4) /* convertir et
                           recopier bit
                           pour bit les données de l'image*/
{
    punct1=(iptr1+ii); /* lecture mot pour mot*/
    punct2=(iptr1+ii+1);
    punct3=(iptr1+ii+2);
    punct4=(iptr1+ii+3);
    *(lptr2+tt)  &=0x00000000; /* vider
l'espace-mémoire avant d'y écrire quelque chose*/
    *(lptr2+tt+20)&=0x00000000;
    for(zz=15,xx=31;zz>=0;--zz,xx-=2) /* lecture
                                       bit pour bit*/
    {
        point1=punct1;
        point2=punct2;
        point3=punct3;
        point4=punct4;
        plot1=(long)((point1>>zz)&1);
        plot2=(long)((point2>>zz)&1);
        plot3=(long)((point3>>zz)&1);
        plot4=(long)((point4>>zz)&1);
        plot1=plot1<<(xx-1);
        plot2=plot2<<xx;
        plot3=plot3<<(xx-1);
        plot4=plot4<<xx;
        *(lptr2+tt)  |=plot4;
        *(lptr2+tt)  |=plot3;
        *(lptr2+tt+20)|=plot2;
        *(lptr2+tt+20)|=plot1;
    }
    ++tt;
    if(tt%20==0)
        tt+=20;
}

iptr1=(int *)ram1;
iptr2=(int *)ram2;
for(ii=0;ii<=16000;+ii) /* recopier l'image
                           convertie */
    *(iptr1+ii)=*(iptr2+ii);
}

while((touche=((Crawio(0xFF)/0x10000)&0xFF))!=0x61)
/*Scan-Code;pour sortir, appuyer sur UNDO*/
{
    pointeur=ram2;
    if(invers==FALSE) /*image dans son état
                       originel

```

```

    strcat(chemin, "\\");
}
strcat(chemin, extender); /*reprendre l'extender*/
fsel_input(chemin, nom_fichier, &button); /*indiquer le
                                         chemin actuel*/
*(chemin+(strlen(chemin)-3))=0; /*effacer l'extender
                                dans le chemin d'accès*/
strcpy(buffer_aux, "\\0"); /*vider le buffer*/
if(button!=0)
{
    strcpy(chemin_accès, chemin); /*transmettre le chemin
                                d'accès au chemin principal*/
    strcat(chemin_accès, nom_fichier); /*ajouter le nom du
                                fichier à l'intitulé du chemin d'accès*/
}
return(button);
}

```

7.11.2 Insérer des "Doodles" dans vos propres programmes

Vous avez sans doute déjà souhaité pouvoir insérer des images dans vos programmes, que ce soit comme page de titre, comme schéma explicatif ou pour faire une 'projection' de vos oeuvres: malheureusement, vous n'y êtes pas parvenu faute de savoir comment faire. Nous vous proposons une petite routine pour vous montrer le principe de fonctionnement permettant d'exaucer vos souhaits.

Dans cette routine, nous nous sommes limités au format doodle (32000 octets par image), qui représente la taille exacte d'une mémoire d'écran ordinaire. Ceci vous permet par exemple d'avoir une image accompagnant tout programme capable de mémoriser une copie-écran dans un fichier; si le programme ne peut s'en charger, vous pouvez faire une copie 'manuelle'. Voici la séquence des commandes nécessaires:

```

int fhandle;
char fname[13];
long adr_ecran;

adr_ecran=Logbase(); /*adresse logique de l'écran*/
strcpy(fname, "image.pic"); /*nom de l'image*/
fhandle=fopen(fname, 2); /*ouvrir le fichier pour y
                        écrire*/
/*inscrire 32000 octets dans le fichier à partir de
l'adresse écran*/
fwrite(fhandle, 32000L, adr_image);
fclose(fhandle); /* refermer le fichier */

```

Avant de vous communiquer le texte du programme, voici quelques explications au sujet des routines utilisées:

Malloc()**GEMDOS-Numéro 0x48**

Appel: long Malloc(quantité)
 long quantité;

Lorsque 'quantité' est égale à -1, Malloc vous indique la quantité d'octets encore libres dans le plus grand bloc. Si 'quantité' est différente de -1, Malloc sert à réserver la quantité d'octets indiquée sous 'quantité', et vous renvoie un pointeur sur cette place-mémoire. Si cette dernière s'avère insuffisante, vous recevez le résultat 0L.

Fopen()**GEMDOS-Numéro 0x3D**

Appel: int Fopen (nom_fichier,mode)
 char *nom_fichier;
 int mode;

La fonction Fopen sert à ouvrir un fichier portant le nom indiqué sous 'nom_fichier'; les caractéristiques attribuées à ce fichier sont déterminées à l'aide de 'mode', qui peut prendre les valeurs suivantes:

0	lecture seule
1	écriture seule
2	lecture et écriture

Si le processus se déroule convenablement, vous recevez en retour le numéro-handle du fichier; sinon, vous recevez un numéro d'erreur négatif.

Fread()**GEMDOS-Numéro 0x3F**

Appel: long Fread(handle,quantité,buffer)
 int handle;
 long quantité;
 char *buffer;

Cette fonction sert à prendre dans le fichier portant le numéro-handle consigné sous 'handle', la quantité d'octets indiquée sous 'quantité', pour les enregistrer dans le buffer désigné sous 'buffer'. Si le processus se déroule convenablement,

```

                                (non invertie)*/
    {
        for(ii=0;ii<32000;++ii)
            *(pointeur+ii)=*(ram1+offset+ii);
    }
    else /*inversion de l'image*/
    {
        for(ii=0;ii<32000;++ii)
            *(pointeur+ii)=(*(ram1+offset+ii));
            /*formation du complément*/
    }
    Vsync(); /*attendre un interrupt VBL*/
    Setscreen(-1L,ram2,-1); /*afficher l'image*/
    switch(touche)
    {
        case 0x48: /*'up'*/
            if((offset+80)<=(64000-32000))
                offset+=80;
            break;
        case 0x50: /*'down'*/
            if((offset-80)>0)
                offset-=80;
            break;
        case 0x4B: /*'left'*/
            if(++offset<=(64000-32000))
                offset++;
            break;
        case 0x4D: /*'right'*/
            if(--offset>=0)
                offset--;
            break;
        case 0x62: /*'save'*/
            imagenr++;
            sprintf(numero,"%d",imagenr);
            /*numéro actuel de l'image*/
            strcpy(saveimage,"save"); /*déterminer
                                         le nom du fichier*/
            strcat(saveimage,numero);
    strcat(saveimage, ".pic");

            shandle=Fcreate(saveimage,0x0); /*créer
                                                le fichier*/

    Fwrite(shandle,32000L,ram2);/*enregistrer l'image*/
            Fclose(shandle); /*fermer le fichier*/
            break;
        case 0x47: /*inverser l'image*/
            if(invers==TRUE)
                invers=FALSE;
            else

```

```

                invers=TRUE;
                break;
            }
        }
        Vsync(); /*attendre un interrupt VBL*/
        Setscreen(-1L,phys_image,-1); /* revenir à
                                        l'écran d'origine*/
    }
    else
    {
        form_alert
        (1,"[1][erreur dans les fichiers][interruption]");
        Fclose(fhhandle);
    }
    Mfree((long)ram1); /*vider l'espace-mémoire occupé*/
    Mfree((long)ram2);
}
else
{
    form_alert
    (1,"[1][pas assez de place-mémoire][interruption]");
}
}
}

int chercher_chemin(chemin_acces,nom_fichier,extender)
/*communiquer les chemins d'accès et les noms de
fichiers*/
char chemin_acces[],
    nom_fichier[],
    extender[];
{
    char buffer_aux[80],
        chemin[80];

    int index,
        drive,
        button;

    if(((*chemin_acces)=='\0')&&((*nom_fichier)=='\0'))
    {
        drive=Dgetdrv(); /*identificateur de l'unité de disque
                           actuelle*/
        *chemin=drive+'A'; /*à reprendre dans le chemin
                           d'accès*/
        strcpy((chemin+1),":");
        Dgetpath(buffer_aux,drive); /*indiquer le chemin
                                     actuel*/
        strcat(chemin,buffer_aux); /*et le reprendre dans le
                                     chemin d'accès*/
    }
}

```

la valeur retournée correspond au nombre des octets enregistrés; sinon, vous recevez un numéro d'erreur négatif.

Fwrite()**GEMDOS-Numéro 0x40**

Appel: long Fwrite(handle,quantité,buffer)
 int handle;
 long quantité;
 char *buffer;

Cette fonction sert à enregistrer dans le fichier portant le numéro-handle consigné sous 'handle', la quantité d'octets indiquée sous 'quantité', prise dans le buffer désigné sous 'buffer'. La valeur retournée correspond soit à la quantité d'octets enregistrée, soit à un numéro d'erreur négatif.

Fclose()**GEMDOS-Numéro 0x3E**

Appel: int Fclose(handle)
 int handle;

La fonction Fclose permet de refermer le fichier désigné par le numéro-handle figurant sous 'handle'. En cas de succès, la valeur retournée est 0, sinon vous recevez un numéro d'erreur négatif.

Mfree()**GEMDOS-Numéro 0x49**

Appel: long Mfree(pointeur)
 long pointeur;

La fonction Mfree permet de libérer l'espace-mémoire réservé sous Malloc et dont l'adresse de début figure sous 'pointeur'. En cas de succès, la valeur de retour est 0L, sinon un numéro d'erreur négatif.

Physbase()**XBIOS-Numéro 2**

Appel: long Physbase();

Cette fonction vous transmet l'adresse physique de base de l'écran.

Vsync()**XBIOS-Numéro 37**

Appel: void Vsync();

Cette fonction permet de synchroniser l'affichage graphique avec les retours d'écran en attendant l'interrupt VBL suivant.

Setscreen()**XBIOS-Numéro 5**

Appel: void Setscreen(logbas,physbas,res);
 long logbas,
 physbas;
 int res;

Cette fonction permet de déterminer les adresses de base de l'écran ('logbas' pour l'adresse logique et 'physbas' pour l'adresse physique) ainsi que sa résolution. Un paramètre recevant une valeur négative sera un paramètre non utilisé.

```

/* Pour insérer des images .PIC dans vos propres programmes */
/* IMG.C */

#include <osbind.h>
#include <stdio.h>

main()
{
    int appl_id;
    char nom_image[13];

    appl_id=appl_init();           /* annoncer le programme */

    strcpy(nom_image,"save01.pic"); /* initialiser le nom de
                                    l'image */
    show_image(nom_image);        /* charger et afficher
                                    l'image */
    appl_exit();                  /* sortir du programme */
}

int show_image(nom_image)
char *nom_image;
{
    int fd;                       /* numéro du fichier */
    long buffer,                  /* pointeur vers l'espace mémoire de 32
                                    K-octets */
        buffer_auxiliaire;       /* variable auxiliaire */

```

```

buffer=Malloc(32000L);      /* rechercher l'espace-mémoire par
GEMDOS */
buffer_auxiliaire=buffer; /* sauvegarder le pointeur */
buffer=((buffer_auxiliaire/256)+1)*256;      /*ajuster adr
standard */

if( (fd=Fopen(nom_image,0)) != NULL)
{ /* ouvrir le fichier pour le lire */
  Fread(fd,32000L,buffer);      /* lire l'image */
  show(buffer);                  /* afficher l'image */
  Mfree(buffer_auxiliaire);      /* vider l'espace-mémoire */
}
else
  form_alert(0,"[1][Image introuvable][ FIN ]");
}

int show(buffer)
long buffer;                  /* adresse de début de l'image */
{
  long adresse;              /* ancienne adresse vidéo */

  adresse=Physbase();        /* sauvegarder ancienne adresse vidéo */
  Vsync();                    /* synchronisation avec le retour de
                             ligne */
  Setscreen(-1L,buffer,-1); /* passer à la nouvelle adresse
                             vidéo */
  Cconin();                   /* attendre action d'une touche */
  Vsync();                    /* synchronisation avec le retour de
                             ligne */
  Setscreen(-1L,adresse,-1); /* revenir à l'adresse vidéo */
}

```

7.11.3 Des graphiques stables

Imaginez que vous soyez chargé d'écrire un programme de jeu sur ordinateur. Les déplacements du héros sur l'écran seront commandés par le joystick (manette de jeu). Rien de plus simple, allez-vous dire, nous allons nous fabriquer un sprite dont la silhouette s'effacera à sa position d'origine pour se rematérialiser à sa nouvelle position d'arrivée.

Le résultat de ce programme ne sera malheureusement pas très satisfaisant: l'ordinateur a besoin d'un délai relativement long pour effacer la silhouette et pour la redessiner, si bien que votre héros disparaîtra de la vue de l'utilisateur, qui aura l'impression d'un scintillement de l'image.

Pour écarter le risque de scintillement, nous allons faire travailler l'ordinateur au niveau interne sur deux écrans, dont l'un reste visible tandis que l'autre est invisible. Admettons que les deux écrans reflètent exactement la même situation dans le jeu. Nous allons nous servir de l'écran invisible pour déplacer la silhouette du héros, qui va donc temporairement disparaître, pour la faire réapparaître à sa nouvelle position. Ceci étant fait, nous allons dire à l'ordinateur d'effacer de l'écran l'image qui était visible pour y envoyer l'image qui était jusqu'ici invisible. Le déplacement suivant sera assuré de la même manière dans l'image qui est devenue maintenant invisible et la permutation des écrans va se poursuivre de cette façon à chaque déplacement de la silhouette.

Le terme 'écran' que nous venons d'employer ne désigne pas votre moniteur en tant qu'appareil. Il s'agit en fait de l'espace-mémoire contenant les données qui, une fois traitées par une puce de votre Atari (le shifter), deviennent un signal vidéo envoyé à votre moniteur. Cet espace-mémoire contient pour chaque point de l'image une information précisant si ce point est activé ou non ainsi que la couleur qu'il doit prendre. Pour mémoriser une page d'écran, l'Atari a besoin de 32000 octets très exactement. On appelle cette mémoire de l'écran tout simplement la RAM-Vidéo.

Naturellement, il est tout à fait possible d'installer plusieurs RAM-Vidéo dans votre ordinateur. Certaines fonctions du système d'exploitation, sur lesquelles nous allons revenir, permettent alors d'indiquer à l'ordinateur quelle est la RAM-Vidéo qui contient l'image que l'on désire voir s'afficher à l'écran. Cet espace-mémoire s'appelle 'mémoire physique de l'écran' car il vient s'afficher sur un appareil physiquement bien concret: votre moniteur.

Le tout ne peut s'avérer avantageux pour nous que si nous pouvons simuler un espace-mémoire d'écran dans lequel se déroulent les traitements graphiques: comme nous voulons éviter le scintillement de l'écran, il faut que les déplacements de la silhouette ne se fassent pas dans la mémoire physique. C'est ce deuxième espace-mémoire que nous appelons 'mémoire logique de l'écran', et c'est là que vont se dérouler les traitements graphiques exécutés par le système d'exploitation.

Nous allons vous illustrer tout cela à l'aide d'un petit exemple. Le programme ci-dessous dessine une ellipse pourvue d'un motif de remplissage, qui va en diminuant et en augmentant. Nous avons choisi une ellipse car cette figure géométrique est particulièrement longue à dessiner. Notre objectif consiste donc à disposer d'une ellipse dans la Vidéo-RAM physique pendant que la seconde est en train de se préparer dans la mémoire logique.

```

/*****
/* Graphiques stables      MP 03-06-88      GRAPH.C      */
*****/

#include <osbind.h>          /*système d'exploitation
- Defs.*/

int  work_in[11],           /* array du VDI */
     work_out[57],
     contrl[12],
     intin[128],
     ptsin[128],
     intout[128],
     ptsout[128],
     vdi_handle,
     xmax,
     ymax;

long screen1,
     screen2,
     memory,
     malloc();

open_vwork()  /*ouvrir la station virtuelle*/
{
    register int i;
    for (i=0; i<10; work_in[i++] = 1);
    work_in[10]= 2;          /*coordonnées*/
    v_opnvwk(work_in, &vdi_handle, work_out);
    xmax = work_out[0] / 2;  /*consigner résolution de
                             l'écran*/
    ymax = work_out[1] / 2;
}

init_screens()
{
    screen1 = Logbase();     /*adresse de début de la
                             Vidéo-RAM*/
    if ((memory = malloc(32256)) == 0L) /*installer nouvelle
                                         RAM-Vidéo*/
    {
        printf ("Place insuffisante en mémoire!\n");
        Cconin ();
        Pterm0 ();
    }
    screen2 = (memory & 0xfffff00) + 256; /*doit se trouver à la
                                         limite*/
}

```

```

}

swap_screens()                                /*basculer de l'écran 1
vers 2*/
{
  if ( Logbase() == screen1 )
  {
    Vsync();
    Setscreen(screen2, screen1, -1);
  }
  else
  {
    Vsync();
    Setscreen(screen1, screen2, -1);
  }
}

circle(radius)                                /* radian entre 0 et 50 */
int radius;                                  /* (en fonction de la taille */
{                                             /* de l'écran) */
  int rx, ry;

  rx = radius * xmax / 50;                  /* conversion en radian */
  ry = radius * ymax / 50;                  /* de l'écran */
  v_ellipse (vdi_handle, xmax, ymax, rx, ry);
}

main()
{
  int radius,
    direction;

  open_vwork ();                            /* déclarer auprès du VDI */
  init_screens ();                          /* initialisation */
  v_hide_c (vdi_handle);                    /* déconnecter la souris */
  vsf_interior (vdi_handle, 3);             /* motif de remplissage */
  vsf_style (vdi_handle, 1);

  direction = 1;                             /* 1 = augmentation de taille */

  for (radius = 5; !Cconis (); radius = radius + direction)
  {                                           /* fin, si une touche appuyée */
    v_clrwk(vdi_handle);
    circle(radius);
    swap_screens();

    if (radius > 25 || radius < 5)         /* valeur finale? */
      direction = -direction;
  }
  Cconin();                                  /* rechercher la touche */
}

```

```

Setscreen(screen1, screen1, -1); /* retour à l'écran normal */
v_show_c (vdi_handle, 1);      /* réactiver la souris */
v_clsawk (vdi_handle);         /* sortir du VDI */
}

```

La première fonction remarquable utilisée ici est sans doute 'init_screens()'. Cette fonction recherche deux espaces-mémoire dans lesquels nous allons mémoriser nos deux écrans. Les bases (adresses de débuts) de ces espaces-mémoire sont sauvegardées dans les variables 'screen1' et 'screen2'. Nous admettons tout simplement que l'écran numéro 1 est celui qui est affiché à ce moment. Son adresse de début nous est fournie par la fonction du XBIOS nommée Logbase() (=base de la Vidéo-RAM logique; normalement, l'écran logique et l'écran physique sont identiques).

Nous devons encore demander un espace-mémoire de 32000 octets pour notre deuxième écran. Nous rencontrons alors un petit problème: un espace-mémoire pour l'écran doit commencer obligatoirement à une 'limite de page', c'est à dire à une adresse dont les deux chiffres les plus bas (écrits en hexadécimal) soient zéro, comme par exemple dans \$F8000 ou \$100. On exprime la même chose en disant que le nombre doit être divisible par 256 (= \$100). Pour résoudre ce problème, nous demandons un espace de 32256 octets, et nous portons l'adresse de début de cet espace-mémoire au nombre supérieur le plus proche divisible par 256. Nous utilisons pour cela l'opérateur logique ET avec \$ffffff00 et l'addition de 256. Nous utilisons la fonction 'malloc()' pour obtenir l'adresse de début elle-même.

Il nous faut maintenant une fonction servant à dessiner une ellipse. Dans notre programme, elle porte un nom qui prête à confusion: 'circle'. Cette fonction 'circle' se voit attribuer un paramètre nommé 'radius' qui doit être compris entre 0 et 50, ceci pour que notre programme puisse tourner quelle que soit la résolution de l'écran. 'radius' contient donc une valeur toute relative (50 = maximum). La résolution de l'écran est consignée sous 'v_opnvwk'.

Il nous reste à assurer le passage d'un écran à l'autre, ce dont se charge la fonction du XBIOS nommée 'setscreen' (base logique, base physique, résolution). Une valeur restant inchangée est désignée par un -1.

'Résolution' peut prendre les valeurs suivantes: 0 résolution basse, 1 résolution moyenne, 2 résolution haute. Un changement de résolution n'a de sens qu'avec un moniteur-couleur pour passer de 0 à 1 ou inversement. Attention: les paramètres transmis à 'setscreen' ne sont pris en compte qu'au retour d'image (VBL) suivant. 'Setscreen' n'est pas pour autant obligé d'interrompre le programme jusqu'au

retour d'image, car cette fonction ne sert qu'à consigner le changement à venir, lequel est réalisé à proprement parler par une routine-interrupt.

En assembleur, on écrit pour 'setscreen':

move.w	#nouvelle résolution, -(sp)	
pea	base logique	
pea	base physique	
move.w	#5, -(sp)	; numéro de fonction
trap	#14	; XBIOS
add.w	#12, sp	; correction du stack

7.11.4 Bande annonce

Voici un autre exemple de réalisation faisant appel à deux écrans, cette fois en assembleur. Le programme ci-dessous ne tourne qu'avec un moniteur monochrome, mais il vous sera facile de l'adapter aux deux autres résolutions. Il recourt à des routines 'scroll' (défilement) déjà présentées ci-dessus, et fait glisser vers la gauche une ligne de texte pixel par pixel.

Comme un caractère a une largeur de 8 points, il nous est possible d'afficher un nouveau caractère à l'extrême-droite de l'image tous les 8 appels de la routine 'scroll'. Nous faisons glisser la ligne lentement de 8 points vers la gauche, provoquons l'affichage à droite d'un nouveau caractère etc etc. Le tout vise à donner l'impression d'un texte défilant sans arrêt.

En principe, ce programme fonctionne exactement comme celui que nous venons de vous donner ci-dessus en langage C. Le sous-programme 'switch' présente cependant quelques particularités. Souvenez-vous de ce que nous avons écrit au tout début de ce paragraphe. Nous avons en effet supposé l'existence de deux espaces-mémoire au contenu rigoureusement identique avant qu'un mouvement ne se produise dans l'un. Pour le dessin des ellipses, cela suffit puisqu'il faut de toute façon redessiner entièrement chaque ellipse. Il en va autrement dans le programme d'écriture cursive: nous devons d'abord créer le contenu identique, si bien que la procédure 'switch' contient une petite routine de recopiage.

Nous ne quittons pas le sous-programme immédiatement après le recopiage, car il peut encore s'écouler un petit délai jusqu'à ce que le changement d'écran annoncé par 'setscreen' se réalise effectivement lors d'un VBL. Mais cela signifierait que nous provoquons le 'scroll' suivant dans le programme principal alors que l'écran soi-disant logique est en fait encore l'écran physique. Résultat: un scintillement de l'image. Nous devons attendre le VBL suivant, qui provoquera

le passage de l'écran logique vers l'écran physique. Il existe pour cela une fonction du XBIOS nommée 'Vsync', qui porte le numéro 37 et ne nécessite aucun paramètre.

```

;
; Ecriture cursive simple          DEFILE.S
;      MP      04-06-88
;
gemdos      =      1
xbios      =      14
print      =      9
constat    =      $b
super      =      $20
logbase    =      3
getrez     =      4
setscreen  =      5
vsync      =      37

_v_bas_ad  =      $44e

section text

clr.l      -(sp)          ;passer en mode superviseur

move.w     #super,-(sp)
trap       #gemdos
addq.l     #6,sp
move.l     d0,save_ssp

move.w     #logbase,-(sp) ; rechercher la RAM Vidéo
trap       #xbios
addq.l     #2,sp
move.l     d0,a4          ; a4:premier écran
move.l     d0,a3          ; a3:'écran d'origine'

move.l     #screen2,d0    ; deuxième écran
andi.l     #$ffffff00,d0 ; qui débute à une
                                limite de page

addi.l     #256,d0
move.l     d0,a5

clr_loop:  move.w     #31999,d0      ; vider l'écran
clr.b     (a4,d0.w)
clr.b     (a5,d0.w)
dbra     d0,clr_loop
move.w     #-1,-(sp)          ; Setscreen
move.l     a4,-(sp)          ; base physique

```

```

        move.l   a5,-(sp)      ; base logique
        move.w   #setscreen,-(sp)
        trap     #xbios
        add.l    #12,sp
main:    lea     string,a2      ; là se trouvent les
                                caractères
main_loop:  tst.b    (a2)          ; fin du string?
           beq.s  main         ; recommencer au début
           move.b (a2)+,output+4 ; écrire et afficher
                                les caractères
           pea    output       ; dans le string output
           move.w #print,-(sp)
           trap   #gemdos
           addq.l #6,sp

           moveq  #7,d2        ; laisser circuler un
                                caractère
inner_loop: bsr    scroll_left  ; 1 caractère = 8 pixels
           bsr    switch       ; changer d'écran

           move.w #constat,-(sp) ; touche appuyée?
           trap   #gemdos
           addq.l #2,sp
           tst.w  d0
           bne.s  ende         ; si oui, interruption

           dbra   d2,inner_loop

           bra.s  main_loop

ende:    move.l   save_ssp,-(sp) ; retour au mode user
        move.w   #super,-(sp)
        trap     #gemdos
        addq.l   #6,sp

        move.w   #-1,-(sp)      ; setscreen en état
                                normal

        move.l   a3,-(sp)
        move.l   a3,-(sp)
        move.w   #setscreen,-(sp)
        trap     #xbios
        add.l    #12,sp

        clr.w    -(sp)
        trap     #1
scroll_left:  move.l  _v_bas_ad,a6
           lea    2560(a6),a6    ; scrolling une ligne
sur deux

lt_lp1:    moveq  #15,d0        ; 16 scanlines par ligne
           moveq  #39,d1        ; 40 mots par ligne de
                                pixel

```

```

lt_lp2:      move    #0, ccr           ; effacer le bit-X
             roxl    -(a6)
             dbra   d1, lt_lp2
             dbra   d0, lt_lp1
             rts

switch:     exg     a4, a5           ; échanger l'adresse
                                                logique avec
                                                ; l'adresse physique
             move.w # -1, -(sp)     ; puis changer d'écran
             move.l a4, -(sp)     ; nouvelle adresse
                                                physique
             move.l a5, -(sp)     ; nouvelle adresse
                                                logique
             move.w #setscreen, -(sp)
             trap   #xbios
             add.l  #12, sp

             lea   1280(a4), a0     ; recopiage dans le
                                                nouvel écran
             lea   1280(a5), a1     ; logique
copy_loop:  move.w  #320, d0
             move.l (a0)+, (a1)+
             dbra   d0, copy_loop

             move.w #vsync, -(sp)   ; attendre un VBL
             trap   #xbios
             addq.l #2, sp

             rts

             section data
output:     dc.b    27, 'Y', 33, 111, ' ', 0; placer le curseur
                                                sur la dernière colonne de la 2ème ligne
                                                ; et afficher un caractère
                                                ; (qui vient s'insérer)

string:     dc.b    'MicroAPP vous présente : TRUCS & ASTUCES '
             dc.b    'sur ST ; Une mine de trouvailles pour tous
                                                ceux qui '
             dc.b    'possèdent cet ordinateur. Trucs et astuces
                                                en Basic, '
             dc.b    'en C et en assembleur. Quelques exemples :

comment '
             dc.b    'cacher, comprimer ou encoder des fichiers'
             dc.b    ' - GEM-Autostarter - '
             dc.b    ' REM-Killer pour GFA-BASIC - Touches

paramétrables.'
             dc.b    ' - Des fontes GEM dans vos programmes avec

GDOS'

```

```

dc.b ' - Rechargement de programmes - Véritable multitasking'
      dc.b ' - Réglage de la vitesse de la souris,
etc...'
      dc.b '          ',0
      section bss
save_ssp: ds.l 1
screen2: ds.b 32256
end

```

7.11.5 Affichage clignotant

On aimerait souvent pouvoir attirer l'attention de l'utilisateur sur un endroit particulier de l'écran, par exemple lorsqu'il y a affichage d'un message d'alarme ou d'une mention comme 'appuyez sur une touche pour continuer'. Comme l'être humain perçoit vite et bien les choses qui bougent, on pense tout de suite à un affichage clignotant. Contrairement aux ordinateurs compatibles IBM-PC, l'Atari n'est pas équipé d'origine de cette possibilité d'affichage. Mais comme bien souvent, le software peut ici remédier aux déficiences du hardware.

Que faut-il pour faire clignoter une partie de l'écran? Pour qu'un mot clignote, il faut qu'il s'efface et se réécrive à intervalles réguliers, ce qu'on peut formuler de la façon suivante: nous avons deux images du même texte à l'écran, dans l'une le mot n'existe pas alors qu'il est présent dans l'autre et il suffit alors de faire alterner les deux images.

Nous allons résoudre ce problème toujours à l'aide de nos deux écrans, en installant nos deux RAM-Vidéo internes. Les caractères qui doivent clignoter ne seront présents que dans un seul écran, le reste du texte dans les deux écrans. Il nous suffit alors de basculer régulièrement d'un écran à l'autre. Si nous faisons intervenir un interrupt, le programme pourra même continuer de tourner tout à fait normalement et ses affichages clignoter.

Examinons le texte du programme: il semble bien long à première vue, mais la routine interrupt par elle-même commence au label 'vbl' et ne comprend que 14 lignes en assembleur. Le reste sert à initialiser l'écran, passer en mode superviseur, insérer la routine de clignotage dans le VBL-Queue ainsi qu'à éditer quelques lignes d'exemples dont certains mots vont clignoter.

Etudions d'abord l'interrupt: on commence par y décrémenter un compteur, pour que la fréquence de clignotage ne soit pas trop élevée. Si le compteur reste supérieur à zéro après avoir été décrémenté, il ne se passe rien. Sinon, il faut activer et afficher l'écran resté jusque là caché: la variable 'shown' détermine quel était l'écran affiché jusqu'à présent. La nouvelle adresse de base Vidéo est alors transmise à D0 et de là dans le registre adéquat de la puce Vidéo. Attention: le registre de cette puce n'a que deux octets pour les adresses d'écran, le high-byte et le mid-byte. Il manque un low-byte, qui est toujours sur zéro: vous vous rappelez que la RAM vidéo doit toujours commencer à une limite de page.

Vous vous étonnerez peut-être de voir le compteur se recharger tantôt avec 25 et tantôt avec 35. Il ne s'agit pas d'une erreur, car cela assure un affichage plus long du premier écran par rapport au deuxième. Si le morceau de texte qui doit clignoter se trouve dans le premier écran, il sera plus facilement lisible puisqu'il restera plus longtemps présent qu'absent à l'écran.

Venons-en à l'affichage du texte: le texte normal, celui qui ne clignote pas, doit se trouver dans les deux écrans logiques. C'est pourquoi on appelle deux fois la routine d'affichage, en changeant d'adresse vidéo la deuxième fois. Toutes les routines d'affichage du système d'exploitation se servent de cette adresse pour savoir où se trouve l'espace-mémoire contenant ce qui doit s'afficher. Nous pouvons modifier cet espace-mémoire pour que l'affichage se fasse la deuxième fois dans le deuxième écran qui est (à ce moment-là) encore invisible. Le morceau de texte qui doit clignoter ne se trouve lui que dans la première RAM-Vidéo.

Notons en passant que ce processus vous permet aussi d'embellir votre curseur, en lui donnant par exemple la forme d'un tiret clignotant. Vous pouvez aussi faire passer un morceau de texte de son état normal à son état en inversion vidéo (écriture blanche sur fond noir). Voici un petit problème pour nos lecteurs qui sont des programmeurs avertis: Essayez d'insérer dans l'émulateur VT-52 les commandes de clignotage: utilisez pour cela la fonction du BIOS nommée 'Bconout'. Les caractères à afficher doivent se trouver dans les deux écrans; assurez-vous que les commandes habituelles du VT-52 soient envoyées vers l'affichage par mot entier et non caractère par caractère. Voilà un petit casse-tête qui n'est pas des plus simples! Mais vous allez pouvoir étudier l'exemple ci-dessous et y repêcher quelques idées:

```

;
; Affichage clignotant           CLIGNOTE.S
;      MP      04-06-88
;
gemdos      =      1

```

```

xbios      =      14
conin      =      7
print      =      9
constat    =      $b
super      =      $20
keep       =      $31
logbase    =      3
setscreen  =      5
superexec  =      38
logique    =      $44e
_vblqueue  =      $456      ;pointeur vers une liste de
                               ;routines VBL
nvbls      =      $454      ;Nombre de routines VBL autorisées
midbyte    =      $ffff8203 ;ici se trouve l'adresse physique
highbyte   =      $ffff8201 ;de la RAM Vidéo

        section text
move.l     4(sp),a0      ;calculer l'espace-mémoire
        move.l     #$100,d6
        add.l      12(a0),d6
        add.l      20(a0),d6
        add.l      28(a0),d6

        move.w     #logbase,-(sp)      ;rechercher la RAM vidéo
        trap       #xbios
        addq.l     #2,sp
        move.l     d0,screen0          ;premier écran

        move.l     #screen2,d0          ;deuxième écran
        andi.l     #$ffffff00,d0       ;commencer à une limite
                                         de page

        addi.l     #256,d0
        move.l     d0,screen1

        clr.l      -(sp)                ;passer en mode
                                         superviseur

        move.w     #super,-(sp)
        trap       #gemdos
        addq.l     #6,sp
        move.l     d0,save_esp

        bsr        init                 ;initialisation
                                         (VBL-Queue)

        dc.w       $a00a                ;Hide cursor

        pea        noblink              ;Affichage du texte
                                         normal

        move.w     #print,-(sp)
        trap       #gemdos
        addq.l     #6,sp

```

```

        pea      blink                ;Texte devant clignoter
        move.w   #print,-(sp)
        trap     #gemdos
        addq.l   #6,sp

        move.l   screen1,logique     ;Base vidéo logique

        pea      noblink              ;le texte normal
                                           figure aussi
        move.w   #print,-(sp)        ;dans le deuxième écran
        trap     #gemdos
        addq.l   #6,sp

move.w   #conin,-(sp)
        trap     #gemdos
        addq.l   #2,sp

move.l   vbl_slot,a0                 ;désactiver le VBL
        clr.l   (a0)

l'état normal
        move.w   #-1,-(sp)           ;setscreen pour

        move.l   screen0,-(sp)
        move.l   screen0,-(sp)
        move.w   #setscreen,-(sp)
        trap     #xbios
        add.l   #12,sp

        dc.w    $a009                ;shown cursor

        move.l   save_esp,-(sp)      ;retour au mode user
        move.w   #super,-(sp)
        trap     #gemdos
        addq.l   #6,sp

        clr.w   -(sp)
        trap     #1

init:
        move.w   nvbls,d0             ;nombre de routines VBL
        lsl.w   #2,d0                ;multiplié par 4
        move.l   _vblqueue,a0        ;liste des adresses
        clr.w   d1

search:
        tst.l   (a0,d1)              ;si libre, on trouve ici
                                           ;un zéro

        beq.s   found
        addq.w   #4,d1                ;sinon scruter
                                           l'enregistrement suivant
        cmp.w   d0,d1                ;tous scrutés?
        bne.s   search

```

```

        illegal                ;si oui, bombes

found:   lea      (a0,d1),a0    ;calculer adresses des
        move.l   a0,vbl_slot  ;les consigner pour
        move.l   #vbl,(a0)    ;plus tard et
        rts                    ;inscrire le vecteur
                                dans slot

vbl:     subi.b   #1,counter    ;boucle d'attente
        bne.s   finish

        bchg     #0,shown      ;changer d'écran
        bne.s   label        ;physique
        move.l   screen0,d0
        move.b   #35,counter  ; recharger le
                                compteur (long)

label:   bra.s    label2
        move.b   #25,counter  ;recharger le
                                compteur (court)

label2:  move.l   screen1,d0
        lsr.l   #8,d0
        move.b   d0,midbyte
        lsr.w   #8,d0
        move.b   d0,highbyte

finish:  rts

        section data

shown:   dc.b    0              ;écran actuel
counter: dc.b    35            ;compteur de délai

noblink: dc.b    27,'E',10,10,'Ce programme vous montre
comment '
        dc.b    'provoquer un affichage clignotant
en',13,10
        dc.b    'utilisant deux écrans.',13,10,0

blink:   dc.b    13,10,10,' Cette ligne va clignoter.',0

        section bss

screen0: ds.l    1
screen1: ds.l    1

vbl_slot: ds.l   1
save_ssp: ds.l   1

```

```
screen2:      ds.b      32256
              end
```

Encore un dernier conseil pour finir: si vous êtes absolument sûr de ne jamais faire tourner votre programme que sur un moniteur couleur, vous disposez d'une possibilité bien plus simple pour faire clignoter un morceau de texte. Si vous écrivez normalement en rouge sur fond blanc, vous pouvez sélectionner deux registres couleurs (tous deux rouges) pour votre texte, l'un des deux passant du rouge au blanc et inversement par le biais de l'interrupt. Vous épargnez ainsi les 32 octets du deuxième écran ainsi que le double affichage; bien entendu, cela ne fonctionnera qu'avec un moniteur couleur, mais c'est une solution tout à fait honnête dans un programme de jeu.

7.11.6 Fondu enchaîné des images

On aimerait souvent ne pas être contraint de faire apparaître/disparaître brutalement les illustrations dans un programme, mais les voir 'monter' lentement sur l'écran: en terme de musique ou de technique vidéo, on appelle cela le 'fading'. Cette technique est d'un bel effet lorsqu'elle s'applique à des pages de titre ou des illustrations de commentaires.

Quel est le principe de base du fading? Il consiste à faire augmenter ou décliner très progressivement l'intensité de toutes les couleurs utilisées dans l'image. Cette notion d'intensité des couleurs vous amène déjà à penser que ce procédé ne pourra s'appliquer qu'avec des applications couleur.

Le programme ci-dessous est écrit en Basic-GFA, il vous montre une image de test se composant de huit couleurs. Il peut donc tourner en résolution basse. Durant le processus d'apparition de l'image, tous les registres couleurs sont placés sur blanc à l'aide de la commande 'setcolor', si bien que vous ne voyez rien. Les couleurs voulues sont reprises dans les lignes data et portées dans 'array F%()', leurs composants rouge, vert et bleu (RVB) étant séparés. Comme nous n'utilisons que 8 couleurs sur 16, nous n'utilisons que 8 crayons différents.

Chacun des trois composants RVB peut prendre une valeur comprise entre 0 et 7. Sur 0, la couleur est désactivée, et sur 7 elle est à son maximum d'intensité. Il ne nous reste donc qu'à faire parcourir une à une toutes les étapes comprises entre l'intensité 0 et l'intensité 7. C'est la variable S% qui prend ces valeurs successivement; la formule

intensité maximale * S% / 7

nous permet de calculer facilement les valeurs nécessaires.

Encore une petite astuce: on peut combiner ce procédé avec l'animation d'images décrite au chapitre 'programmes résidents' (7.3). Il vous reste cependant à calculer les différentes intensités de couleur et à vous en confectionner un tableau.

Voici le texte de ce programme:

```
'
' Fondu-enchaîné des images en couleur      FADE.BAS
'      MP      25-06-88
'
Dim F%(7,2)
'
For I%=0 To 7          !valeur des intensités, en
commençant
  For J%=0 To 2        !par tout désactiver
    Read F%(I%,J%)    !tout est noir
  Next J%
  Setcolor I%,0,0,0
Next I%
'
For I%=8 To 79        !peindre l'image
  Color I% Mod 8      !avec les 8 premières couleurs
  Defill I% Mod 8,1,1
  Pcircle Random(320),Random(200),Random(20)+10
Next I%
'
S%=1                  ! S% représente l'intensité de 0 à 7
Increment%=1
Repeat
  Void Xbios(37)      ! attendre un retour-image
  For I%=1 To 7
    Setcolor
    I%,Int(F%(I%,0)*S%/7),Int(F%(I%,1)*S%/7),Int(F%(I%,2)*S%/7)
  Next I%
  Pause 3              ! déterminer ici le temps nécessaire
  If S%=0 Or S%=7    ! pour faire monter l'image
    Increment%=-Increment%
    Pause 30
  Endif
  Add S%,Increment%
Until Bios(1,2)      ! attendre action d'une touche
'
Data 0,0,0, 7,0,0, 0,7,0, 0, 0, 7, 2,2,2, 1,3,5, 6,7,4, 3,4,8
'
```

```
Setcolor 0,7,7,7
Setcolor 1,0,0,0           ! couleurs à-demi normales
```

7.11.7 Soft-scrolling

Nous allons maintenant vous présenter des routines un peu plus compliquées, que vous pourrez reprendre telles quelles dans vos programmes, mais qui pourront aussi vous servir d'exemples vous incitant à expérimenter et à les améliorer. Elles servent à faire défiler le contenu de l'écran pixel par pixel tant dans le sens vertical qu'horizontal (en anglais: scroll = un rouleau de papier).

Mais le soft-scrolling sollicite beaucoup les possibilités de l'Atari, puisqu'il s'agit de mettre en mouvement 32000 octets. Il est donc évident qu'il faut ici utiliser l'assembleur.

Vous connaissez maintenant l'organisation interne de la mémoire-écran. Vous vous rappelez que cette mémoire varie quelque peu selon le degré de résolution de l'image: c'est pourquoi, pour ne pas trop ralentir notre routine, nous avons écrit deux programmes séparés, l'un pour les moniteurs monochromes et l'autre pour les moniteurs couleurs.

Nous ne vous donnerons que les routines pour la résolution haute et moyenne.

Les explications qui suivent se rapportent toujours à la résolution haute. Commençons par le plus simple, le scrolling vertical. Il s'agit tout simplement d'une boucle de recopiage: en descendant, il suffit de décaler d'une ligne vers le bas les lignes 0 à 398 (nous comptons en lignes-pixel). On lit d'abord la ligne 398 qu'on reporte dans la ligne 399, puis la ligne 397 qui est reportée dans la 398 etc etc jusqu'à la ligne 0 qui devient la ligne 1. S'il s'agit par contre d'un défilement vers le haut, nous travaillons en ordre inverse: la ligne 1 passe à la ligne 0, la ligne 2 à la ligne 1 etc jusqu'à la ligne 399 qui passe en 398. Les adresses de départ de la source et de la cible se différencient d'une ligne soit 80 octets.

L'affaire se complique un peu avec le défilement horizontal, car nous ne pouvons plus faire glisser des octets ou des mots entiers. Nous devons travailler au bit près. Heureusement, il nous suffit de décaler ces bits d'un seul bit, ce que nous pouvons faire à l'aide de 'roxr' et 'roxl': ces deux commandes servent à décaler un mot de la mémoire d'un bit vers la droite ou vers la gauche. Pendant cette 'rotation', le bit X du registre des états passe dans le premier bit libéré, pendant que le bit créé par le processus lui-même passe dans le X-bit. Cela nous permet de faire glisser très

facilement les mots de la mémoire situés l'un à côté de l'autre: une petite boucle suffit, qui contient les commandes 'roxl/roxr'.

Attention: nous avons tenu à vous présenter un programme s'articulant en sous-programmes facilement compréhensibles et non un programme très long, très compliqué et hyper-rapide. Nos routines n'ont pas un rendement maximal en matière de rapidité d'exécution, si bien que vous pouvez considérablement les améliorer. En étudiant bien le chapitre consacré au 'movem', vous allez certainement trouver une astuce tout à fait brillante...

```

;
; routines de défilement          SCROLL_H.S
;   MP      10-04-88
;

Xbios  = 14
Gemdos = 1

        section text

        move.w #3, -(sp)
        trap  #Xbios
        move.l d0, Physbase

scroll_up:  move.l  Physbase, a5
            lea    80(a5), a6
            move.l #32000/4-21, d0
up_loop:   move.l  (a6)+, (a5)+
            dbra  d0, up_loop

scroll_down:  move.l  Physbase, a5
            lea    32000(a5), a5
            lea    -80(a5), a6
            move.l #32000/4-21, d0
down_loop:   move.l  -(a6), -(a5)
            dbra  d0, down_loop

scroll_right:  move.l  Physbase, a6
            move.w #399, d0
rt_lp1:      moveq   #39, d1
            move   #0, ccr          ;effacer le bit-X
rt_lp2:      roxr   (a6)+
            dbra  d1, rt_lp1
            dbra  d0, rt_lp1

```

```

scroll_left:   move.l   Physbase,a6
               lea     32000(a6),a6
               move.w   #399,d0
lt_lp1:       moveq    #39,d1
               move     #0,CCR
lt_lp2:       roxl     -(a6)
               dbra    d1,lt_lp2
               dbra    d0,lt_lp1

               clr.w   -(sp)
               trap    #Gemdos

               section bss

Physbase:     ds.l 1
               end

```

Nous allons vous donner le même programme appliqué à la résolution moyenne, pour bien vous montrer le type de modifications nécessaires. La chose est simple pour le défilement vertical: nous devons juste tenir compte du fait qu'une ligne d'écran se compose alors de 160 octets (et non plus de 80). Pour le reste, les deux routines sont identiques.

Les choses se compliquent avec le défilement horizontal. Les mots ne se suivent plus directement l'un derrière l'autre, ils occupent un mot sur deux dans la mémoire-écran. Entre les deux, nous trouvons un mot d'une autre couleur. Nous allons résoudre ce problème en parcourant deux fois la boucle, une fois par couleur. D'où l'apparition de la commande 'addq.l #2,a6' pour sauter un mot.

```

;
; Routines de défilement (résolution moyenne)
SCROLL_M.S
;          MP          10-04-88
;

Xbios     = 14
Gemdos    = 1

               section text

               move.w   #3,-(sp)
               trap    #Xbios
               move.l   d0,Physbase

scroll_up:   move.l   Physbase,a5
               lea     160(a5),a6

```

```

up_loop:      move.l   #32000/4-41,d0
              move.l   (a6)+,(a5)+
              dbra     d0,up_loop

scroll_down:  move.l   Physbase,a5
              lea     32000(a5),a5
              lea     -160(a5),a6
              move.l   #32000/4-41,d0
down_loop:    move.l   -(a6),-(a5)
              dbra     d0,down_loop

scroll_right: move.l   Physbase,a6
              move.w   #199,d0
rt_lp1:      moveq    #39,d1
              move     #0,ccr           ; effacer le bit X
rt_lp2:      roxr     (a6)+
              addq.l   #2,a6
              dbra     d1,rt_lp2
              dbra     d0,rt_lp1
              move.l   Physbase,a6
              move.w   #199,d0
rt_lp3:      moveq    #39,d1
              move     #0,ccr           ; effacer le bit X
rt_lp4:      addq.l   #2,a6
              roxr     (a6)+
              dbra     d1,rt_lp4
              dbra     d0,rt_lp3

scroll_left:  move.l   Physbase,a6
              lea     32000(a6),a6
              move.w   #199,d0
lt_lp1:      moveq    #39,d1
              move     #0,ccr
lt_lp2:      roxl     -(a6)
              subq.l   #2,a6
              dbra     d1,lt_lp2
              dbra     d0,lt_lp1
              move.l   Physbase,a6
              lea     32000(a6),a6
              move.w   #199,d0
lt_lp3:      moveq    #39,d1
              move     #0,ccr
lt_lp4:      subq.l   #2,a6
              roxl     -(a6)
              dbra     d1,lt_lp4
              dbra     d0,lt_lp3

              clr.w   -(sp)
              trap    #Gemdos

```

```

                section bss

Physbase ds.1 1

                end

```

7.11.8 Programmation de symboles 'Sprite'

Si, avant d'avoir un Atari, vous possédiez un C64, vous connaissez déjà ce terme 'sprite'. Il désigne un petit symbole graphique que l'on peut déplacer à volonté sur l'écran. Dans le C64, cela se faisait au niveau du hardware, et le choix se limitait à 8 possibilités. Dans l'Atari ST, nous opérons au niveau du software, ce qui nous ouvre un nombre quasi illimité de possibilités.

Seule limite: le sprite doit compter au maximum 16 fois 16 points. Les exemples les plus connus sont la flèche de la souris ou la petite abeille, qui sont engendrées par les mêmes fonctions. Vous n'avez pas besoin d'être un programmeur chevronné si vous utilisez le bon langage de programmation. En langage C par exemple, vous ne pourrez pas programmer de sprite sans une bibliothèque complémentaire de routines LineA. En Basic-GFA par contre, l'interpréteur vous offre des commandes vous permettant de créer facilement des sprites:

```

'
' Programme : SPRITE.BAS
'
' Ce programme vous montre la programmation des sprites
' en basic GFA.
'
Hidem                                ! Cacher la souris
Dim Sprite1$(16),Sprite2$(16),Sprite3$(16) ! champs Sprites
Dim Maske1$(16),Maske2$(16),Maske3$(16)    ! champs Masques
'
For I%=0 To 15
  Read Sprite1$(I%)                      ! données Sprite 1
Next I%
'
For I%=0 To 15
  Read Maske1$(I%)                        ! données Masque 1
Next I%
'
For I%=0 To 15
  Read Sprite2$(I%)                        ! données Sprite 2

```

```

Next I%
'
For I%=0 To 15
  Read Maske2$(I%)           ! données Masque 2
Next I%
'
For I%=0 To 15
  Read Sprite3$(I%)         ! données Sprite 3
Next I%
'
For I%=0 To 15
  Read Maske3$(I%)         ! données Masque 3
Next I%
'
Spr1$=Mki$(1)+Mki$(1)+Mki$(0)+Mki$(0)+Mki$(1) ! Préparer les
                                                sprites
Spr2$=Spr1$
Spr3$=Spr1$
'
For I%=0 To 15               ! 16 Mots
  Mask1_dat%=0
  Spr1_dat%=0
  Mask2_dat%=0
  Spr2_dat%=0
  Mask3_dat%=0
  Spr3_dat%=0
'
For J%=1 To 16               ! 16 bits
  Mul Mask1_dat%,2
  If Mid$(Maske1$(I%),J%,1)="*" ! Bit positif ?
    Add Mask1_dat%,1
  Endif
'
  Mul Spr1_dat%,2
  If Mid$(Sprite1$(I%),J%,1)="*"
    Add Spr1_dat%,1
  Endif
'
  Mul Mask2_dat%,2
  If Mid$(Maske2$(I%),J%,1)="*"
    Add Mask2_dat%,1
  Endif
'
  Mul Spr2_dat%,2
  If Mid$(Sprite2$(I%),J%,1)="*"
    Add Spr2_dat%,1
  Endif
'
  Mul Mask3_dat%,2
  If Mid$(Maske3$(I%),J%,1)="*"
    Add Mask3_dat%,1

```

```

    Endif
    '
    Mul Spr3_dat%,2
    If Mid$(Sprite3$(I%),J%,1)="*"
        Add Spr3_dat%,1
    Endif
Next J%
Spr1$=Spr1$+Mki$(Mask1_dat%)+Mki$(Spr1_dat%) ! réunir les
                                                données
Spr2$=Spr2$+Mki$(Mask2_dat%)+Mki$(Spr2_dat%)
Spr3$=Spr3$+Mki$(Mask3_dat%)+Mki$(Spr3_dat%)
Next I%
'
' Spielanfang
'
X_off%=1                                !
Y_off%=1
Dim Spr_an%(11)                          ! Flag Sprites
Dim Spr_ennemi$(10)                      ! Place pour 10 ennemis
'
New_level:
'
For I%=1 To 10
    Spr_an%(I%)=1
    Spr_ennemi$(I%-1)=Spr2$
Next I%
Spr_an%(0)=10
'
X2%=25
Y2%=20
Do
    Exit If Spr_an%(0)=0
    Add X2%,X_off%
    If X2%>=50 Or X2%<=1
        X_off%=-X_off%
        Add Y2%,Y_off%
    Endif
    For I%=0 To 9
        If Spr_an%(I%+1)=1
            Sprite Spr_ennemi$(I%),X2%+I%*64,Y2%
        Endif
    Next I%
'
X1%=Mousex
Sprite Spr1$,X1%,360
'
If Mousek=1 And Schuss_flag%=0
    Schuss_flag%=1
    X_schuss%=X1%
    Y_schuss%=340

```

```

Endif
'
If Schuss_flag%=1
  Sub Y_schuss%,10
  Sprite Spr3$,X_schuss%,Y_schuss%
  If Y_schuss%<=Y2%+20
    Sprite Spr3$
    Schuss_flag%=0
    Sp_nr%=(X_schuss%+X2%+32) Div 64
    Sp_off%=(X_schuss%) Mod 64
    If Abs(Sp_off%-X2%)<10 And Spr_an%(Sp_nr%)=1
      Sprite Spr_ennemi$(Sp_nr%-1)
      Spr_an%(Sp_nr%)=0
      Dec Spr_an%(0)
    Endif
  Endif
Endif
Endif
'
If Y2%>320
  Alert 2," GAME OVER | NEW GAME ?",1," Oui | Non ",Wahl%
  If Wahl%=1
    Run
  Endif
  Run
Endif
'
If Mousek=2
  Do
    Exit If Mousek<>2
  Loop
Do
  Exit If Mousek=2
  Loop
  Do
    Exit If Mousek<>2
  Loop
Endif
'
Loop
'
Alert 2," WOUAHHH...! YOU WIN...!",1,"OK",Dummy%
X_off%=Abs(X_off%)+1      ! Difficulté plus grande
X_off%=5
  X_off%=2
  Inc Y_off%
Endif
Goto New_level
'
' Données Sprite
'

```



```

Data "      * * * * * "
Data "      * * * * * "
Data "      * * * * "
Data "      * * * "
Data "      * "
Data "      "
Data "      "
Data "      "
Data "      "

```

Voici la structure du string pour le Sprite:

Mot	Signification
1	position x du point d'action (important pour la souris)
2	position y du point d'action
3	mode: 0=normal 1=mode Xor
4	couleur du masque: 0=blanc 1=noir
5	couleur du Sprite: 0=blanc 1=noir
6-37	en alternance: masque et Sprite

La position où vient se dessiner le Sprite est calculée par rapport au point d'action qui se trouve, pour plus de simplicité, en haut à gauche (0.0).

Nous avons inséré dans le programme une petite routine qui vous aidera à créer vos propres Sprites. Dans les lignes Data, nous avons symbolisé par * les bits positifs; les espaces vides représentent des bits nuls. Ceci vous permet dès la saisie d'avoir déjà une idée assez précise de l'aspect de votre Sprite.

Il y a un petit problème lorsque deux Sprites se chevauchent. Quand le programme dessine un Sprite sur l'écran, il sauvegarde auparavant dans une mémoire temporaire la partie de l'écran qui va être recouverte afin de pouvoir ensuite la restaurer. Si un Sprite vient chevaucher un autre Sprite, c'est ce dernier qui va être mémorisé. S'il vient à se déplacer avant celui qui le recouvre, le système d'exploitation va restaurer la partie d'écran à partir des anciennes données du Sprite, si bien que celui-ci aura l'air d'être resté sur place. Nous avons tourné ce problème en faisant de telle sorte que le projectile disparaisse juste avant de heurter le Sprite.

Si vous faites tourner ce jeu, vous allez constater que le programme est considérablement ralenti par la présence de si nombreux Sprites en Basic-GFA. Le programme s'accélère au fur et à mesure que les ennemis disparaissent. Si vous essayez d'améliorer le jeu, par exemple en comptant les coups, vous ralentirez encore plus le programme. C'est la raison pour laquelle les professionnels écrivent

les programmes de jeu en Assembleur; c'est aussi pourquoi nous allons vous donner un exemple de Sprite en Assembleur. Notez cependant que ce programme n'est pas un jeu complet, il ne sert qu'à vous montrer le maniement des fonctions servant à la programmation des Sprites.

```

*****
* Exemple de programmation de SPRITES *
*****

        clr.w        d0          * position X du Sprite
        clr.w        d1          * position Y du Sprite
        lea          sprite,a0   * pointeur sur les données du
                                * Sprite
loop:   lea          spr_sv,a2   * pointeur sur la sauvegarde

        movem.l     d0/d1/a0/a2,-(sp) * sauver le registre
        .dc.w       $a00d       * dessin du Sprite
        bsr.s       wait        * nécessaire pour y voir
                                * quelque chose
        lea          spr_sv,a2   * pointeur sur la sauvegarde
        .dc.w       $a00c       * effacer le Sprite
        movem.l     (sp)+,d0/d1/a0/a2 * reprendre le registre
        addq.w      #1,d0        * augmenter la position x
        addq.w      #1,d1        * augmenter la position y
        cmp.w       #640,d0     * arrivé à la fin?
        bne.s       no_x_end    * non
        clr.w       d0          * sinon position x = 0
no_x_end:
        cmp.w       #400,d1     * arrivé à la fin?
        bne.s       no_y_end    * non
        clr.w       d1          * sinon position y = 0
no_y_end:
        btst        #1,$dfe     * touche gauche de la souris enfoncée
        beq.s       loop        * non, continuer
*
        clr.w       -(sp)       * sortir du programme
        trap        #1          *
*
wait:   move.w       #1000,d7    * sous-programme boucle d'attente
                                * valeur de départ
w_loop: dbra        d7,w_loop   *
                                * jusque -1
        rts         * terminé
*****

        .data
sprite: .dc.w 0          * position x du point d'action
        .dc.w 0          * position y du point d'action

```

```

                .dc.w 0          * flag du mode
                .dc.w 0          * couleur du masque
                .dc.w 0          * couleur du Sprite
                .dc.w 0          * Masque1
                .dc.w 0          * Sprite1
                .dc.w 0          * Masque2
                .dc.w 0          * Sprite2
etc
etc
.bss
spr_sv:        .ds.w 37        * arrière-fond

```

Ce programme crée un sprite que vous connaissez déjà; une boucle décroissante à partir de 1000 est là pour que nous ayons le temps de voir ce qui se passe. Vous pouvez interrompre le programme en appuyant sur la touche gauche de la souris; si vous possédez un Méga ST, appuyez sur la touche droite.

Vous constatez que notre sprite scintille même dans ce programme en assembleur. C'est pourquoi vous devriez dans vos propres programmes vous efforcer de redessiner le nouveau sprite juste après avoir effacé l'ancien, sans intercaler d'autres commandes entre les deux. Ceci fera quasiment disparaître tout scintillement.

L'autre cause de scintillement se trouve dans le retour de ligne. Si le sprite est dessiné juste au moment où le retour de ligne se trouve au milieu de la ligne en cours, le bas du sprite apparaîtra en premier, et il faudra attendre le prochain retour d'image pour voir apparaître le reste du sprite. Il en va de même lorsqu'il s'agit de l'effacer. La solution réside dans le fait d'attendre un VBL, le rayon se trouvant alors au tout début de l'image, ce qui se fait en utilisant la routine 'Vsync' du système d'exploitation (Xbios(37)).

Dans certaines applications, il se peut qu'un si petit sprite devienne inutilisable en raison des grandes plages utilisées. En Basic GFA, nous pouvons remédier à ce problème grâce aux commandes GET et PUT, alors qu'en assembleur il faudra dessiner plusieurs sprite l'un à côté de l'autre ou l'un en-dessous de l'autre. Cette solution n'est pas envisageable avec le Basic GFA en raison de la perte de vitesse et du scintillement qui en résulterait. Attention: si vous juxtaposez plusieurs sprites, faites attention qu'ils ne viennent pas à se chevaucher.

Chapitre 8

Trucs et astuces pour le Hardware

Nous voudrions dans ce dernier chapitre vous présenter quelques améliorations que vous pourriez facilement apporter à votre ordinateur. Nous nous en sommes tenus à des choses très simples, réalisables par n'importe qui. Malgré tout, nous vous conseillons de vous exercer un peu auparavant à la manipulation d'un fer à souder pour éviter d'endommager votre appareil.

Attention: dès que vous ouvrez vous-mêmes l'ordinateur, la garantie est caduque.

En matière d'outils, vous aurez besoin d'un petit tourne-vis de taille moyenne ou petite, d'une pince plate ou effilée, d'un fer à souder de 30 Watt maximum, ou mieux d'un fer électrique avec compensation de potentiel.

Très important: avant de commencer à réaliser une opération, lisez attentivement tout le paragraphe la décrivant; vous éviterez ainsi de commettre de grosses erreurs pouvant coûter la vie à votre ordinateur.

Après toutes ces mises-en-garde, voici les travaux pratiques.

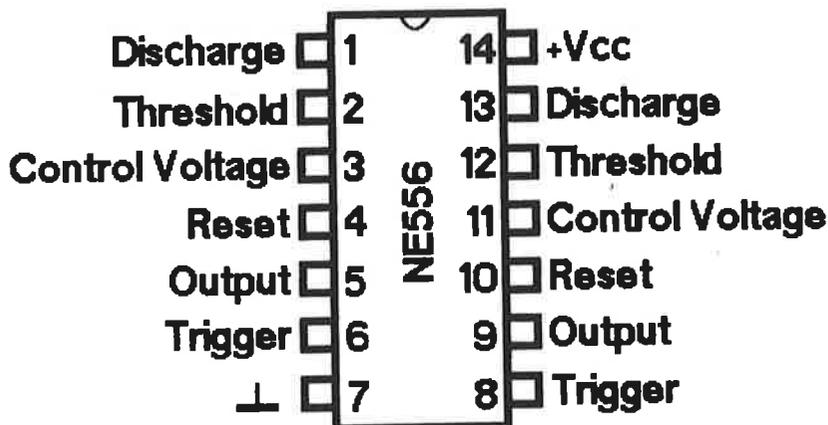
8.1 Comment allumer votre configuration complète

à partir d'un seul interrupteur

Les propriétaires de disque dur ou d'imprimante à laser connaissent bien ce problème: ils doivent d'abord allumer leur disque dur ou leur imprimante à laser, et attendre au moins 15 secondes avant d'allumer l'unité centrale. C'est assez gênant, car ceci nous empêche d'allumer toute la configuration à partir d'un seul interrupteur. Naturellement, on peut s'acheter un relais de décalage qui se chargera de décaler l'allumage de l'ordinateur, mais ceci vous coûtera au moins 300FF. Nous vous proposons une solution beaucoup plus économique.

Etudions d'abord le déroulement du processus d'allumage. Après son allumage ou après un reset, l'ordinateur doit revenir à son état initial: pour cela, les connexions 'Arrêt' et 'Reset' du processeur doivent rester un certain temps sur le 'low' logique. Lors de l'allumage, ce délai doit durer au moins 0,1 seconde et 0,5 milliseconde dans le cas du 'reset'.

Ceci se fait dans l'Atari ST par un composant nommée NE556: c'est un timer de haute précision, très coûteux, ayant peu de connexions extérieures et un courant de sortie relativement élevé. Le NE556 abrite deux timers du type NE555, l'un servant lors de l'allumage pour un délai assez long, l'autre lors du reset pour un délai plus court. Voici le plan des connexions du timer:



Nous allons nous intéresser de plus près au pôle numéro 12. Entre cette connexion et la masse se trouve un condensateur de 22 microfarad, qui sert surtout à déterminer la durée du délai de décalage à l'allumage, à l'aide de deux résistances. C'est là que nous allons intervenir, en soudant en parallèle un deuxième condensateur raccordé au premier, si bien que nous rallongeons le délai. L'ajout d'un condensateur de 1000 microfarad porte le délai à environ 15 secondes, ce qui nous suffit en principe amplement. Si vous avez malgré tout quelques problèmes, n'hésitez pas à allonger ce délai.

Venons-en au montage concret: commencez par débrancher complètement votre unité centrale de tous ses périphériques et surtout du secteur. Retournez votre unité centrale et mettez-la sur le clavier: dévissez toutes les vis visibles sur le dos. Attention: toute intervention dans l'ordinateur entraîne la nullité de la garantie. Remplacez l'unité centrale dans sa position normale et soulevez le capot.

Vous débranchez le clavier, qui n'est plus relié à l'unité que par une prise assez large. Vous devez ouvrir la carrosserie: dévissez les vis cruciformes et redressez les languettes métalliques. Dans le 1040ST, éloignez aussi la partie alimentation électrique. Ceci fait, vous avez accès à la platine.

Il ne vous reste plus qu'à trouver le composant NE556. En général, il porte une inscription commençant par 556 suivi d'autres chiffres. Dans le 1040ST, il se trouve derrière à gauche et dans le 260/520ST derrière à droite. Vous ne pouvez pas vous tromper, car il n'y a qu'un composant de ce type.

Il est temps de mettre à chauffer votre fer à souder, car on ne peut bien souder que s'il est suffisamment chaud: il faut lui laisser le temps d'arriver à la température correcte. Profitez-en pour préparer votre mèche d'étain: vous devez absolument vous procurer de l'étain spécial pour souder sur matériel électronique, car les autres sortes peuvent contenir une forte proportion de liquide corrosif. N'utilisez en aucun cas un produit de mauvaise qualité.

Examinons les petites pattes de notre composant, dont le schéma ci-dessus vous montre la répartition et la numérotation. Si l'entaille supérieure manque, la patte 1 est repérable grâce à un petit trou dans le composant. Nous intervenons sur les pattes 7 et 12: si elles ne sont plus parfaitement lisses et brillantes mais un peu ternes, essayez de les nettoyer en les frottant prudemment avec un bout de toile émeri. La soudure ne tient que sur du métal bien propre.

Prenez maintenant le nouveau condensateur: l'un des contacts doit porter le signe 'plus' ou 'moins', sans doute symbolisé par '+' ou '-'. Comme il s'agit d'un condensateur à polarisation, il faut tenir compte de la répartition des connexions.

Repérez une place libre sur la platine: dans les 260/520ST, on peut souder le nouveau condensateur directement sur le composant, dans le 1040ST, vous devrez les relier l'un à l'autre par deux petits fils et rechercher un emplacement suffisant, car vous devez garder de la place pour replacer la partie alimentation électrique. Dans le cas du 260ST ou du 520ST, repliez les pattes de votre condensateur de telle sorte que le pôle 'moins' vienne sur la patte 7 et le pôle 'plus' sur la patte 12 du composant. Si les pattes sont trop longues, recoupez-les à l'aide de ciseaux rogneurs. Dans le cas du 1040ST, préparez d'abord vos deux petits fils de raccordement puis soudez-les sur le condensateur.

Vous devez relier le compensateur de potentiel sur la masse de l'ordinateur dans la mesure où votre poste de soudure le permet. Vous étalez de la soudure sur les pattes du composant: attention à ne pas laisser s'échauffer les pattes du composant qui pourrait être détruit. Soudez ensuite le condensateur directement sur les pattes du composant; si vous soudez deux fils de raccordement, faites attention à bien respecter la polarité du condensateur.

En dernier lieu, vous devez contrôler si ce condensateur ne va pas provoquer un court-circuit pour enfin le fixer solidement quelque part sur la platine.

Après avoir remonté votre unité centrale et rebranché dessus les appareils périphériques, vous pouvez utiliser un interrupteur général de mise sous-tension de l'ensemble de la configuration, y compris le disque dur et l'imprimante à laser. L'écran va d'abord rester noir, puis vous envoyer après quelques 15 secondes une fenêtre blanche et l'ordinateur va s'initialiser comme d'habitude. Si ça ne marche pas, re-testez le tout; vous pouvez aussi contrôler le résultat d'un appui sur la touche 'reset'.

Nous n'allons pas vous cacher un petit inconvénient lié à ce montage: nous venons d'ajouter un condensateur, et comme les autres connexions sont restées telles qu'elles étaient, ce condensateur se décharge lentement. Vous devrez donc attendre environ 30 secondes pour rallumer votre ordinateur après l'avoir éteint. Mais nous pensons qu'il s'agit là d'un inconvénient très mineur, puisque généralement vous réinitialisez votre ordinateur à l'aide de la touche 'reset'.

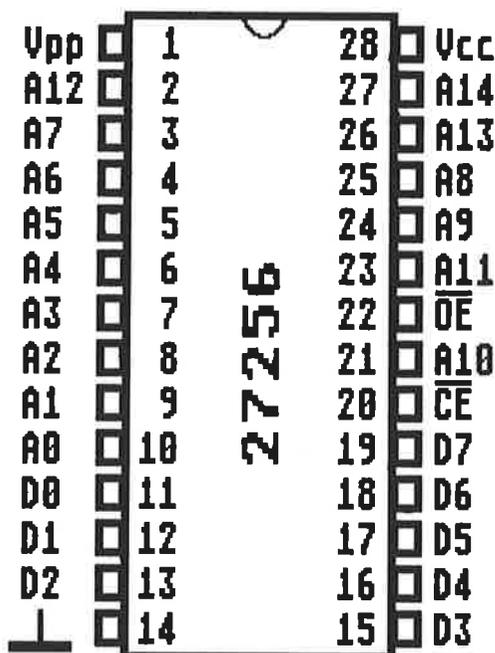
8.2 Pour passer du TOS ordinaire au Blitter-TOS

Atari a profité de l'introduction du Méga ST sur le marché pour fournir un nouveau TOS, capable de gérer une horloge en temps réel ainsi qu'un blitter servant à déplacer rapidement de gros blocs-mémoire. Il en résulte quelques modifications,

si bien que certains logiciels tournant avec l'ancien TOS, qui supposait l'usage direct de variables non-attestées ou de sauts dans le système d'exploitation, ne fonctionnent plus sur le Méga ST.

Comme ce nouveau TOS va être implanté également dans l'Atari 1040ST/F, certains programmes ne tourneront plus non-plus sur cet ordinateur. L'utilisateur a deux solutions: soit attendre que les firmes de software changent leurs logiciels, soit mettre la main à la pâte.

Dans l'Atari ST, le système d'exploitation réside dans six composants, qu'on appelle des Prom (Programable Read-Only Memory = mémoires programmables n'autorisant ensuite que la lecture). Chacune de ces Prom à une taille de 32 K-octets, ce qui donne toutes réunies 192 K-octets. La répartition des connexions sur ces Prom est la même que sur les Eprom (mémoires effaçables) portant la référence 27256.



Le schéma ci-dessus vous montre qu'une Eprom dispose de 15 connexions d'adresses numérotées de A0 à A14 et de 8 connexions de données numérotées de D0 à D7. On s'aperçoit immédiatement que l'Eprom ne peut ainsi fournir qu'un octet à la fois. Comme le ST est une machine fonctionnant avec des mots, il sollicite

toujours deux Eprom simultanément. Les lignes de données de la première Eprom aboutissent sur le bus de calcul aux entrées D0 à D7 et celles de la deuxième Eprom aux entrées D8 à D15.

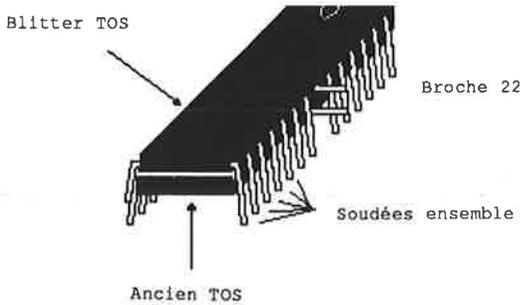
Plusieurs participants dépendent d'un tel bus de données, qui ne doit donc comporter que les données appelées qui sont nécessaires à un moment donné. Le reste du temps, l'Eprom doit se comporter comme si elle n'existait même pas: on nomme cela le 'tri-state', le troisième état. Il existe deux possibilités de placer une Eprom en état tri-state, c'est le rôle des connexions OE et CE. 'OE' est l'abréviation de 'output enable' qui signifie à peu près 'sortie autorisée' et 'CE' est l'abréviation de 'chip enable' signifiant 'composant autorisé' (on trouve parfois 'CS' pour 'chip select'). Ces deux connexions sont 'low-active', ce qui signifie qu'il faut un signal négatif pour les activer, si bien que l'Eprom n'est reliée au bus que si CE et OE sont sur 'low'.

En examinant le plan d'un ST, on s'aperçoit que les connexions pour 'output enable' sont regroupées par trois: sur U2, U3, U4 (première Eprom) et sur U5, U6, U7 (deuxième Eprom).

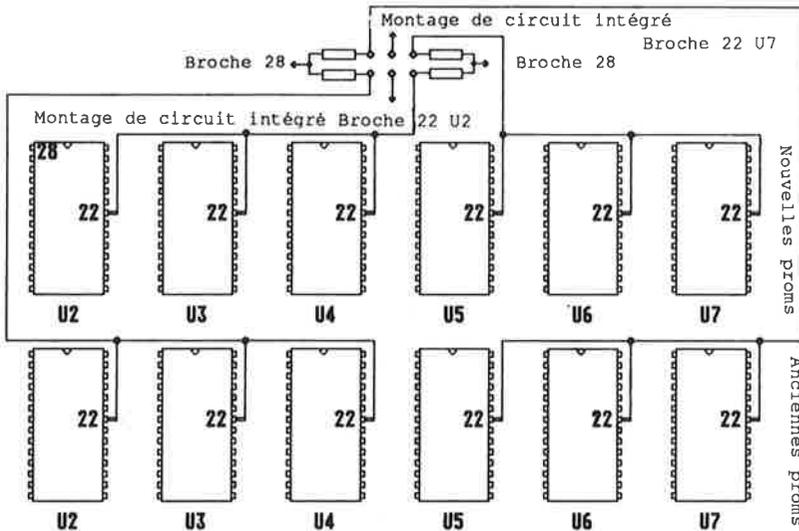
Nous allons pouvoir modifier cela en nous servant des connexions OE, où il nous suffit de passer sur une connexion à deux pôles et d'avoir 4 résistances de chacune 2,7 kilo-ohm. Nous devons aussi nous procurer un jeu d'anciennes et de nouvelles TOS-ROMs soit en PROM soit en EPROM, mais ces dernières ne pourront être d'une vitesse inférieure à 200 ns (plus elles seront rapides, mieux elles conviendront).

Commencez par démonter votre ordinateur comme nous l'avons expliqué au début du paragraphe précédent. Nous devons cette fois sortir complètement la platine pour souder sur le dessous.

Sur chaque Prom (ou Eprom) nous allons plier la patte 22 de telle sorte qu'elle soit à l'horizontal (très prudemment pour ne pas la casser). Sur la broche 22 des composants numérotés U2 et U7, nous allons souder un fil suffisamment long. Nous ré-enfichons les anciennes Prom et relient ensemble d'un côté les pattes pliées à l'horizontale de U2, U3 et U4 et de l'autre côté les pattes 22 de U5, U6 et U7. Sur U2 et U7, nous soudons encore un fil pas trop court. Nous enfichons les nouvelles Proms sur les anciennes et soudons toutes les pattes deux à deux sauf la broche 22.



Attention: placez bien l'une au-dessus de l'autre les Proms adéquates, car il vous sera très difficile ensuite de redécoller deux Proms soudés l'une sur l'autre. Faites de même avec la broche 22. Il ne nous manque plus qu'un fil partant de la broche 28 d'une Prom et véhiculant du +5V. Voici le schéma de câblage:



Il nous manque encore les raccords sur le commutateur et les résistances pull-up, que nous pouvons souder directement sur le commutateur. Reportez-vous pour cela au schéma ci-dessus. Il ne vous reste plus qu'à trouver une place suffisante pour votre commutateur, et à percer un trou pour le fixer. Tout ceci étant fait, remontez votre unité centrale.

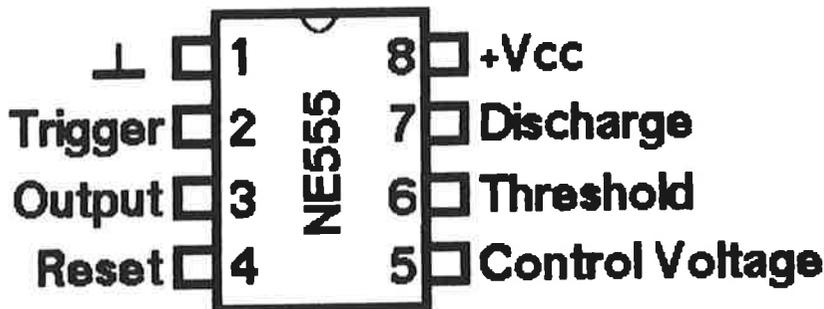
Encore quelques conseils: vous ne pouvez réaliser ce changement de système d'exploitation qu'après avoir complètement débranché l'unité centrale. Les derniers modèles sortis n'ont que deux Proms à la place des 6 Prom décrites ci-dessus. Comme on ne peut acheter l'ancien TOS que sous la forme de 6 Proms, une réadaptation paraît en ce cas quasiment impossible. Nos lecteurs capables de construire par eux-mêmes une platine n'auront aucun mal à fabriquer une adaptation de niveau professionnel, il suffit de s'en tenir à des conducteurs courts.

Pour ceux enfin qui n'osent pas intervenir dans l'ordinateur (ces interventions entraînent tout un travail de soudure), signalons qu'on trouve souvent dans les magazines informatiques des publicités pour des platines d'adaptation.

8.3 Régler la vitesse du processeur

Nous avons présenté dans ce livre un procédé permettant de ralentir, au niveau du software, de nombreux programmes à l'aide d'un interrupt. Ceci ne fonctionne cependant qu'avec des programmes ne se chargeant pas automatiquement. Nous pouvons éliminer cette limitation par une petite modification au niveau du hardware.

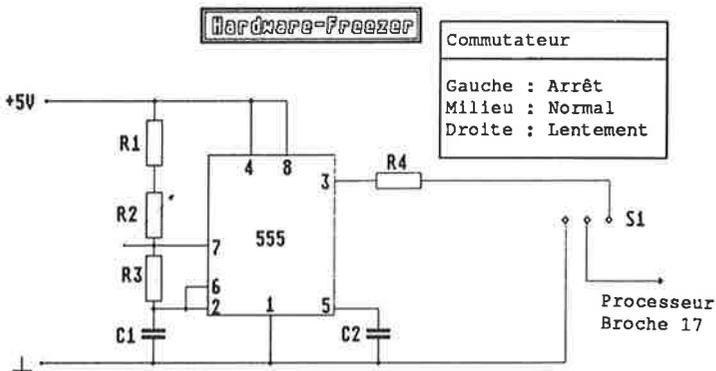
Le cœur de la connexion se fait ici par un composant timer NE555, que nous connaissons déjà dans sa version NE556. Ce composant peut devenir à peu de frais un véritable petit timer.



Commençons encore par étudier le principe de la connexion. Dans le paragraphe consacré à l'instauration d'un délai d'attente lors de l'allumage, nous vous avons déjà parlé un peu du connecteur de marche/arrêt du processeur, qui doit, lors de l'allumage ou d'un reset, se trouver sur 'low'. En exploitation normale cependant, on peut placer ce connecteur marche/arrêt sur 'low' sans passer par un reset. Après avoir terminé le traitement de la dernière commande, le processeur passe alors en état Tri-state et attend, pour reprendre le traitement de la commande suivante, que le connecteur marche/arrêt reprenne la position 'high'. Dans la pratique, ceci s'utilise par exemple avec le DMA, la transmission directe des données dans la mémoire vive sans passer par le processeur.

Notre amélioration consiste donc à placer cette ligne de temps en temps en position 'low'. Qui plus est, le temps d'attente sera réglable, ce qui nous permettra d'influer sur la vitesse de traitement d'un programme. Enfin, on devrait pouvoir stopper l'ordinateur (Freezer).

Voici un schéma de la connexion que nous voulons ajouter:



- S1: 1*UM avec position intermédiaire R4: 330 Ohm
 R1: 47 Ohm C1: 15 nF
 R2: Poti 270 Ohm C2: 15 nF
 R3: 47 kOhm

Voici la formule permettant de calculer la fréquence de travail du timer:

$$f = 1,44 / ((R1+R2+2*R3) * C1)$$

Après avoir monté ce commutateur dans l'ordinateur, et refermé celui-ci, placez le commutateur en position moyenne lors de l'allumage. Le freezer est alors désactivé. C'est après l'apparition du bureau GEM que vous pourrez modifier la

position du commutateur. Selon la position choisie, vous ne pourrez peut-être plus déplacer le curseur de la souris (position freezer). Dans la position opposée, le potentiomètre vous permet de régler la vitesse de calcul de l'ordinateur. Les effets sont particulièrement visibles lorsqu'on fait tourner le programme d'écriture cursive. Attention: ne sélectionnez cette position que lorsque vous vous trouvez dans un programme, faute de quoi vous pourriez perturber la synchronisation avec le processeur du clavier.

Chapitre 9

Annexe

9.1. Tableau des codes ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		000		000	eeE	PPP	'`	ppp	ccC	EEÉ	ééé	ššš	ııııj	ooO	ααα	εεε
1	100	111	!!!	111	AAÁ	QQQ	aaa	ooO	uuU	zzZ	ııı	ööÖ	ııııı	uuU	BBB	±±±
2	100	222	"""	222	BBB	RRR	bbb	rrr	eee	eff	óóó	ooO	KKK	ggG	rrr	±±±
3	100	333	##	333	CCÇ	SSŞ	ccc	sss	ááá	ööÖ	uuU	BBB	ııııı	ggG	KKK	≤≤≤
4	100	444	\$\$	444	DDD	TTT	ddd	ttt	ššš	ööÖ	KKK	EEÉ	ııııı	kkk	ΣΣΣ	ııııı
5	100	555	%%	555	EEÉ	UUU	eee	uuU	ááá	ööÖ	KKK	EEÉ	ııııı	ııııı	ooO	ııııı
6	100	666	&&	666	FFF	UUU	fff	uuU	ááá	ööÖ	EEÉ	AAÁ	KKK	BBB	UUU	+ + +
7	100	777	'`	777	GGG	HHH	ggg	hhh	ccC	uuU	EEÉ	AAÁ	ııııı	KKK	ııııı	±±±
8	100	888	((888	HHH	XXX	hhh	xxx	eeE	uuU	ééé	ööÖ	ııııı	ııııı	ööÖ	ııııı
9	100	999))	999	ııııı	ııııı	ııııı	ııııı	eeE	ööÖ	ııııı	ııııı	ııııı	ııııı	ooO	ııııı
A	100	100	* * *	ııııı	ııııı	ııııı	ııııı	ııııı	eeE	uuU	ııııı	ııııı	ııııı	ııııı	ııııı	ııııı
B	100	100	? ? ?	ııııı												
C	100	100	<<<	ııııı												
D	100	100	--	ııııı												
E	100	100	>>>	ııııı												
F	100	100	/ / /	ııııı												

9.2. Tableau des codes SCAN du clavier

⁵⁴ 3B	⁵⁵ 3C	⁵⁶ 3D	⁵⁷ 3E	⁵⁸ 3F	⁵⁹ 40	^{5A} 41	^{5B} 42	^{5C} 43	^{5D} 44
---------------------	---------------------	---------------------	---------------------	---------------------	---------------------	---------------------	---------------------	---------------------	---------------------

01	⁷⁹ 02	^{7A} 03	^{7B} 04	^{7C} 05	^{7D} 06	^{7E} 07	^{7F} 08	⁸⁰ 09	⁸¹ 0A	⁸² 0B	⁸³ 0C	0D	29	0E	62	61	63	64	65	66	
0F	10	11	12	13	14	15	16	17	18	19	1A	1B		53	52	48	47	67	68	69	4A
	^{1D} 1E	1F	20	21	22	23	24	25	26	27	28		1C	2B	⁷³ 48	⁷⁴ 50	4D	6A	6B	6C	4E
^{2A} 60	2C	2D	2E	2F	30	31	32	33	34	35	³⁶							6D	6E	6F	72
³⁸	39										^{3A}								78	71	

Index

!

.PRG	1-11
1st Word Plus	3-53

A

Accessoire	2-36, 6-186
Antibomb.s	5-138
ASCII	3-59
Assembleur	4-70
Assign.sys	6-144
Atteinte à un privilège	5-137
Attribut d'un fichier	2-28
Auto	1-11
Autodate.bas	1-12
Autogem.s	1-23

B

Bombes	5-136
BPB	2-36
Buffer DTA	2-48
Bureau GEM	1-22

C

Calendrier	1-12
CHERCHE.C*/	6-159
CHIFFRES.S	7-240
Clavier.s	4-79

CLIGNOTE.S 7-301
Code Scan 7-192
Commande illégale 5-137
Commentaires 4-67
Comp.bas 4-73
Conv1asc.bas 3-59
Conv_A_A.bas 3-63
Convasc1.bas 3-61
Conversion 3-53
Copie d'écran 5-87
Copyram.c 2-49
Critical error handler 5-124

D

Data 4-70
Data.bas 4-71
Date 1-16
DEFILE.S 7-297
DEMOACC.C 6-187
Demofont.c 6-145
DEMOVBL.S 7-205
Desktop 6-148
Desktop.c 6-150
Desktop.inf 1-17
Dispatcher 5-117
Disque RAM 2-43
Division par zéro 5-137
Dossier 2-35

E

Ecran 5-113
Encodage des fichiers 2-29
Encoder.s 2-30
Erreur d'adresse 5-137
Erreur de bus 5-137
Evtnt_multi 4-78

F

FADE.BAS	7-306
Fenêtre	6-172
FENETRE.C	6-180
Fichier-ressource	4-76
Format BCD	7-222
Freezer	5-109
Freezer.s	5-111
Fréquence	5-140

G

Garantie	8-321
GDOS	1-19
Gdosmake.bas	1-19
Getindex.c	4-77
GRAPH.C	7-293

H

Hardware	8-321
HBL	7-210
HD_SHIP.BAS	7-230
Hide.s	2-28
Horloge	7-213
HORLOGE.S	7-222

I

Images	7-280
IMG.C	7-290
Impression	5-121
Imprimante	1-17
INPUT.S	7-237

J

Joystick5-104

K

Keyboard.lst4-75

L

Liste.lst2-43

LOADED.C7-227

LOADER.C7-227

M

Masque5-101

Mémoire-tampon7-195

Messages d'erreur5-136

MFP 689015-117

Monitor7-253

Mousedit.lst5-98

Mousenew.lst5-103

Movem7-224

Multi-tâches5-116

Multitask.s5-117

N

NOMBRES.S7-244

P

Pointeur5-101

Pointeur de la souris5-89

POURCENT.S	7-218
Print.s	5-124
Processeur du clavier	5-92
Programmation graphique	7-279
Prop.s	5-95

R

RAM-Vidéo	7-219
RANDOM.S	7-251
Readbin.lst	3-53
REM-Killer	4-67
Remkill.bas	4-68
Rename.c	2-37
Reset	5-109
Retour de ligne (HBL)	7-208
Retour-image(VBL)	7-202
ROULETTE.LST	7-257
Routine résidente	4-78

S

SAVEIMG.C	7-280
Saver.prg	5-113
SCROLL_H.S	7-308
SCROLL_M.S	7-309
Slow.s	5-93
Soft-scrolling	7-307
Sound	7-271
Speed.s	5-92
Speeder.s	2-34
Sprite	7-311
SPRITE.BAS	7-311
STRINGS.S	7-248
Synchronisation interne	5-116
Système d'exploitation	5-121

T

Time	1-16
TIME.S	7-253
TIMEDEMO.BAS	7-255
Timer A	5-120
TOS	1-11
TOSLDD.C	7-228
Touches	4-74
Touches spéciales	7-190
Touches.par	4-75
Trap	7-216

V

Variables locales	7-231
Vecteurs	5-139
Vsync	7-297

Achévé d'imprimer
sur les presses de l'imprimerie IBP
à Rungis (Val-de-Marne 94) (1) 46.86.73.54
Dépôt légal - Avril 1989
N° d'impression: 5106

PAULY/SCHEPERS/SCHULZ

TRUCS ET ASTUCES

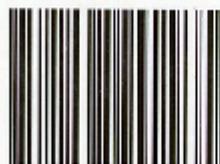
Tout ce que vous ne pensiez pas pouvoir faire avec votre ATARI ST, TRUCS ET ASTUCES va vous le permettre !

Fabriquer ses accessoires, lancer un programme résident, convertir des images, éviter les 'bombes' ou modifier la fréquence du processeur... Voici une véritable mine d'astuces, dont nombreuses sont inédites, présentées sous forme de programmes en code compilé, langage C, Assembleur et GFA Basic.

Si vous n'êtes pas un adepte des discours théoriques, vous pourrez immédiatement exploiter ou modifier tous les programmes du livre et de la disquette, et par la suite approfondir vos connaissances techniques dans les domaines les plus variés : hardware, software, langages, périphériques, entrées/sorties...

Principaux sujets traités :

- Installation du GDOS et de ses accessoires.
- Dissimuler, reformater, coder des fichiers...
- Accélérer la gestion de vos disquettes : le floppy-speeder.
- Affecter les fonctions au clavier, créer des raccourcis clavier.
- Recherche rapide et paramétrable des fichiers sur disque.
- Conversion et insertion des images graphiques aux formats BitMap, Degas...
- Des routines graphiques hyper-rapides.
- Créer et installer vos propres accessoires de bureau.
- Comment renommer un dossier.
- Reset, extinction automatique d'écran par une simple touche.
- Réaliser des hardcopies d'écran, des scrolling doux...
- Programmer en multi-tâche, détourner les interruptions du 68000...



9 782868 991935

Réf. : ML 651. Prix : 299 F
246/ISBN : 2 86899 193 9/ISSN : 0980-1928

EDITIONS MICRO APPLICATION
58, RUE DU FAUBOURG-POISSONNIÈRE
75010 PARIS. TÉL. : (1) 47 70 32 44