April 2018 Martin Head

DisCharge, SuperCharge Decompiler

DisCharge (working title) is a decompiler for SuperCharged programs. It is still very much a work in progress, and does not yet know how to handle all SuperBASIC commands. And as it is still in development, anything, and everything is libel to change at any time.

DisCharge cannot just convert an executable file back into a ready to run SuperBASIC program. It requires manual intervention, and the resulting SuperBASIC program will need some tidying up.

This document is a walk through of the steps required to convert a SuperCharged executable back into a SuperBASIC program.

For this example I have created a small SuperBASIC program (**sample_original**) and compiled it to **sample_exe** with SuperCharge version 2.00 in Qemulator. To see what **sample_exe** does, **EXEC_W** it. The rest of the decompiling will be done in QPC2, and a Windows based text editor. If you use a different setup, you may need to make a few changes to the manual aspects of decompiling.

Step 1 - Disassemble the executable program.

We first need to disassemble the executable file to a text file. I use the Assembler Workbench by Talent/Quanta. If you use another disassembler, you may need to make changes to the **ProcessDump_bas** program.

I have supplied a disassembly of **sample_exe**, named **sample_dmp**

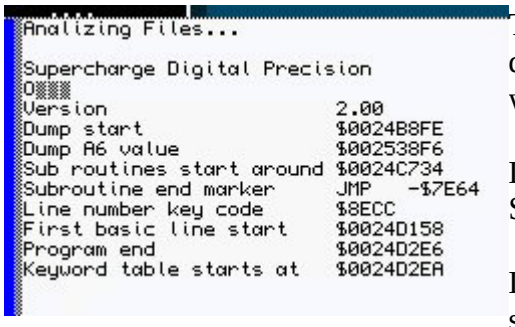
Step 2 - Process the dump file.

Load the **ProcessDump_bas** program, Edit the filenames near the start of the program to suit your system. And **RUN** the program.

This will create two new files **sample_dmp_lib**, and **sample_codes**.

The **sample_dmp_lib** file is the disassembly with the various machine code routines for the SuperBASIC program separated out. It misses the start of the first routine, which I will come back to later.

The **sample_codes** file is a list of these routines code numbers, This is used later to create an array used in the decoding the executable program.



The program will display some information on the disassembled program. A couple of pieces of information will come in useful later. So make a note of them.

Dump A6 value, This value is used extensively by SuperCharge.

Line number key code, This is the code used to identify the start of a SuperBASIC line.

I have supplied copies of the generated files, **sample_dmp_lib_bak**, and **sample_codes_bak**.

Using the Line number key code from above, you can now update the **sample_codes** file in a text editor.

If you look at the **CodeArrayKeys** document. You will see that Index 0, is the Start of a program line.

So edit the line '-1,8ECC' to '0,8ECC'. All of the lines in the **sample_codes** file will eventually need to be edited to remove the minus ones.

Step 3 - Identify the routines.

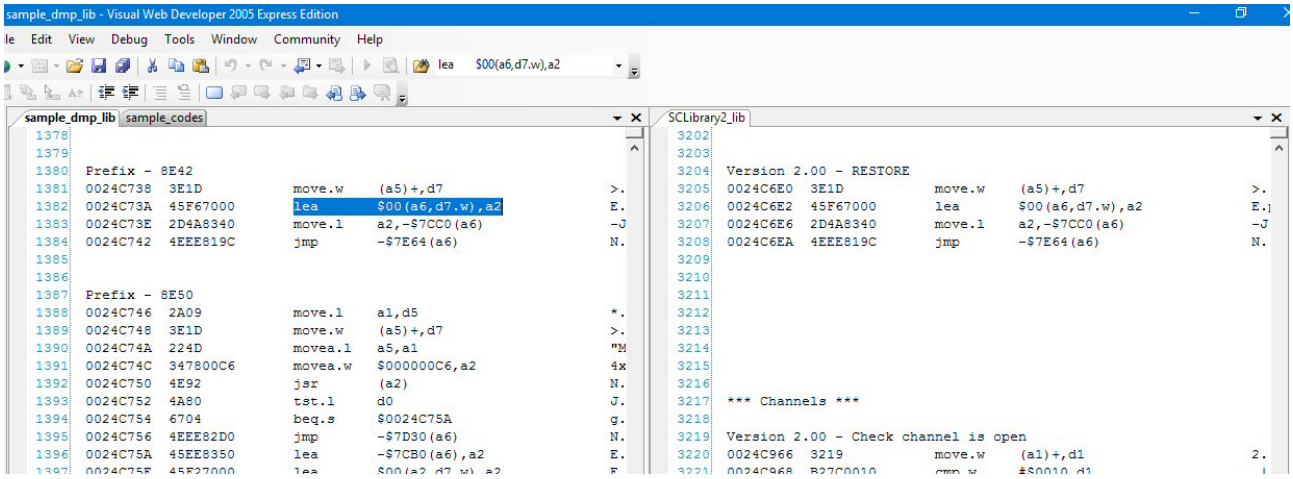
You will now have to go through all the machine code routines headed by '**Prefix - xxxx**'.

There are two reference library files supplied, **SCLibrary1_lib** and **SCLibrary2_lib**. These files contain identified sample routines to compare with your **sample_dmp_lib** file.

The first file contains routines for SuperCharge under version 2.00, and the second file for version 2.00. For this example you will only require **SCLibrary2_lib**.

I will now do a walk through of identifying the first routine **Prefix - 8E42**.

Pick a line in the routine, I have chosen '**lea $00(a6,d7.w),a2**' and I have searched for this in **SCLibrary2_lib**, There is more than one match, but only one routine has the same number of lines and matches, '**Version 2.00 - RESTORE**'



So Prefix - 8E42 is RESTORE. It's worth updating the **sample_dmp_lib** file, so you can keep track of which routines you have identified. So update it to something like '**Prefix - 8E42 - RESTORE**'

If you look at the **CodeArrayKeys** document. You will see that Index 165 is RESTORE, and you can update the **sample_codes** file line to **165,8E42**.

You now need to repeat this process for all the 'Prefix - xxxx' routines.

Be careful with some of the routines as they look similar but have small differences

If you cannot identify a routine, leave it and come back later when you start actually decompiling.

```
165,8E42
-1,8E50
-1,8E84
-1,8E92
-1,8EC4
0,8ECC
100,8ED2
96,8F0E
55,8F14
97,8F1A
180,900E
113,9034
57,90C2
110,90E0
112,917C
58,91A8
76,91AE
63,91B6
98,91C4
64,94E8
61,951A
151,95A2
3,95B0
140,965A
60,9674
84,9684
56,97D2
20,97DC
109,97E4
161,9818
-1,9820
```

The **sample_code** file should now look something like this.

Notice that some codes could not be identified. Codes 8E50, 8E84, 8E92 and 8EC4 are not part of the SuperBASIC program, and can be deleted.

The last code 9820 can also be deleted as it is not the start of a routine. Note that all the routines end with ‘jmp -$7E64(a6)’

Some of the routines take the form

```
jsr     -$7DEA(a6)
jmp     -$7E64(a6)
```

To work out the address in the ‘jsr’ line, use -$7DEA+the value of A6 we noted earlier. -$7DEA+$2538F6 = $24BB0C

Don’t worry if you delete a needed code, it will be highlighted when you run the decompiler. And you can put it back in.

I have supplied a filled in **sample_dmp_lib** and **sample_codes** files.

In the **sample _dmp_lib** file, I have also included the original SuperBASIC lines of code in case you want to try to understand how the SuperBASIC program is stored in the compiled program. I wont go into details in this document, as it’s only meant to be a walk through to get you started.

Step 4 - Do the decompilation.

Load the program **DisCharge_bas**, Edit the filenames near the start of the program to suit your system. And **RUN** the program.

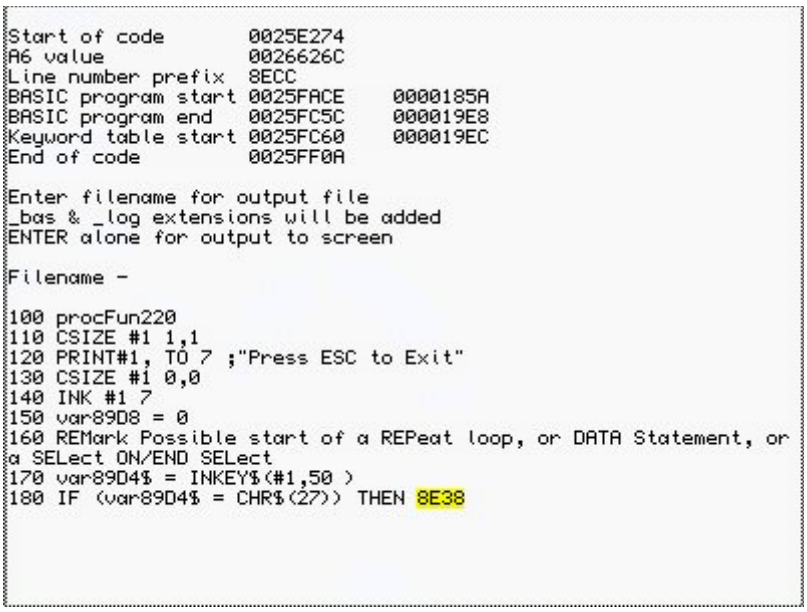
You only need to **RUN** the program once, after that us **GO TO 1000**.

Use **GOTO 1000**.

When asked for a filename, just press Enter.

The decompilation will begin, Press any key to continue one line at a time.

When you reach line 180, you will see a highlighted code after THEN. This indicates the program has encountered a code it does not know how to deal with.



Remember I said earlier that the **ProcessDump_bas** program misses the start of the first routine. This is it.

To find the start of this routine in the **sample_dmp_lib** file, use the formula ‘A6 value-($10000-code)’ in this case $2538F6-($10000-$8E38) = $24C72E

0024C72E	3E1D	move.w	(a5)+,d7
0024C730	4BF67000	lea	$00(a6,d7.w),a5
0024C734	4EEE819C	jmp	-$7E64(a6)

It’s GO TO with an index of 160. Add this to the **sample_codes** file and decompile again.

This time the decompilation will complete without error.

REMark out the line 1880 to read .. 1880 REMark IF lineno>100 THEN PAUSE -1

Decompile again, this time entering a filename when asked. Two files will be created with your filename, and extensions of _bas and _log

The _bas file will be the BASIC program, and the _log file will contain any warning, or errors, and a list of Procedure/Function line numbers.

Step 5 - Tidy the SuperBASIC programs

The SuperBASIC program produced is unlikely to Load without errors, So load the produced BASIC program into a text editor.

Here is the program produced

```
100 procFun220
110 CSIZE #1 1,1
120 PRINT#1, TO 7 ;"Press ESC to Exit"
130 CSIZE #1 0,0
140 INK #1 7
150 var89D8 = 0
160 REMark Possible start of a REPeat loop, or DATA Statement, or
a SELect ON/END SELect
170 var89D4$ = INKEY$(#1,50 )
180 IF (var89D4$ = CHR$(27)) THEN GO TO 220 : END IF
190 AT #1 5,20  : PRINT#1,var89D8
200 var89D8 = (var89D8 + 1)
210 GO TO 170
220 DEFine PROCedure/FuNction procFun220
230 MODE 4
240 WINDOW #1 256,128,128,64
250 BORDER 2,2
260 PAPER #1 0  : INK #1 4
270 CLS
280 RETurn/END DEFine : STOP
```

Notice that there are missing comma’s after the channel numbers in some of the commands.

Looking at the code between lines 160 and 210, Line 210 goes to 170, which suggests it’s a REPeat loop. Also the GO TO 220 in line 180 looks like a EXIT loop.
Edit line 160, 180, and 210

Line 220, DisCharge cannot tell the difference between a DEFine PROCedure, and a DEFine FuNction. Line 100 tells you it is a Procedure, so edit line 220 to DEFine PROCedure procFun220.

Likewise in line 280, Discharge cannot tell the difference between a RETurn and a END DEFine. Edit line 280 to END DEFine, and you can remove the STOP, as its not needed.

You can now load the program into your QL/emulator and run it.