Infiltrator Disassembler Version 1.0 Manual

Table of Contents

List Of Abbreviations (uncompleted)

ASCII	American Standard Code for Information Interchange
BCS	Branch on Carryflag Set
BEQ	Branch on EQual
CIA	Complex Interface Adapter
CPU	Central Processing Unit
CSDb	The C-64 Scene Database
et seq.	and the following
etc.	and so on
IDE	Integrated Development Environment
IRQ	Interrupt Request
FLI	Flexible Line Interpretation
JSR	Jump to SubRoutine
KERNAL	Keyboard Entry Read, Network, And Link
MOS	Metal Oxide Semiconductor
NTSC	National Television Systems Committee
OP-Code	OPeration-Code
PAL	Phase Alternation Line
PEBKAC	Problem Exists Between Keyboard And Chair
PETSCII	Personal Electronic Transactor Standard Code of Information Interchange
RAD	Rapid Application Development
RAM	Random Access Memory
RGB	Red Green Blue
ROM	Read Only Memory
SID	Sound Interface Device
TSM	The Shaolin Monastery
UPX	Ultimate Packer for eXecuteables
VIC II	Video Interface Controller
VICE	Versatile Commodore Emulator

1 Introduction

Once upon a time everyone was eager to find sprites, bitmaps, music or code somewhere in the RAM. Yea, we used the “Action Replay” and other cheat technology to get what we wanted. Infiltrator comes with those basic functionalities including some hopefully nice updates.

The disassembler uses forward interpretation with all the implied problems of this method. The basic concept is to process the complete memory, that´s why you can import PRG files as well as VICE snapshot files. Using VICE snapshots will naturally result in a lot of false interpretations. The included set of tools and methods may help you to master them.

Please note that some support tools where made a long time ago, so they may not have all comfort you know from somewhere else(sprite animations, etc.). The primary purpose of these tools is to help identifying memory areas as graphics, code, etc.

For quick results on your side the manual refers to several programs from different cracking / demo groups using the VICE 2.3 version. Getting these releases in your hands is recommended. Download the latest VICE version here: http://vice-emu.sourceforge.net/

Requirements: You should have at least basic knowledge of all MOS Technology chips and a standard computer using the Microsoft Windows XP ServicePack 3 operation system. The software is not tested on any 64 bit operating system yet. You are welcome to try on Vista / Windows 7 and submit any results to me.

Porting requests: The application is programmed in the Lazarus IDE using standard components only. It should be possible to compile the code-lines on various platforms. If you like to volunteer for the job, I will be very pleased. However, give me some time to wait and react on major bugs reported by someone in the first place(plus clean up some source crap).

You are welcome to drop any comment or request to my CSDb mailbox. Search for user
RHX / Excess / Secret Lab Productions (SLP)

Cheers,
Gerald

1.1 Terms of Use

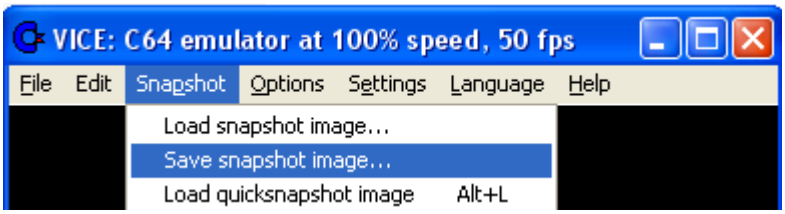
2 Getting Started (Showcase)

For the showcase I opted the “Judge Dredd” crack intro from “IKARI & TALENT + TSM”. Oh no, why this one...?! Because it is less complex which makes it perfect for a showcase. In addition, it features many visual standard techniques you meet in most programs. In case you did not have the release, grab it here: http://noname.c64.org/csdb/release/?id=17220.

Once you have loaded the crack using VICE 2.3, the emulator shows you this neat old school intro.

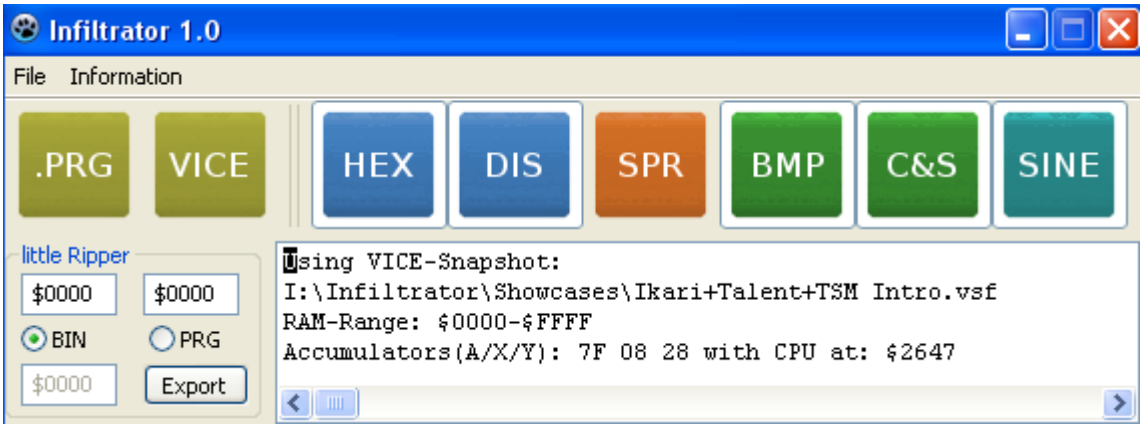
Save a VICE snapshot on your hard disk:

.



2.1 *Importing a File*

Simply choose the file type (VICE for the showcase) and select the snapshot you saved before. Depending on the import type, you may get some additional information.



3 Standard Support Tools

Before pushing any buttons, you should always do some preliminary considerations about any program you want to rape. In this case you can expect to find the following:

- the “TSM” sprites (having x and y movements)
- a bitmap font (charset) for the scroll text (“This Game Was ...”)
- the scroll text data
- the charset(s) of IKARI and TALENT logos (probably all in one)
- char data tables of the logos (displayed with some x-sine movement)
- the main screen where everything is displayed
- the music
- some sine for the movements

The main objective is to identify most of these program parts. Later, you can advice the disassembler to ignore parts of the memory, that will reduce errors and false friends.

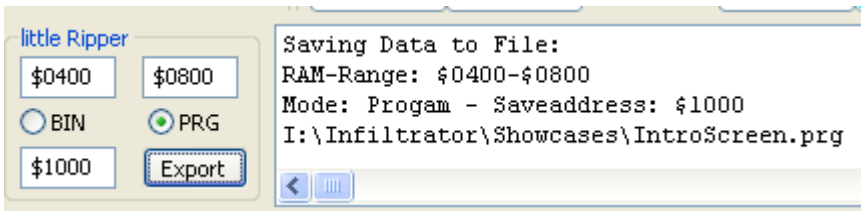
3.1 Unholy Buttons

This is for all people who don't care about disassembling, but just want to rip off something. You may already stumbled upon some of these buttons.

- Export as BMP
- Export (little Ripper)
- Make KOALA (see chapter 4)
- Play Data as Wave (see chapter 4)
- Save Data as Wave (see chapter 4)

Export as BMP: Saves the currently shown graphic in a bitmap file (RAD tools, I love you!). You may use Gimp & Adobe products plus Kickass for something bad.

Export (little Ripper): Saves the specified RAM range to a file. Depending on the chosen file type you must enter a valid save address, too. Here is an example how to save the screen data to a new address.



Tip: You can't re-import binary files since the memory allocation is missing. You can still use binary files in your Kickass code.

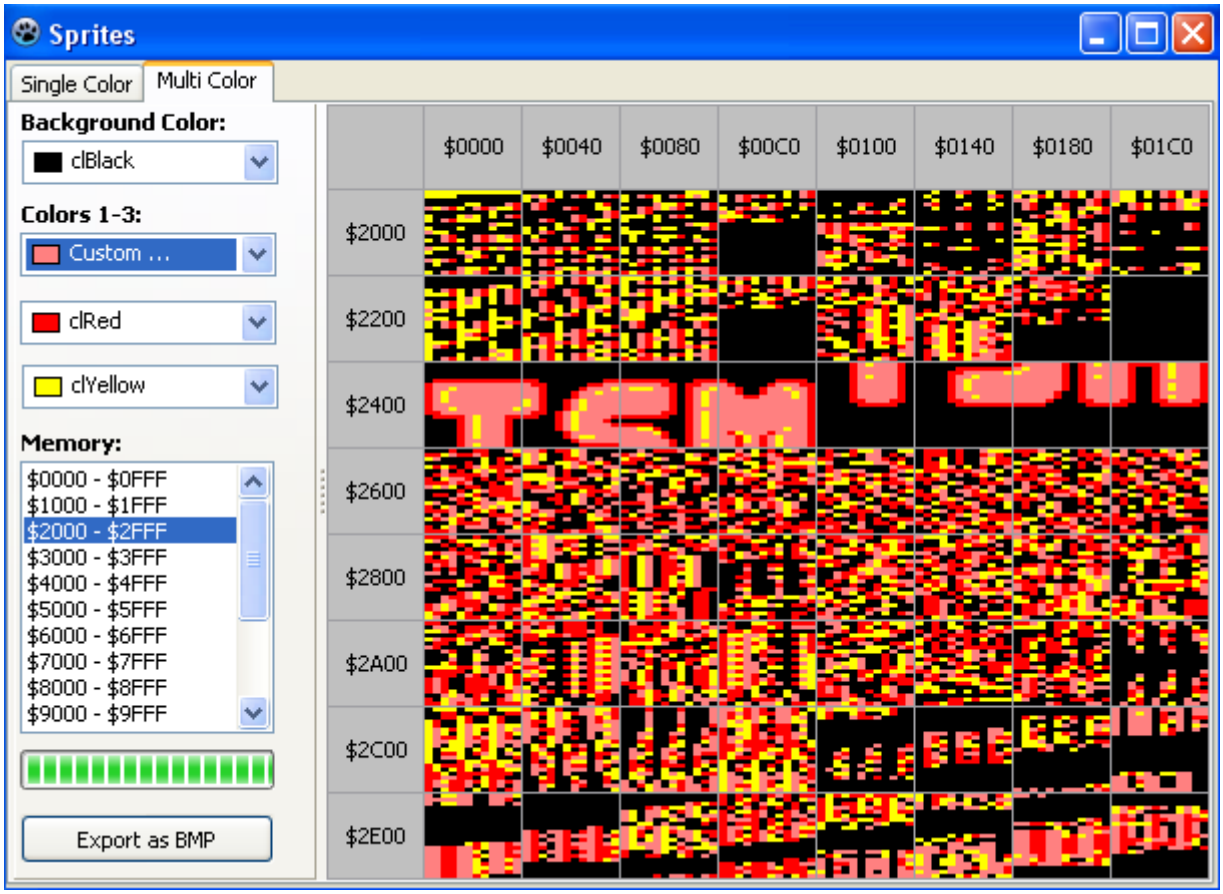
Tip: Make sure to write the correct suffix in all save dialogs (“.prg”, “.bin”, “.bmp”), I haven't implemented automatism routines for them.

3.2 Sprite Pad (SPR)

Select this button to get there:

Simply choose the memory area you want to display. The multicolored “TSM” sprites can be found at $2400 - $25FF and are build out of eight single sprites. You should remember this memory range for the disassembling showcase job(chapter 6).

Tip: The “TSM” sprites memory position(VIC Bank I) indicates that other graphics and the main screen can be found there($0000 - $3FFF).



3.3 Bitmaps (BMP)

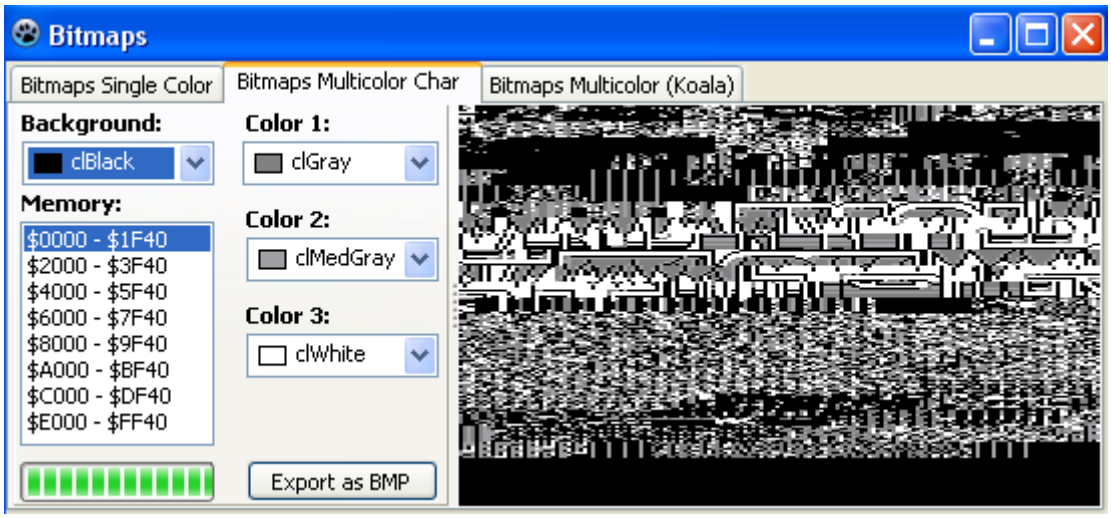
Select this button to get there:

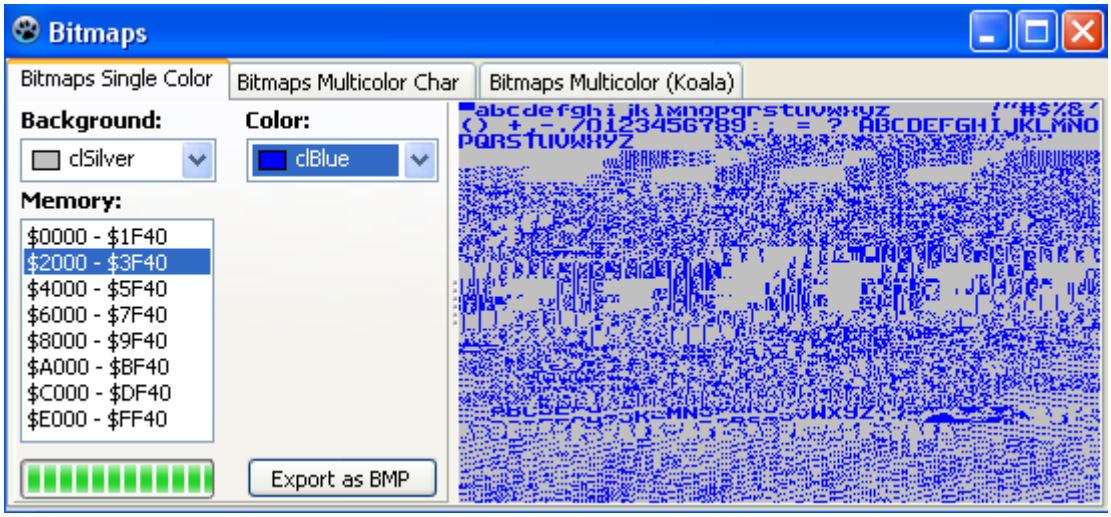
You know the selection procedure, so let's have the results only:

- char based graphic at $0800 et seq. (exact size unknown yet)
- font bitmap is located at $2000 et seq. (exact size unknown yet)

=> both has to be verified for the disassembling!

Tip: See chapter 4 for the “Bitmaps Multicolor (Koala)” tab.





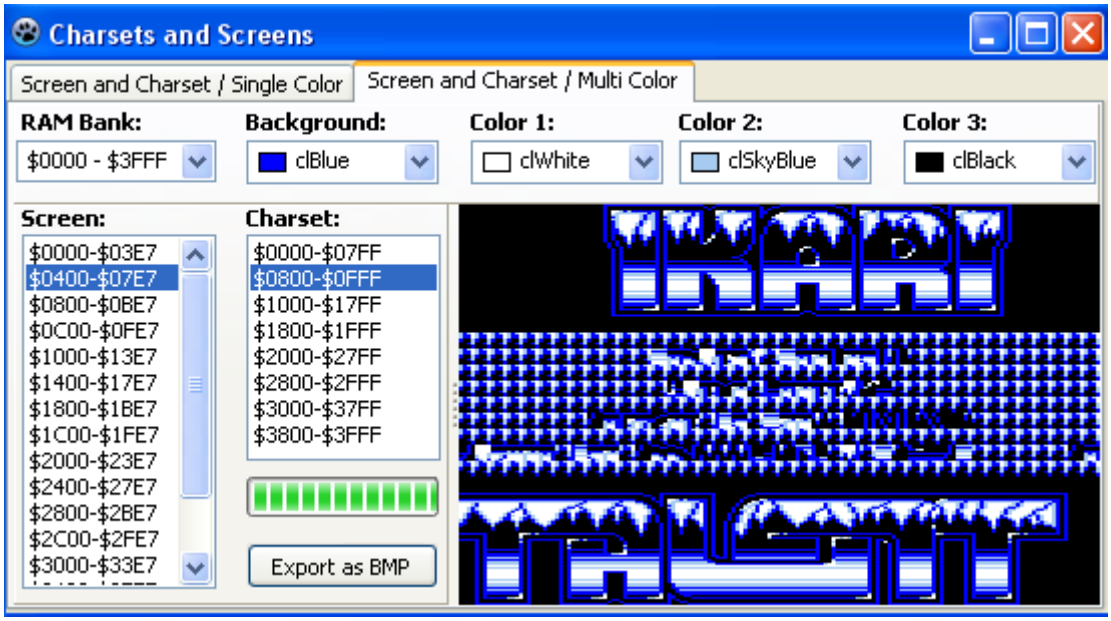
3.4 Charsets and Screens (C&S)

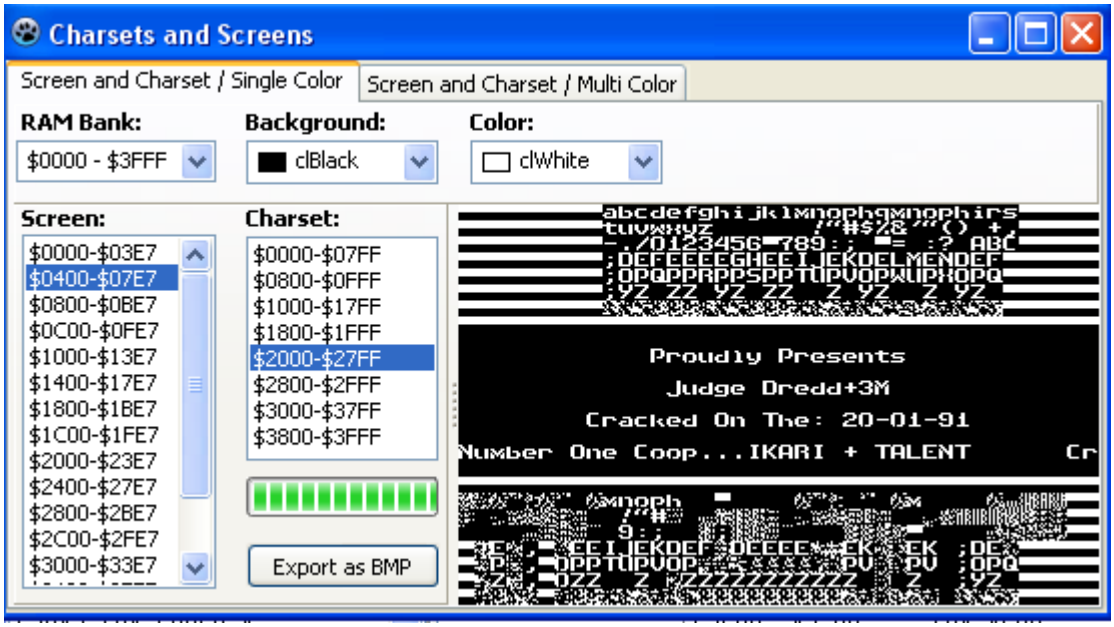
Select this button to get there:

Based on the findings made before it's obvious to search in VIC(RAM) Bank I ($0000 to $3FFF). Search for the screen using the identified charsets, results:

- standard screen is used at $0400
- IKARI and TALENT logos using the same charset

There's no wow function here, so let's continue.





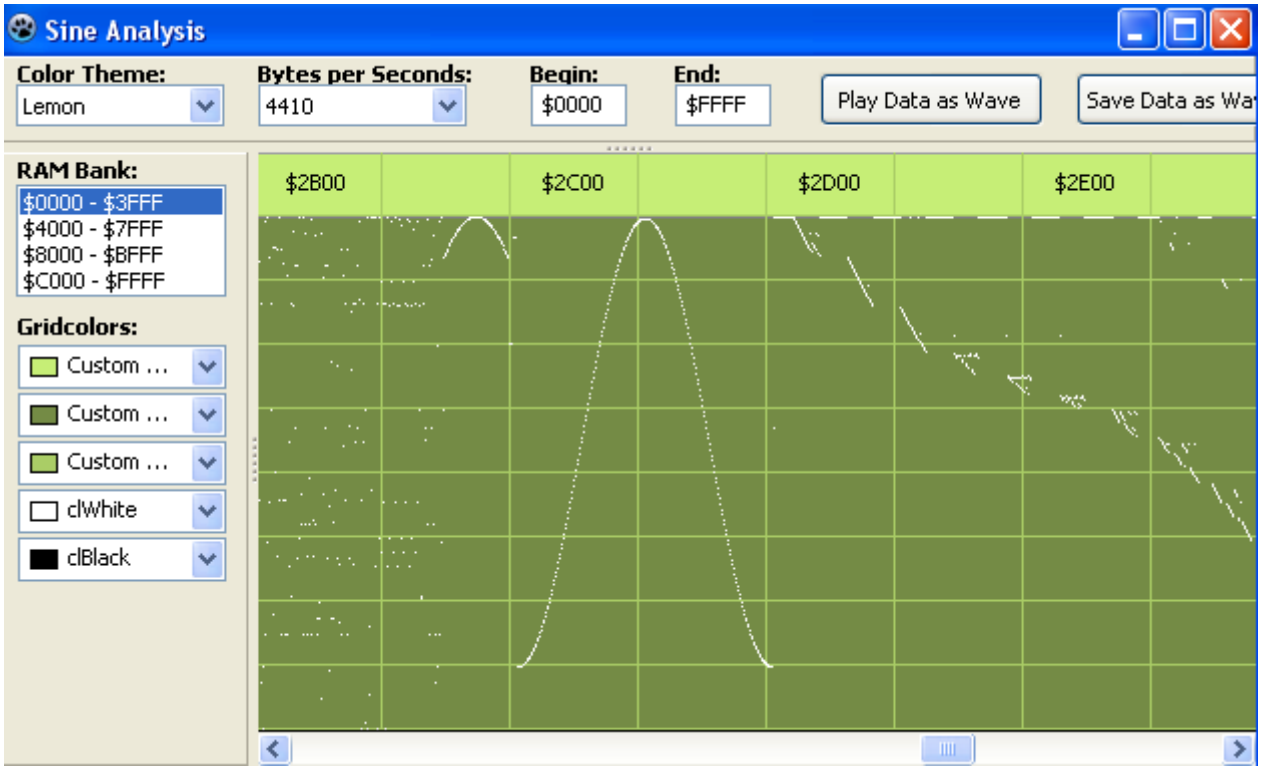
3.5 Sine Analysis (SINE)

Select this button to get there:

What's this? Well, it displays the RAM values one after the other as pixels in a bitmap. In other words, a neat thing to identify sine waves.

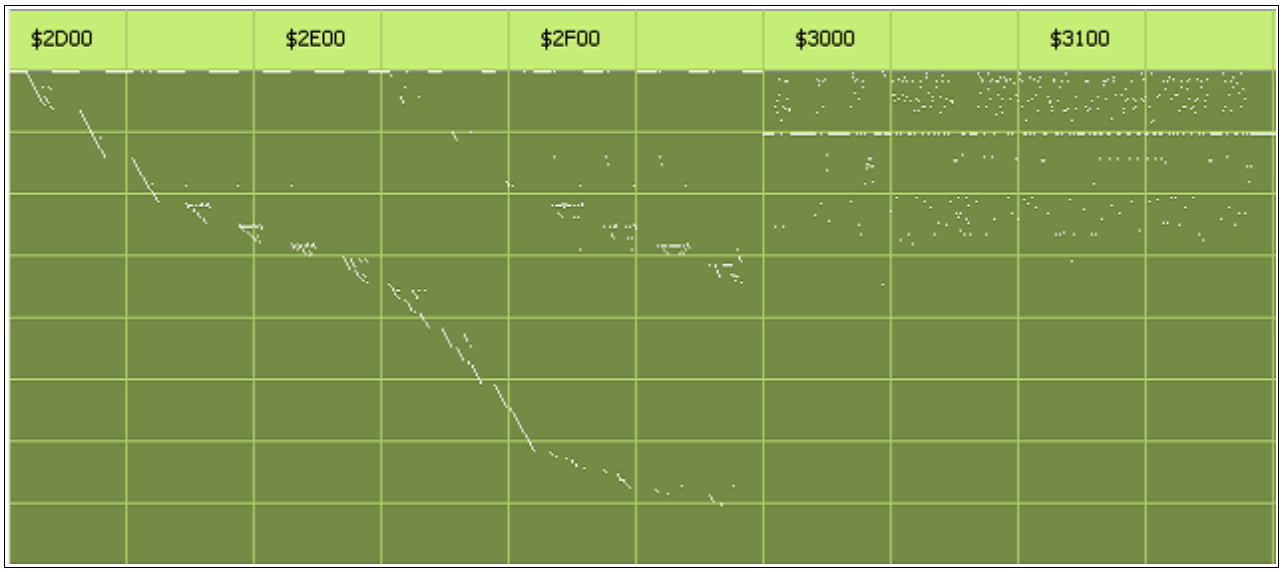
You can easily detect the sine waves used for the logos and sprites. Remember that the TSM sprites moving in x and y direction. Take a **close look**:

- the bouncing half sine uses approx $40 bytes and starts somewhere at $2BC0
- the full sine uses approx $100 bytes but **does not** start exactly at $2C00

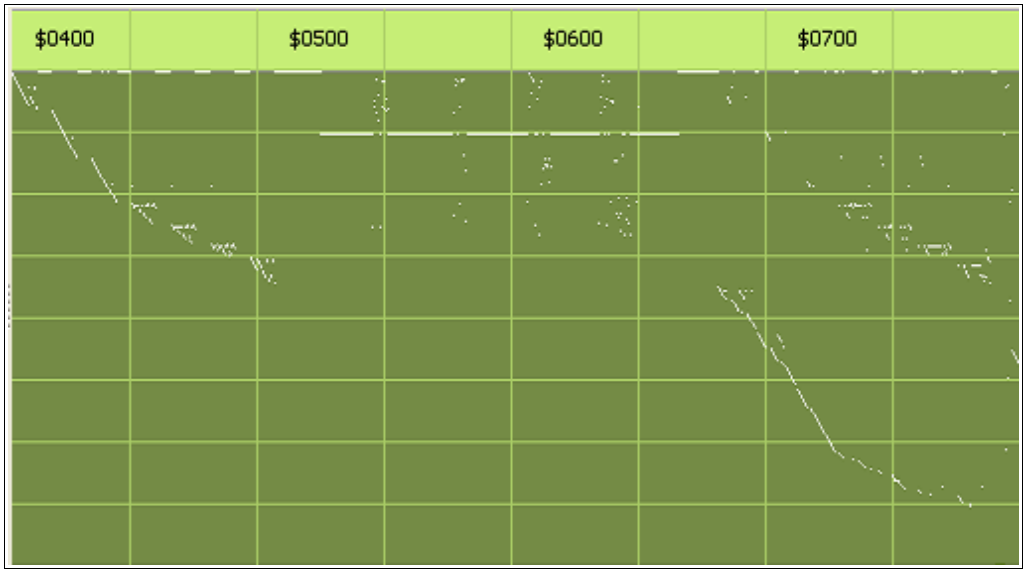


The showcase uses a lot of char based graphics, so let's have a look at it. The memory range from $2D00 to $2FFF shows stored char data tables that are used to build IKARI and TALENT logos. The fringed diagonal line indicates to an equal char tool. The standard screen from $0400 to $07E7 shows the logos too. But of course, only parts of them!

IKARI and TALENT logos and the scroll text, stored in the RAM:



IKARI and TALENT logos and some text, displayed on the screen:



Text & Scroll-Text:

A small standard font plus the upper letters “A..Z” are used, so you can expect data values from $01 to $5F. That's what is shown from $3000 to $3200 and from $0540 to $06C0. However, the most used char in any text is “Space”($20).

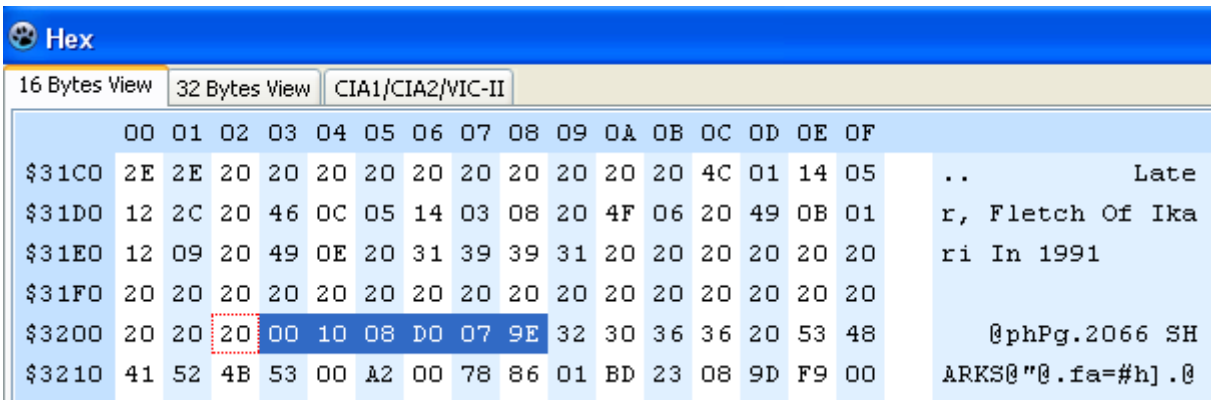
3.6 Hex Pad (HEX)

Select this button to get there:

A simple dump of the RAM including a small PETSCII to ASCII conversion. When using VICE as input file, you will have access to additional dumps of some MOS chips(CIAs and VIC-II). Since this is almost standard, anybody knows how it works.

Let's have some findings and updates for the showcase:

- charset from $0800 to $0FFF seem to include color tables at $0F10(logo color flasher?)
- music player plus music data is from $1000 to $1A7D
- some comments and blanks from $1A7E to $1FFF (can be ignored)
- scroll text use a charset from $2000 to $22FF (standard charset plus upper letters)
- bouncing sine from $2BBE to $2BFD or $2BFE
- full sine from $2C07 to $2D06 (=> $2C00 to $2C06 maybe pointers)
- RAM from $2D08 to $2FFF is stored char data for logo shakers
- some text at $3000 and scroll text from $3080 to $3203 (endbyte is $00, see picture)
- byte $9E at $3208 seems to be a basic SYS start command(see picture)



That's a all you need know for the showcase disassembling. The following chapter covers the Koala graphics and discuss some typical shapes that may come about in the Sine Analysis tool.

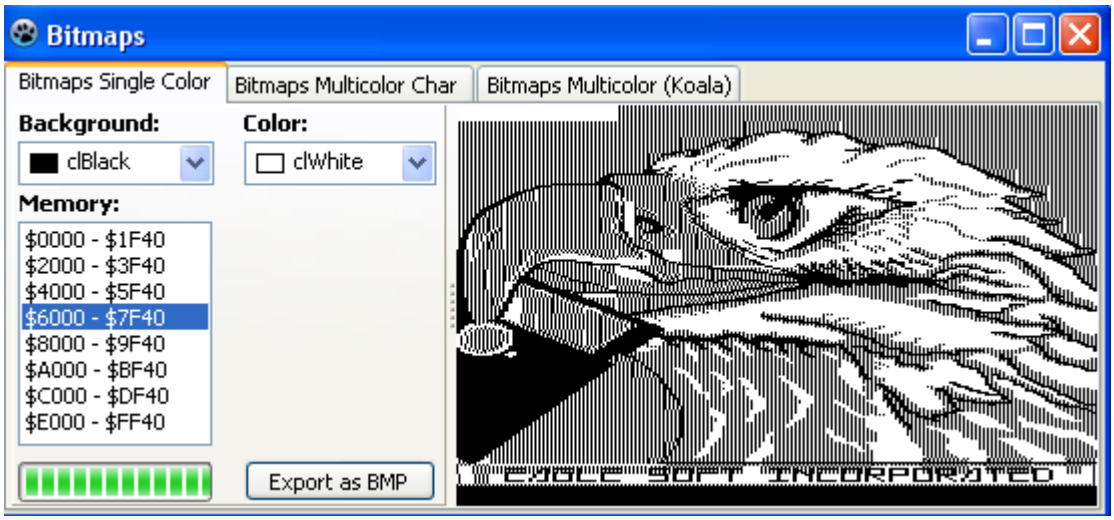
4 Lekker Bratwurst

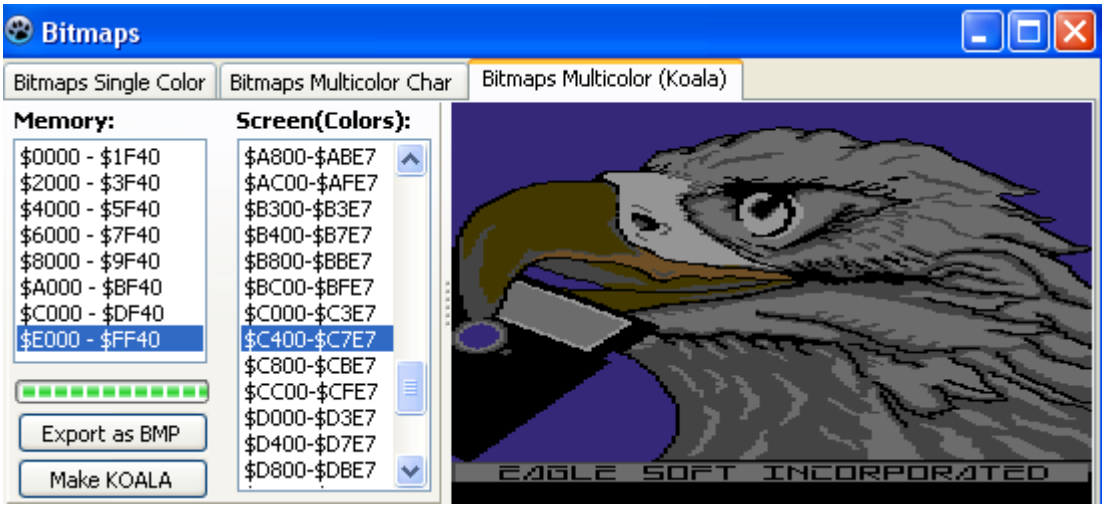
Guess what's my favourite food. Hmmm... this chapter dealing with stuff out of the showcase. Bratwurst rules – and Currwurst too.

4.1 Rip Off Koala

When using VICE you can snatch Koala graphics pretty easy. Do a quick scan on **all** possible memory locations to get the correct bitmap position, switch to Koala mode and search for the screen colors.

But please pay attention, you may come across false friends. Example: The well known “noble bird” is copied by the program for a “fade in” effect from $6000 in $E000. You will get stuck by searching for the color table in VIC Bank II only. (Oops... guess what happened to me.)





Because Koala uses screen data to define the pixel colors, an internal palette is used for the re-creation and display. An export to a bitmap file will use this palette too, so you can not define that on your own. Here are the RGB values in hexadecimal and integer - in case you need them:

- `$000000 // 0,0,0 (black)`

- $FFFFFF // 255,255,255 (white)
- $2B3768 // 104,55,43 (red)
- $B2A470 // 112,164,178 (cyan)
- $863D6F // 111,61,134 (purple)
- $438D58 // 88,141,67 (green)
- $792835 // 53,40,121 (blue)
- $6FC7B8 // 184,199,111 (yellow)
- $254F6F // 111,79,37 (orange)
- $003943 // 67,57,0 (brown)
- $59679A // 154,103,89 (light red)
- $444444 // 68,68,68 (dark grey)
- $6C6C6C // 108,108,108 (grey)
- $84D29A // 154,210,132 (light green)
- $B55E6C // 108,94,181 (light blue)
- $959595 // 149,149,149 (light grey)

The “Make KOALA” button streams the selected RAM to a Koala formatted program file, so you should enter the suffix “.prg”. You can't set the PETSCII code $C1, it won't be accepted as part of a Windows XP file name. Use the “DirMaster V2/Style” to do that.

In case you are looking for similar robbery routines... nope! Dozens of interlaced and FLI graphic formats cruising around, that's not Infiltrators' assignment yet. There is a nice tool named “Vice Snapshot Grabber 4.2” by Ian Coog/HVSC Crew dealing with this.

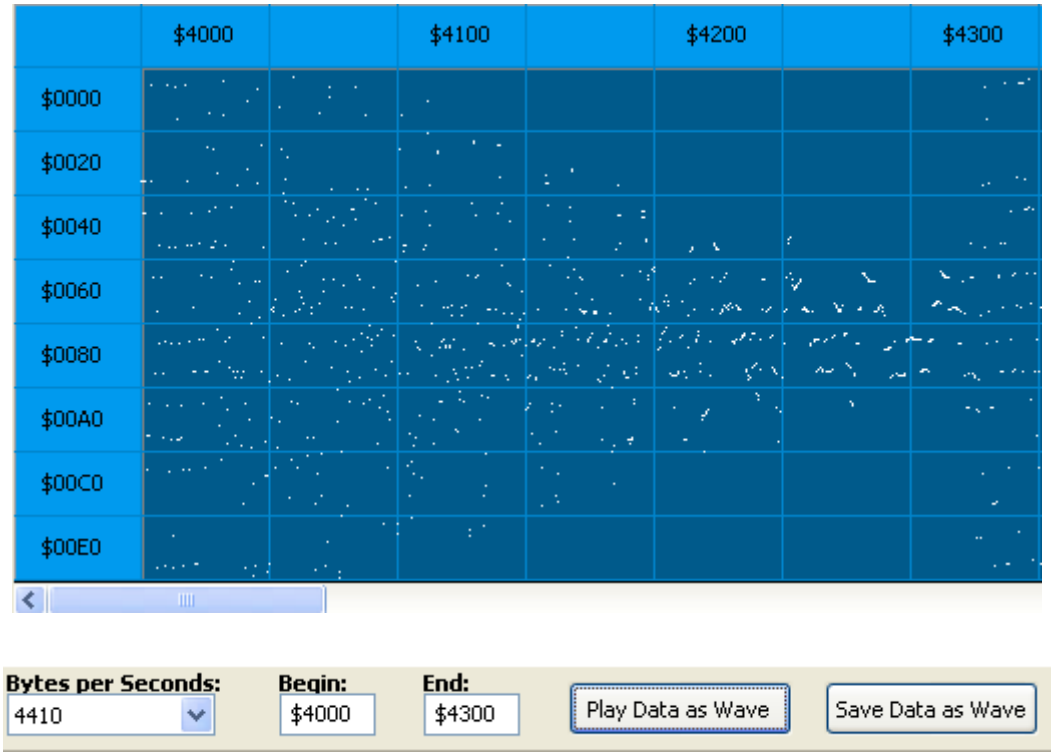
4.2 Sine Analysis (more typical shapes)

8-Bit Samples

Whenever a program uses 8-bit audio samples you get typical audio shapes. I made an internal "data to sound" converter, that's why you can listen to the samples. The little more weird ones can "listen" to graphic, code or whatever else they like. By the way, pay some attention on the BPS value you choose. Playing the complete memory using 441 BPS will take approx 148,6 seconds of unstoppable mythical sound(no thread programming, you know).

The save button creates a Wave formatted file that can be played with any external media player or editor. To make sure it will be saved as ".wav" that signature is always added - whatever you enter as file name.

Here is an example taken from a Megastyle Inc. product:



So, you are actually able to play Cycleburner sounds. Remember that it does not emulate technical effect programming on the SID chip, but just plays the original sample. This feature is rarely tested and I haven't implemented any variations(4-bit, 12-bit) yet.

Fun stuff: In case you got the HVSC collection, you can do this: Make a copy of "I-Ball.sid" (Rob Hubbard) and change the file suffix from ".sid" into ".prg". Import the PRG file. Due to it's signature "RSID...." it will be loaded from $5350 to ???? – but that doesn't really matter. Adjust the play beginning and have fun.

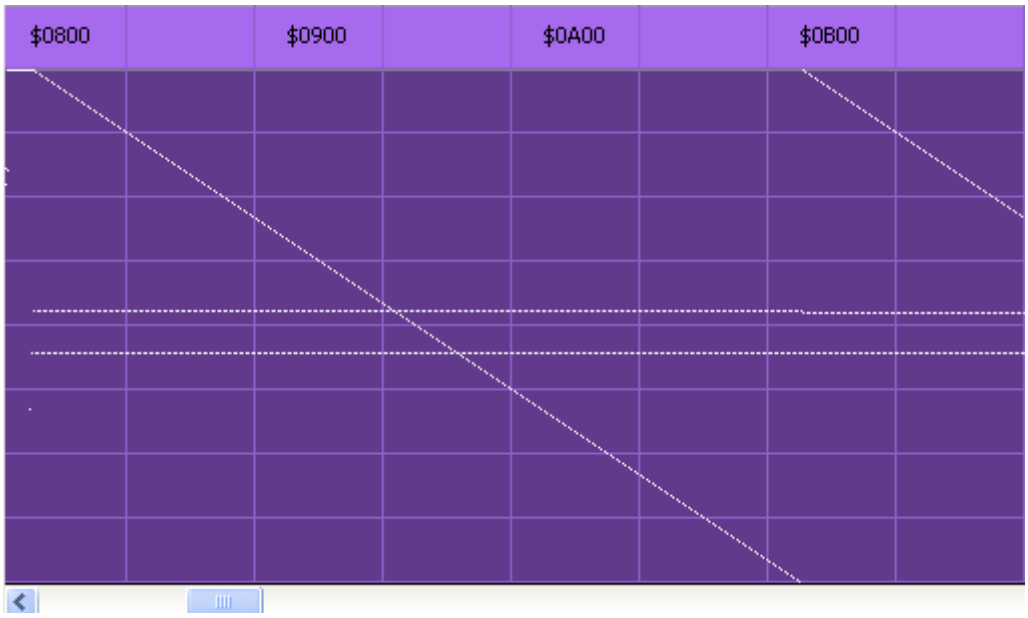
Speed Optimized Code

Code without loops always takes a lot of memory, but is often used in bottleneck situations.

Example 1: Let me start with a code snippet writing accumulator A into RAM $78XX.

Offset	OP-Code	Low Byte	High Byte	Interpreter
$0822	8D	00	78	STA $7800
$0825	8D	01	78	STA $7801
$0828	8D	02	78	STA $7802
$082B	8D	03	78	STA $7803
$082E	8D	04	78	STA $7804
$0831	8D	05	78	STA $7805
$0834	8D	06	78	STA $7806
$0837	8D	07	78	STA $7807
$083A	8D	08	78	STA $7808

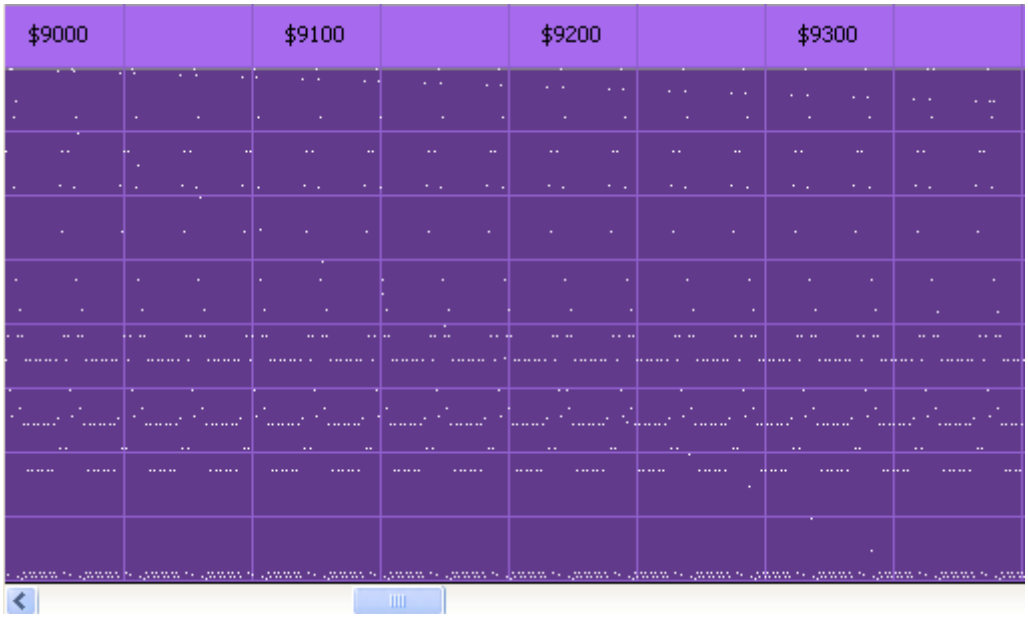
Because OP-Code $8D and High Byte $78 are used frequently, both are shown as horizontal, dotted lines. The diagonal line represents the increasing Low Byte.



Example 2: The next one shows OP-Codes $8C and $A0 wrapping up several VIC II registers at $D0XX. It's the holy “NU FLI” display routine, © by Crest!

So there's not necessarily a diagonal line, more important are horizontal lines at an OP-Code position.

Example 3: This one uses the indirect addressing mode “LDA ($FB),Y” and “STA ($FD),Y”.



4.3 CRAP

CRAP allows very little automatism scripting. It's in an early development stage. The following table shows the available commands.

Command	Parameter1	Parameter2	Parameter3	Parameter4
SAVE BINARY	$Startaddress	$Endaddress	#filename.bin	-
SAVE PROGRAM	$Startaddress	$Endaddress	$Targetaddress	#filename.prg
MOVE RAM	$Startaddress	$Endaddress	$Targetaddress	-
TRANSFER COLORRAM	$Startaddress	$Endaddress	$Targetaddress	-
TRANSFER CIA1	$Startaddress	$Endaddress	$Targetaddress	-
TRANSFER CIA2	$Startaddress	$Endaddress	$Targetaddress	-
TRANSFER VIC	$Startaddress	$Endaddress	$Targetaddress	-
FILL	$Startaddress	$Endaddress	$Fillbyte	-
INJECT	$Startaddress	$Amount of Bytes	$Bytes, Comma separated	-
UPDATE GRIDS	-	-	-	-

Because some commands modify the internal memory array, your last command should always be UPDATE GRIDS. In case you use the disassembler you should do a re-disassemble for a correct display.

The imported chip data is stored in several arrays, use TRANSFER to access them. The start address of TRANSFER commands should always be $0000, see examples. This makes only sense when you imported VICE files. Otherwise you get initial values.

For hexadecimal parameters always use **UPPER letters** and don't forget the '$'. The '#' is used as a parameter signal for file names, so enter it before you type the file name. There is some error handling implemented – but in an early stage. Guess why I called it CRAP.

Here is a senseless example script:

```
SAVE PROGRAM $1000 $1FFF $1000 #audio.prg
FILL $2000 $2FFF $00
TRANSFER COLORRAM $0000 $07E7 $2000
MOVE RAM $0400 $07E3 $2800
TRANSFER CIA1 $0000 $000F $2C00
TRANSFER CIA2 $0000 $000F $2C10
TRANSFER VIC $0000 $002F $2C20
INJECT $2D00 $0C $20,$44,$E5,$A9,$00,$8D,$20,$D0,$8D,$21,$D0,$60
SAVE BINARY $2000 $2DFF #screen_and_chip_rip.bin
UPDATE GRIDS
```

5 The Disassembler

Select this button to get there:

I do not want to bore you too much, but reading this is mandatory.

5.1 The Label Concept

Labels are generated for any direct JMP, JSR and the branches(BNE, BEQ, BCS, etc.). Every label includes the targeting and calling memory address.

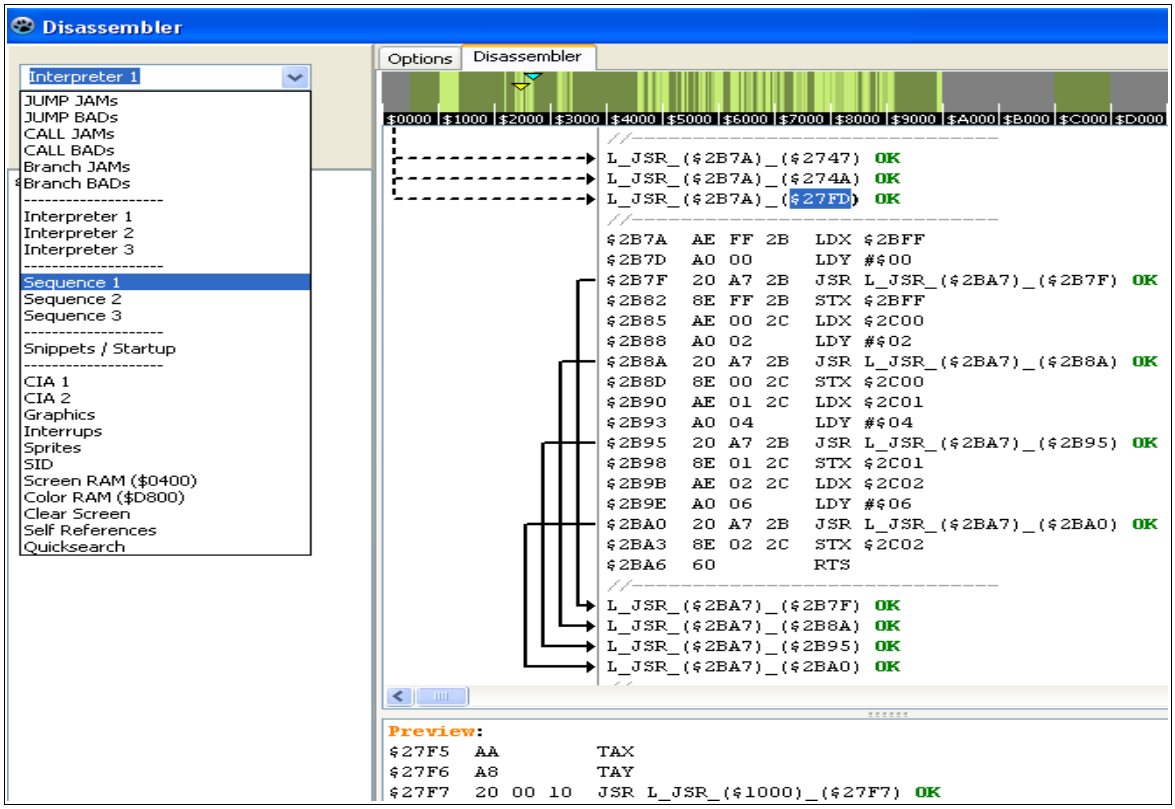
Label definition:

Label-Layout	Values
Signal	L (Label)
Interpreter	JSR, JMP or BRS(Branch)
Target($)	($XXXX)
Source($)	($XXXX)

Here is a code line example:

Offset	OP-Code	Low Byte	High Byte	Interpreter	Label	Rating
$274A	$20	$7A	$2B	JSR	L_JSR_($2B7A)_($274A)	OK

Here is a look-and-feel:



The good OK, the bad BAD, and the ugly JAM

You may have wondered about the “OK”s above. There is a quality check on every label's destination address. These four results are possible:

Rating	Meaning	Comment
OK	valid OP-Code found	be aware of false friends
BAD	valid OP-Code, but found inside another code line	evidence for crap or incorrect interpretation at target
JAM	invalid OP-Code found	evidence for crap at source and/or target
nothing	target is excluded	

When disassembling, you get BAD and JAM lists for your convenience. I prepared some extra functions to handle them, but first.....options!

5.2 Options and Searchfunctions

Options, quick wins:

Autoadd Comments: Will add clever(?) information to the code.

Colorizer: Gives you some color themes.

CONCAT Bad Bytes: Let them stick together, they may build a row.

Font-/Line Size: Adjust your glasses for free.

Ignore Trap Sequences: Prevents you from let you get thousands of crap results.

Insert Beauty Blanklines: Inserts an extra line after every end of IRQ, RTS, etc.

Progress: Shows the progress of the disassembling.

Start: Start of the program.

Options, Explained Later:

Autobusy: Re-disassemble immediately.

Snatch Snippets: Give me some code fragments.

The Exclude Disassembly Memory List:

Memory ranges to be ignored by the disassembler. Range overlapping is allowed and will not have any effect. Range autosort is activated. You need to re-disassemble after changes were made.

Add to List: Adds an entry to the list.

Delete: Deletes a selected entry, use right mouse button for a pop-up menu.

Save: Save you work to a file(exl = Exclude List).

Load: Import your work done before.

Reset: Give me the initial values.

Linear Interpreter Scan:

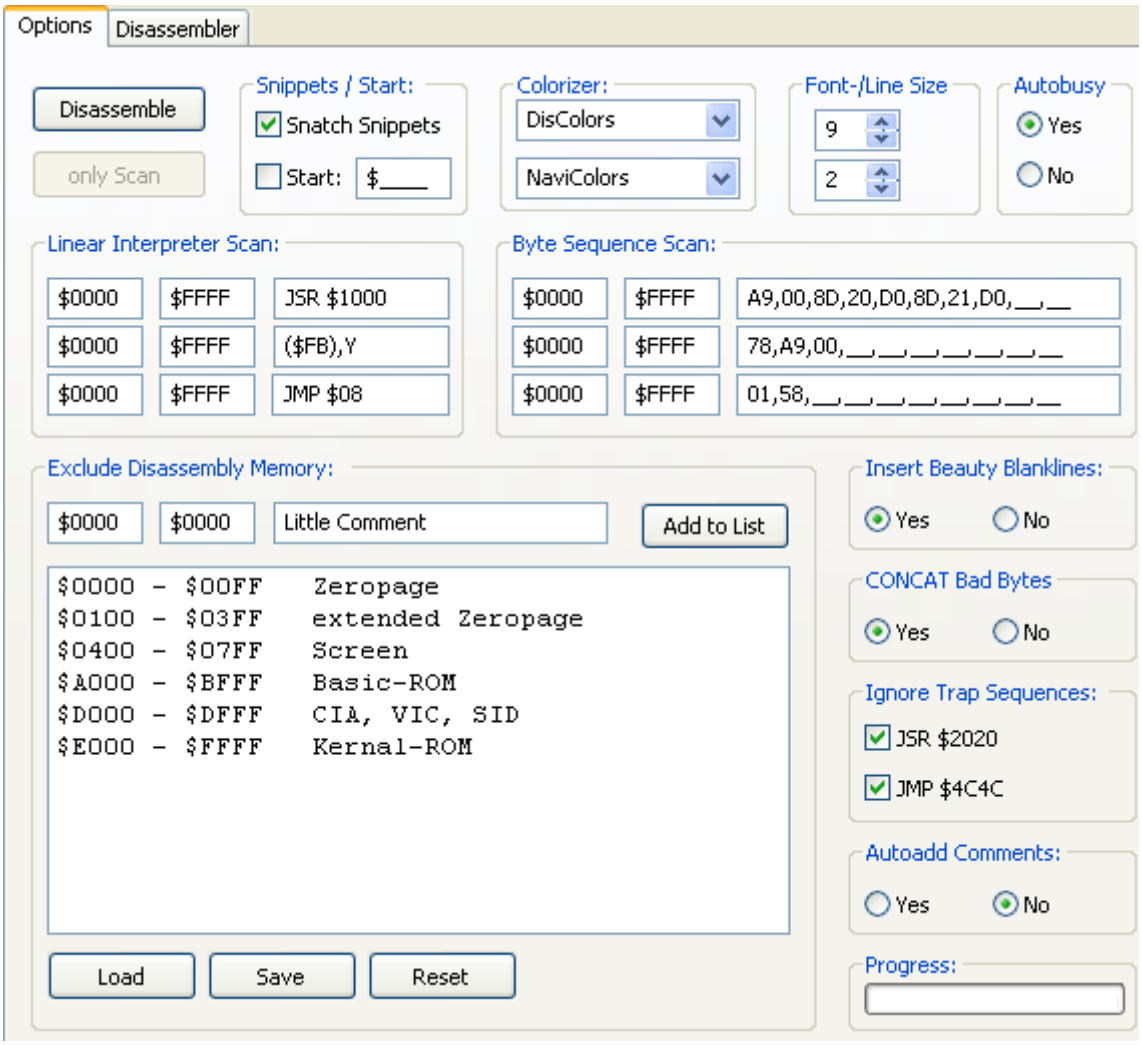
Performs a search on the interpreter terms. You can enter a phrase partly, an example: enter "**),Y**" for all interpreter codes using the indirect Y-indexing addressing mode. Keep in mind that any excluded memory will have an impact on the results.

Byte Sequence Scan:

Performs an old school search that is not affected by any excluded memory.

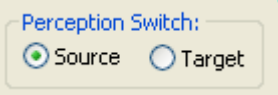
„only Scan" Button:

Performs the searches explained above, but don't disassemble.



5.3 Searchlists

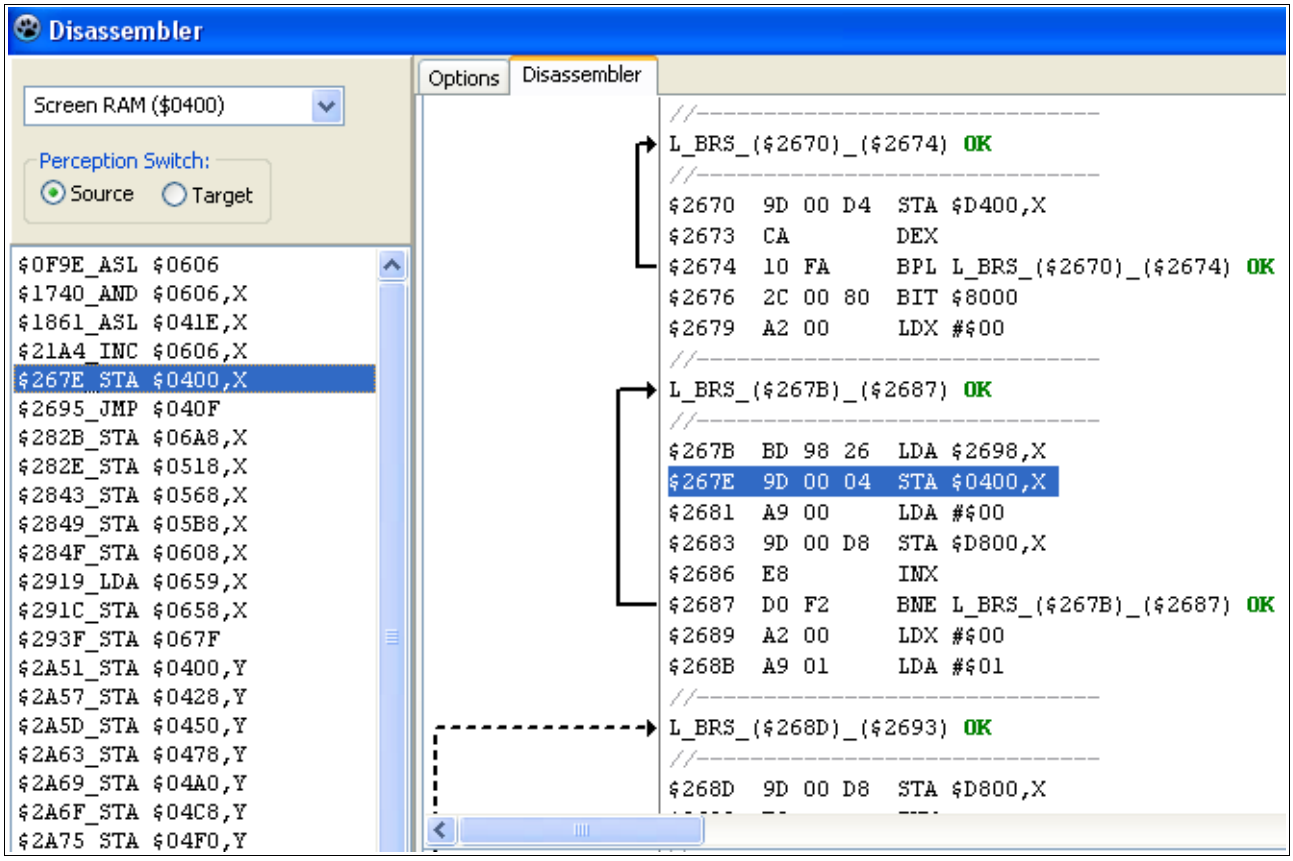
Use the dropdown component to select a generated list and push the left mousebutton to choose an entry. You will be instantly routed to the codeline. Corrupt rated codelines are sorted in six different lists, you can quickly inspect them with this:



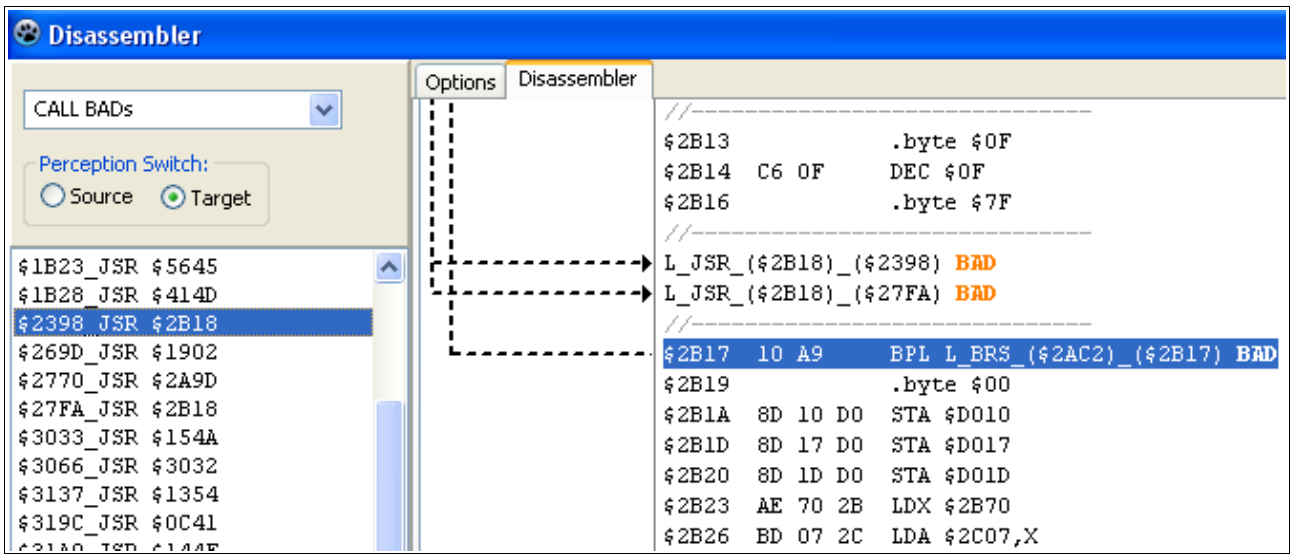
Here is a table with detail information:

Listselection	Description	Perception Switch	Search Type
JUMP JAMs	see above	yes	interpreter
JUMP BADs	see above	yes	interpreter
CALL JAMs	see above	yes	interpreter
CALL BADs	see above	yes	interpreter
Branch JAMs	see above	yes	interpreter
Branch BADs	see above	yes	interpreter
Interpreter 1	see above	no	interpreter
Interpreter 2	see above	no	interpreter
Interpreter 3	see above	no	interpreter
Sequence 1	see above	no	sequence
Sequence 2	see above	no	sequence
Sequence 3	see above	no	sequence
Snippets / Startup	Grabs an unlabeled code line having a preceding line of no code, end of IRQ or RTS. Ten lines of valid code must follow to get an entry. You may find the programstart, code fragments or some IRQ stuff. The given startaddress (options) will show up here, too.	no	interpreter
CIA 1	$DCXX	no	interpreter
CIA 2	$DDXX	no	interpreter
Graphics	$D011, $D016, $D018, $D020, $D021, $D022, $D023, $DD00	no	interpreter
Interrups	$D011, $D012, $D019, $D01A, $0314, $0315, $0316, $0317, $0318, $0319, $FFFE, $FFFF, $DC0D, $DD0D, $EA31, $EA7B, $EA81	no	interpreter
Sprites	$07F8-$07FF, $D000-$D00F, $D013-$D015, $D01B-$D01F, $D025-$D02E, $D010, $D017	no	interpreter
SID	$D4XX	no	interpreter
Screen RAM ($0400)	$0400-$07E7	no	interpreter
Color RAM ($D800)	$D800-$DBE7	no	interpreter
Clear Screen	JSR $E544, JSR $FF81	no	interpreter
Self References	Code target equals code offset	no	interpreter
Quicksearch	Grabs all direct code references to a given address or range. Part of some disassembler popup tools, see chapter 5.8	no	interpreter

Example 1: This is an example how to find anything on the standard screen, so it should not be hard to find scroll or shake routines. The search routines grab any possible OP code combination.



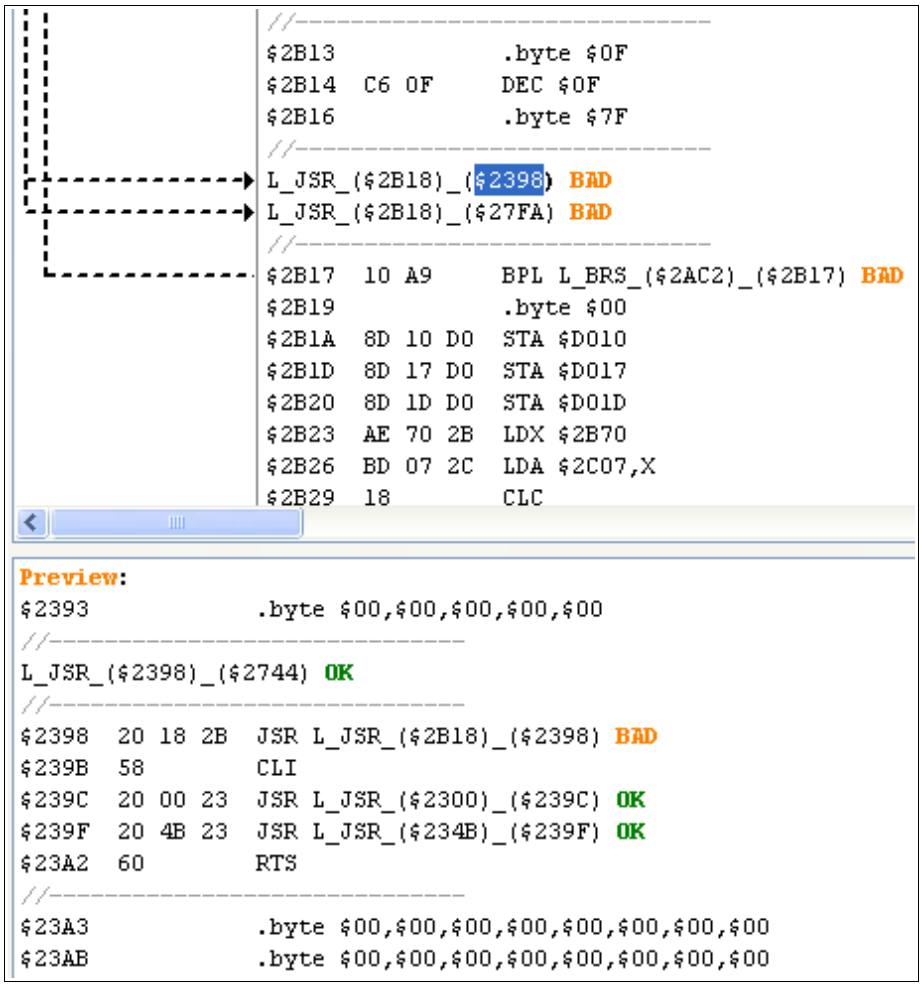
Example 2: This is a typical disassembling error, and a real BAD one too. RAM $2B18 is called two times(see CALL BADs), additionally the value $A9 indicates to an LDA #$00. These are inherent errors triggered by a false interpretation of $10 at $2B17. See chapters 5.7 to 5.9 how to analyse and handle this quickly.



5.4 Preview Window

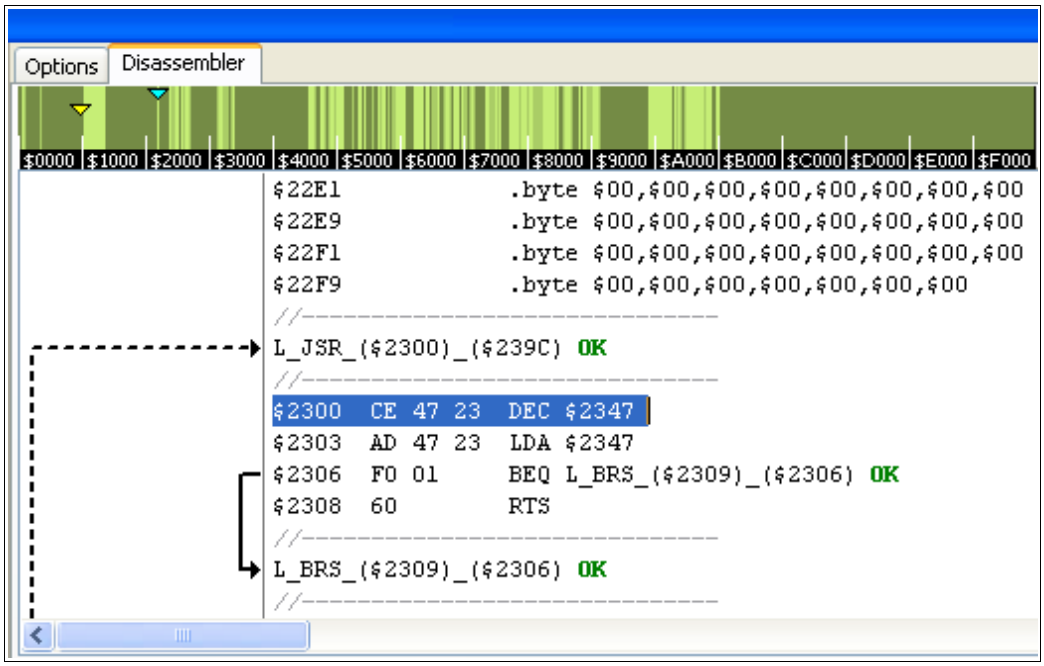
Since you can't use the “Perception Switch” on many lists there is a preview window. A double-click on any valid RAM address will automatically show a preview. It likes every three byte OP-code as well as addresses in labels .

You can quickly stab around if you are curious.



5.5 NaviMap

This is a concept of weighting the memory. The RAM is portioned in $80 byte pieces and checked for the code dose inside. The result is a map of the memory that can be used to navigate through the disassembling. A click on the map will route to the selected memory area. The cyan coloured triangle is the current position of your disassembling, while the yellow shows the preview position.



There is the color spread formula.

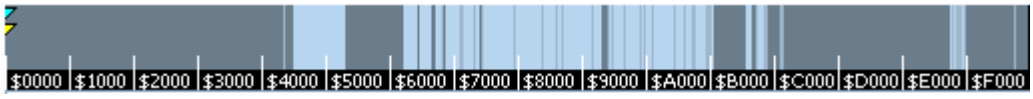
Colors	Memory Rating
black	unloaded
gray	excluded
dark green	almost crap
green	some code
light green	code

Let's have some application guessing:

'NUFLI' in VICE with standard excluded areas:

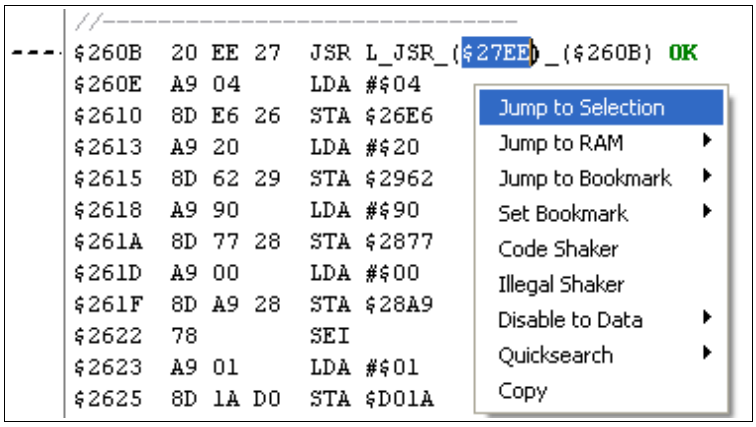
A crunched PRG file, in case you like unpacking routines:

Wizball, in game:



5.6 Pop-Up Menu (Basics)

The disassemble window uses a pop-up menu that gives you access to some more methods.



Jump to Selection: Go to the selected offset address.

Jump to RAM: Let you jump around.

Jump to Bookmark: Go to a bookmarked address.

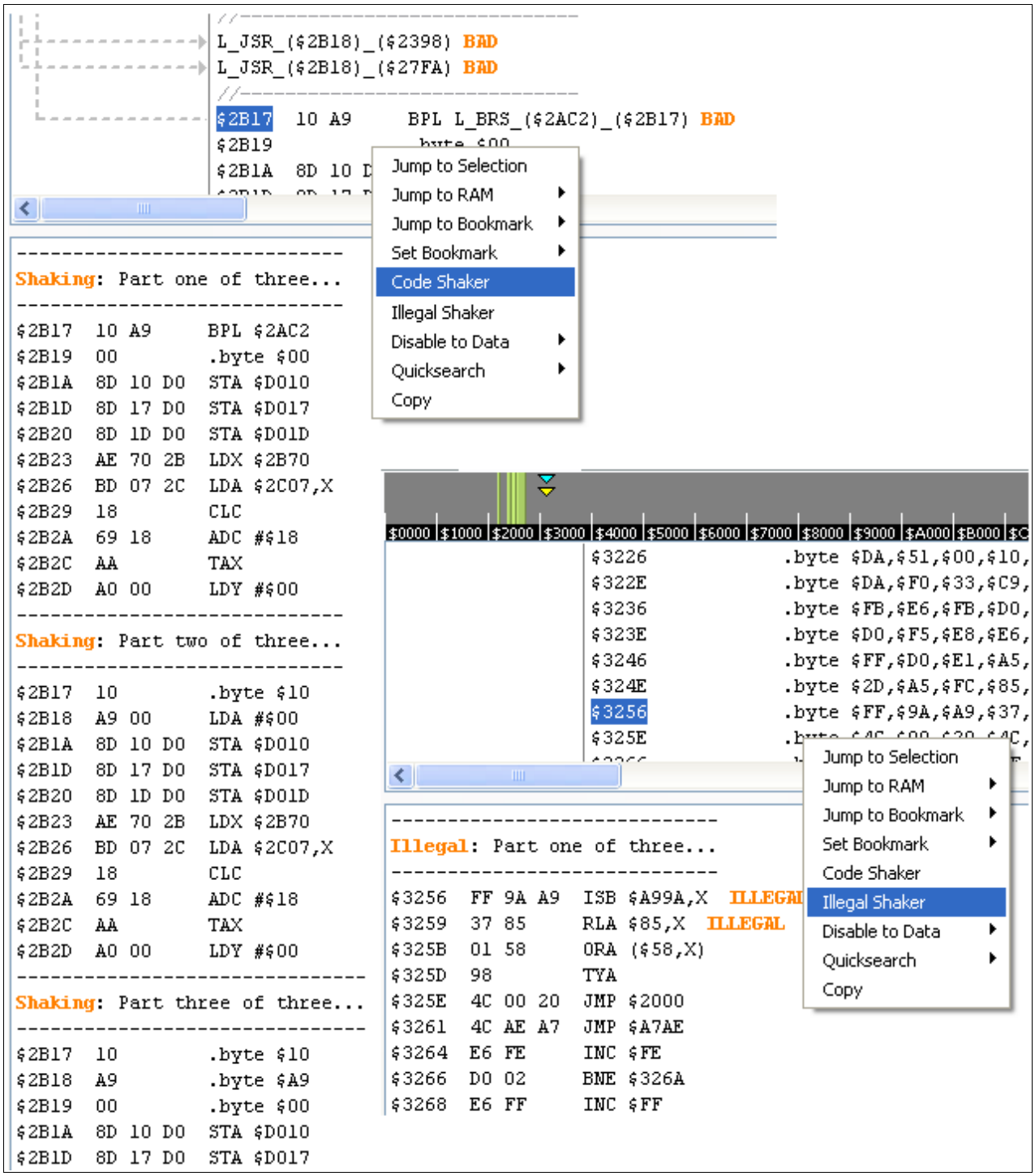
Set Bookmarks: Set a bookmark for a marked address.

Copy: Copy marked text to the clipboard

The little more complex items "Code Shaker", "Illegal Shaker", "Disable to Data" and "Quicksearch" are introduced below.

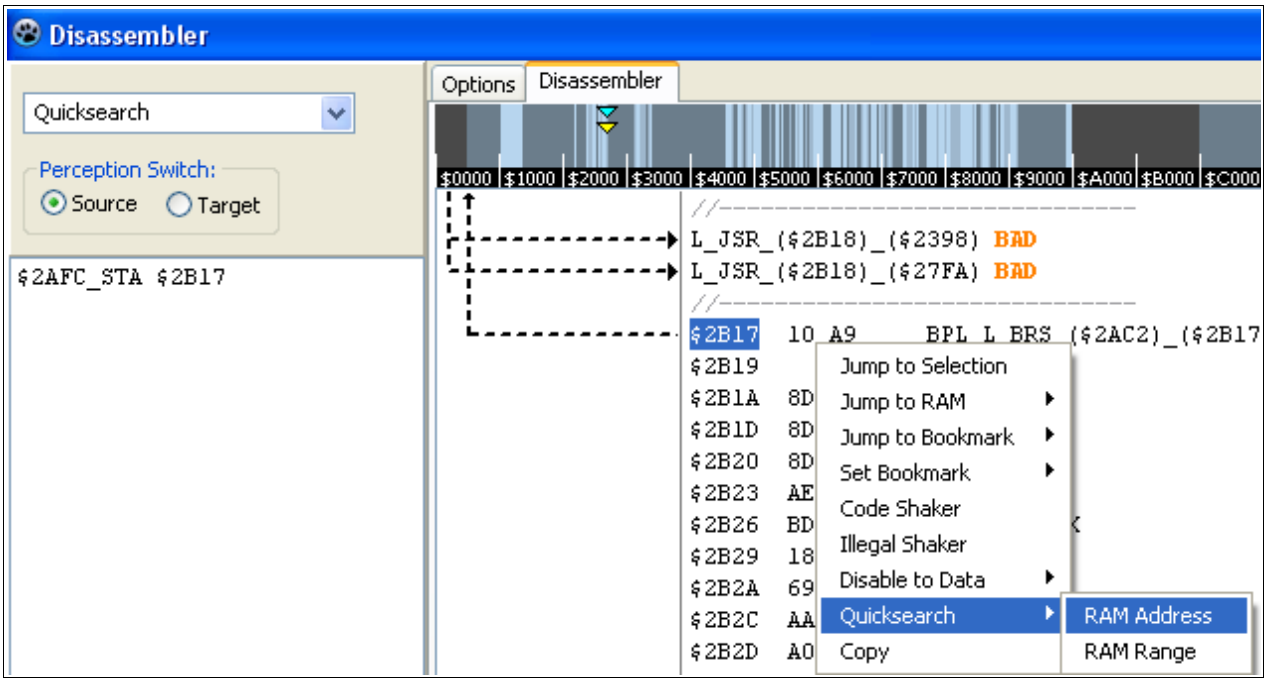
5.7 Code Shaker and Illegal Shaker

Whenever you are not sure about alternative disassemble interpretations, try “Code Shaker” or “Illegal Shaker” with a marked offset address. Both also like disabled RAM offsets. The results are shown in the preview window. Illegal CPU instructions get an extra marking .

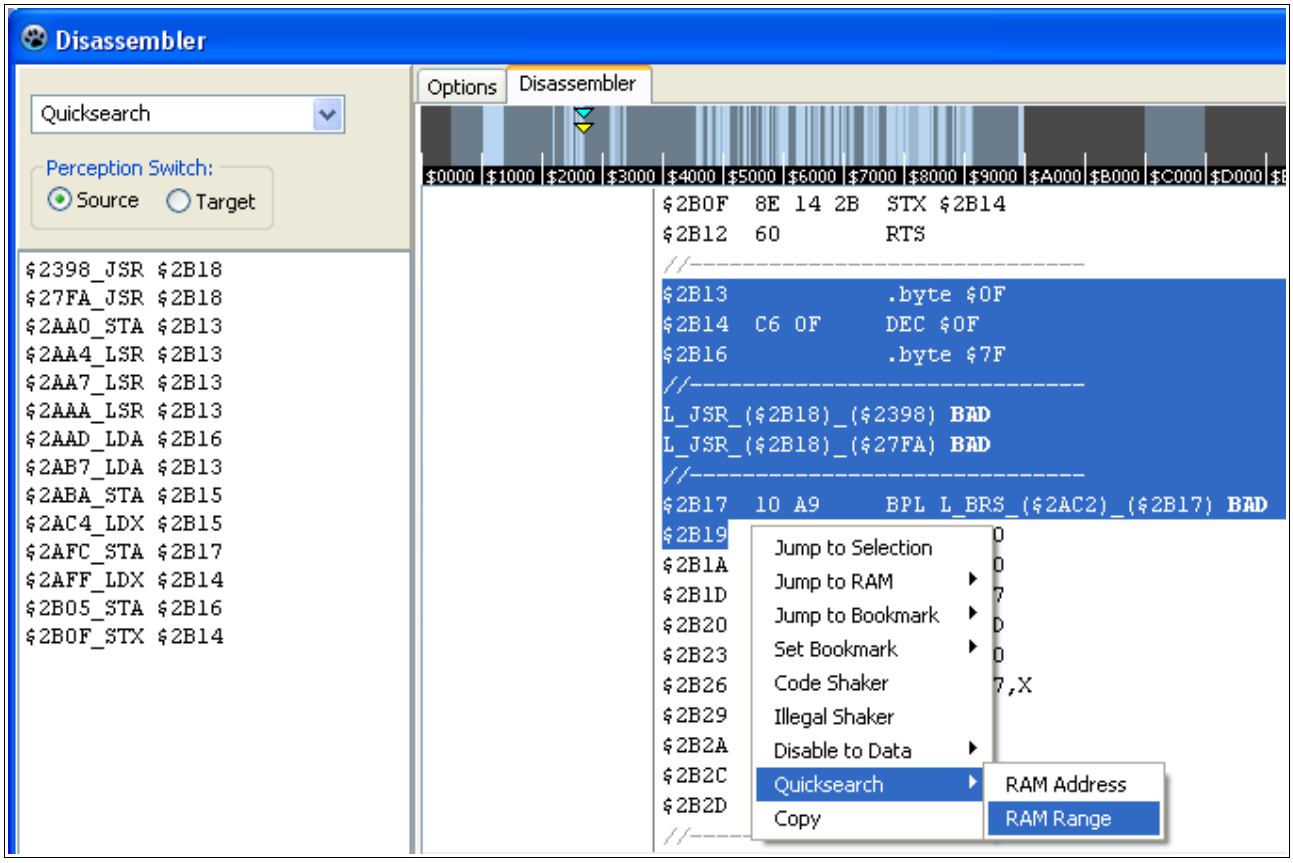


5.8 Quicksearch and the Quicksearchlist

Quicksearch performs an interpreter scan on the marked offset address, the output is transferred to the Quicksearchlist. Please note that branches will not be handled and not be found.

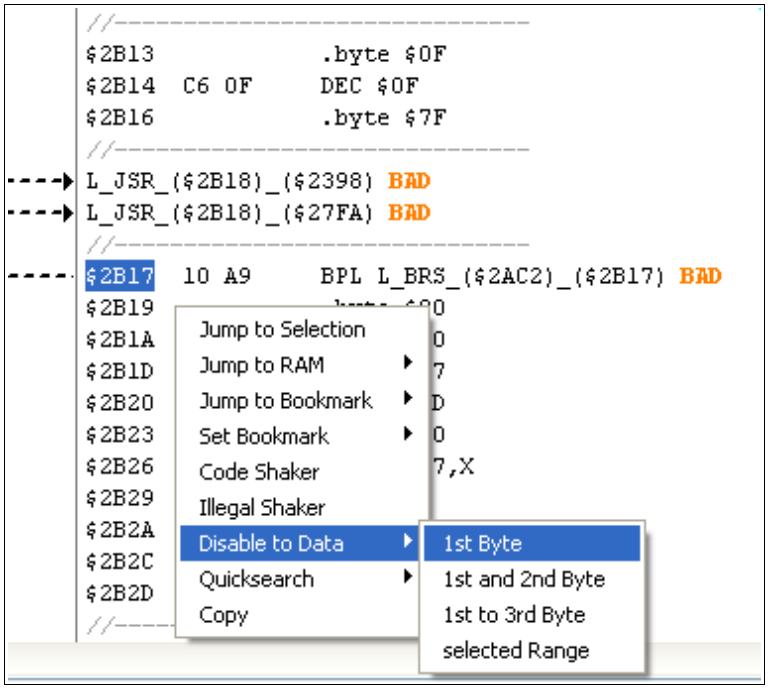


The subitem “RAM Range” is an extended version. This can be very useful when facing a table of pointers. It only accepts a range selection the way shown below. Please note that the last offset address is not part of the range. So in this case, the range is $2B13 - $2B18.

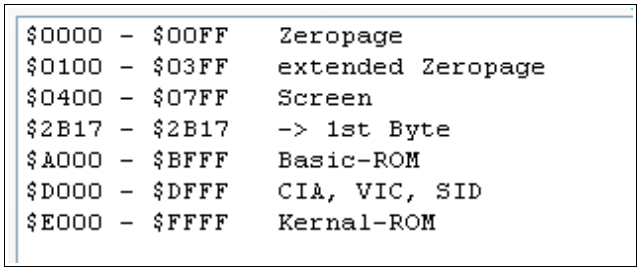


5.9 Disable to Data

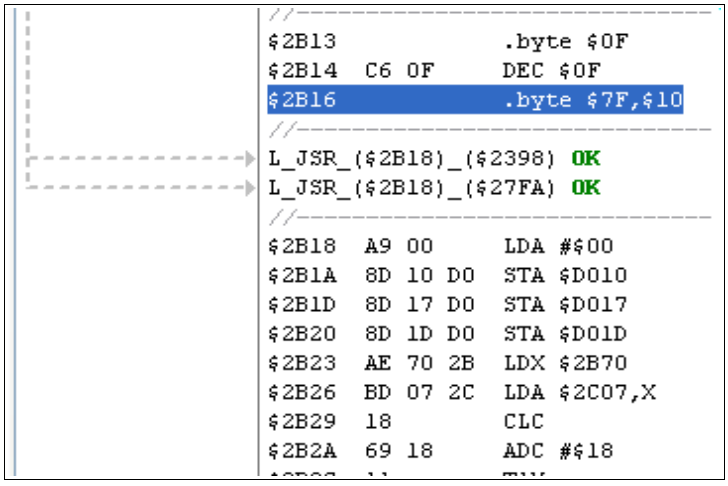
This method can add bytes to the excluded list using the interpreter view. Depending on the option “Autobusy”, a re-disassembling is done immediately. You may use a range instead of handpicking single bytes.



The disabled byte is automatically added to the excluded list.



Result:

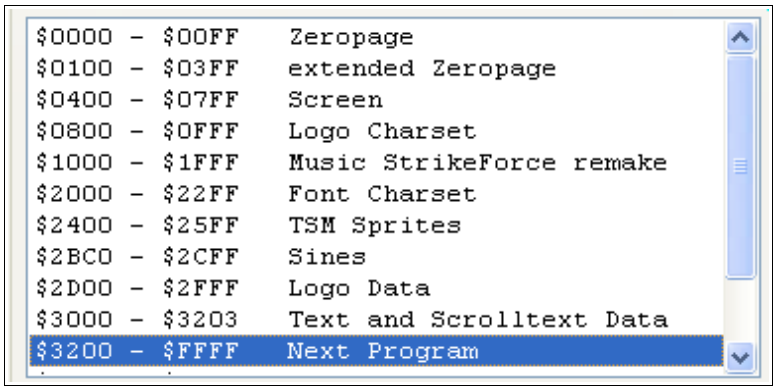


6 Showcase

Welcome to this little showcase. This shall give you some information about how to find several program routines. So let's start with the excluded areas identified in chapter 3.

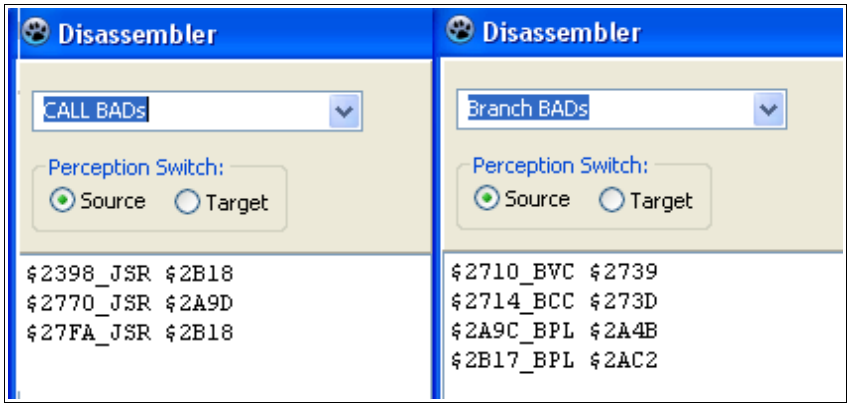
6.1 Fill the Excluded List

In many cases you can never be sure what's really going on – before you really look inside. So, the excluded list is just a draft and not the ultimate final one. You should always put the music on the list. The player, especially it's data, produce additional errors. This is how the excluded list may look like before you push the disassemble button. Because large RAM areas are excluded it shouldn't take longer that one or two seconds.



6.2 Solve the BADs and JAMs

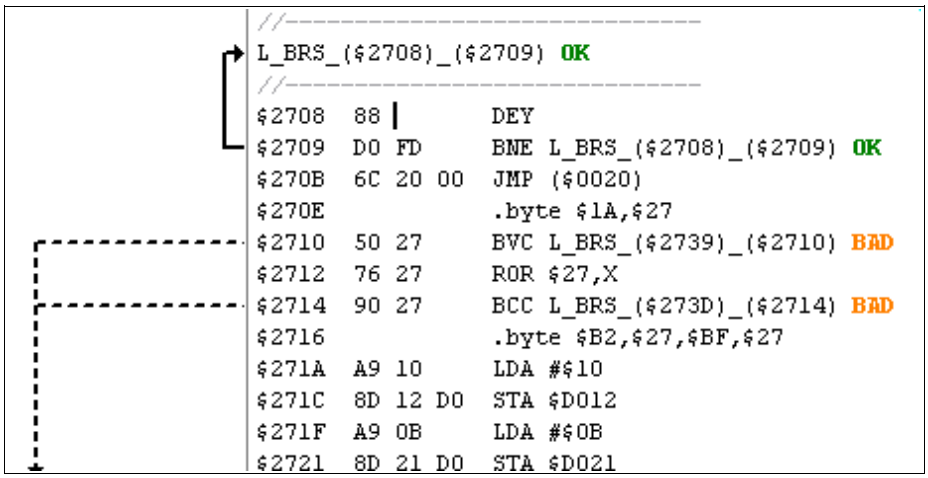
Checking the BAD and JAM entrys is the very first thing you should do. Because “JUMPs” and “CALLs” have the tendency to be multiple inherited errors, you should take care of them in the first place. Solving them means to get rid of most problems. Due to the excluded list entry, there are only seven BAD errors and no JAMs. You may got little different results here, it depends on the time your VICE snapshot was made(pointers for movements, color tables and scroll text).



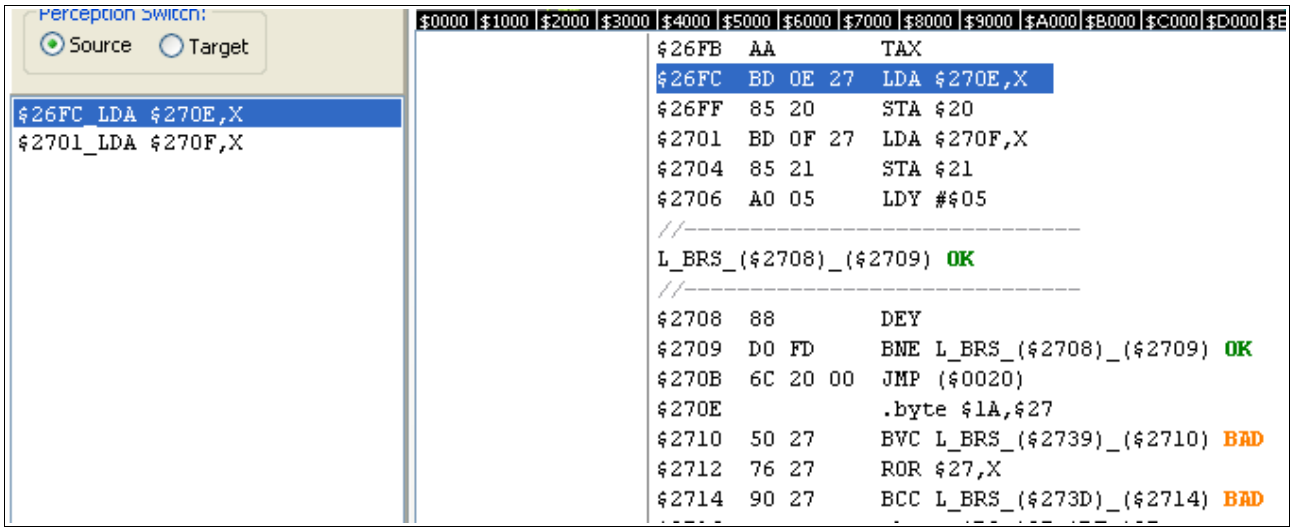
All three “CALLs” are inherited errors triggered by false interpretations at $2B17 and $2A9C, so disabling $2B17 and $2A9C is a good thing. The $2B17 problem was discussed in chapter 5, so there's no need to do this again. $2A9C is pretty much the same, just remember to use the “Perception Switch”, “Codeshaker”, “Quicksearch” and “Disable to Data” for this.

=> Errors at **$2710** and **$2714** remaining.

Both errors are close together and in between valid code lines. The indirect “JMP ($0020)” at $270B suggests that the errors are not executable code.



It looks a lot like an internal table with many $27 values used for something else . To be sure of that, you can use the “Quicksearch” functionalities. I used the range $270E to $2719($271A).



Yes, it's a jump vector table for $0020/$0021. => exclude!

A jump vector table is always a nice thing to have. Since Infiltrator can't produce labels for indirect jumps, I got some additional information about the program design. However, they may appear in the “Snippets” list. These are the table values without any code interpretation:

$270E .byte $1A,$27,$50,$27,$76,$27,$90,$27

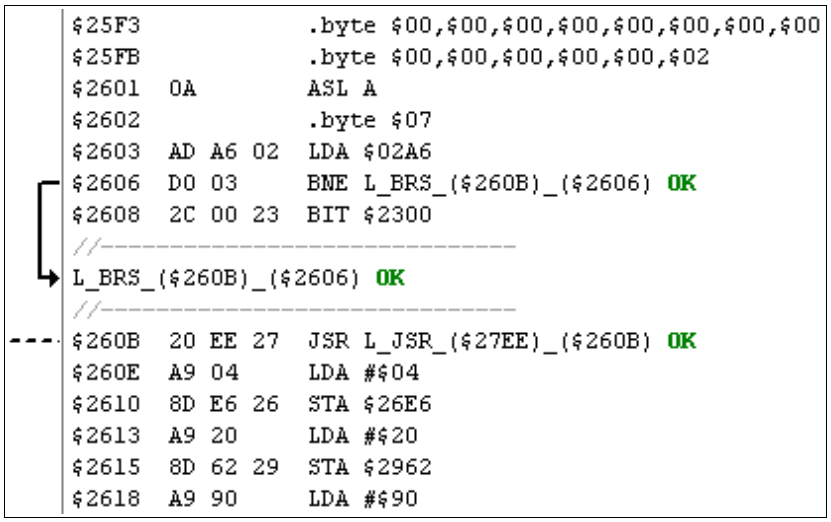
$2716 .byte $B2,$27,$BF,$27

Time to take care of the basic framework.

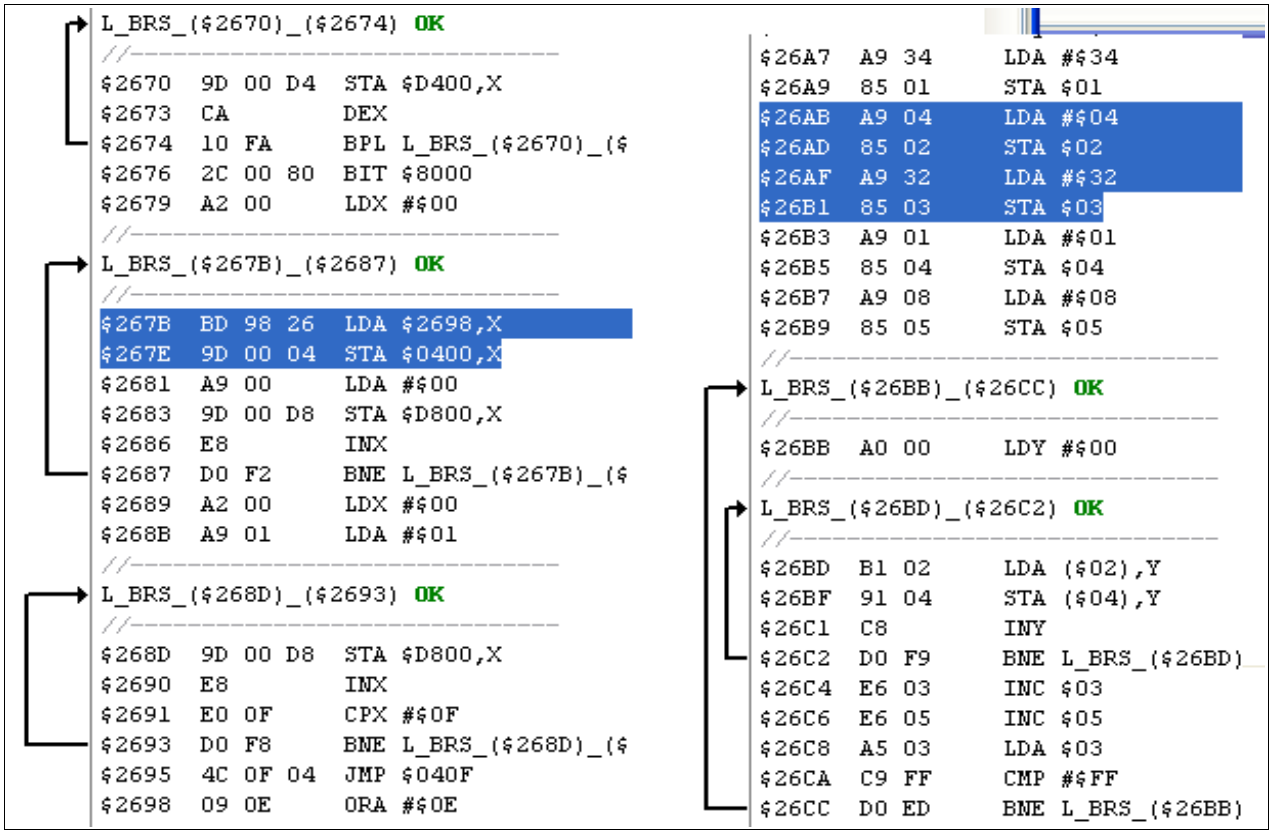
6.3 Understand the Program Framework

Catching the start and end of a program is not always that easy. But since we are facing an intro it should not be that hard. I don't like to discuss all attempts, so let's try something simple:

CALL for music player initialization (Interpreter Search 1): CALL at $27F7, part of a subroutine at $27EE which is called form $260B. That routine starts at $2603 with “LDA $02A6” (checking for the PAL/NTSC version) and does not have a label. Gotcha!



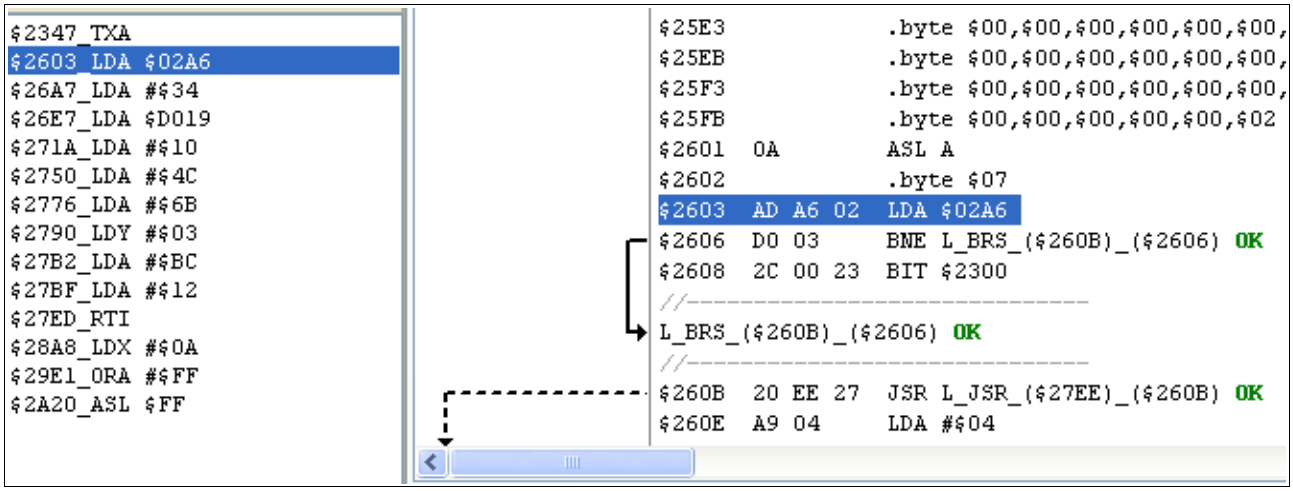
Searching for the end: Easy, because “Space” activates the end! Use the “CIA 1” list with register $DC01, you get the loop for the keyboard scan. You can catch the memory move routine along the way. (LDA $2698,X → STA $0400,X → JMP $040F → LDA $3204,Y → STA $0801,Y)



Get IRQ start: Use the “Interrupt” list to get $26E7

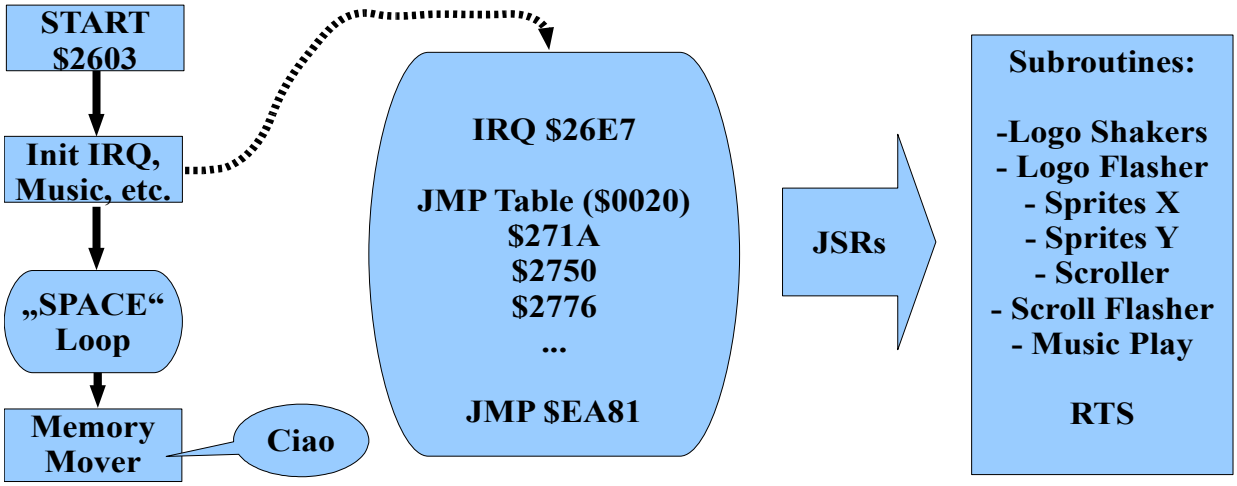
```
$2622  78        SEI
$2623  A9 01     LDA #$01
$2625  8D 1A D0  STA $D01A
$2628  A9 7F     LDA #$7F
$262A  8D 0D DC  STA $DC0D
$262D  A9 1B     LDA #$1B
$262F  8D 11 D0  STA $D011
$2632  A9 E7     LDA #$E7
$2634  8D 14 03  STA $0314
$2637  A9 26     LDA #$26
$2639  8D 15 03  STA $0315
```

Or use the “Snippets”: Grabs the main start, the IRQ start, the jump vector table addresses seen before and of course some crap.



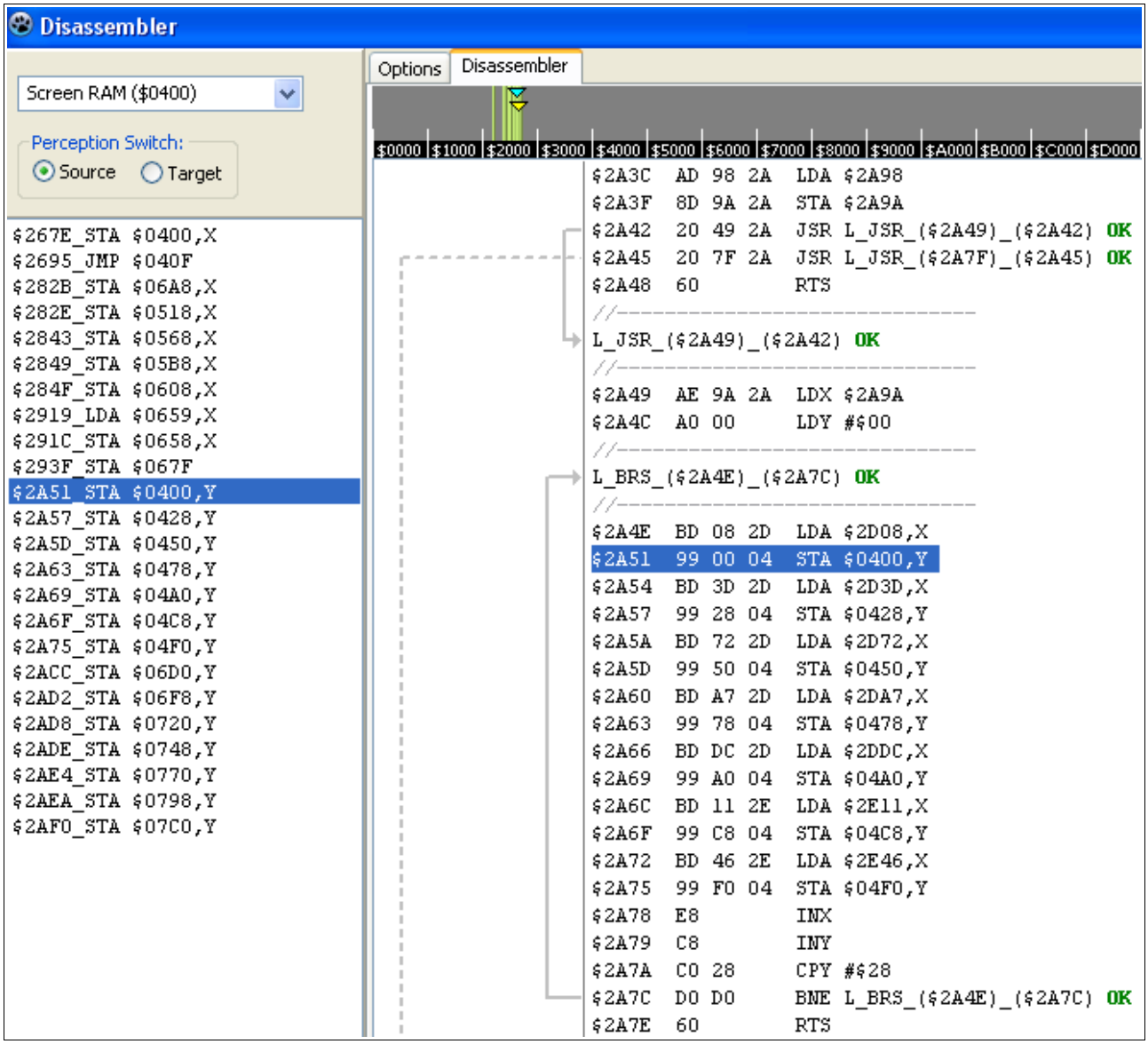
You may also try the standard sequence 3 ($01, $58) for the final decompression command. It doesn't work here, because the JUMP to $2603 was originally placed on the screen and is gone. So it depends on the used packer, cruncher and the memory usage. You may receive some decompression code fragments in excluded areas, use “Code Shaker” for a peek.

Intro framework:



6.4 Get the IKARI Logo Shaker

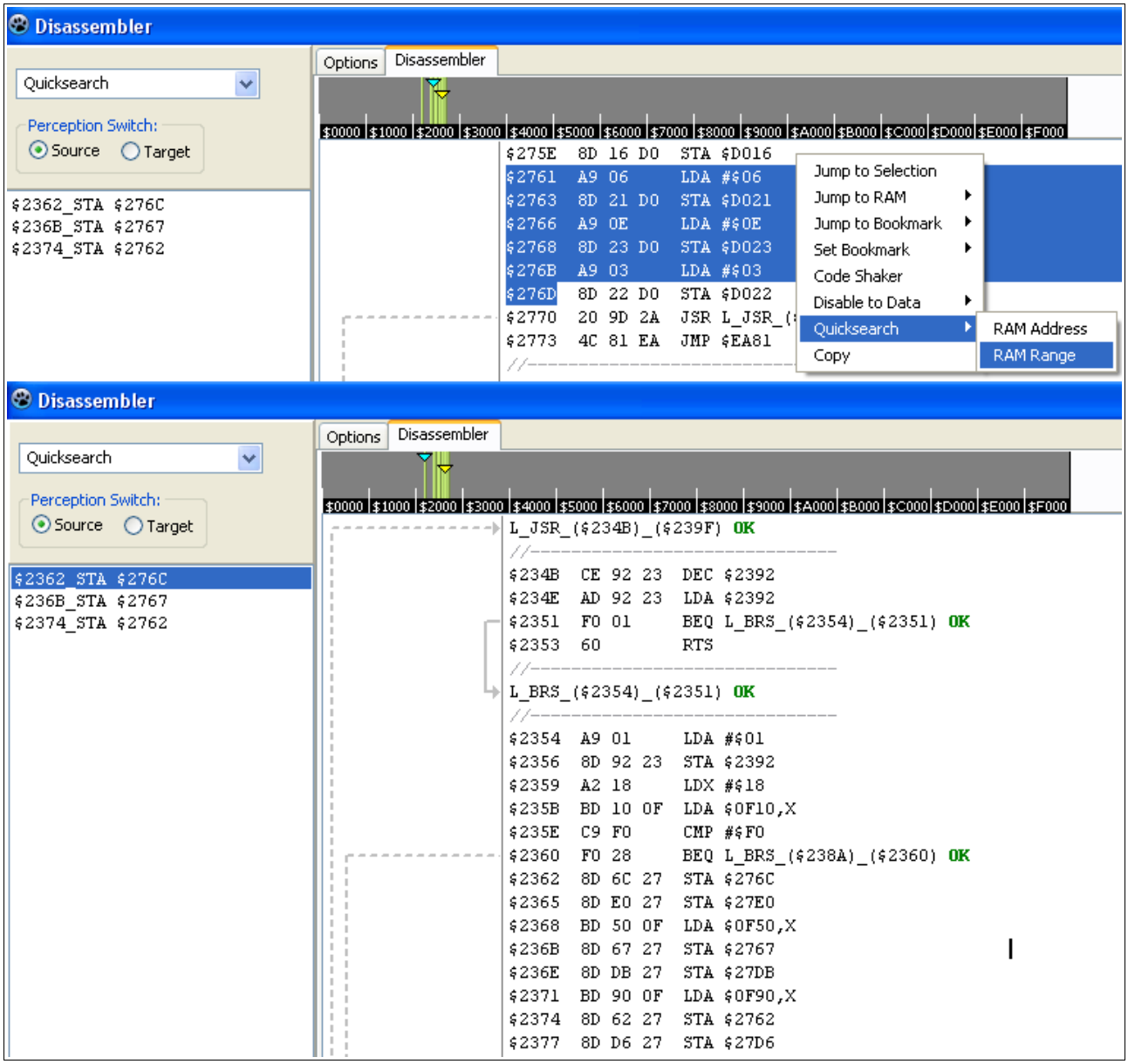
Using the “Screen RAM” list is an efficient approach. The X sine position is read from $2A9A(calculated somewhere before). Finding the TALENT logo and the scroll is one and the same thing.



6.5 Get the Logo Flash Routine

Use the “Graphics” list and try for $D022 or $D023, you will find the typical IRQ constructs. Since the color bytes are updated by another routine it's obvious to do a “Quicksearch”.

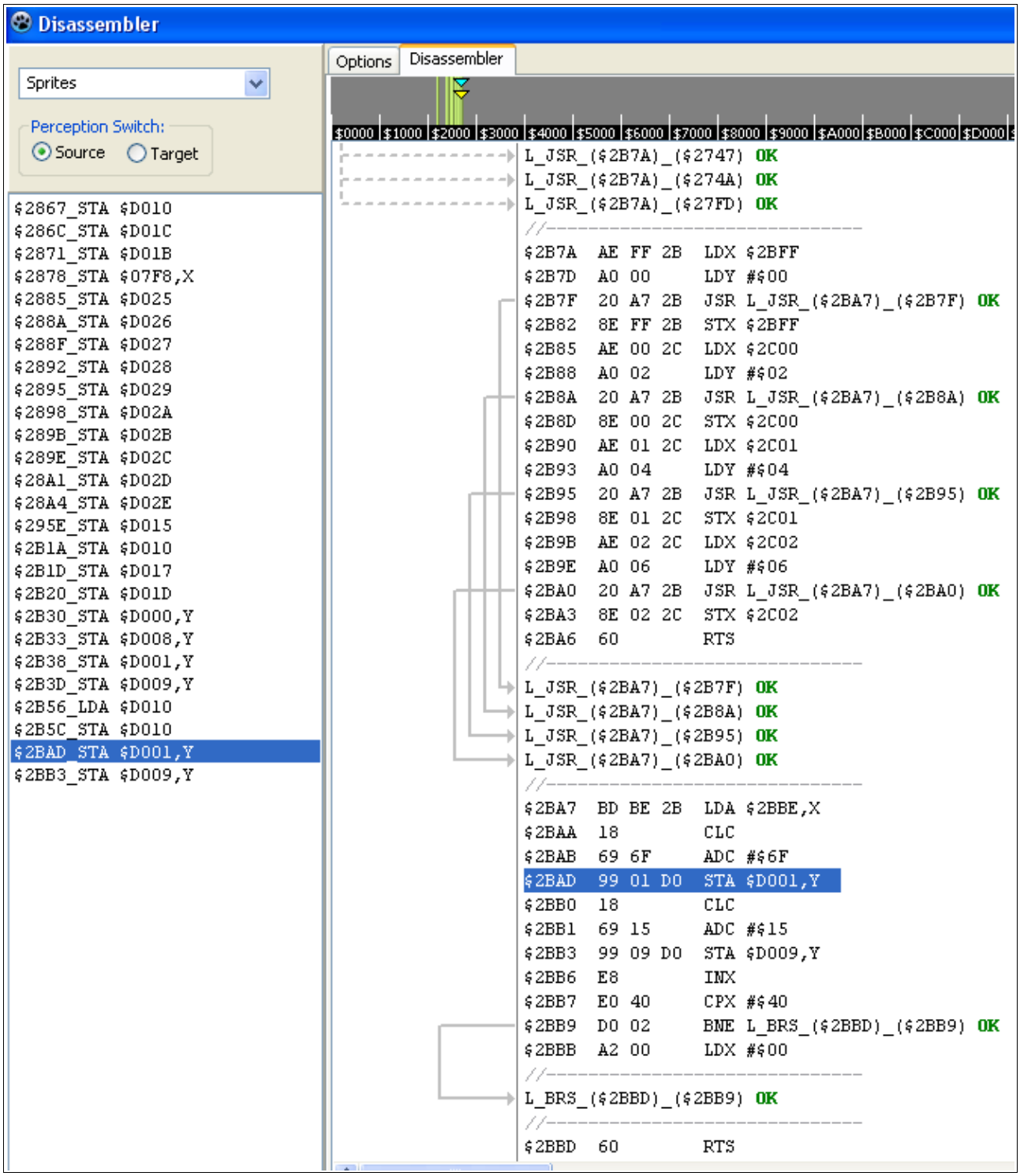
This routine uses color tables at $0F10, $0F50 and $0F90.



6.6 Get the TSM Y-Movement Routine

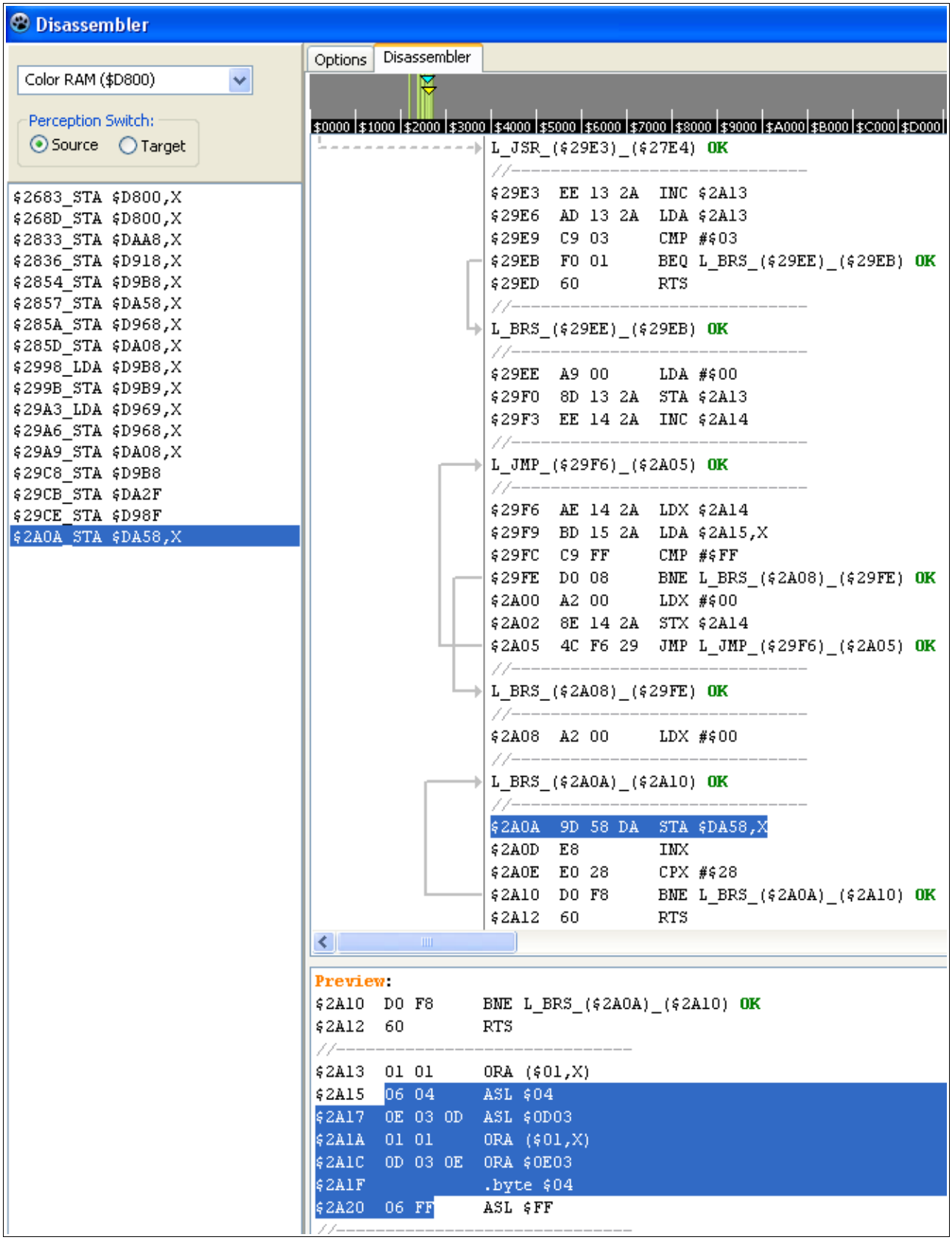
Just use the “Sprites” list and choose. You may wonder about $2B38 and $2B3D feeding the Y sprite registers with static values. Well, I don't know – maybe the programmer intended to charge X and Y registers in the same routine.

However, $2B7A is what we are looking for. The routine is called three times(speeding up the movement) and uses the bouncing sine at $2BBE.



6.7 Get the Scroll Text Flasher

Use the “Color RAM” list. The routine uses a small color table and has a delay of three frames.



7 Appendix

7.1 FAQ

- I'm using VICE 2.2! Does it work anyway? The VICE Development Team made some major changes, but I guess it will work. However, I recommend to get the 2.3 version.
- Where are the illegal OP-Codes? I think it does not make sense to do that for the complete memory. Use the illegal shaker in case you think you are facing them.
- What about other emulator imports? One day, maybe.
- Can I have several program instances? Yes!
- Where is the OP-Code “BRK”? The OP-Code $00 is internally handled as an unknown CPU instruction, so it can be put in rows by the CONCAT. Hope you don't mind too much.
- What about IRQ labels? Simple forward interpretation could result in incomplete or even incorrect results. I don't know how to solve this without writing an OP-Code emulator yet.
- What about generating labels for static offsets (LDA,STA,etc. $XXXX): Might be useful in case you like to rip off speed code, but also may result in tens of thousands useless labels. However, I guess this function will be used rarely. So... maybe.
- What about an extended CRAP version with IF ELSE and LOOP commands? Maybe.
- Why does Infiltrator use so much RAM? Blame the Lazarus Development Team. At least I used UPX 3.07 to compress the executable file.
- What about function trees? Planned.

7.2 Known Bugs

- The connecting code lines in the disassembler are not redrawn when using the mouse wheel. Until now they are only drawn on a canvas element when pushing the cursors, page up/down keys or the mouse buttons.
- When closing a tool window not using the main Infiltrator form buttons you have to push the button twice for a reinitialisation.
- The CRAP error handling is insufficient.

7.3 "AS IS" Warranty Statement

ATTENTION: BY DOWNLOADING AND USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE FOLLOWING TERMS. IF YOU DO NOT AGREE TO ALL OF THESE TERMS, DO NOT DOWNLOAD AND USE THE SOFTWARE ON YOUR SYSTEM.

"AS IS" WARRANTY STATEMENT

DISCLAIMER. TO THE EXTENT ALLOWED BY LOCAL LAW, THIS SOFTWARE PRODUCT ("SOFTWARE") IS PROVIDED TO YOU "AS IS" WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, WHETHER ORAL OR WRITTEN, EXPRESS OR IMPLIED. THE OWNER OF THE COPYRIGHT SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

LIMITATION OF LIABILITY. EXCEPT TO THE EXTENT PROHIBITED BY LOCAL LAW, IN NO EVENT WILL THE OWNER OF THE COPYRIGHT BE LIABLE FOR DIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR OTHER DAMAGES (INCLUDING LOST PROFIT, LOST DATA, OR DOWNTIME COSTS), ARISING OUT OF THE USE, INABILITY TO USE, OR THE RESULTS OF USE OF THE SOFTWARE, WHETHER BASED IN WARRANTY, CONTRACT, TORT OR OTHER LEGAL THEORY, AND WHETHER OR NOT ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.