

ASSEMBLEUR ASM

1. L'EDITEUR

- 1.1. Introduction
- 1.2. Insertion dans la source
- 1.3. listing de la source
- 1.4 Remplacement
- 1.5. Effacement du bloc
- 1.6. Printing de la source
- 1.7. Déplacement d'un bloc (Move)
- 1.8. Recopie d'un bloc (Repeat)
- 1.9. Recherche et change

2. LES UTILITAIRES

- 2.1. Commandes fichier.
- 2.2. Divers

3. ASSEMBLEUR

- Syntaxe
- Expressions numériques
- Pseudo opérateurs
- Assemblage.
- Optimisation

4. DEBUGGER

- Entrée.
- Indications données par debug :
- Commandes (Valables seulement en mode debug)
- Exécution de programmes objets

5. DESASSEMBLEUR.

- Etape 1. chargement du programme à désassembler.
- Etape 2. Programmation du désassembleur

APPENDICE A. CLAVIER

- Table 1. Mouvements.
- Table 2. Touches de fonction
- Table 3. Codes ASCII
- Table 3 bis. Codes ASCII
- Table 4. Control+Lettre. (mot clef)

APPENDICE B. EDITION DE LA SOURCE

- Table 5. Définitions de base.
- Table 6. Définition du bloc

APPENDICE C

- Table 7. Commandes globales du désassembleur.
- Table 8. Noms des labels
- Table 9. Zones de désassemblages.
- Table 10. Programme des zones A
- Table 11. Messages d'erreur

APPENDICE D SOLUTION DES EXERCICES.

6. NOUVEAUTÉS TT(68030)

ASSEMBLEUR ASM

L'assembleur ASM contient à lui tout seul:

Un éditeur de symboles ASCII très perfectionné.

Un ensemble d'utilitaires.

Un assembleur entièrement compatible avec l'assembleur SEKA mais de 30 à 50% plus rapide avec possibilité d'optimisation. Cet assembleur produit un code immédiatement exécutable.

Un débogueur très sophistiqué rendant la programmation en assembleur aussi facile qu'en BASIC.

Un désassembleur programmable créant un fichier source avec labels à partir du code objet. Ce désassembleur est aussi rapide que l'assembleur.

(Z) Les paragraphes marqués (Z)-(ZF), celui-ci mis à part, (ZF) doivent être sautés en première lecture.

1. L'EDITEUR

1.1. Introduction

Les commandes de type éditeur permettent de créer et de modifier la source (ou code source). ASM peut être dans divers modes. Les modes principaux sont les modes ASMA et Source à partir desquels s'effectuent les commandes. Le mode dans lequel se trouve ASM est indiqué sur la première ligne de l'écran, la ligne de tête, qui est réservée au système. Elle indique aussi le nombre de lignes modifiées depuis la dernière sauvegarde, et le nombre de

lignes de la source.

Appuyez sur F10 : ASM passe en mode ASMA. Appuyez sur sF10 (ie sur Shift et F10 que nous noterons aussi sF10): ASM passe en mode source. Tapez maintenant :

Bonjour [CR]

où [CR] désigne la touche Return ou Enter. Après l'appui sur CR, la ligne se trouve dans la source. Vérifiez le en tapant F1.

Revenez maintenant en mode ASMA (Tapez F10) et retapez la même ligne. Vous obtenez un message d'erreur.

Tapez maintenant :

= [CR]

C'est une commande. Les commandes commencent par un caractère autre qu'une lettre A-Z (ou a-z, ASMA ne faisant pas la distinction entre minuscules et majuscules dans presque toutes ses commandes). Elles sont validées par appui sur CR (que nous omettrons désormais).

Après la commande "=", ASM écrit des renseignements sur l'utilisation de la mémoire. Passez de nouveau en mode source (sF10) et retapez la même commande. Vous obtenez la même réponse. Vérifiez (F1) que la source n'a pas été modifiée.

Ainsi donc, les commandes de l'éditeur peuvent être effectuées en mode source ou ASMA. Seules des lignes commençant par A-Z, ";" ou ":" peuvent être insérées dans la source en mode source.

Passage mode source vers ASMA :

F10

* Les * en début de ligne sont ignorés et font passer en mode ASMA. "*****=" est donc

équivalent à "=".

Passage mode ASMA vers source :

sF10

-ligne Indique qu'on désire insérer des nouvelles
lignes après la ligne.

Passage mode quelconque vers ASMA :

Esc Cette touche interrompt toutes les commandes
sauf lorsque le disque tourne.

L'éditeur pleine page.

A l'aide des touches mouvements (Table 1 de l'appendice A), on peut se déplacer et écrire facilement sur tout l'écran sauf la ligne de tête. L'écran est formé de 24 lignes écran et d'un nombre plus réduit de lignes texte.

ligne texte

Après clr ou s+up les lignes texte sont les lignes écran. Lorsqu'on écrit en passant à la ligne suivante, on crée une ligne texte formée de 2 lignes écrans. Il faut vraiment écrire, passer avec le curseur ne suffit pas. On peut aussi utiliser s+down pour rallonger la ligne texte du curseur.

nota 1) En appuyant sur s+left et s+right on déplace le
curseur aux extrémités de la ligne texte.

2) La ligne texte peut occuper tout l'écran.

3)Noter que Insert, Delete, Backspace agissent sur la ligne texte du curseur.

4)Après appui sur CR, c'est la ligne texte qui est validée. Si cette ligne était la dernière de la page, la première ligne texte de la page disparaît laissant de la place pour une nouvelle ligne.

mots clefs

L'appui sur Control+lettre de A à Z écrit un mot usuel,si on se trouve en extérieur de guillemets (voir Table 4 de l'appendice A).

1.2.Insertion dans la source

L'insertion se fait devant le pointeur d'insertion (M),en mode source.Il faut donc positionner ce pointeur et passer en mode source.

positionnement graphique du pointeur d'insertion M

Si on valide la ligne:

123>_....

ou 123._....

ou 124>^....

ou 124.^....

(les indiquent des caractères quelconques)

le pointeur M se positionne de sorte que l'insertion se fait après la ligne 123 et avant la ligne 124.

(Z) Après passage en mode source par sF10, il est conseillé de relister la page par F9. En effet si on remonte le curseur sur une ligne comme

```
10.EXG A0,A1
```

et qu'on appuie sur CR croyant l'insérer, en réalité on modifie la ligne 10 et on place le pointeur M sur la ligne 11. Après F9 la ligne est réécrite sous la forme

```
10>EXG A0,A1
```

(ZF) et peut être insérée.

-ligne

Cette commande permet à la fois de positionner le pointeur M et de passer en mode source. Exemples :

- Insertion en tête de la source
- D Insertion en fin de la source
- 12 Insertion après la ligne 12
- gamma Insertion après la ligne commençant par "gamma:"

(Z) Pour la syntaxe complète de cette commande voir Table 5 de l'appendice B.

nota Ne pas écrire "-12," par erreur (voir Effacement)

(ZF) qui aurait un effet désastreux.

Exemple

Après "-" le programme écrit

1>

Tapez

debut: ctrl+U D0-D7/A0-A7 ctrl+P [CR]

ctrl+M ctrl+X D0,D2 [CR]

Remontez sur la ligne précédente, changez D2 en A3

et tapez sur [CR].Vérifiez la source obtenue (F1).

Tapez

-debut

add #5,D2

Vérifiez la nouvelle source.

Remarques

Les modifications que vous faites sur des lignes précédentes ne sont validées qu'après CR.Si ces modifications ne vous conviennent plus, vous pouvez retrouver l'ancien écran par F9.

L'éditeur insère :

si la ligne texte commence par "nnnn>" toute la ligne
texte après "nnnn>"

sinon ,et si le premier caractère est A-Z ";" ou ":",
toute la ligne texte.

Les blancs avant le premier caractère, avant le premier
";" et ":" non entre guillemets ne sont pas insérés.

On peut insérer une ligne vide.

(Z) Après l'insertion les pointeurs M,A,B,D sont modifiés de
sorte qu'ils continuent à pointer les mêmes lignes que
(ZF) précédemment même si leur numéro a changé.

1.3.listing de la source

touches de fonction.

Voir la table 2 de l'appendice

#ligne

liste à partir de la ligne donnée.

nota:1)Le listing est de la forme n. (en mode ASMA) et de la forme n> en mode source.

2)Le listing est formaté:les espaces devant les débuts de ligne et ";" n'existent pas dans la source en mémoire

(Z) 3)#bloc (Voir table 6 la syntaxe de bloc)

(ZF) liste et définit le bloc de lignes.

1.4 Remplacement

n.instr Remplace la ligne n par "instr",et déplace le pointeur d'insertion pour insertion après la ligne n.

n. Efface la ligne n

Pour corriger une ligne de la source, par exemple après détection d'une erreur, si cette ligne est listée sous la forme n.etc

il suffit de corriger et d'appuyer sur CR pour valider la correction.

(Z) Ces commandes peuvent être effectuées en mode source.Ainsi

108>7.

efface la ligne 7.

Après effacement les numéros des lignes changent. En particulier les lignes listées après une ligne qui vient d'être effacée ont des numéros faux. Il est alors dangereux de conserver un tel écran, et la prudence conseille d'appuyer sur F9. On remarquera d'ailleurs qu'après effacement, le curseur remonte, pour rester sur une ligne avec son vrai (ZF) numéro .

1.5.Effacement du bloc

On désigne par bloc un ensemble de lignes consécutives de la source. Les pointeurs A et B repèrent la première et dernière ligne du bloc respectivement.

-bloc

où bloc n'est pas de la forme "ligne". (Table 6)

Cela efface irréversiblement le bloc.

nota Il est plus prudent de procéder comme suit:

1) Définir le bloc.

On peut définir le bloc graphiquement par exemple

12857.`etc

13589.`etc

ou par

#12857,13589 (avec listing)

ou encore par

alt+B12857,13589

2) Vérifier que c'est bien le bloc voulu. (F8)

3) Effacer par :

- alt+B

Après cette instruction on se trouve en mode source, pour insertion à l'endroit où se trouvait le bloc.

Exercice 1.5

Comment vider la source ?

1.6. Printing de la source

alt+P [bloc]

Imprime le bloc (par défaut toute la source)

L'instruction

PAGE titre

insérée dans la source permet de mettre un titre en haut de chaque page du listing.

1.7. Déplacement d'un bloc (Move)

alt+M [bloc] [:ligne]

Déplace le bloc devant la ligne.

Si le bloc n'est pas donné, il s'agit du dernier bloc défini.

Si la ligne n'est pas donnée, le bloc est déplacé devant le pointeur d'insertion.

Après le déplacement, le pointeur d'insertion est après le bloc, et les pointeurs A et B sont sur la nouvelle position du bloc.

Exemple

-10		Met les lignes 100, 107 à 113
alt+M100		après la ligne 10
alt+M107,3		

1.8.Recopie d'un bloc (Repeat)

alt+R [bloc] [;ligne]

Comme alt+M, mais l'ancien bloc existe toujours. Après la copie, les pointeurs A et B restent sur l'ancien bloc, et le pointeur d'insertion se trouve après le nouveau bloc.

1.9.Recherche et change

On dispose de 2 commandes identiques de recherche alt+S et alt+T mais définissant 2 chaînes différentes S et T, et d'une commande de changement alt+C qui change la chaîne S en T.

Ces commandes sont très perfectionnées : on peut rechercher/changer des chaînes formées de caractères définis séparés par des caractères non précisés et qui peuvent s'étendre sur plusieurs lignes.

Recherche de S

alt+S [chaîne] [alt+B bloc] [rep] [alt+H]

Les 3 derniers arguments peuvent être donnés dans un ordre quelconque. Ils définissent les options suivantes :

bloc Dans lequel s'effectue la recherche. Par défaut toute la source.

rep C'est l'un des caractères alt+U (arrêt après la première occurrence) ou alt+R (répète la recherche, qui est la valeur par défaut)

alt+H En cas de répétition, la recherche ne s'arrête pas toutes les 16 lignes. (Hold)

La chaîne est par défaut la dernière chaîne S définie. Elle est formée d'au plus 48 caractères. Les caractères suivants ont un sens particulier:

alt+S Désigne un caractère autre que 0-9, A-Z et a-z (séparateur). Par exemple

alt+S alt+S M O alt+S

trouvera TST MO mais pas MOVE D0,D1

alt+Z Désigne la séparation entre 2 lignes de la source (zero : en effet les lignes de la source sont séparées par des octets nuls)

alt+F Désigne n'importe quelle chaîne, éventuellement vide, incluse dans une ligne source donnée. alt+F doit être précédé et suivi d'un caractère différent de alt+F.
Exemple: alt+S W alt+F alt+Z alt+F W qui cherche 2 lignes consécutives contenant "W" toutes les deux. (filler)

Si la chaîne est donnée, elle est mémorisée dans S et pourra être recherchée de nouveau sans être réécrite.

Recherche de T

C'est la même commande que alt+S, mis à part qu'elle agit sur la chaîne T. Le caractère de commande est alt+T au lieu de alt+S.

Change S en T

alt+C [chg] [alt+B bloc] [rep] [alt+H]

Les 3 derniers arguments ont le même sens que pour les commandes de recherche. L'argument chg peut avoir les formes suivantes:

(omis) Change la chaîne S en T. Il s'agit des chaînes en mémoire

alt+I (Inverse) On permute d'abord les chaînes S et T en mémoire avant d'effectuer le changement.

[chaîneS][alt+T chaîneT] Cela permet de définir la chaîne S et ou T, si une est omise on prend l'ancienne.

(Z) La chaîne S (resp T) définit les zones 1,2,... (resp 1',2',...)

alt+S 1111111 alt+S ... alt+F 222222 alt+F 333333 alt+S ...

Ces zones ne contiennent ni alt+S ni alt+F mais peuvent contenir alt+Z. Dans chaque région séparée par les alt+F il y a une seule zone qui commence au premier caractère ou au deuxième si le premier est alt+S.

La commande exige que le nombre de zones de S et T soit identiques. Elle effectue alors le remplacement de la zone 1

(ZF) par la zone 1', de la zone 2 par la zone 2'...

Exemple: alt+C : alt+T : alt+Z alt+B 2

coupe la ligne 2 en deux après :

Notez que dans la donnée des chaînes S et T les espaces sont significatifs, y compris au début de S ou T. Il est également

possible de mettre des espaces significatifs en fin de ces chaînes, ils vont jusqu'au premier argument donné.(par exemple alt+R)
Notez également que minuscules et majuscules diffèrent dans les chaînes S et T.

Exercice 1.9.1

Comment peut on remplacer le label "A" par le label "apluslong"? La difficulté vient de ce que beaucoup de A ne doivent pas être changés.(L'utilisation d'un tel label est très malencontreuse).

Exercice 1.9.2

Comment créer un fichier source contenant les lignes:

987

654

321

On rappelle qu'il n'est pas possible d'insérer des lignes commençant par autre chose que A-Z,a-z, ";" et ":"

(Z) Exercice 1.9.3

Comment peut-on raccourcir(compactifier) le code source ?

(ZF)

2. LES UTILITAIRES

2.1.Commandes fichier.

alt+Vpath (View) Contenu du disque.

Cela définit l'environnement "path" (Exemples A:\chess\ ou A:\). Ecrit la taille disponible, puis pour l'environnement, la liste des fichiers (y compris les fichiers cachés) et leur taille. Le type des fichiers est indiqué par un symbole :

normal	#	nom
!	lecture seul	% sous répertoire
"	système	& écrit et refermé

alt+KF (Kill File) Efface un fichier. Donner le nom complet avec l'extension.

alt+A (Append) Rajoute à la fin de la source un fichier disque

alt+W (Write) Ecrit la source sur disque.

(Z) Si on veut un fichier ASCII où les lignes sont séparées par chr(13) chr(10), il faut utiliser l'extension .S .Sinon le fichier créé est un fichier ASCII les lignes étant séparées par chr(0). Par défaut l'extension est .Z

Les fichiers lus par alt+A peuvent être de ces deux types, sans que l'extension suive nécessairement les règles d'écriture ci dessus. ASM se charge de la conversion au type avec séparateur chr(0), qui est la représentation interne utilisée.

alt+AI (Append Image) Lit un fichier dans la mémoire

Pour lire le fichier en entier donner -1 pour la fin.

alt+WI Ecrit une zone mémoire sur disque.

Par défaut, l'extension est .IMG

alt+AO Lit un fichier objet. Extension .PRG par défaut.

Cette commande permet de lire un programme (.TOS ou .PRG) en vue de désassemblage ou d'exécution.

Un tel fichier commence par un en-tête de 28 octets:

0 .W \$601A

2 .L c=longueur du code

6 .L d=longueur des données

\$A .L a=longueur additionnelle

\$E .L s=longueur des symboles

\$12 14 octets inutilisés

Suivent c+d+s octets, puis une table de relocation.

La table de relocation permet de retrouver les adresses des mots longs dépendant de l'implantation en mémoire. La table commence par un mot long qui donne la distance entre la première adresse et le début du programme. Si l'adresse suivante est $254*q+r$ octets plus loin ($0 < r < 255$), la table contient q octets 1 suivis de r. Fin indiquée par un octet nul.

La commande alt+AO demande une adresse de chargement ([CR] met au dessus de la source) puis charge :

l'en-tête dans rel-\$1C à rel

les c+d+s octets dans objet

la table de relocation dans rel.

Ensuite l'objet est modifié suivant la table de relocation pour le rendre exécutable sur place. Un programme de désassemblage (utile pour la première étude du programme) est mis à la fin de la source.

La page de base de ASM est modifiée de sorte qu'il est maintenant possible par la commande alt+GS de lancer et suivre le programme à l'aide du débogueur. Les quelques instructions en S avant le programme servent à simuler le TOS. Nota: Il faut utiliser ASM.TOS ou ASM.PRG avec la même extension que le programme.

alt+WP Ecrit un fichier objet.

Le fichier correspond à la partie du programme assemblé après une adresse paire donnée.

Cette adresse est d'abord demandée.

Cela permet de disposer de programmes utilitaires, placé au début de la source, et qui ne seront pas transformés en objet.

alt+WO Ecrit un fichier objet. Le fichier correspond au programme assemblé. On peut aussi écrire les fichiers (ZF) lus par alt+AO.

(Z)(Z)

alt+AL Lit un fichier link.

alt+WL Ecrit un fichier link.

Nous déconseillons l'utilisations des fichiers link,
qui sont là seulement pour la compatibilité avec
l'assembleur SEKA.

(ZF)(ZF)

2.2.Divers

! Sortie de ASM.(Bye,Quit)

= Carte mémoire.

? ad Ecrit la valeur d'une expression numérique.Exemple:

? QA+[10 * [7+\$25]]/3

où QA est label qui prend sa valeur dans le code
objet. La valeur est aussi donnée en fonction des
labels qui l'encadrent. On peut utiliser des nombres
hexadécimaux (\$64), décimaux (100), octaux (@144),
binaires (%1100100) et des constantes ASCII ("d").
Les opérations + - * / &(and) !(or) ~(eor) > = < sont
effectuées dans l'ordre de lecture sauf s'il y a des
crochets [].

alt+N.s ad Modifie la mémoire en ad avec la taille s(L,W ou B).

alt+O (Old) Essai de récupérer la source après -,
Peut également permettre de retrouver une source
éditable après une destruction.

alt+Q.s ad (Query) Examine la mémoire en ad

Si ad est omis on utilise la dernière valeur+128.

(Z)

alt+Z Recherche de suites d'octets dans la mémoire.

Exemple: mettre à l'adresse P le programme de recherche en assemblant

P:D.B 8,"ATARI ST",0

Puis appeler la commande, donner P comme adresse, [CR] pour début et fin. Cela localise les chaînes de 8 octets "ATARI ST". Lorsqu'on appuie sur CR la valeur est 1. Si fin <= debut on change fin en \$F8000 (haut de la mémoire). On peut rechercher des choses plus compliquées ainsi

P:D.B \$85,"A-Za-z",0,0,\$81,"Tt",0,0,0

recherche 5 lettres suivies de "T" ou "t", et

P:D.B \$81,"--",0,"+",0,0,0

recherche l'octet "-", "+" ou 0. La longueur des suites est limitée à une vingtaine d'octets.

alt+X Copie le bloc debut... fin-1 en dest.

alt+F (Fill) Remplit debut...fin-1 avec le même octet.

alt+E Compare debut,...,fin-1 avec dest,dest+1,...

alt+H ad Désassemble quelques lignes en ad

alt+L ad (Limit) ASM restitue la mémoire au dessus de ad au TOS. Utile pour utiliser le GEM. En effet, ASM utilise sinon toute la mémoire.

alt+Y Rk Pour le débbugger, modifie le registre

D0,D1...A7,SP,USP,SR

alt+J ad (JMP) Saut au sous programme d'adresse ad. Le retour

à ASM se fait soit sur "ILLEGAL" (ou autre exception), soit sur "RTS" avec la même hauteur de pile qu'en entrée. Cette fonction diffère de la fonction J du débbugger seulement par le fait que Esc ne produit pas le retour à ASM.

(ZF)

(Z)(Z)

alt+KL Vide le link

alt+XL Copie objet dans le link

(ZF)(ZF)

3. ASSEMBLEUR

Syntaxe

Une instruction a la forme générale :

[label:] [instr] [;commentaire]

Pour les instructions 68000, nous conseillons le livre de C.Vieillefond "Mise en oeuvre du 68000" (Sybex 1984). Nous ne décrivons pas le jeu de mnémoniques du 68000, mais faisons seulement quelques remarques.

Dans MOVE D0,D1 un espace doit séparer MOVE et D0, par contre dans MOVE.B D0,D1 l'espace entre .B et D0 est inutile. On peut aussi écrire sans mettre d'espace entre opérateur et opérand:

MOVE#100,D1 TRAP#4 ...

Certains mnémoniques ont été simplifiés, ainsi ADDA et ADDI ont été remplacés par ADD. Noter cependant que CMPM n'a pas été simplifié.

Les instructions BTST#k,d(PC) et BTST Dk,d(PC,Xi) ... sont correctes bien que non référencées dans le livre de Vieillefond et non assemblées par SEKA.

Pour la notation des branches, voir l'optimisation.

Expressions numériques

Voir la commande "?". Noter que dans l'assemblage le pointeur de programme est représenté par *.

Pseudo opérateurs

Pour créer un programme avec table de relocation utiliser

CODE (peut être omis)

prgr1

DATA

T1:D.W 0,0,0,0 (ou DC.W)

CODE

prgr2

M1:D.B "ASM",0

EVEN ;opposé ODD

DATA

T2:BLK.B 100,0

END ;l'assembleur ignore ce qui suit

L'alternance CODE/DATA fait qu'après assemblage on aura l'ordre prgr1, prgr2, T1, T2.

Voici d'autres pseudo ops :

ORG | Utilisation déconseillée

LOAD |

EQU On peut écrire W:EQU \$70000
ou W=\$70000

MACRO Une MACRO ne peut pas appeler une autre MACRO. La MACRO

ENDM doit être définie avant l'appel. Exemple :

VIDE:MACRO

MOVEQ#1,D?2 ;?2 deuxième argument

MOVEQ#7,D0

I?0:CLR(?3)+ ;?0 varie d'un appel à l'autre.

DB?1 D0,I?0

ENDM

...

VIDE RA,7,A5

ILLEGAL

Nous examinons les différences avec l'assembleur SEKA

BLK.s n,p n n'est plus limité à 2^{16} .

REL k où k=-1,0 ou 1

Ce pseudo op permet de vérifier les mots et octets.

Ainsi pour SEKA

D.W alpha

D.B beta

est ok même si alpha(beta) n'est pas un mot(octet).

Le pseudo op REL k délimite des zones où k prend une

certaine valeur (par défaut k=0). Dans les zones k=0, alpha et beta peuvent être quelconques, comme dans SEKA. Dans les zones où k=-1 il faut que :

$$-2^{15} \leq \alpha < 2^{15} \quad \text{et} \quad -2^7 \leq \beta < 2^7.$$

Dans les zones où k=1 il faut que :

$$0 \leq \alpha < 2^{16} \quad \text{et} \quad 0 \leq \beta < 2^8.$$

Exemple:

```
REL -1      | Permet de recalculer sp1.L sans erreur
V:D.W sp1-* |
REL 0      |
...        |
LEA V,A0   |
ADD (A0),A0 |
```

AA Ces trois pseudos op donnent des instructions au debugger.

AAS l'objet lui même n'est pas modifié.

AAE AA définit un breakpoint devant l'instruction qui le suit.

"AA ad" définit un breakpoint en ad. Le nombre de ces breakpoints est limité à 10.

AAS et AAE définissent le début (Start) et la fin (End)

d'une zone pas à pas. Ainsi dans l'exécution pas à pas de:

MOVEQ#1,D0

MOVEQ#2,D2

AAS

EXG D0,D2

AAE

MOVEM D0/D0,-(SP);noter que MOVEM D0,-(SP) est illégal
le débarrer ne s'arrête que sur la ligne EXG D0,D2. S'il
n'y a pas de "AAS" ou "AAE", toute la mémoire est en pas à
pas. Sinon seulement les blocks limités par AAS...AAE. On
peut définir 10 zones de pas à pas. On peut également
utiliser "AAS ad1", "AAE ad2",... à condition que les
adresses ad1, ad2 ... forment une suite croissante.

DO.s n{n} Comme D.s mais adresse impaire possible.

Assemblage.

/ Fait partir l'assemblage.

Options: V listing écran

P ou E listing printer (diffère de alt+P car sort le code
assemblé)

H arrêt après chaque page.

L sort code link (n'est là que pour les SEKAistes).

Il est conseillé d'assembler sans donner d'options. En cas
d'erreur ASM écrit la ligne en erreur avec l'endroit où l'erreur
s'est produite, et la ligne assemblée (# si MACRO).

Optimisation

Donner l'option O ou B (branches seulement) dans
l'assemblage. Les optimisations effectuées sont écrites dans la
source.

1)Optimisation des branches.Dans les instructions Bcc et BSR on
peut préciser la longueur(Défaut .K)

BRA.S Branche courte

BRA.M Branche longue non optimisée

BRA.K |Branche longue optimisable

BRA.L | "

L'optimisation mettra une branche courte .S ,si c'est possible, sauf pour les branches .M ,dans ce dernier cas il écrit que la branche n'est pas optimale, sans la modifier.

Cas des macros : Les branches des macros ne sont pas modifiées, elle doivent obligatoirement être données sous la forme .S (Sortie erreur si longue) ou .M.

2)Optimisation du mode d'adressage Adr.L

Dans des instructions comme

```
MOVE LA,D0
```

LA peut être optimisé et transformé en LA(PC).

Pour cela on doit être en mode CODE et LA doit être une adresse(+ une valeur numérique éventuellement) dans un segment CODE ou DATA.

Pour interdire l'optimisation il faut écrire:

```
MOVE LA.L,D0
```

Si ce mode d'adressage n'est pas optimal,ASM écrit un message.

Nota:L'assembleur transforme les branches .S qui sont longues en branches .K et les adressages LA(PC) qui sont longs en LA même sans optimisation.

3)La transformation du mode d'adressage Adr.L en Adr.W n'est pas faite.Lorsque cette transformation est possible on obtient le diagnostic "not optimal".

4)Contraction de la source. Après optimisation la source a sensiblement augmenté de taille.On peut la contracter par alt+U (Voir l'exercice 1.9.3)

Exercice 3.1

Le programme suivant calcule la somme des mots dans la table TDEBUT :

```
LEA TDEBUT,A0
CLR D0
I:ADD(A0)+,D0
CMP.L#TFIN,A0
BLT I
ILLEGAL
TDEBUT:BLK.W 10
TFIN:
```

Le changement de LEA DEBUT,A0 en LEA DEBUT(PC),A0 fait gagner 4 cycles à l'exécution et 2 octets code. Cependant dans cet exemple, pourquoi est il déconseillé de faire ce changement et de se contenter d'optimiser sur les branches (B) ?

4. DEBUGGER

Le débbugger permet de contrôler l'exécution d'un programme. On peut partir d'un point quelconque, exécuter en pas à pas certaines portions, mettre des points d'arrêts, exécuter jusqu'à ce qu'un registre ou mémoire change ou prenne une valeur donnée (G), sortir d'un sous programme (D), exécuter un nombre de fois

prédéterminé une instruction (F et H). L'écran du programme n'est pas modifié de sorte que l'exécution par le débbuger produit en général les mêmes résultats que l'exécution libre. En appuyant sur la touche Esc, on revient en général à ASM, même lors d'une exécution du programme. Ainsi par exemple dans :

```
I10:MOVEQ#5,D0
```

```
I11:MULU #3,D1
```

```
DBRA D0,I10
```

```
I12:MOVE D1,PUIS
```

vous avez fait un boucle infinie par erreur. Si vous lancez le programme par alt+J I10, La seule échappatoire sera d'éteindre la bécane. Si par contre vous lancez ce programme par le débbuger, il vous suffira d'appuyer sur Esc pour revenir en mode ASMA. Ce retour est rendu possible par une déviation de l'interruption de niveau 4 (VBL) qui est générée 60 ou 70 fois par secondes. Pour que Esc soit actif vous devez ne pas masquer cette interruption, (par exemple OR#\$700,SR mis après la ligne I10 inhiberait Esc) ni supprimer l'exécution du programme vectorisé en \$70. Notez que Esc est masqué pendant les opérations disque. De plus cette fonction est paramétrée par l'octet en \$200. Si cet octet est nul, Esc est actif à chaque interruption. Si cet octet est positif Esc est actif une fois sur 256 (environ toutes les 5s), et s'il est négatif Esc est inhibé. Utiliser ce masquage est parfois nécessaire car le test de Esc vole les caractères entrés au clavier. Nota: Esc n'est possible qu'avec le TOS en ROM

Exemple Indiquons comment mettre au point le programme I10 donné plus haut. On part de l'adresse I10 par alt+GI10. Le débbugger se positionne sur la première instruction. Il est ensuite conseillé d'exécuter chaque instruction au moins une fois en mode pas à pas. Appuyez sur CR : la première instruction a été exécutée, le débbugger est positionné sur la 2ème ligne. Appuyez encore 2 fois sur CR, Le débbugger revient ligne I10, alors que vous auriez voulu aller ligne I11: Voila l'erreur. Une fois l'erreur corrigée, vous reprenez ligne I10, mais maintenant vous voulez aller jusqu'en I12 rapidement : appuyez sur 3 seulement (la ligne I12 est numérotée par 3) le débbugger exécute la boucle en vitesse réelle et se positionne en I12.

(Z) Principe du débbugger

Le débbugger utilise le mode trace du 68000 pour les commandes suivantes: CR D G. L'exécution du programme est alors très ralentie, car après chaque instruction on examine s'il faut continuer l'exécution ou revenir au débbugger. Le débbugger implante des instructions illégales dans les commandes : 0 à 9, B, C, F et H. Dans ces commandes le code est exécuté en vitesse réelle (Sauf l'instruction cible dans les commandes H et F). Ces commandes sont interdites dans la ROM (CR D et G y sont valables).

(ZF)

Entrée. (Passage du mode ASMA ou Source en mode débbug)

alt+G [AD] On part de l'adresse AD qui peut être donnée par une expression numérique avec des labels,après assemblage.Si AD est omis on part de l'ancienne adresse. Dans les deux

cas les registres ont leurs anciennes valeurs, qui ont pu être modifiées par alt+Y. Toutefois si AD est donné le registre SR est mis égal à \$2000, et si vous désirez partir en mode User il faut revenir à ASMA (A) modifier SR (alt+YSR) puis revenir au débbugger sans mettre AD.

Indications données par débbug :

Les valeurs des registres sont comparées aux valeurs lors du précédant passage dans debug. Si une valeur n'a pas été modifiée elle est suivie du signe =. PreC donne la valeur de PC lors du passage précédant dans débbug, sauf après la commande G où PreC est l'adresse de la dernière instruction exécutée. On trouve ensuite les 16 octets à partir d'une adresse que vous pouvez fixer par " alt+Q ad". ad est soit une adresse numérique soit un registre, par défaut ad est le registre SP.

Suivent 10 instructions désassemblées à partir de l'adresse AD et numérotées de 0 à 9. Ces instructions sont écrites en fonction des labels ou de façon absolue (voir commande L). La ligne 0 est l'instruction à exécuter en premier. Après alt+G aucune instruction du programme à débbugger n'a été exécutée. Le débbugger attend l'appui sur une touche suivant un menu affiché. Ce sont les commandes du débbugger.

Commandes (Valables seulement en mode débbug)

CR Exécute l'instruction de la ligne 0 et revient en mode debug sur l'instruction suivante. C'est le pas à pas. En appuyant seulement sur CR on peut ainsi suivre le

programme dans toutes ses instructions. Si les zones AAS à AAE ont été définies dans la source, le débogger ne s'arrête que dans ces zones.

I Lance le programme sans s'occuper de l'arrêter. On revient en mode ASMA lorsque le programme trouve l'instruction "ILLEGAL", que vous avez eu le soin de mettre à la fin de votre programme.

J Identique à la commande alt+J, mis à part que Esc est actif.

O Arrêt sur les breakpoints AA (définis dans la source).

1 à 9 Lance le programme et s'arrête sur la ligne de numéro donné ou sur un breakpoint AA.

A Arrêt. Retour en mode ASMA. Il est alors possible de modifier les registres par alt+Y, de modifier la mémoire par alt+N, d'examiner la mémoire par alt+Q. On peut lister la source et aussi la modifier. Il est conseillé de réassembler après une modification, mais tant que le message "objet détruit" n'est pas sorti, l'ancien objet est intact. On peut ensuite reprendre l'exécution du programme par alt+G.

B Breakpoint. Le débogger attend l'entrée d'une adresse validée par CR. Comme d'habitude, on peut donner cette adresse par une expression numérique. Il est encore

possible de revenir en mode ASMA en appuyant sur Esc ou en rentrant une adresse illégale. Après la donnée du Breakpoint, le programme s'exécute jusqu'à l'adresse donnée ou jusqu'à un breakpoint AA.

C Demande plusieurs breakpoints. On peut rentrer i de 1 à 9 ce qui met un breakpoint sur la ligne i affichée. Après le dernier breakpoint entrer CR pour valider la commande. On peut utiliser jusqu'à 15 breakpoints au total en comptant les breakpoints AA. Le programme s'exécute jusqu'au premier de ces breakpoints atteint.

D Exécute jusqu'à sortir de la subroutine en cours. Par exemple:

```
O:BSR B
A:MOVE D0,D5
.....
B:SWAP D0
  BSR C
  SWAP D0
  RTS
```

Après alt+G O le debugger va exécuter la ligne O. Appuyez sur CR, le debugger va ligne B. Appuyez sur D, le debugger va ligne A.

E Remet l'écran du programme, ses couleurs, sa résolution et son curseur. Attend ensuite une commande du debugger. Il est

conseillé d'utiliser K, si on veut continuer ou A si on veut s'arrêter.

F Demande nombre de fois et un breakpoint (on peut rentrer i de 1 à 9 pour la ligne i, et CR sans rien d'autre pour la ligne 0). Le débbuger lance le programme et s'arrête lorsqu'il a trouvé le breakpoint le nombre de fois donné.L'instruction de la ligne breakpoint aura donc été exécutée fois-1 fois.

G (Gardien) Cette commande permet de surveiller un mot long qui est soit un registre soit 4 octets en mémoire en ne gardant que les bits donnés par le masque. On exécute le code jusqu'à ce que ce mot soit change de valeur soit prenne une valeur prédéterminée. L'instruction ayant modifié la valeur se trouvait en PreC .

Exemples 1) Un programme s'autodétruit en XZ. Pour comprendre exécuter en surveillant XZ ,masque -1, C (jusqu'à changement)

2) Simulation de breakpoints en ROM. Surveiller PC, masque -1, valeur=breakpoint

Nota: Les zones non pas à pas ne sont pas surveillées.

Dans l'entrée des données de cette commande, on peut taper CR pour garder l'ancienne donnée.

H Exécute jusqu'au retour à l'instruction de la ligne 0. Cette commande équivaut donc à la commande F avec fois=2

et breakpoint=(vide) [CR].

K Remet l'écran du débbugger (utile après E)

L Change le listing de désassemblage (avec ou sans labels)

Exécution de programmes objets

Le débbugger permet aussi d'exécuter des programmes objets qui n'ont pas été assemblés par ASM. Voir la commande alt+AO.

Exécution d'un programme utilisant le GEM

Dans ce cas il faut changer le nom ASM.TOS en ASM.PRG avant de charger l'assembleur. Cela vous confisque 10000 octets environ et fait apparaitre la souris. Appelons O le label de la 1ere ligne du programme. Ce programme doit être écrit sans appel de la fonction \$4A du TRAP#1 (réservation de l'espace mémoire). Pour réserver l'espace nécessaire au programme O utiliser:

alt+L fin

On suppose que le programme n'utilise pas la mémoire au dessus de fin. Après Cette commande le GEM dispose d'espace pour ses fonctions. ASM ne touche plus à la mémoire au delà de fin.

Mettre au point le programme O par le debugger alt+GO ... Vous ne rajouterez au programme la réservation de l'espace mémoire qu'au moment où vous voulez le rendre indépendant de ASM.

5. DESASSEMBLEUR.

Nous avons vu la commande alt+H ad qui désassemble quelques lignes en code 68000. Les adresses sont alors repérées par des valeurs numériques. Nous allons voir ici le désassembleur programmable qui permet de créer un code source avec des labels alphanumériques. Ce code permet de modifier facilement un programme donné.

Etape 1. chargement du programme à désassembler.

Cette étape n'est pas nécessaire si on veut désassembler le TOS !

Plusieurs méthodes sont possibles.

1) Charger le prgr à une adresse assez haut dans la mémoire, par exemple en \$A0000, avec la commande alt+Al. Si cette méthode est employée on a un programme à désassembler implanté en n'=\$A001C qui correspond à un programme réel exécutable en n=0.

2) Charger le prgr dans le fichier objet par alt+AO. A l'aide de la commande = on obtient les limites du programme (ligne objet) par exemple de s à f. Le programme dans le fichier objet est implanté en n'=s et correspond à un programme réel exécutable en n=s et de longueur e=f-s.

Le chargement de l'objet doit être fait à une adresse assez haute dans la mémoire pour que la source puisse être remplie sans détruire l'objet. Noter que le désassembleur peut aussi détruire l'objet s'il est trop haut en mémoire, car le haut de la mémoire est utilisé pour la table des labels.

La 2ème méthode est recommandée pour les programmes. Elle permet en effet de faciliter le désassemblage en tenant compte de

la table de relocation. Pour un fichier binaire la 1ère méthode est préférable.

Etape 2. Programmation du désassembleur

Il s'agit de créer en tête du fichier source un programme se terminant par la ligne

F e

Lorsqu'on lance le désassembleur par la commande alt+D, ce programme de désassemblage crée un code symbolique qui est mis dans la source à partir de la fin. Les lignes de la source après la ligne F sont conservées mais ne font pas partie du programme de désassemblage. Les commandes du désassembleur sont détaillées dans l'appendice C.

Le programme de désassemblage commence par :

S n | n adresse d'exécution.

I n' | n' adresse d'implantation

Pour tenir compte de la table de relocation (n=n') rajouter

U r | r adresse de la table

Les nombres doivent être écrits en hexadécimal, non précédés de \$.

Mis à part ces commandes S, I et U, toutes les adresses sont les adresses relatives à n. Par exemple :

C 0 | Les adresses relatives 0 à d (réelles de n à n+d)

D d | sont labélisables.

Si l'objet contient seulement du code 68000 il suffit d'ajouter:

P 0 | e est la longueur de l'objet, c'est aussi

F e |l'adresse relative de la fin.

En général l'objet contient aussi des data (messages, mots...).

Le programme de désassemblage sera par exemple :

P 0 Zone [0,a[en code 68000

M a Zone [a,b[messages (D.B "message",13,10,0...)

P b Zone [b,c[en code 68000

A c | Zone [c,d[du type D.W L1-*,m

W * |La zone A est désassemblée suivant les instruction

W | W,L, B,...

V d Zone [d,e[en BLK.B (données non initialisées)

F e

La difficulté est de déterminer ces diverses zones. Il est utile de commencer par le programme suivant:

N 0 |Sort des D.B "message assez long" et des BLK.B k

F e |On peut ainsi repérer les zones message.

En fait après chargement par alt+AO, la source contient ce programme avec les lignes S, I, U, B, C et éventuellement une ligne V, correspondant au programme chargé.

On essaiera ensuite :

P 0

M a En supposant que [a,b[et [c,e[sont apparus comme

P b zones messages.

M c

F e

On cherchera dans la source les messages d'erreur (alt+S;*).

En général elles correspondent à des zones data. Le type de

désassemblage à effectuer dans ces zones nécessite l'étude du code. Pour l'étude du code les touches sF4 et F4 sont très utiles . Ces zones seront désassemblées en mode A, qui permet le désassemblage de tous les types usuels de data utilisés par les programmeurs.

APPENDICE A. CLAVIER

Table 1.Mouvements.

Codes	Touches	Effets
ASCII		
1	up	
2	down	
3	right	Mouvements du curseur
4	left	
5	home	
6	clr	Vide l'écran
7	delete	Efface et tire la ligne texte
8	backspace	Efface la fin de ligne texte
9	tab	Va sur le séparateur suivant
A	shft+left	Va en debut de ligne texte
B	shft+down	Connecte à la ligne texte suivante
C	shft+up	Brise les lignes texte en lignes écran
D	Return	Valide la ligne texte
E	shft+right	Va en fin de ligne texte
F	shft+Insert	Toggle résolution (basse/moyenne)
10	Insert	Insertion dans la ligne texte
11	Help	Toggle (intérieur/extérieur) de ""
12	Undo	Va en bas de page

Table 2. Touches de fonction

Effet des touches de fonction en extérieur de guillemets.

F1 Liste depuis le début

sF1 Liste jusqu'à la fin

F2 Liste jusqu'au pointeur d'inscription.

F3 Liste 1/2 page plus bas.

F4 Liste la dernière page mémorisée par sF4.

sF4 Met la page en mémoire (pile de 4 pointeurs).

nota Seul le pointeur de page est conservé, pas la page.

F5 Liste la page précédente.

F6 Liste la page suivante.

F7 Liste une ligne plus bas.

F8 Liste le bloc.

F9 Liste de nouveau la même page.

F10 Met en mode *

sF10 Met en mode > (Source)

Table 3.Codes ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		Ins	SP	0	@	P	`	p	alt+ç	alt+A	alt+D	alt+B	F6			
1	up	Help	!	1	A	Q	a	q	alt+à	alt+Z	alt+F	alt+N	F7			
2	down	Undo	"	2	B	R	b	r	é	alt+E	alt+G		F8			
3	rght	ctrl+S	#	3	C	S	c	s	alt+-	alt+R	alt+H		F9			ctrl+lft
4	left	ctrl+T	\$	4	D	T	d	t		alt+T	alt+J		F10	shft+F1		ctrl+rgt
5	home	ctrl+U	%	5	E	U	e	u	à	alt+Y	alt+K			shft+F2		
6	clr	ctrl+V	&	6	F	V	f	v		alt+U	alt+L			shft+F3		alt+1
7	del	ctrl+W	'	7	G	W	g	w	ç	ù	alt+M			shft+F4		ctrl+hom
8	back	ctrl+X	(8	H	X	h	x		alt+O	shft+alt+ù			shft+F5		°
9	tab	ctrl+Y)	9	I	Y	i	y		alt+P	ctrl+£	¨		shft+F6		alt+2
A	shft+lft	ctrl+Z	*	:	J	Z	j	z	è	alt+I				shft+F7		alt+3
B	shft+dwn	esc	+	;	K	[k	{				F1		shft+F8		alt+4
C	shft+up	ctrl+<	,	<	L	\	l		£	alt+W	F2			shft+F9		alt+5
D	CR	ctrl+§	-	=	M]	m	}		alt+X	F3			§		alt+6
E	shft+rgt	ctrl+^	.	>	N	^	n	~		alt+Q	alt+C	F4		shft+F10		alt+7
F	shft+ins	ctrl+del	/	?	O	_	o			alt+S	alt+V	F5				alt+8

Ces codes hexadécimal sont obtenus en intérieur de guillemets.

Par exemple, en appuyant simultanément sur Shift, Alternate et ù on

obtient le code ASCII \$A8. Ce code écrit sur l'écran un "?"

renversé. Les codes en blanc ne sont pas délivrés par le clavier.

La table 3b donne les caractères obtenus (rendus approximatifs pour certains)

Table 3 bis. Codes ASCII

codes des caractères Atari st

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		[0]	¸	0	@	P	`	p	Ç	É	á	ã	ij	Ɔ	α	≡
1	↑	[1]	!	1	A	Q	a	q	ü	æ	í	õ	lj	γ	β	±
2	↓	[2]	"	2	B	R	b	r	é	Æ	ó	Ø	κ	ϣ	Γ	≥
3	→	[3]	#	3	C	S	c	s	â	ô	ú	ø	ϛ	Ϙ	π	≤
4	←	[4]	\$	4	D	T	d	t	ä	ö	ñ	œ	λ	ρ	Σ	∫
5	✘	[5]	%	5	E	U	e	u	à	ò	Ñ	Œ	τ	ϛ	σ	∫
6	clr	[6]	&	6	F	V	f	v	â	û	ä	À	η	ψ	μ	÷
7	🕒	[7]	'	7	G	W	g	w	ç	ù	ó	Ã	ι	τ	τ	≈
8	✓	[8]	(8	H	X	h	x	ê	ÿ	¿	Ö	ι	ι	Φ	°
9	🕒	[9])	9	I	Y	i	y	ë	Ö	Γ	ˆ	η	ι	Θ	•
A	🔔	ə	*	:	J	Z	j	z	è	Ü	¬	'	υ	□	Ω	•
B	🎵	esc	+	;	K	[k	{	ï	¢	1/2	†	ι	η	δ	√
C	ff	ctrl+<	,	<	L	\	l		î	£	1/4	¶	κ	Υ	ƒ	ⁿ
D	cr	ctrl+\$	-	=	M]	m	}	ì	¥	í	©	λ	§	φ	²
E	atari g	ctrl+^	.	>	N	^	n	~	Ä	β	«	®	η	^	ε	³
F	atari d	ctrl+del	/	?	O	_	o	Δ	Å	f	»	™	η	∞	η	-

Symbole obtenu par alt+lettre ou ctrl+lettre

	+alt	+ctr
A	Ç	
B		
C	«	
D	á	
E	é	
F	í	
G	ó	
H	ú	
I	Ü	
J	ñ	
K	Ñ	
L	ä	
M	ö	
N		
O	ÿ	
P	Ö	
Q	β	
R	ô	
S	f	[3]
T	ö	[4]
U	û	[5]
V	»	[6]
W	1/4	[7]
X	ì	[8]
Y	ò	[9]
Z	æ	ə

Table 4.Control+Lettre.

L'appui simultané sur Control et une lettre, en extérieur de guillemets délivre un mot clef. Un espace est indiqué par _.

A	ADDQ	J	JMP_	S	SUBQ
B	BSR_	K	BLK.B_	T	TRAP_#
C	CMP	L	LEA_	U	MOVEM.L_
D	DBRA_D	M	MOVE	V	.B
E	EXG_D	N	NEG_	W	.W
F	BRA_	O	(A0)	X	.L
G	BEQ_	P	,-(SP)	Y	MACRO
H	BNE_	Q	(SP)+	Z	ENDM
I	JSR_	R	RTS		

APPENDICE B. EDITION DE LA SOURCE

Table 5. Définitions de base.

source Ensemble des lignes éditables en mémoire. Les lignes sont numérotées 1,2,3,...

pointeur d'insertion La ligne devant laquelle on peut insérer de nouvelles lignes dans la source.

bloc Ensemble de lignes consécutives bien déterminé, pouvant être copié, effacé,...

pointeur A La première ligne du bloc.

pointeur B La dernière ligne du bloc.

label Nom alphanumérique apparaissant une fois en tête de ligne suivi de ":". Par exemple :

debut;

Caractères autorisés : A-Z _ a-z 0-9 (A=a,B=b etc)

Le premier caractère ne doit pas être un chiffre

Le label repère également la ligne du source où il apparaît sous cette forme.

n Désigne un entier décimal et la ligne du source de même numéro.

sligne Désigne les façons suivantes de définir une ligne

du source:

M Ligne du pointeur d'insertion

D Dernière ligne de la source.

A Pointeur A (1ère ligne du bloc)

B Pointeur B (dernière ligne du bloc)

label La ligne du label

ligne Désigne les façons suivantes de définir une ligne

du source:

Ligne "0" (qui n'existe pas!)

(" " désigne l'absence de symboles)

n Line n

sligne

sligne+n | sligne décalé de n

sligne-n |



Table 6. Définition du bloc

Les écritures suivantes permettent de définir le bloc. "l" et "l'" ci dessous sont du type ligne.

Le bloc peut aussi être défini graphiquement

Exemple: 157.`... Défini le bloc de 157 à 245

245.'... après validation par "Return".

Pour plus détails voir l'Insertion.

alt+A Toute la source

alt+B L'ancien bloc.

n,p Bloc de la ligne n jusqu'à la première
ligne de numéro se terminant par p

Ex: 10087,03 | bloc de 10087 à

10087,103 | 10103

10087,0103 |

l, Bloc de l à la fin

,l Bloc de 1 à l

, Toute la source

l,l' (si différent de n,p):

Bloc de la ligne l à l'

l,+n Bloc de n lignes à partir de la ligne l

l Bloc formé de la seule ligne l

APPENDICE C

Table 7. Commandes globales du désassembleur.

Les nombres doivent être écrits en hexadécimal, non précédés de \$.

Mis à part n , n' et r , dans les commandes S, I et U, toutes les adresses sont les adresses relatives à n .

S n n est l'adresse de l'origine du programme réel. L'adresse réelle A doit être donnée par son adresse relative $a=A-n$. (Par défaut $n=0$)

I n' n' est l'adresse où l'origine du programme réel est implantée. Si le programme est exécutable là où il est implanté il faut donner $n'=n$. (Par défaut $n'=0$)

U r Tenir compte de la table de relocation d'adresse r . Il faut mettre cette commande juste après S et I avec $n=n'$.

G n'' n'' sert à la définition du nom des labels. Ainsi à l'adresse relative a correspond un label de nom (lettre) $\text{hexa}(a+n'')$.

C k Labélise seulement les adresses k à l . Les autres

D l adresses restent numériques. Par défaut k =début de la première zone, l =fin du programme.

X a [$;$ nom] Force la création du label à l'adresse a . Si nom est donné, ce sera le nom de ce label.

X a,l [$;$ nom] Les adresses $a+i$ ($0 \leq i < l$) sont sorties sous la forme $\text{nom}+i$. Les commandes X doivent être avant la première zone de désassemblage.

H s Options d'impression suivant les bits mis à 1 de l'octet s: (Par défaut: s=12 ie bits 1 et 4). Ces impressions se font en commentaire, après l'instruction désassemblée.

bit 0 sort l'adresse réelle (a+n)

bit 1 sort l'adresse label (a+n')

bit 2 sort l'adresse relative (a)

bit 3 sort l'adresse d'implantation (a+n')

bit 4 sort les octets de l'instruction.

Y Indiquent les parties du code désassemblé qui vont

Z être sorties dans la source.

Exemples:

P x	Z
A a	P x
Y	P a
...	Y
P b	P b
...	...
P c	F c
Z	
F d	
sort [a,c[sort [a,c[

Le flag de sortie est modifié Y(on) et Z(off) pour la zone dans laquelle se trouve Y ou Z. Il y a deux passages, le premier calcule les labels et ne sort rien, mais modifie le flag de sortie suivant Y/Z. Le

deuxième passage sort le code source seulement si ce
flag est on. Par défaut le flag est on.

;comment Le commentaire est récrit dans la source en début de la
zone où il se trouve.

a n,k Les $d(A_k)$ (pour $k=0$ à 7 et $n < > 0$) sont mis sous la forme
label-labeln(A_k). Pour annuler donner $n=0$.

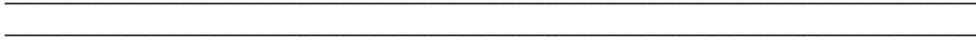


Table 8. Noms des labels

A chaque label créé est associé un flag. Chaque fois que le label est rencontré un des bits du flag est mis à 1 suivant le mode d'adressage. Le nom définitif du label commence par la lettre du bit le plus bas mis à 1. Ainsi si un label L apparaît dans BRA.S L (bit 0) et dans MOVE.L #L,A5 (bit 4) son nom commencera par S. Pour renommer les labels utiliser la commande X.

bit 1ère

0	S	Adresse relative 1 ou 2 octets.
1	W	Créé dans D.W et D.L
2	T	Modes d'adressages Adr.W et Adr.L
3	U	" Adr(PC)
4	V	" #Adr
5	X	Défini par limite de zone ou commande X
6	E	Erreur

Table 9. Zones de désassemblages.

Les commandes suivantes délimitent des zones et fixent le type de désassemblage réalisé.(J q n'est pas un délimiteur)

P a	code 68000
V a	met des BLK.B (limités par les labels)
M a	met D.B "alphanumérique",13,10,0 etc (zone messages)
J q	Met des D.B "alphanumérique" seulement si les noms
N a	alphanumériques sont de longueur $\geq q$.Par défaut $q=8$ Sinon met des BLK.B k
F a	Fin du programme.a-1 est la dernière adresse désassemblée.
A a	Désassemblage par programme.Le programme est répété autant de fois que nécessaire pour arriver en fin de zone A.Voir table 10

Table 10. Programme des zones A

- B [k] k entre 1(valeur par défaut) et 14(hexa).
met D.B u,v,...,z avec au plus k octets.
on obtient moins d'octets que demandé s'il y a un label
ou en fin de zone A.
- W D.W m où m est numérique.
- W l D.W L1-L Création des labels L (correspondant à
l'adresse l) et L1 (correspondant à l'adresse
l+m où m désigne la valeur numérique du mot).
D.W L1 Si l=0
- W l,q D.W L1-L+Q Q est la valeur numérique m.AND.q
Les labels L et L1 sont créés.
Exemple courant q=8000.Le bit de poids fort étant utilisé
pour définir une option.
- W * D.W L1-* Crée le label L1
- W *,q D.W L1-+Q Analogue à W l,q
- w ... Analogue à w ... l'adresse peut être impaire (DO.W ...)
- l ... Analogue à L ... l'adresse peut être impaire (DO.L ...)
- E |met D.B m ou rien pour rendre l'adresse paire (E)
- O |ou impaire(O).

Q k k de 1 à 3C (défaut k=3C)

D.B "alphanumérique", "x" + \$80

sur k octets au plus. Cette forme de codage de mots clefs est très usuelle.

En cas d'échec sort un seul octet D.B m et reprend le programme A au début.

q k k de 1 à 3C (défaut k=3C)

D.B "alphanumérique", 0 sur k octets au plus.

En cas d'échec sort un seul octet D.B m et reprend le programme A au début.

K q,k Sort k fois D.B u,v,...,z (suite de q octets)

R k Sort k fois D.W L1-*

T k Sort (k de 1 à 10) D.W m1,m2,...,mk

où les k nombres m1,...,mk sont numériques.

Table 11. Messages d'erreur

Repérables dans la source grâce à ;*

;*I Label interne au code 68000 Sort L:=*-offset;*I

;*D Dépassement de zone. Au lieu de dépasser on met D.B m

;*E Les labels hors de la zone de désassemblage (si on a utilisé C et D) sont sortis comme L=m;*E

;*R Erreur détectée grâce à la table de relocation

APPENDICE D SOLUTION DES EXERCICES.

Exercice 1.5 -,

Exercice 1.9.1

alt+S alt+S A alt+S Cherche les "A" entre séparateurs, car le label "A" ne peut apparaître qu'entre séparateurs.

alt+T alt+S apluslong alt+S Cherche si la chaîne apluslong existe. Supposons que non, on obtient le diagnostic "pas trouvé".

alt+C alt+H Change "A" en "aplong" sans s'arrêter après 16 changements.

alt+C alt+I alt+H Vous avez un regret, trouvez que le nouveau nom est trop long. Vous pouvez annuler le changement ainsi (parce qu'il n'y avait pas de chaîne "aplong" à l'origine)

Exercice 1.9.2

-, Vide la source précédente, se met en insertion
;987 On rentre les nombres en les faisant
;654 précéder de " ;"
;321

alt+C ; alt+T alt+R Change " ;" en chaîne vide.

F1 Liste la source.

(Z)Exercice 1.9.3

alt+C _ _ alt+T _ alt+H Remplace 2 espaces par un seul.

alt+C _ # alt+T # alt+H L'espace devant un séparateur est facultatif.Utiliser avec les séparateurs " , " - " , ...

alt+C .L_ alt+T .L alt+H L'espace suivant .size est facultatif.
Répéter avec les autres .size

alt+C BRA.L alt+T BRA_ alt+H Il est inutile d'indiquer les branches longues.Répéter avec Bcc.k ,bSr.L etc (toutes combinaisons de minuscules et majuscules et les size .L et .K

En réalité, il y a une commande, alt+U ,qui effectue toutes ces opérations de compactification. Il faut d'abord assembler avec optimisation pour que les branches BRA.L ... soient réécrites sous la forme BRA.K.Cette commande évite l'inconvénient désagréable suivant des transformations par alt+C.Les chaînes entre guillemets ou situées dans des commentaires sont aussi modifiées, ce qui peut être gênant.

(ZF)

Exercice 3.1.

Dans "LEA DEBUT(PC),A0" les bits 31-24 de A0 peuvent prendre une valeur quelconque ! Il faudrait donc comparer A0 à A1 par exemple où A1 est chargé par "LEA TFIN(PC),A1" et non à #TFIN (qui a ses bits 31-24 nuls).

6. NOUVEAUTÉS TT(68030)

Clr

dans l'écran du programme du debugger -> vide l'écran.

Shift gauche + Shift droit

stoppe exécution lancée par °G

debugger mode trace 2

stoppe après RTS, BRA, etc.

if/else/endif peuvent être imbriqués

macro peuvent s'appeler entre elles

labels locaux

U:

\0: a pour nom complet U\0

\A03: a pour nom complet U\A03

V:

°I0 met 68030 off

°I3 met 68030 on

°lautre écrit l'état 68000/68030

Pour ?, °Q, °, etc.

accepte A0, (D0), D1+25, etc.

	size	optimisable
Bcc.S	b	oui
Bcc.K	w	oui
Bcc.M	w	non
Bcc.L	l	oui
Bcc.N	l	non

PROCESSOR k

pseudo-op met mode 68000 si k=0 ou 68030 si k<>0

DIVUL.L <ea>,dl ou DIVUL.L <ea>,dl,dl équivaut à DIVU.L <ea>,dl

Le ":" officiel dans MULU.L <ea>,dh:dl et autres est remplacé par
", "

Bit field

<ea> {offset,width} doit être écrit

<ea>,offset,width

.L peut être omis dans DIVUL.L, DIVSL.L et EXTB.L

CAS2 Dc1:Dc2,Du1:Du2,(Rn1):(Rn2) doit être écrit

CAS2 Dc1,Dc2,Du1,Du2,Rn1,Rn2

7. Autres infos (2021)

Symboles

Version ST

Les symboles doivent commencer par (A-Z a-z). Les caractères suivants peuvent être (A-Z a-z), underscore (_) ou un chiffre (**0-9**). Les symboles ABC, Abc, aBc, etc. sont considérés identiques.

Version TT

Les symboles doivent commencer par (A-Z a-z) ou antislash (\). Les caractères suivants peuvent être (A-Z a-z), antislash (\), underscore (_) ou un chiffre (**0-9**). Les symboles ABC, Abc, aBc, etc. sont considérés identiques.

Equ

On peut donner une valeur à un symbole :

symbole :equ expression

symbole = expression

Il n'est pas permis de changer sa valeur :

A=13

B=14

Expression

Voir la commande ? **ad** (§2.2 divers)

Les symboles dans expression doivent être préalablement définis

Exemple:

QA:NOP

QB=QA+[10 * [7+\$25]]/3

On peut utiliser des nombres hexadécimaux (\$64), décimaux (100), octaux (@144), binaires (%1100100) et des constantes ASCII ("d").

Les opérations + - * / &(and) !(or) ~(eor) > = < sont effectuées dans l'ordre de lecture sauf s'il y a des crochets [].