==========

TFB-Disasm
The Flaming Bird Disassembler

==========

HISTORIQUE

Date	Version	Description
08/92	1.0b	First draft
11/92	1.0b2	Trucs en plus...
12/92	1.0b3	Améliorations...
02/93	1.0b4	La saga continue
10/93	1.0b5	Assymptote
12/93	1.0	First release

==========

Intro

Yeah... Voici la toute première mega-production du PHC entièrement sous GS/OS. Si si, c'est bien un fichier S16. A vrai dire, on adore le bon vieux DOS 3.3, mais on a jugé qu'il ne serait pas forcément tout à fait adapté à cette production. Alors voilà, on se retrouve avec un soft comme les aime Apple à ceci près qu'on est sur Apple II...

TFBD, comme son nom semble bien l'indiquer, est un désassembleur. (Mais attention: un désassembleur Merlin™. Chose rare). Il vient donc de ce fait s'ajouter aux deux autres courament utilisés: Orca™ et Sourceror. Ça en fait trois, et c'était indispensable pour ceux qui voulaient les sources Merlin™ avec la puissance d'Orca™ et la convivialité du mode texte. En fait, à l'origine, je m'étais mis à coder ce truc pour désassembler un fichier duquel aucun désassembleur ne pouvait sortir un truc correct: Start.GS.OS. Essayez, vous comprendrez vite pourquoi. Mais voilà qu'à force de rajouter des options a un soft qui ne devait être à l'origine qu'un Sourceror amélioré, on se retrouve avec un truc professionnel sur les bras, qui va jusqu'à nous obliger à écrire une doc... Que voici.

==========

Considérations

“The Flaming Bird Disassembler” est un shareware.

Ou presque. Vous êtes libre de le copier et de le diffuser, et ce dans sa version intégrale et d'origine (uniquement). Vous êtes également libre de l'utiliser autant que vous voulez, et ce durant une période illimitée. A vous de juger la valeur de ce soft et de m'envoyer ce que bon vous semble: un chèque, un de vos logiciels commercialisés, ou bien simplement une lettre pour me dire que vous trouvez ce soft chouette, voire même rien du tout si vous le trouvez nul, ou que votre compte est a zéro, ou que ça vous empêcherait de dormir d'envoyer 100 balles pour avoir economisé quelques milliers d'heures de programmation.

Je réponds dans la mesure du possible aux lettres qu'on m'envoie, les classant en un ordre de priorité dépendant de leur contenu. Ce dernier détermine aussi les privilèges de chacun vis-à-vis des versions futures, peut-être commerciales.

==========

Généralités

Mémoire: TFBD tourne sans aucun problèmes avec 1.2 Mo de RAM sous système v6.0 et au-delà. La RAM maximale prise par TFBD en cours de désassemblage est de moins de 300K en comptant le soft lui-meme, le code à désassembler et les données qui lui sont relatives. Le chargement ou l'édition d'un fichier script prend plus de 64k supplémentaires. La place prise par les templates depend de leur nombre mais n'éxède que très rarement les 60 ko. Au total, pour se servir de TFBD au maximum de ses capacités, il est utile d'avoir 400 à 500 ko de libre dans la machine. Compte tenu du fait qu'il n'utilise aucun toolset “ram”, c'est jouable même avec 1.2 Mo.

Vitesse: Au delà d'un certain stade du désassemblage (si le nombre de constantes ou de labels devient assez conséquent), TFBD peut subir de sérieux ralentissements; aussi, une carte accélératrice sera la bienvenue si vous désassemblez un fichier de quelques centaines de Ko.

Resources: TFBD conserve certaines de ses données (préférences, pathnames) dans son resource fork, mais peut fonctionner sans lui. Si le resource fork est absent (par exemple s'il a été détruit par un utilitaire de copie ignorant les resources), TFBD prendra ses options par défaut internes, mais qui ne seront alors plus modifiables.

Syntaxe: TFBD n'accepte comme paramètres de commande que des nombres hexadecimaux. Une adresse peut ou non etre precedée de $, mais ce n'est pas utile, sauf au cas où celle-ci commence par une lettre ($FF69 par ex.) pour ne pas

être confondue avec un label. Un paramètre ne representant pas une adresse n'a pas a être précédé de $.

Bugs: N'étant pas tous les jours un programmeur absolument parfait et exempt d'inattention, cette version de TFBD peut encore receler quelques bugs qui auront echappé a ma vigilance. En cours de désassemblage, faites donc des sauvegardes fréquentes des templates de manière à éviter de tout perdre en cas de plantage définitif du soft (Je ne saurais trop vous recommander d'avoir en memoire le QuitCDA du Phœnix corp., qui résoud ce petit problème de la sauvegarde avec TFBD).

Si vous trouvez un bug (même mineur), n'hésitez pas à prendre la plume et a m'écrire, en indiquant bien les circonstances du bug, la manière de le reproduire, sur quel fichier il s'est produit (envoyez-le moi si nécessaire), etc... Idem bien sûr si vous avez des idées, des sous, des softs ou des suggestions, tout peut être intéressant. Mes coordonnées:

Philippe Savitch
19 rue de la Duée
75020 Paris - France

==========

Doc

Lorsque vous entrez dans TFBD, tout ce que vous pouvez voir est une petite ligne de texte en haut, et un curseur en bas, qui clignote nonchalament. Cela signifie tout simplement qu'il attend une commande de votre part. L'écran vide au milieu, lui, ne demande qu'à se remplir, et on se doute avec quoi. En l'absence de fichier objet chargé, les commandes possibles sont les suivantes:

?		Pages d'aide
$		Pages de shareware
PFX	[path]	Sélection du prefix par défaut (0:)
CAT	[path]	Catalog du directory “path” ou 0: si non precisé
POP		Remonte dans le prefix
MD	name	Crée un directory.
LOAD	filename	Charger un fichier objet
RLOAD	file [,t,id]	Charger une resource
CFG	name	Charger une config
QUIT		Quitter TFB-Disasm

ROM	Désassemblage des ROMs
SLOAD [pathname]	Chargement d'un fichier Script
SEDIT	Edition des scripts
SSAVE [pathname]	Sauvegarde des scripts
DSK	Accès au desktop
HIST	Historique

Description

SLOAD, SEDIT, SSAVE : cf section “Les scripts” plus loin.

?

'?' affiche les pages d'aides de TFB-Disasm, qui ne sont autre que la liste relativement exhaustive des commandes possibles. Au vu du peu de place disponible sur un écran et dans un segment de code, elles ne sont pas très detaillées; c'est plus un aide-mémoire qu'autre chose.

$

'$' affiche les pages de shareware. Il est toujours bien de les avoir lues au moins une fois, quoi...

PFX [path]

Permet, au même titre que son homonyme que l'on recontre un peu partout, de changer le prefix par defaut (le zero, et uniquement le zero pour l'instant). Si le directory n'est pas specifié, elle permet de le choisir de manière “ergonomique”, en se baladant avec les flèches.

Flèche	gauche:	descend d'un niveau
"	droite:	monte d'un niveau
"	bas...:	dir suivant
"	haut..:	dir précédent
<ESC>.....:		cancel
<CR>.....:		accept.

CAT [path]

Liste les fichiers du directory courant ou du directory “path”. Le formattage du listing est un peu particulier en ce sens que les tailles en blocks et bytes des forks data et resources sont séparées (oui, j'ai méchament sacrifié la date de creation du fichier pour ça, en attendant les écrans texte 160 colonnes).

POP

Fait remonter le préfixe courant (0:) d'un niveau. Arrivé au directory principal, cette commande n'a plus d'effet.

MD filename

Crèe le sous-directory “0:filename”.
Pas grand-chose à dire de plus.

LOAD [Path:] Filename

Charge le fichier à désassembler.

Dans le cas d'un fichier non relogeable (SYS, BIN, ou même un fichier data), le code objet est désassemblé par défaut a partir de l'adresse zéro à moins que son Type ne la spécifie (par exemple $2000 pour SYS, l'AuxTupe pour BIN …), en bank zéro.

Dans le cas d'un Load File (Types $B1 a $BE), le premier segment est chargé et désassemblé à partir de $01/0000. Les infos OMF du segment sont chargées elles aussi et TFBD s'en aide pour le désassemblage. Si le fichier débute par un segment ExpressLoad, c'est le second segment qui est d'abord chargé et désassemblé en $02/0000 (il est toujours possible de charger le segment ExpressLoad, mais c'est sans grand intérêt).

La touche command (pomme) enfoncée au moment de l'ordre de chargement change le status du fichier entre Load File & Data File. Ça peut être utile pour charger certains Load Files sauvés sous un type spécial (il en existe par exemple en type OS, $F9) ou au contraire pour charger un Load File tout d'un block, en-tête de segment et infos OMF comprises (attention à la limite des 64K).

RLOAD [Filename [,rType,rID]]

Charge une resource a désassembler.

Le Filename est facultatif dans la mesure ou l'on veut charger une resource du fichier déjà en cours de désassemblage. Si l'on ne précise rien ou bien seulement le Filename mais pas rType et rID, une fenêtre de sélection s'ouvre, permettant de choisir sa resource dans une liste.

Sur une ligne de la liste sont indiqués: si la resource a besoin d'un convertisseur (“c”), son type (hexa & nom) et son ID. On peut charger n'importe quel type de resource, code ou data. TFBD se base sur le fait que la resource a besoin d'un convertisseur pour savoir si c'est ou non du code. Au cas où il se trompe de type de donnée, faire comme avec LOAD: Pomme-Ouverte au moment de la sélection change le statut de ce qui est chargé entre OMF et Data.

Quelques rTypes code:	8017	rCodeResource
	8018	rCDEVcode
	801C	rCtlDefProc
	801E	rXCMD
	801F	rXFCN

Attention: TFBD n'est pas DeRez, et ne peut faire ce que fait DeRez (ce n'est pas son but). Pour TFBD, chaque resource est traitée comme un fichier à part, indépendament des autres resources, et indépendament du data fork. Ainsi, chaque resource a SON fichier template, et en passant d'une resource à l'autre ou du data fork au resource fork, il ne faut pas oublier de le sauver.

CFG [/S ou /D] CfgName

Sauvegarde, délétion ou chargement d'une configuration. Les fichiers configs de TFBD sont dans le directory 1:Configs et sont de type $5A/$8040. Ils contiennent l'état actuel de TFBD: le préfixe courant, le pathname du fichier en cours de désassemblage ainsi que celui de ses templates et du fichier script chargé, le segment courant (y compris resource), la position courante de l'écran et des flags divers, notament l'état de ^C, ^R et ^S (voir les commandes de contrôle, plus loin).

Sauver la config courante:	CFG /S MyConfig
Charger une config:	CFG MyConfig
Eradiquer une config:	CFG /D MyConfig

La commande CFG sert également à charger et sauver la resource définissant les préférences de TFBD (rType=1 et rID=1), à savoir les états de ^R, ^C, ^S et les

tabulations. Par défaut, ^R est actif (relocs en inverse dans le dump hexa), ^C et ^S sont inactifs (constantes non inversées et mnémoniques en majuscules). Ces préférences sont automatiquement chargées au démarrage.

Sauver les préférences: CFG /S
Charger les préférences: CFG

QUIT ou BYE

Quitte TFBD et retourne au launcher. Je ne pense pas qu'il y ait grand-chose à dire là-dessus.

ROM

Commande de désassemblage des ROMs. Pour l'instant, ROM ne permet de désassembler que les ROMs 01.

DSK

Une petite commande rajoutée après les supplications repetées de Bandit II qui tenait absolument à ses petits NDAs. Mais ne rêvons pas: la version desktop de TFBD n'est pas pour demain; un désassembleur est exactement le genre de soft que la “Human Interface” rend parfaitement inutilisable... Un équivalent plus rapide de DSK est ^* (cf commandes de contrôles). L'accès normal au desktop s'effectue en mode 640; si l'on veut y accéder en mode 320, il faut enfoncer la touche pomme au moment de l'appel (DSK + pomme-return ou pomme-ctrl-*).

HIST

Historique des commandes. Affiche les 18 dernières commandes entrées.

Maintenant qu'un superbe code objet dont on est avide d'extirper les entrailles est chargé, on a accès a l'ensemble des commandes de TFBD. Les voici, décrites par section.

Commandes Control-.

Les commandes Control-qqchose sont accessibles à tout moment, même tout en tapant une commande ligne. Les voici:

^S: Désassemblage des opcodes en majuscule/minuscule.

^H: Passe de l'affichage opcode à l'affichage hexa.
En mode hexa:
- ^R: Affiche en inverse video les zones de relocations (OMF ou REL). Ca permet de localiser assez vite les tables d'addresse dans un fichier OMF, ou de voir si on a pas oublié de REL dans un fichier non relogeable.
- ^C: Affiche en inverse video les zone où ont été définies des constantes.

^+: Empile la position courante de l'écran pour y revenir plus tard.
^-: Dépile la dernière position empilée par ^+.
16 positions au maximum sont empilables.

^B: Saute au debut du code
^N: Saute à la fin du code

^*: Accès au desktop (idem commande DSK)
Pomme-^* : desktop 320

^Flèche haut: remonte l'historique des commandes
^Flèche bas: descend l'historique des commandes
(50 commandes memorisées)

Les commandes changeant un état du désassembleur, càd ^H (dump hexa), ^S (maj/min), ^R (relocs inverses) & ^C (csts inverses) en l'absence de fichier objet son enregistrées et prennent effectivement effet au moment du désassemblage.

Commandes générales

LIST $Adr ou LIST Label

Désassemble le code objet à partir de $Adr ou Label. Par défaut, l'adresse est automatiquement réalignée sur le début de l'instruction ou de la constante si $Adr tombe en plein milieu. Ca permet de s'y retrouver. Si par contre l'on veut volontairement couper une instruction (c'est assez fréquent), il faut rajouter “*” derrière l'adresse. Ex: LIST $2543*

SEG [n]

Charge puis désassemble le segment n. L'adresse de désassemblage par défaut est $n/0000. (Ne sert évidement à rien pour un fichier SYS ou BIN)...

Si le numero de segment n'apparait pas, c'est une liste de sélection qui apparait, contenant les numéros et noms des segments du fichier. Oui, cette commande marche aussi pour les resources multi-segments (il est d'ailleurs assez amusant de voir Orca/Disasm™ s'obstiner à ne désassembler que leur segment ExpressLoad. Meuh non, je ne critique pas...).

SRC [range] File

Génère le fichier source File.S du segment courant. Si celui-ci devient trop long, sont générés les sources File.2.S, File.3.S, etc. Si le segment utilise des equates ou des externals, les fichiers File.E.S et File.X.S sont créés.

Le paramètre facultatif [range] sert à ne générer que le source partiel de la zone qu'il spécifie. Sa syntaxe est un peu particulière:

[adr1.adr2] (crochets obligatoires): zone de adr1 a adr2 compris.
[adr1.] : de adr1 a la fin du code.
[.adr2] : du debut du code a adr2.

Au total, SRC genere trois types de fichiers:

- File.S [File2.S, File3.S, ...] .... Source du segment
- File.E.S ..... Equates de TFBD.Data
- File.X.S ..... EXTs et user EQUs

PS : Désolé pour la lenteur de cette fonction, mais je n'ai pas encore trop cherché l'optimisation. Et puis on s'en fout. Si vous partez d'un code objet de 50K, allez vous faire couler un bain. Le temps n'est jamais perdu.

INFOS

Affiche les infos sur le segment en cours de désassemblage. En fait, rien de plus que le dump commenté du Segment Header, mais il recèle parfois des infos intéressantes.

Constantes

TFBD gère à l'heure actuelle 15 constantes (il en manque une ou deux, mais il faut attendre que j'en aie besoin, question de flemme):

DB	Byte	1 octet(s)
DW	Word	2
DDB	Double Byte	2
DA	Address	2
ADR	Address	3
ADRL	Long address	4
FLO	IEEE extended	10
HEX	Hexa	-
DS	Define Space	-
ASC	ASCII	-
REV	ASC renverse	-
DCI	Asc, Msb-ended	-
STR	String	-
STRL	C1 String	-
CHK	Checksum byte	1

La syntaxe utilisée pour placer une constante a un endroit du code objet est la suivante:

cst addr [,num] pour une constante à longueur determinée

et:

cst addr [,len]

ou cst addr [.addr2] pour une constante à longueur indeterminée

“cst” est le pseudo-opcode de la constante, càd DW, ADR, STR, etc...
“addr” est l'endroit où on la place (adresse hexa ou label).

“num” est le nombre qu'on en met à la queue-leu-leu. Marche pour:

DB, DW, DA, DDB, ADR, ADRL, FLO, STR, STRL, DCI & CHK.

“len” est sa longueur.
“addr2” est son point d'arrêt (compris dans la constante). Pour:
HEX, DS, ASC & REV.

Si seule l'adresse de début est indiquée pour une constante de longueur indeterminée, la longueur par défaut sera de 1 octet pour DS et HEX, et jusqu'au prochain zéro pour ASC (pratique pour les CStrings). Mais attention: si ASC ne rencontre pas de zéros, il ira jusqu'à la fin du code objet...

Lors de la génération en série de DCIs, TFBD considère qu'ils sont tous du même type (que tous les caractères de fins ont le même bit de poids fort), ce qui permet de gérer les DCIs de 1 caractère.

Vu que je n'avais aucune doc sur le SANE, la constante FLO m'a quand même couté un désassemblage sauvage des “run-time” d'un compilateur dont je tairai le nom. La routine de conversion ASCII tente de faire les choses intelligement, c'est à dire que les trucs du genre “0.00000000000000000e+0” sont dans la mesure du possible evités...

STR et STRL doivent pointer, bien évidemment, sur l'octet ou le mot de longueur de la chaîne, sinon petites surprises !

Pseudo-constantes:

CS Addr[,num]

Une variante a ASC pour générer des C-Strings. L'avantage de cette commande est de pouvoir les générer en série. Si num n'est pas spécifié, CS n'en génère qu'une (comme ASC).

C1 Addr[,num]

Une variante de STRL, mais pour les C1-Strings trop longues pour tenir sur une ligne. C1 désassemble un DW pour la longueur du texte puis désassemble le texte via ASC.

CS et C1 peuvent etre placées dans des structures, au même titre que toutes les autres constantes. Ex: “[DW C1] c1out”. (voir section “Les structures”, plus loin).

Remarque: En interne, TFBD gère les constante de longueur fixe (DA,...) en un seul bloc lorsqu'elles sont générées en série avec le paramètre “num”. Ça permet de limiter la place prise par son enregistrement et d'accélérer le désassemblage. Ainsi, quand c'est possible, générez toujours les constantes par gros blocs.

Labels

Ben oui, y'a des labels. Ca serait malheureux pour un désassembleur quand même... Je vous donne les commandes en vrac:

LAB $Adr,Label
: Affuble l'adresse $Adr de l'étiquette “Label”.

LAB Label,Label2
: Renomme le label “Label1” en “Label2”.

LAB $Adr ou LAB Label
: Efface le label défini a $Adr ou le label “Label”.

ENT $Adr,Label
: Crèe un label intersegment

ENT Label
: Transforme le label “Label” normal en label Entry.

EQU $Adr,Equate
: Définit un equate (Equate = $Adr).

GENLAB

: Génère les labels à partir des infos OMF, des Relocs (cf plus loin), et du code machine. Les labels générés sont toujours alignés sur le début des instructions et des constantes. Si le code puise une information au nième octet d'une constante, l'opérande sera de la forme “Label+n”. Voir la commande MATCH pour plus de renseignements.

Les labels sont de la forme usuelle; ils commencent par une lettre ou par “~” ou “_” ou “:” et peuvent contenir tout cela sauf “:”, et des chiffres.

Directives

Trois directives sont actuellement gérées: MX, ORG et DBR.

MX $Adr,%xx (xx= 00, 01, 10 ou 11)

Ajoute au source la directive de changement de taille de registres processeur, puis redésassemble le code objet a partir de cette adresse jusqu'a ce que les tailles recorrespondent.

MX $Adr

Enlève la directive; remet la taille des registres en place en se basant sur leur taille en ($Adr - 1).

ORG $Adr,$xxxxxx

Change, à partir de Adr, l'adresse d'assemblage du code objet.

ORG $Adr

Retour a l'origine principale a partir de Adr, ou annulation de la directive si un changement d'origine a été défini a Adr.

DBR $Adr,xx   (xx= Bank #)

Indique au désassembleur que le Data Bank Register (registre B) est différent du bank courant; cela permet au générateur de labels (voir GENLAB) de générer les labels correctement.

DBR $Adr

Remet la valeur de B au registre K ou annule le DBR $xx s'il a été précédement défini a $Adr. La directive DBR n'est affichée que dans le désassembleur; elle n'apparait bien sur pas dans le fichier source, cette directive ne servant qu'au générateur de labels.

Commentaires

COM addr,texte du commentaire
Place un commentaire a addr

COM addr
Enlève le commentaire de addr

Le commentaire n'a pas à être entre guillemets, et un espace est automatiquement ajouté après le point-virgule (esthétisme, voyons...). On ne peut pas mettre de commentaires de début de ligne (*).

Relocations/Offsets

REL $Adr [,Size [,Shift,$Ref [±Disp]]]

Ajoute un enregistrement de relocation à $Adr, défini sur Size octets, qui référence $Ref±Disp avec un décalage de Shift bits. Cette commande n'est généralement utile que pour les fichiers non relogeable, les relocations étant définies dans les infos OMF.

Par exemple, on a:
```
Adr1    PEA  $0000
Adr2    PEA  $7DD1
```

qui correspond en fait a un PushLong #Label, avec Label= $007DD1. Pour que le source généré soit correct, il devrait être de la forme:

```
Adr1    PEA  ^Label
Adr2    PEA  Label
```

pour ce faire, il faut indiquer a TFBD (qui ne peut pas le savoir !) que le $0000 est la partie haute de Label et $7DD1 la partie basse:

REL $Adr1+1,2,-10,Label
REL $Adr2+1,2,0,Label

le -10 (hexa !) indique que l'adresse est décalée de 16 bits vers la droite, et que l'on se retrouve donc avec sa partie haute. Un -8, par exemple, indiquerait un décalage de 8 bits vers la droite et un 8 un décalage d'autant vers la gauche.

Mais pour cet exemple precis, il est suffisament fréquent pour avoir sa commande spéciale: PHL $Adr1, qui va chercher elle-même les deux parties de l'adresse et effectuer la relocation.

Imaginons maintenant le cas suivant:

```
Adr     ADRL  $00007DD1
```

En ce cas, il suffit de taper:

REL $Adr

puis la ligne devient:

```
Adr     ADRL  Label
```

La routine a été chercher elle-même l'adresse de référence ($7DD1) et la taille (celle de la constante, ici 4). Le décalage et le déplacement sont mis par défaut a 0.

Et ca marche aussi dans ce cas:

```
Adr     LDX  #$7DD1
```

REL $Adr est equivalent a: REL $Adr+1,2,0,$007DD1

Donc: Adr LDX #Label

La commande REL permet une autre chose amusante, evoquée plus haut: les déplacements. Si l'on a:

```
Adr  ADRL  $00007DD1
```

taper:

REL $Adr,4,0,$7DD2-1

```
donnera:    Adr     ADRL  $00007DD2-1
ou:         Adr     ADRL  Label-1          si Label= $007DD2
```

Tous les déplacements sont possibles entre -80 et +7F.

Il existe également des variantes bien utiles a la commande PHL. Pour commencer, celle-ci ne vérifie pas la présence des PEA, c'est à dire qu'elle marche aussi pour:

```
Adr     LDY  #$0000
        LDX  #$7DD1
```

La commande RPHL s'occupe du cas ou le mot de poids faible vient en premier:

```
Adr     LDX  #$7DD1
        LDY  #$0000
```

La commande PHL2 permet la meme chose dans le cas ou les deux parties d'une adresse sont eloignees:

```
Adr1    LDA  #$7DD1
        STAL Pointer
Adr2    LDA  #$0000
        STAL Pointer+2
PHL2 $Adr2+1,$Adr1+1
Adr1    LDA   #Label
        STAL  Pointer
Adr2    LDA   #^Label
        STAL  Pointer+2
```

(La premiere adresse donnee a PHL2 doit pointer sur le mot de poids fort de l'adresse referencee)

Voila à peu près de quoi générer rapidement et correctement les labels exacts des applications non relogeables. Plus ceci:

OFF $Adr

Condidère le mot stocké a $Adr comme un offset a partir de sa position. Exemple:

```
Adr     DA      $0068
OFF $Adr
Adr     DA      Label-*
```

Avec Label= Adr + $68

Très pratique pour désassembler les segments ExpressLoad, ce qui est parfaitement inutile...

MREL [±Disp,] Adr,n [,Ref]

MREL permet de générer des enregistrements de relocation en série. Ça fait gagner quelques heures dans les tables d'adresses. MREL référence n adresses à partir de Adr subissant un déplacement de Disp (optionnel). Ref n'est utilisé que pour passer à MREL la partie haute de la référence si celle-ci n'est pas spécifiée dans le code objet (par exemple une table de DA qui référencent un autre banc mémoire). Pour une table d'ADR ou ADRL, le paramètre Ref est ignoré. La spécification d'un déplacement commence obligatoirement par + ou -.

Exemple:

```
            Adr     DA $XXXX
                    DA $YYYY
                    DA $ZZZZ
```

MREL -1,Adr,3,020000

```
=>          Adr     DA $02xxxx-1
                    DA $02yyyy-1
                    DA $02zzzz-1
```

Puissant, non ?

Correctifs

Différentes commandes permettent de corriger des imperfection de désassemblages sans avoir à magouiller pendant des heures. La première est simple, même évidente:

```
            REM C|L|D,addr1.addr2
ou          REM C|L|D,addr,len
ou          REM C|L|D,addr
```

Enlève les Constantes/Labels/Directives entre deux adresses.
Exemple:

```
            REM CL,$7DD1,4
```

Enlève les définitions de constantes et de labels de $7DD1 a $7DD4.

Si l'adresse de fin ou la longueur ne sont pas specifiées, les objets seront enlevés à la seule adresse. Si rien n'est spécifié, les objets sont enlevés de tout le segment.

```
            MATCH C|L|D
```

Aligne les directives sur les débuts d'instructions et élimine les labels et les constantes non alignées, et ce sur l'ensemble du fichier.

Exemple:

```
            LDA   #^Addr
            STA   Label2
            LDA   #Addr
```

```
                    STA   Label
                    RTS
Label       DS      2
Label2              DS    2
```

Label et Label2 faisant partie d'un même pointeur, on tape:

DS Label,4

```
Et:                 LDA   #^Addr
                    STA   Label2
                    LDA   #Addr
                    STA   Label
                    RTS
Label       DS      4
```

Et on constate avec horreur que (certaines fois, mais c'est le cas dans notre exemple) Label2 est toujours défini et utilisé en lieu et place de Label+2.

Eh bien MATCH L corrigera ceci en éliminant Label2 et en donnant a Label un champ d'action de 4 octets au lieu de 2 précédement.

```
Resultat:   LDA   #^Addr
                    STA   Label+2
                    LDA   #Addr
                    STA   Label
                    RTS
Label       DS      4
```

Un source impeccable, et sans frotter.

Il est recommandé d'utiliser la commande MATCH L avant toute génération de labels et MATCH CL avant un TSAVE (bien que ce ne soit pas indispensable dans ce dernier cas). D'une manière générale, lorsque vous voyez quelquechose qui paraît anormal dans le désassemblage au niveau de la disposition des labels par exemple, essayez la commande MATCH, puis si ça ne marche vraiment pas, voyez ça avec la commande TC (correction des templates, cf plus loin).

Attention: pour les commandes REM et MATCH, les commentaires et relocations sont traités comme directives.

BUG [$Adr][/n]

Recherche les defauts de désassemblage. Si aucun paramètre n'est spécifié, BUG recherche tous les défauts dans tout le fichier. Sinon, il recherche à partir de $Adr les défauts de type n, puis ensuite repart du début du fichier pour le type n+1, jusqu'à la fin.

n=0: BRK. Cherche les breaks, qui en général ne doivent pas se trouver dans un code (ou rarement).

n=1: Branchements jamais pris. Recherche les séquences du type SEC/BCC, CLV/BVS, etc. Peut être très utile dans certains cas. Si on désassemble la ROM, par exemple: souvent, l'opérande du branchement correspond au positionnement inverse de la condition. (CLC, etc). Ex: CLC, BCS *+$38: $38=SEC.

n=2: Mauvais branchements. Détecte les branchements sur les opérandes ou sur les constantes. Correspond souvent a des zones mal désassemblées (constantes oubliées ou en trop).

Lorsqu'un bug est détecté, sa description s'affiche sur la ligne du bas; à ce moment, on peut reprendre la main avec Return, ou continuer la recherche avec Space. Taper “BUG” de nouveau sans paramètres reprendra la recherche là où elle a été interrompue. Taper “BUG addr” reprendra la recherche à addr avec le même n.

Les templates

Un fichier templates de TFBD est un fichier de type $5E, aux $8002 qui contient tous les enregistrements éffectués sur un code objet (constantes, directives, labels, commentaires, relocations, offsets, enfin bref tout) et qui permet de sauvegarder le désassemblage et de le reprendre au point ou on l'a laissé. Voici les commandes:

TLOAD FileName.T ou TLOAD

Charge le fichier FileName.T ou le dernier fichier template chargé ou sauvé si aucun nom n'est specifié. Le “.T” est optionel, mais il vaut mieux le mettre pour s'y retrouver après.

TSAVE FileName.T ou TSAVE

Exactement la même chose que TLOAD, mais pour sauver.

TCLR

Efface tous les enregistrements présents en mémoire (Idem que REM CLD, mais pour tout le fichier et pas seulement pour le segment courant).

TC

Templates Correction. Elimine les défauts des enregistrements présents en ram. Cette commande était surtout utile pour moi lors du debugging de TFBD, mais elle peut aussi servir a récupérer le plus grand nombre possible d'enregistrements dans un fichier templates qui a été endommagé. Elle élimine les enregistrement qui ne correspondent à rien, ou qui font référence a des adresses extérieures au code objet. Elle réactive egalement des enregistrements qui auraient été “virtuellement” éliminés du désassemblage par des remaniements successifs d'une même zone.

Recherches

FIND [range] [hexs] ["text"] [>adr] [>>adr] [^adr] [:cst]

Cette commande marche comme à peu près tous les “finds” qu'on peut trouver un peu partout, à part une ou deux spécificités. Quand elle a trouvé une occurence de la chaine, elle le fait savoir en listant l'endroit concerné (qu'elle place à peu près au milieu de l'écran) dont l'adresse précise apparait en bas. A ce moment, frapper <ESC> ou <CR> rend la main, une autre touche continue la recherche. Pour reprendre la recherche là où elle a été interrompue ou recommencer la même recherche, taper FIND sans plus de paramètres.

range est l'espace de recherche. S'il n'est pas spécifié, c'est
l'ensemble de l'objet. Sa syntaxe est la même que pour SRC:
[adr1.adr2] (crochets obligatoires): zone de adr1 a adr2 compris.
[adr1.] : de adr1 a la fin du code.
[.adr2] : du debut du code a adr2.

hexs est une chaine quelconque de mots hexadécimaux de 1 a 8 chiffres.
Tout mot de plus de 8 chiffres est tronqué a son long mot de
poids faible.

"text" est une chaine ASCII case-insensitive (càd "Z"="z") et également
hi-bit-insensitive ("Z"='Z'). Tous les caractères non spéciaux y

sont admis, bien que le "?" ait une fonction un peu particulière.

>adr représente une instruction de saut ou de branchement long vers adr (qui peut etre un label): JMP, JSR, JSL, BRL ou JMPL adr.

>>adr représente toutes les instructions de saut ou de branchement court ou long vers adr (branchements conditionnels etc).

^adr représente une référence à adr, quelle qu'elle soit (en opérande, en constante, etc) et quelle que soit sa taille et son décalage (shift count).

:cst est une constante supposée être à cet endroit, qui peut y être ou pas au moment de la recherche. Si la chaine entrée contient des constantes, FIND propose de les mettre en place quand il trouve une chaine qui a l'air de coller.

? est un “wild nibble” dans un nombre hexa ou un “wild char” dans du texte. Ne peut être utilisé dans une référence (^, > ,>>).

Exemples de chaines valides:

```
FIND  A2 0902 22 E10000
FIND  C9 "M.K."
FIND   [Start.] A? 2000
FIND  [.Label] A9 ???? >Routine
FIND   >Print :CS
FIND  22 E100A8 20?? ^Parms
```

etc...

Attention: FIND recherche la chaine dans le code objet et non dans le source équivalent. Ainsi, les fantaisies du genre “FIND "LDA #$0005"” ne marchent pas.

Remarque 1: Le calcul d'une référence dans le code prend beaucoup de temps. Le recherches comme “FIND ^adr” peuvent parfois (souvent) être très lentes. De ce fait, quand on sait un peu plus précisement ce qu'on cherche, mieux vaut le faire savoir. Si on cherche un branchement, le “FIND >adr”, de nature moins générale, sera déjà beaucoup plus rapide. Et si on sait qu'on veut un JSR, alors “FIND 20 ^adr” est quasi-instantané.

De plus, le plus souvent, une référence correspond à sa valeur exacte dans le code. Quand c'est possible, remplacez les “^adr” par “adr”, tout bêtement. Ça accélère.

Remarque 2: FIND estime parfois mal la taille réelle d'une référence, comme par exemple dans le cas ou une reference sur 3 octets est divisee en deux Relocs de 16 et 8 bits successifs dans l'OMF: FIND verra les deux références séparément; Ca peut poser des problèmes au niveau du placement des constantes. Raison de plus pour désigner quand c'est possible (et suffisant) une référence par sa valeur exacte.

Remarque 3: La sytaxe des noms de constantes est la meme que pour les structures (la routine trainait par là...), càd que l'on peut fort bien entrer des choses du genre: FIND *[choses]* :DW*2 *[choses]* :HEX(10)*3 ...
Mais n'abusons pas, quand même. Une constante peut également apparaitre en première position.

SCAN [range] [kinds]

Celle-ci permet de rechercher un type particulier de données dans tout ou partie du code objet.

range A la même syntaxe que pour FIND.
Si non specifié, recherche dans tout le segment.

kinds Caractères représentant le type de données a chercher.
Si non specifiés, recherche tous les types.
A l'heure actuelle:
A : tables d'adresses
S : p-strings

Lorsque SCAN a trouvé une zone correspondant à l'un des types à chercher, elle le fait savoir en dumpant la zone dont l'adresse apparait en bas de l'écran (dans le dumping, le debut de la zone est à la 3è ligne, histoire de pouvoir voir ce qui se passe un peu au-dessus et au-dessous). Si l'interprétation de la zone est correcte (conseil: toujours vérifier !), taper Y pour mettre les constantes en place. <ESC> et <CR> permettent de reprendre la main, et une autre touche de continuer la recherche.

Remarque: SCAN S ne recherche pas les p-strings trop courtes (moins de 5 caractères) et n'y accepte comme caractères de controles que CR, LF & BELL. On est plus ou moins forcé de poser de telles restrictions, sinon la routine verrait partout des strings qui sont tout sauf des strings (peut-on vraiment se permettre ici de considerer un zéro comme une string vide ?...).

Les commandes qui suivent sont plus des commandes de confort qui servent dans des cas assez particuliers. Je les ai placées pour en avoir eu besoin une fois ou deux, et elle sont restées au cas où vous aussi...

STOOL addr,tsnum

Désassemble un en-tête de toolset système. Met en place la table d'ADRLs, avec leur déplacement de -1, et met les labels correspondant aux noms des fonctions. Les noms sont pris du fichier TFBD.Data.

“addr” pointe sur le début de l'entête (nombre de fonctions).
“tsnum” est le numéro du toolset.

MLABS

En cours de désassemblage des ROMs, met en place les labels du moniteur pris dans TFBD.Data.

Les scripts

Ce qui était le nec en matière de désassemblage. En fait, un mini-langage interprété qui permet à l'utilisateur de générer ses propres algorithmes de désassemblage.

Un fichier script doit être de type texte (TXT ou SRC), et peut être écrit avec l'éditeur de Merlin. Il se charge comme suit:

SLOAD FileName ou SLOAD :Path:FileName

Si aucun nom de fichier n'est spécifié, le nom par défaut est: 1:ScriptFile.S (ou le dernier fichier script chargé). Ce nom par défaut est dans les resources de TFBD et peut être modifié.

Un appel de script s'effectue de la manière suivante:

\ ScrName addr ou \ * addr

Si le caractère * est utilisé à la place du nom du script, une petite fenêtre s'ouvre et on choisit son script dans la liste (pratique). Si aucune adresse n'est spécifiée, l'adresse par defaut envoyée au script sera celle de la première ligne de code de la fenêtre courante.

\ + Control

Taper ^-<CR> au moment d'éxécuter un script permet de tracer celui-ci pas à pas en visualisant l'état de ses variables. Assez utile pour debugger les scripts. Si on choisit son script dans la liste, il faut taper ^-<CR> après le “\ * Adr” et non en le choisissant. En cours de traçage, n'importe quelle touche (sauf shift, control, capslock, option, pomme, reset et ^C) éxécute la ligne courante et passe à la ligne suivante. ^C stoppe l'éxécution du script. D'ailleurs, même si l'on est pas en train de tracer, on peut stopper l'éxécution du script par le même biais.

La programation des scripts s'effectue à peu près de la même manière que ceux d'Orca/Disasm™, et ceux-ci peuvent être assez aisement transférés vers TFBD.

Plus de précisions dans une prochaine doc...

SEDIT

(En cours de developpement)
Un éditeur plein-écran permettant d'écrire ses scripts “à la volée”. Très pratique, bien qu'encore pas très rapide et ne permettant pas de sauver le nouveau fichier scripts. L'éditeur est du même type que celui de Merlin, mais assez diminué.

SSAVE [Pathname]

Sauvegarde d'un fichier script. Si aucun nom de fichier n'est specifé, c'est le dernier nom passé a SLOAD ou SSAVE qui est utilisé. Le fichier est sauvé au format TXT ($04) en ASCII inferieur. Merlin™ ayant une gestion assez particuliere des espaces, il est nécessaire de taper “FIXS” dans le command-line editor pour remettre les tabulations en place. Le suffixe “.S” n'est pas rajouté d'office.

Les fichiers d'extension

Un nouveau pas vers la toute-puissance... Les fichiers d'extention de TFBD sont des fichiers de type TLK ($BC), placés dans le directory 1:Expand. Ils permettent un désassemblage automatique beaucoup plus puissant et rapide que les scripts, qui sont plutôt faits pour les structures simples de petite taille. Ils sont prévus pour pouvoir être programmés par n'importe qui en fonction de ses besoins, et disposent à cet effet d'un point d'entrée dans TFBD permettant d'appeler les différentes fonctions de désassemblages à la manière des toolsets (macros & supermacros comprises !) ainsi que d'information générales sur le fichier ou segment en cours de désassemblage (type/auxtype, infos OMF, segment header...).

Un fichier d'extention s'appelle tout simplement en tapant son nom, ou bien via la commande “-” suivi du nom. Exemple: “-TOOLS” et “TOOLS” appellent toutes deux le fichier d'extention TOOLS. La commande “-” n'est vraiment utile que dans la mesure où un fichier d'extention peut porter un nom identique à une commande interne ou à une structure (cf section “les structures”, plus loin).

Si pour une raison quelconque on veut stopper l'éxécution d'une extention, on tape l'ordre d'arrêt pomme-point (.); TFBD demande alors si l'on veut vraiment stopper l'éxécution; taper Y pour confirmer.

Attention: l'ordre d'arrêt ne fonctionne que dans la mesure où l'extention appelle TFBD; si par exemple elle tourne en boucle infinie en elle-même, on ne peut pas l'arrêter comme ça.

Pour plus d'informations sur la structure et la programmation des fichiers d'extentions, lire la documentation qui leur est consacrée (Expand.Doc, en théorie).

Les fichiers d'extention actuellement livrés avec TFBD sont:

- DOS (Désassemblage des tables de paramètres GSOS)
- Tools (Désassemblage des tables de paramètres ToolBox)
- FTypes (Fichiers de types du Finder)
- Express (Segments ExpressLoad)

Les structures

Une autre carte dans la famille «Gagnons des heures». Les structures sont des assemblages simples de constantes, généralement de tailles fixes. Dans la mesure ou un programme contient assez souvent des structures répétées fastidieuses à désassembler manuellement et qu'un script ne convient pas toujours, ce niveau intermediaire entre les commandes de base et les scripts a été rajoute.

La définition d'une structure est relativement simple: une liste de constantes entre crochets suivie d'un nom. Exemple: [DW ADRL] OS, qui definit la structure “OS” comme etant un mot 16 bits suivi d'une adresse 32 bits.

Pour placer une structure à un endroit précis du code, c'est encore plus simple: il suffit de taper le nom de la structure suivi de l'adresse ou on la place plus optionellement d'une virgule et du nombre de fois qu'on la répète. Exemple: OS $2000, qui placera un DW en $2000 et un ADRL en $2002. En fait la syntaxe est rigoureusement la même que la définiton d'une constante toute bête.

Autre exemple: si on a en $1234 une table de 8 adresses avec un caractère ascii entre chaque adresse, on définit la structure “chrad” par: [ASC(1) DA] chrad, puis on fait: chrad $1234,8.

Comme l'illustre l'exemple précédent, on spécifie le nombre d'octets pris par une constante à longueur indéfinie par un mot entre parenthèses. Par exemple HEX(20) indique une table de 32 octets hexas. Spécifier la longueur d'une constante telle que DA ou STR n'a aucun effet. La longueur par défaut est de 1 octet.

Autre détail: pour placer à la suite une quirielle de constantes identiques, on tape son nom puis *nbrdefoikonlaplace. Ex: [HEX(8)*5 DB*2].

SKP(n) dans une structure ‘saute’ n octets. C'est une pseudo-constante qui représente en fait une absence de constantes.

Pour l'instant, le processus de définition de structures n'est pas recursif, c-à-d qu'on ne peut pas placer de structures dans une structure. Mais ce sera fait dès que j'en aurai besoin.

END OF DOC FEROX,
Clermont-Fd, 12/11/92

==========
==========

Updates

==========
==========

Cette partie de la doc n'est utile que si vous avez déjà reçu une version de TFBD. Elle contient les updates successifs depuis la version 1.0b, ce qui est assez ancien...

Trucs en plus... - v1.0b2

Ajouts:

- Commande RLOAD rType,rID,Filename
- Pages d'aide (?)
- Commandes Control-.
- Commande BUG [addr][/n]
- Commande DOS

Modifs/Améliorations:

- LOAD: Pomme-Ouverte
- Génération de constantes en série

Améliorations - v1.0b3

Debugging:

- Relocation des Supers ($F7-01)
- Coupures de constantes
- Défilement du source
- Gestion correcte des 64Ko
- Génération des labels via offsets
- Commandes de contrôle en absence de fichier objet
- TC: offsets reconnus corrects

Ajouts:

- Désassemblages des “imbedded chains” de GS-Bug
- Fichiers d'extention
- Constante REV
- ^S: Désassemblage en minuscules

- POP: Remonte dans le prefix
- DSK: Desktop
- INFOS: Infos segment
- SEDIT: Editeur de script
- SSAVE: Sauvegarde des scripts
- MREL: Relocs multiples
- HIST: Historique des commandes

Modifs/Améliorations:

- LOAD: Gestion partielle des fichiers OBJ ($B1)
- RLOAD/SEG: Gestion des resources multi-segment
- DOS: Commentaires
- SEG: Liste de selection

- Commandes clavier case-insensitives
- Erreurs GS/OS courantes en toutes lettres

La saga continue - v1.0b4

Ajouts:

- Resources rVersion, rComment et Preferences
- Pseudo-constantes CS & C1
- Structures

Modifs/Améliorations:

- Lectures clavier par _GetNextEvent
- DCI, STR & STRL génères en serie.
- DS sur une zone <> 0

- CFG: préférences

Assymptote - v1.0b5

Debugging:

J'ai encore réussi a dénicher des petits bugs du genre aléatoire et bien planqué. Ecrasés, les vilains cafards...

Les routines de chargement de segments ont été un peu débuguées également, pour que les segments aient leur taille réelle, meme si l'en-tête de segment ne la précise pas (ça arrive, pour les segment “Library dictionary” notament).

Modifs/Améliorations:

- SRC: source partiel + EXTs
- DOS: passage en fichier d'extention (cf ci-dessous)
- RLOAD: changement de syntaxe - liste
- CAT: tailles resource fork
- LOAD: seg 2 si ExpressLoad

- Traitement des csts par bloc (interne)
- Chargement OMF v1

Par suite d'une petite limitation de Merlin™ (ce crétin crèe des fichiers LNK de plus de 64Ko qu'il est incapable de linker), la commande DOS est passée en fichier d'extention pour alléger le segment principal. Ce n'est ni moins rapide ni moins performant pour autant et ça permet d'avoir un sample d'extention supplémentaire. Pas négligeable.

Ajouts:

- Constante FLO (Extended IEEE)
- Directive ORG des fichiers non relogeables (cf LOAD)
- Sauvegarde QuitCDA (cf ci-dessous)
- Fonctions fichiers d'extentions (cf ci-dessous)
- Resources des paths par defaut (cf ci-dessous)

- MD: Création d'un directory
- FIND: Recherche explicite
- SCAN: Recherche de types de data

En cas de plantage, si on purge TFBD avec QuitCDA, un fichier “TMP.BAK” est crée dans le directory courant (0:), contenant une sauvegarde du désassemblage en cours. Cette sauvegarde n'est pas créée si on appelle QuitCDA avec la touche Pomme enfoncée.

Les pathnames par défaut de TFBD sont maintenant en resource; ce sont les rWString ($8022) d'IDs $00001001 a $00001006. Elles contiennent les chemins d'accès aux fichiers config et aux extensions, le nom du fichier d'equates, des fichiers script et template par défaut, et le nom du fichier template de backup (cf QuitCDA ci-cessous).

Quelques appels ont été ajoutés aux fichiers d'extention. Voir la doc correspondante. De plus le “-” devant un nom de fichier d'extention n'est plus nécessaire. Taper son nom tout simplement suffit, à moins qu'il ne porte le même nom qu'une commande interne ou qu'une structure. En fait, maintenant, lorsqu'une commande est entrée, TFBD cherche une interprétation possible dans la chaine suivante:

Commandes internes
|
Structures
|
Extentions

First release - v1.0

At last, but not at least...

Une première “release” qui se sera fait attendre près de seize mois, quand même... Mais vous savez ce que c'est, on voit un machin ou deux à rajouter, tiens je pourrais mettre un truc là, cette petite commande en plus ça serait pas mal... Et puis ces satanées options qu'on a la flemme de coder mais qui sont malgré tout indispensables à un désassembleur digne de ce nom.
Mais bon, il faut bien s'y mettre un jour, et voila, ce jour est venu.

Par une froide journée de Décembre, Ferox & le Phœnix corp. mettent une fois de plus à votre disposition un outil puissant et efficace pour passer agréablement ces inévitables nuits ou l'on arrive à allumer que les Apple II...

==========

Ferox, Clermont-Fd, Decembre 1993.

==========