TFB-Disasm
The Flaming Bird Disassembler

HISTORY

Date	Version	Description
08/92	1.0b	First Draft
11/92	1.0b2	Tricks and more...
12/92	1.0b3	Improvements...
02/93	1.0b4	The legend continues
10/93	1.0b5	Asymptote
12/93	1.0	First release

Intro

Yeah... Here is the very first mega-production of the PHC completely under GS/OS. Yes yes, it's indeed a S16 file. To tell the truth, we love the good old DOS 3.3, but we judged that it would not necessarily adapt completely to this production. So here we are, we are left with an app like Apple loves, except that we are on Apple II..

TFBD, as its name well indicates, is a disassembler. (But note: a Merlin™ disassembler. Something rare). It therefore comes to be added to two others commonly used: Orca™ and Sourceror. That makes three, and it was indispensable for those who wanted Merlin™ sources with the power of Orca™ and the user-friendliness of text mode. In fact, originally, I had begun coding this thing to dismantle a file of which no disassembler could correctly bring out: *Start.GS.OS*. Try, and you will quickly understand why. But now due to adding options, we've improved a program which was originally intended to be another Sourcerer. This leaves us with a professional product on our hands, which forces us to write a doc.

This is that doc…

Considerations

"The Flaming Bird Disassembler" is shareware.

Or almost. You are free to copy it and to transfer it, but only in its whole and complete original version. You are also free to use it as much as you want, and during an unlimited period. You decide the value of this software and send me what you like: a check, one of your commercial softwares, or even simply a letter to say to me that you find this app great, even absolutely nothing if you find nothing, or that your account is has zero, or that it would prevent you from sleeping to send 100 balles/bucks/euros to have saved some thousand hours of programming.

I will answer as many letters as is possible, classifying them in an order of priority dependent on their contents. The last one also determines the privileges of each towards the future, maybe commercial versions.

General Info

Memory: TFBD runs without any problems with 1.2MB of RAM under system v6.0 and beyond. The maximum RAM used by TFBD in the course of disassembly is at least of 300KB by counting the app itself, the code to be dismantled and the relative data. Loading or editing a script file uses an additional 64KB. The space taken by the templates depends on their number but very rarely exceeds 60KB. All in all, to use TFBD with maximum capabilities, it is useful to have 400 to 500KB free in the machine. Considering the fact that it uses no Toolset "Ram", it is doable even with 1.2MB.

Speed: Beyond a certain stage of the disassembly (if the number of constants or labels becomes rather substantial), TFBD can undergo severe slow-down; so, an accelerator card would be welcome if you dismantle a file of some hundreds of KB.

Resources: TFBD keeps some of its data (preferences, pathnames) in its resource fork, but can work without it. If the resource fork is absent (for example if it was destroyed by a copy utility ignorant of resources), TFBD will take its internal default options, but they will no longer be modifiable.

Syntax: TFBD only accepts parameters of commands as hexadecimal numbers. An address can (or not) be preceded by $, but this is not useful, except in the case where this one begins with a letter (ex: FF69) but not to be confused with a label. A parameter does not represent an address if it is not preceded by $.

Bugs: Not being every day a programmer is absolutely perfect and exempt from inattention, this version of TFBD may still harbor some bugs which will have escaped my attention. In the course of disassembly, make frequent saves of templates so as to avoid losing everything in case of definitive crash of the app (I cannot recommend enough that you have QuitCDA of Phœnix Corp., loaded in memory, which solves this small problem of the saving with TFBD).

If you find a bug (even minor), do not hesitate to take a pen and write to me, indeed by indicating the circumstances of the bug, the manner to reproduce it, on which file it occurred (send it to me if need be), etc.... Same thing if you have ideas, software or suggestions. Everything can be interesting.

My address:
Philippe Savitch
19 rue de la Duée
75020 Paris - France

Instructions

When you enter TFBD, all that you can see is a small line of text at the top, and a cursor below, which flashes nonchalantly. This simply means it's waiting on a command on your part. The screen is cleared in the middle, it, just waiting to be filled, and we suspect with what. In the absence of a loaded file, the possible commands are the following:

?	Help pages
$	Shareware Page
PFX [Path]	Selection of prefix - by default (0:)
CAT [Path]	Catalog of the directory "Path" Or 0: if not specified
POP	Go back up in the prefix
MD name	Create a directory
LOAD filename	Load a file object
RLOAD file[,t,id]	Load a resource
CFG name	Load a config
QUIT	Quit TFB-Disasm
ROM	Disassembly of ROMs
SLOAD [pathname]	Load of a script file
SEDIT	Edit scripts
SSAVE [pathname]	Save the scripts
DSK	Access to the desktop
HIST	History

Description

(for SLOAD, SEDIT, SSAVE: See section "The scripts" farther down)

?

'?' Display the help pages of TFB-Disasm, which is nothing other than the relatively exhaustive list of the possible commands. In view of little room available on a screen and in a segment of code, they are not very detailed; it is more of a reminder than anything else.

$

$ Display the shareware pages of TFB-Disasm. It is always good to have read them at least once, what…

PFX [path]

Allows, in the same way as its homonym that we always encounter, to change the default prefix (Zero and only zero at the moment). If the directory is not specified, it allows you to choose in an “ergonomic” way, by moving around with the arrow keys.

Left Arrow:	Comes down a level
Right Arrow:	Goes up of a level
Bottom Arrow:	Goes to following level
Top Arrow:	Goes to preceding level
<ESC>	Cancel
<CR>	Accept

CAT [path]

List the files of the current directory or the directory "path".
Formatting of the listing is a little bit particular in the sense that the sizes in bytes and blocks and data forks and resources are separated
(yes, I have cruelly sacrificed the creation date of the file for that, while waiting for 160 column text screens).

POP

Make it go back up one level from the current prefix(0:). If you're already at the root directory, the command doesn't have any effect.

MD filename

Make the sub-directory "0:Filename".
There's not much more to say.

LOAD [path:] filename

Load a file into the disassembler

In the case of a non-relocatable file (SYS, BIN, or even a data file), code object is disassembled by default from the address zero (0) unless it is specified by the type (for example $2000 for SYS, the AuxType for BIN), in bank zero.

In the case of a Load File (Types B1 to BE), the first segment is loaded and disassembled starting at 01/0000. The OMF information of the segment is loaded too and TFBD uses it for the disassembly. If the file begins with an ExpressLoad segment, it is the second segment which is loaded at first and disassembled in 02/0000 (it is always possible to load the ExpressLoad segment, but it is not of any big interest).

The command key (Apple) held down during the load command changes the status of the file between Load File and Data File. That can be useful to load certain Load Files saved under a special type (it exists, for example, in filetype OS, $F9) or on the contrary to load a Load File all as one block, a segment header and OMF information included (attention to the 64KB limit).

RLOAD [Filename [rType, rID]]

Load a resource file into the disassembler

Filename Is optional to the extent that we want to load a resource of the file already in the disassembler. If we specify nothing, or Filename but not rType, a selection window opens, allowing you to choose the resource in a list.

Lines on the list indicate: if the resource needs a converter ("c"), its type (HEX and Name), and its ID. We can load any type of resource, code or data. TFBD is based on the fact

that resource needs a converter to know if it is code or not. In case it makes a mistake about the given type, do the same as we did with LOAD: Open-Apple at the time of the selection changes the status of what is loaded with between OMF and Data.

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

Attention: TFBD is not DeRez and cannot do what DeRez does (that is not its purpose). For TFBD, every resource Is treated as a separate file, independently of other resources, and independently of the data fork. So, every resource has ITS file template, and in passing of one resource to another or of a data fork to a resource fork, you should not forget to save it.

CFG [/S or / D] CfgName

Save, delete or load a configuration. TFBD Config files are in the directory 1:Configs and are of the type $5A/$8040. They contain the current state of TFBD: the current prefix, the pathname of the file currently being disassembled as well as that of its templates and of the loaded script file, the current segment (including resource), the current screen position and the various flags, notably the state of ^C, ^R and ^S (see the control commands, farther down).

Save the current config:	CFG /S MyConfig
Load a config:	CFG MyConfig
Erase a config:	CFG /D MyConfig

The CFG command also serves to load and to save the resource defining preferences of TFBD (rType=1 and rID=1) namely the states of ^R, ^C, ^S and tabulations. By default, ^R is active (relocs reverse in the hex dump), ^C and ^S are inactive (constants are not inverted and mnemonics in capital letters). These preferences are automatically loaded at startup.

Save the preferences:	CFG /s
Load the preferences:	CFG

QUIT or BYE

Quit TFBD and return to the launcher. I do not think that there is much to say about the above.

ROM

Disassemble ROM. At the moment, ROM only allows you to disassemble ROM 01.

DSK

A small command added after the repeated pleas of Bandit II who insisted on his little NDAs. But let us not dream: the desktop version of TFBD is not just around the corner; a disassembler is exactly the kind of software that "Human Interface" makes perfectly unusable... A faster equivalent of DSK is ^* (see Control Commands). Normal access to the desktop is made in 640 mode; if we want to reach it in 320 mode, it is necessary to press the apple key at the time of the call (DSK + apple-return or apple-ctrl-*).

HIST

History of the commands. Show the last 18 entered commands.

Now that a magnificent object code is loaded, (which we are eager to dig out its internals) we have access to all of the TFBD commands. Here they are, described by section.

Control Commands

The Control Commands (control plus another key) are accessible at any time, even while typing a command line. Here they are:

^S: Disassemble opcodes in uppercase/lowercase (small)

^H: Switch from opcode display to hexadecimal display.
In hex mode:
^R: Display in inverse video the relocation zones (OMF or REL). This allows you to quickly locate the address tables in an OMF file, or see if we have forgotten about REL in a non-relocatable file.
^C: Display in inverse video the zone where constants are defined

^+: Push the current position of the screen to return there later.

^-: Pop (restore) the last position piled by **^+**.
You can push up to a maximum of 16 positions

^B: Jump to the beginning of the code

^N: Jump to the end of the code

^*: Access to the desktop (equal to DSK command)

Apple ^*: 320-mode desktop

Up Arrow: Go back through the command history
Down Arrow: Go forward through the command history
(50 commands memorized)

The commands change the state of the disassembler, ie ^H (hex dump), ^S (upper/lower), ^R (relocatable inverse) and ^C (constants inverse) in the absence of an object file are registered/ recorded and come into effect at the time of the disassembly.

General commands

LIST $Adr *or* **LIST** label

Disassemble the object code from $Adr or Label. By default, the address is automatically realigned over the beginning of the instruction or the constant if $Adr falls right in the middle. That allows you to work your way around. If on the other hand we voluntarily want to cut an instruction (it is rather frequent), it is necessary to add "*" behind the address. Ex: LIST 2543*

SEG [n]

Load then disassemble segment *n*. The address of the disassembly, by default, is $n/ 0000. (Obviously doesn't do anything for a file SYS or BIN)...

If the number of the segment does not appear, a selection list will appear containing the numbers and the names of the segments of the file. Yes, this command also works for multi-segment resources (it is moreover rather funny to see Orca/Disasm™ persist in dismantling only their ExpressLoad segment. Uhm no, I do not criticize).

SRC [range] File

Generate the source file "*File.S"* of the current segment. If it becomes too long, then it will generated subsequent source files, "*File.2.S*", "*File.3.S*", etc. If the segment uses equates or externals, the files "*File.E.S*" and "*File.X.S*" are created.

The optional parameter [range] serves to generate only partial source for the zone which it specifies. The syntax is a little bit particular:

[adr1.adr2] (brackets required): it includes adr1 to adr2
[adr1.] : from adr1 to the end of the code
[.adr2] : from the start of the code to adr2

All in all, SRC generates three types of files:

- File.S [File2.S, File3.S, ...] …….. Source of the segment
- File.E.S ..... Equates of TFBD.Data
- File.X.S ..... EXTs and user EQUs

PS: Sorry for the slowness of this function, but I still have not looked too much for optimizations. And we do not care. If you start a 50KB object code, go take a bath or something. The time is never lost.

INFOS

Display information on the segment in currently being disassembled. In fact, nothing more than the dump commented by the segment header, but it sometimes conceals interesting information.

Constants

TFBD manages 15 constants at the moment (one or two are missing, but they'll have to wait until I need them, a question of laziness):

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	Hexadecimal	-
DS	Define Space	-
ASC	ASCII	-
REV	ASCII Inverse	-
DCI	ASCII MSB-ended	-
STR	String	-
STRL	C1 String	-

CHK Checksum byte 1

The syntax used to place a constant at a location in the object code is as follows:

cst addr [,num] for a constant with a known fixed length

And:

cst addr [,len]

Or: *cst* addr [.addr2] for a constant with an indeterminate length

"cst" is the pseudo-opcode of the constant, e.g. DW, ADR, STR, etc...
"addr" is the location where we place it (hex address or label).
"num" is the number is which we put in the end of the line. Works for:
DB, DW, DA, DDB, ADR, ADRL, FLO, STR, STRL, DCI and CHK.
"len" is its length.
"addr2" is the stop point (included in the constant). Used for:
HEX, DS, ASC and REV.

If only the start address is indicated for a constant of indeterminate length, the default length will be 1 byte for DS and HEX, and up to the next zero for ASC (useful for CStrings). But note: if ASC does not encounter zeros, it will go to the end of the object code.

During the mass generation of DCIs, TFBD considers them all as the same type (that all the characters have the same MSB), which allows you to manage 1 character DCIs.

Given that I had no docs on SANE (Standard Apple Numerics Environment), the FLO constant cost me a wild disassembly of "run-time" of a compiler the name of which I shall keep silent about. The ASCII conversion routine tries to make intelligent choices, that is to say that tricks like "0.00000000000000000000e+0" are the types of measures to be avoided.

STR and STRL has to point, naturally, at the byte or the word length of the string, otherwise small surprises!

Pseudo-Constants

CS Addr[,num]

A variant of ASC to generate C-Strings. The advantage of this command is to be able to generate them in series. If *num* is not specified, CS only generates one (same as ASC).

C1 Addr[,num]

A variant of STRL, but for C1-Strings too long to hold on one line. C1 disassembles a DW for the length of the text then disassembles the text via ASC.

CS and C1 can be placed in structures, in the same way as all other constants. Ex: "[DW C1] c1out". (See Section "The structures", farther down).

Note: Internally, TFBD manages fixed length constants (DA,...) in a single block when they are generated in series with the parameter "*num*". It limits the space taken by its recording and accelerates the disassembly. So, when it is possible, always generate constants by big blocks.

Labels

Well yes, there are labels. That would be unfortunate for a disassembler to be without... I give you bulk list of commands:

LAB $Adr,Label
Associate the address $Adr with the label "*Label*".

LAB Label,Label2
Reappoint the label " Label1 " as "Label2".

LAB $Adr
Or: **LAB** Label
Erase the label defined as $Adr or the label "Label".

ENT $Adr,Label
Create an intersegment label

ENT label
Transform a normal label "Label" into label Entry.

EQU $Adr,Equate
Defines an equate (Equate = $Adr).

GENLAB
Generate labels from OMF information, Relocs (compared to farther), and the machine code. The generated labels are always aligned over the beginning of the instructions and the constants. If the code draws information from the *n*th byte of a constant, the operand will be the shape "Label+n". See the MATCH command for more information.

Labels are in the usual form; they begin with a letter or with "~" or “_” or “:” and can contain anything except “:” and digits.

Directives

Three directives are at present handled: MX, ORG and DBR.

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

Add to the source the directive to change the size of processor registers, then re-disassemble the object code from this address up to where the sizes re-correspond.

MX $Adr

Takes the directive; returns the size of registers in that position by based on their size ($Adr - 1).

ORG $Adr,$xxxxxx

Change, from the Adr, the assembly address of the object code.

ORG $Adr

Return to the main origin from Adr, or cancellation of the directive if a change of origin was defined in Adr.

DBR $Adr,xx (xx = Bank #)

Indicate to the disassembler that Data Bank Register (register B) is different than the current bank; it allows the labels generator (see GENLAB) to generate labels correctly.

DBR $Adr

Puts back the value of B to the K register or cancels the DBR $xx if it was previously defined to $Adr. The DBR directive is only shown in the disassembler; it doesn’t appear in the source file, this directive being of use only to the label generator.

Comments

COM addr,text of the comment
Place a comment at addr

COM addr
Remove the comment at addr

The comment does not have to be between quotation marks, and a space is automatically added after the semicolon (aestheticism, let us see...). We cannot put of comments of the beginning of line (*).

Relocations / Offsets

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

Add a relocation record at $Adr, defined by its size in bytes, which references $Ref±Disp with a gap of Shift bits. This command is generally useful for non-relocatable files, the relocations are defined in OMF information.

For example, we have:

```
Adr1        PEA $0000
Adr2        PEA 7DD1
```

Which in fact corresponds to “PushLong #Label”, with Label = $007DD1. So that the source generated if correct, it should be in the form:

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

To make this happen, it is necessary to indicate to TFBD (which cannot infer it!) that $0000 is the high part of Label and $7DD1 the low part:

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

The 10 (hex!) indicates that the address is shifted by 16 bits towards the right, and that one is left with its high part. A negative number such as -8, for example, would indicate a shift of 8 bits towards the right and an 8, a shift towards the left.
But for this particular example, it is common enough to have a special command: command(order): PHL $Adr1, will fetch both parts of the address and make the relocation.

Now imagine the following case:

```
Adr     ADRL $00007DD1
```

In this case, it is enough to type:

```
REL $Adr
```

Then the line becomes:

```
Adr     ADRL Label
```

The routine was to seek itself the reference address ($7DD1) and the size (that of the constant, here 4). The offset and displacement set by default to 0.

And also works in this case:

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

REL $Adr … is equivalent to … REL $Adr+1,2,0,$007DD1

Thus: Adr LDX #Label

The REL command allows something else amusing, evoked higher - the displacements. If we have:

```
Adr     ADRL $00007DD1
```

Type:

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

Will give:

```
Adr     ADRL $00007DD2-1
```

Or: Adr ADRL Label-1 (if Label = 007DD2)

All the displacements are possible between +-80 and 7F.

There are also very useful variants of the PHL command. To begin, it does not verify the presence of the PEA, that is that it also works for:

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

The RPHL command takes care of the case where the low word comes first:

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

The PHL2 command allows same thing in the case whete the two parts of an address are distant:

```
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
```

(The first address given to PHL2 must point to the high word of the address referenced.)
Just like that, you can pretty much generate quickly and correctly the exact labels of non-relocatable applications. Over this:

OFF $Adr

Consider the word stored at $Adr as an offset of its position. Example:

```
        Adr     DA $0068

OFF $Adr
        Adr     DA Label-*
With Label = Adr + $68
```

Very practical for disassembling ExpressLoad segments, which is perfectly useless…

MREL [±Disp,] Adr,n [,Ref]

MREL can generate series of relocation records. That saves a few hours in the address tables. MREL references *n* addresses from Adr, undergoing a displacement of *Disp* (Optional). *Ref* is used only to pass to MREL the upper part of the reference if it is not specified in object code (for example a table of DA which reference another bank of memory). For a table of ADR or ADRL, the parameter *Ref* Is ignored. The specification of a displacement must begins with + or -.

Example:

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

MREL -1,Adr,3,020000

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

Powerful, no?

Fixes

Various commands allow you to correct imperfection of the disassembly without having to fiddle for hours. The first one is simple, even obvious:

REM C|L|D,addr1.addr2
or **REM** C|L|D,addr,len
or **REM** C|L|D,addr

Remove Constants/Labels/Directives between two addresses.
Example:

REM CL,$7DD1,4

Remove the definitions of constants and labels of $7DD1 to $7DD4. If the end address or the length are not specified, the objects will be removed only at that address. If nothing is specified, objects are removed from the entire segment.

MATCH C|L|D

Align the directives on the starts of instructions and eliminates non-aligned labels and constants, and that's on the whole file.

Example:

```
        LDA #^Addr
        STA Label2
        LDA *Addr
        STA Label
        RTS
Label   DS 2
Label2 DS 2
```

Label and Label2 being members of the same pointer, we type:

```
        DS Label,4
And:    LDA #^Addr
        STA Label2
        LDA #Addr
        STA Label
```

```
        RTS
Label   DS 4
```

And we notice with horror that (certain times, but it is the case in our example) Label2 is always defined and used instead of Label+2.

Well MATCH L will correct this by eliminating Label2 and by giving Label 4-byte scope instead of the 2 previously.

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

An impeccable source, and without scrubbing.

It is recommended to use the MATCH L command before any generation of labels and MATCH CL before a TSAVE (although it is not essential in this last case). Generally speaking, when you see something that seems abnormal in the disassembly at the level of the arrangement of labels for example, try the MATCH command, then if that does not really work, see the command TC (correction of templates, further down).

Attention: for the commands REM and the MATCH, comments and relocations are treated as directives.

BUG [$Adr][/n]

Search the disassembly defect. If no parameter is specified, BUG looks for all the defects in all the file. Otherwise, it looks from $Adr for defects of type *n*, and then starts back at the beginning of the file for the type n+1, till the end.

n=0: BRK. Look for breaks, which generally should not be found in code (or rarely).

n=1: Branches never taken. Look for the sequences of the SEC/BCC,CLV/BVS, etc. Can be very useful in certain cases. If one disassembles the ROM, for example: often, the branch operand corresponds to the inverse location of the condition. (CLC, etc). Ex: CLC, BCS *+$38: 38=SEC.

n=2: Bad branches. Detect branches on operands or on constants. Often corresponds to incorrectly disassembled zones (forgotten or spare constants).

When a bug is detected, its description displays on the line at the bottom; at this moment, we can take back control with Return, or continue the search with Space. To type "BUG" again without parameters will resume the search where it was interrupted. Type "BUG addr" will resume the search at addr with the same *n*.

Templates

A TFBD template file is a file of type $5E, auxtype $8002 which contains all the recordings carried out on an object code (constants, directives, labels, comments, relocations, offsets, in short everything) and allows the saving of the disassembly and the resuming at the point when it's allowed. Here are the commands:

TLOAD FileName.T *or* **TLOAD**

Load the file FileName.T or the last file template loaded or saved if no name is specified. The ".T" is optional, but it is better to put it there at the end.

TSAVE FileName.T *or* **TSAVE**

Exactly the same thing as TLOAD, but to save.

TCLR

Erases all records in memory (same as REM CLD, but for the entire file, not just the current segment)

TC

Templates Correction. Eliminate the defects of the recordings present in RAM. This command was especially helpful to me when debugging TFBD, but it can also be used to recover the largest possible number of records in a template file that has been damaged. It eliminates records that correspond to nothing, or that refer to addresses outside of the object code. It also reactivates records that were "virtually" eliminated from the disassembly through successive revisions of the same zone.

Queries

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

This command works like almost all the "Finds" that can be found everywhere, except for one or two specificities. When it has found an occurrence the string, it lets you know by listing the relevant location (that it places around the middle of the screen) whose exact address appears below. At this moment, to strike <ESC> or <CR> returns control, any other key continues the search. To resume the search where it was interrupted or repeat the same search, type FIND without any parameters.

range the space of search. If it is not specified, it is the whole of object. Its syntax is the same that for SRC:

[adr1.adr2] (brackets required): it includes adr1 to adr2
[adr1.] : from adr1 to the end of the code
[.adr2] : from the start of the code to adr2

hexes is any string of hexadecimal words from 1 to 8 digits.
Any word longer than 8 digits is truncated to its long low word.

"text" is an ASCII string which is case-insensitive (eg "Z" = "z") and also hi-bit-Insensitive ("Z" = 'Z'). All the not special characters are allowed, although "?" has rather special function.

>adr represents a jump instruction or a long branch to adr (which can be a label):
JMP, JSR, JSL, BRL or JMPL adr.

>>adr represents all jump or branch instructions, short or long to adr
(conditional branches etc).

^adr represents a reference to adr, whatever kind (operand, constant, etc.) and regardless of its size and offset (shift count).

:cst is a supposed constant at this point, which can be there, or not, at the time of the search. If the string entered contains constants, FIND suggests putting them in place where it finds a string which seems to stick.

? is a “wild Nibble” in a hex number, or a “wild char” in text. It cannot be used in a reference (^, >, >>).

Examples of valid strings:

```
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 looks for the string in the object code and not in the equivalent source. So, fantasies like “FIND "LDA #$0005"” do not work.

Remark 1: The calculation of a reference in the code takes a lot of time. Searches like “FIND ^Adr” can sometimes (often) be very slow. Therefore, when we know a little more

precisely what we are looking for, it is better to let it know. If we look for a branch, the “FIND >adr”, of a less general nature, will already be much more. And if we know that we want a JSR, then “FIND 20 ^adr” is quasi-instantaneous.

Furthermore, in most cases, a reference corresponds to the exact value in the code. When it is possible, replace “^adr” with “adr”, quite silly. That accelerates it.

Remark 2: FIND sometimes poorly estimates the real size of a reference, as for example in the case where one reference with 3 bytes is divided in two Relocs of 16 and 8 successive bits in the OMF: FIND will see both references separately; that can pose problems at the level of the placement of constants. Another reason to designate when possible (and sufficient) a reference by its exact value…

Remark 3: The syntax of constant names is the same as for the structures (the routine was lying there...), ie - we may well get things like:

FIND [things] :DW*2 [things] :HEX(10)*3 ...

But don’t abuse it, all the same. A constant can also appear in first position.

SCAN [range] [kinds]

This allows you to search a particular type of data in all or part of the object code.

range The same syntax as for FIND
If not specified, search the entire segment

kinds Characters representing the type of data to search for
If not specified, search for all types
Currently:
A: tables of addresses
S: p-strings

When SCAN has found an area corresponding to one of the search types, it lets you know in the dump area whose address appears at the bottom of the screen (in the dumping, start of the area is in the 3rd line, just to be able to see what is going on a little above and below). If the interpretation of the area is correct (advice: always verify!), type Y to set up constants. <ESC> and <CR> returns control, any other key continues the search.

Remark: SCAN S does not look for p-strings that are too short (less than 5 characters) and does not accept control characters CR, LF & BELL. We are more or less forced to put such limitations, otherwise the routine would see strings everywhere which are everything except strings (can we really permit it to consider a zero as an empty string?...).

Ammenities

The commands which follow are more commands of comfort which serve in rather particular cases. I placed them to have needed it once or twice, and they stayed in case you need them too...

STOOL addr,tsnum

Disassemble a header of a system toolset. Implements the ADRLs table with their displacement of -1, and puts the labels corresponding to the function names. Names are taken from the file TFBD.Data.

addr Points to the beginning of the header (number of functions).
tsnum is the number of the Toolset.

MLABS

In the course of dismantling of ROMs, sets up the labels of the monitor, taken from TFBD.Data.

The Scripts

That which is the ultimate in disassembly. In fact, an interpreted mini-language which allows the user to generate his own algorithms for disassembly.

A script file must be text (TXT or SRC) and can be written with the Merlin editor. It loads as follows:

SLOAD FileName or **SLOAD** :Path:FileName

If no name of file is specified, the name by default is: 1:ScriptFile.S (or the last script file loaded). This default name is in the resources of TFBD and can be modified.

A script call is made in the following manner:

\ ScrName addr or \ * addr

If the character * is used in the place of the script name, a small window opens and we choose its script in the list (convenient). If no address is specified, the default address sent to the script will be that of the first line of code in the current window.

\ + Control

Type ^-<CR> at the time of executing a script allows to trace it step by step in visualizing the state of its variables. Rather useful for debugging the scripts. If we choose the script from the list, it is necessary to type ^-<CR> after the "\ * Adr" and not when choosing it. While tracing, any key (except shift, control, capslock, option, apple, reset and ^C) executes the current line and goes to the next line. ^C stops the execution of the script. Moreover, if it's not current tracing, we can stop execution of the script in the same way.

Programing the script is made in much the same manner as Orca/Disasm™, and these can be quite easily transferred to TFBD.

More details in an upcoming doc …

SEDIT

(In the course of development)
A full-screen editor to write scripts "on-the-fly". Very practical, although still not very fast and it doesn't allow you to save new script files. The editor is of the same type as that of the Merlin, but stripped down a bit.

SSAVE [Pathname]

Save a script file. If no filename is specified, it is the last name passed to SLOAD or SSAVE that is used. The file is saved in the format TXT ($04) in low ASCII. The Merlin™ software having a rather particular management of spaces, it is necessary to type in the "FIXS" command in the command-editor to fix the tabs. The suffix ".S" is not automatically added.

Extension Files

A new step towards omnipotence... The extension files of TFBD are of type TLK ($BC), placed in the directory 1:Expand. They allow an automatic disassembly much more powerful and faster than the scripts, which are rather made for the small-sized simple structures. They are designed to be programmed by anyone based on their needs, and for this purpose and have an entry point into TFBD for calling the different functions of disassembly in the manner of toolsets (macro and super-macro included!) as well as general information about the file or segment currently being disassembled (type/auxtype, infos OMF, segment header...).

An extension file is simply called by typing its name, either via the command "-" followed by the name. Example: "-TOOLS" and "TOOLS" both call the extension file TOOLS. The command "-" is really useful only as resolving the case where an extension file has a name identical to an internal command or structure (see section "the structures", farther down).

If for some reason you want to stop the execution of an extension, we type the stop command apple-period (.); TFBD then asks if we really want to stop the execution; type Y to confirm.

Attention: the stop command works only as far as the TFBD extension calls; if for example it enters an infinite loop itself, we cannot stop it like that.

For more information on the structure and the programming of the extension files, read the documentation which is dedicated to that (Expand.doc, in theory).

The extension files which come with TFBD are:

- DOS (Disassembly of GSOS parameters tables)
- Tools (Disassembly of ToolBox parameters tables)
- FTypes (Files of type Finder)
- Express (ExpressLoad Segments)

Structures

Another card in the deck of "Saviour of Hours". The structures are simple assemblies of constants, generally of fixed sizes. To the extent that a program quit often contains tedious repetitive structures to manually disassemble and a script is not always appropriate, this intermediary level between the basic commands and scripts has been added.

The definition of a structure is relatively simple: a list of constants between brackets followed by a name. Example: [DW ADRL] OS, which defines the structure “OS” as being a 16-bit word 16 bits followed by a 32-bit address.

To place a structure at a specific location of code, it is even simpler: it is enough to type the name of the structure followed by the address where we place it plus optionally a comma and a number of times to repeat it. Example: OS $2000, which will place a DW $2000 and an ADRL at $2002. In fact the syntax is exactly the same as the definition of a quite stupid constant.

Another example: if we have at $1234 a table of 8 addresses with an ascii character between every address, we define the structure “chrad” as: “[ASC(1) DA] chrad”, then we do: “chrad $1234,8”.

As illustrated in the previous example, we specify the number of bytes taken by a constant with indefinite length by a word in parentheses. For example HEX(20) indicates a table of 32 hex octets. Specifying the length of a constant such as DA or STR has no effect. The length by default is 1 byte.

Another detail: to place following of identical constants, we type its name then size and count in the following format: [HEX(8)*5 DB*2]. (Translator’s Note: the previous sentence seems to have errors in the source document and is likely not accurate or even close to reality.)

SKP(n) in a structure ‘jumps’ n bytes. It is a pseudo-constant which represents in fact an absence of constants.

At the moment, the process of definition of structures is not recursive, in that we cannot nest structures in a structure. But it will be made as soon as I shall need it.

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

Updates

This part of the doc is only useful if you have already received a version of TFBD. It contains the successive updates since version 1.0b, which is rather ancient...

Tricks and more - v1.0b2

Additions:

- Command RLOAD rType, rID, Filename
- Help Pages
- Command Control-.
- Command BUG [addr][/n]
- Command DOS

Modifications/Improvements:

- LOAD: Open-Apple
- Generation of mass constants

Improvements - v1.0b3

Debugging:

- Relocation of Supers $(F7-01)
- Cuts of constants
- Scrolling of Source
- Correct management of 64KB
- Generation of labels via offsets
- Command of control in absence of object file
- TC: correct recognized offsets

Additions:

- Disassembly of "Imbedded Strings" of GS-Bug

· Extension files
· Constant REV
· ^S: disassembly in lower-case letters
· POP: Go back up in the prefix
· DSK: Desktop
· INFOS: Infos segment
· SEDIT: Script editor
· SSAVE: Saving of the scripts
· MREL: Multiple Relocs
· HIST: History of commands

Modifications/Improvements:

· LOAD: Partial management of OBJ files ($B1)
· RLOAD/SEG: Management of multi-segment resources
· COM: Comments
· SEG: Selection List
· Case-Insensitive keyboard commands
· Errors GS/OS commonly printed out

The Legend Continues - v1.0b4

Additions:

· Resources rVersion, rComment and Preferences
· Pseudo-constants CS and C1
· Structures

Modifications/Improvements:

· Reading keyboard by _GetNextEvent
· DCI, STR and STRL generates in series.
· DS on an area <> 0
· CFG: Preferences

Asymptote - v1.0b5

Debugging:

I have still managed to find small, but well-hidden, bugs. Squashed, the naughty cockroaches...

Segment loading routines were also somewhat debugged, so that segments have their real size, even if the segment header does not specify it (that arrives, for segment " Library Dictionary" in particular).

Modifications/Improvements:

- SRC: Partial source + EXTs
- DOS: Passing of extension files (see below)
- RLOAD: Change of syntax - list
- CAT: Sizes Resource Fork
- LOAD: Seg 2 if ExpressLoad
- Treatment of csts by block (internal)
- Load OMF v1

As a result of a small limitation of Merlin™ (this moron creates LNK files of more than 64KB that it is unable to link), the DOS command is passed in extension files to ease the main segment. It is neither slower nor less effective so far and it allows a sample additional extension. Not unimportant.

Additions:

- Constant FLO (Extended IEEE)
- Directive ORG for non-relocatable (see LOAD)
- Saving QuitCDA (see below)
- Extension File Functions (see below)
- Default Path Resources (see below)
- MD: Creation of a directory
- FIND: Explicit search
- SCAN: Search for types of data

In case of crash, if we purge TFBD with QuitCDA, a file "TMP.BAK" is created in the current directory (0:), containing a save of the current disassembly. This save is not created if we call QuitCDA while pushing the Apple key.

Pathnames by default of TFBD are now a resource; They are rWString ($8022) of IDs $00001001 to $00001006. They contain the paths to the config files and extensions, the name of the equates file, script files and default template, and name of the backup template file (see QuitCDA below).

Some calls were added to the extension files. See the corresponding doc. Furthermore "-" in front of the name of an extension file is not necessary any more. To type its name simply is enough, unless it uses the same name as an internal command or structure. In fact, now, when a command is entered, TFBD looks for a possible interpretation in the following order:

Internal Commands
|
Structures
|
Extensions

First Release - v1.0

At last, but not at least…

The first "Release" which will have been waiting about sixteen months, all the same... But you know what that is, one sees a thing or two to add, wish I could put something there, plus this little command would not hurt… And then those pesky options we were too lazy to code, but are nonetheless essential to a disassembler worthy of the name.
But hey, you have to put it out one day, and here we are, this day has come.

On a cold day in December, Ferox & Phœnix corp. once again put at your disposal a powerful and effective tool to happily spend those inevitable nights where we manage to turn the Apple II…

Ferox, Clermont-Fd, December 1993.

(fan translation by: Dagen Brock, not associated with the authors.)