Hacking a Custom Intro into SNES ROMs

Introduction

This tutorial aims to simplify the learning curve for hacking your own intros into SNES ROMS.

The tutorial will be kept updated HERE.

If you'd like a play-by-play walk-through of this tutorial, you can check out the YouTube video HERE.

If you have questions or problems, feel free to leave comments on the video.

(Last Update: 2018/07/08)

Credits

- DaMarsMan (This tutorial expands upon his original tutorial and is based upon his original code)
- MrRichard999 (He suggested I take a look at the old tutorial and organized/edited things)
- Manz (for his updated code of GIF2SOPT)
- Pinswell (for compiling GIF2SOPT)

- Neviksti (for PCX2SNES)
- byuu and Revenant (for bsnes-plus)
- Geiger (for his SNES9x debugger)
- DackR (I wrote this tutorial and made a video on YouTube about it)

Requirements

- Windows PC
- Tutorial files extracted to the folder of your choice. This folder will be referred to as: {introfiles}
- LoROM formatted SNES ROM (this tutorial only covers LoROM currently) The ROM used in the tutorial is Little Master Episode 3.
- An original intro image (256x224 pixels, 256 Colors) - An example is included.
- Hex Editor (HxD, Hex Workshop, etc.)
- xkas v0.06 (Common SNES assembler) - Placed in the {introfiles} folder.
- Geiger's SNES9x Debugger or bsnes-plus
- gif2sopt (accepts gif87a format) OR pcx2snes (for pcx files) - A 32 bit version of gif2sopt is included. pcx2snes must be extracted in the {introfiles}\image\pcx folder.
- Image Converter app (XNView, Gimp, Photoshop)

Preparing the Intro Image

You can use pretty much any image you want as long as it is prepared properly. Keep in mind some limitations:

- The image must be 256x224 in size.
- Number of colors must be limited to 256.

Once the image has been created, it's time to save it in the correct format that can then be used to prepare the image to be used by our code.

There is an intro.png file included with this tutorial ({introfiles}\image\original\). To prepare the image, we've got two options: GIF2SOPT and PCX2SNES.

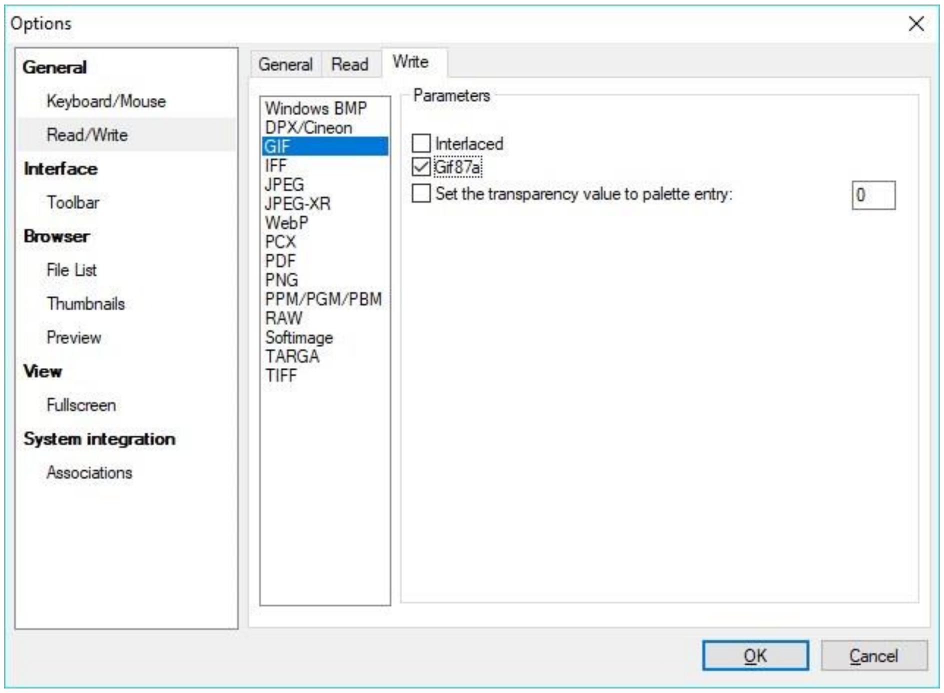
Option #1: Using GIF2SOPT and XNView

GIF2SOPT requires that you provide an image in GIF87a format. Newer image editor applications almost always export in GIF89a format. **If you feed GIF2SOPT a GIF89a formatted gif file, the conversion will fail.**

Okay, now that that's sorted out, if that sounds like a headache to you, then you may just want to skip to the next option.

To convert your image into a gif (the example file is in png format):

- Open up XnView and go to the directory with intro.png ({introfiles}\image\original\).
- Right click intro.png and click "batch convert" or "Batch Processing...", depending on the version of XNView you are using.
- Change the format dropdown to GIF (Compuserve GIF).
- Next, click "Options" and **make sure that Gif87a is checked** under the gif category (as seen below).



Interlaced ☐
Gif87a ☑
Set the transparency value to palette entry: ☐ 0

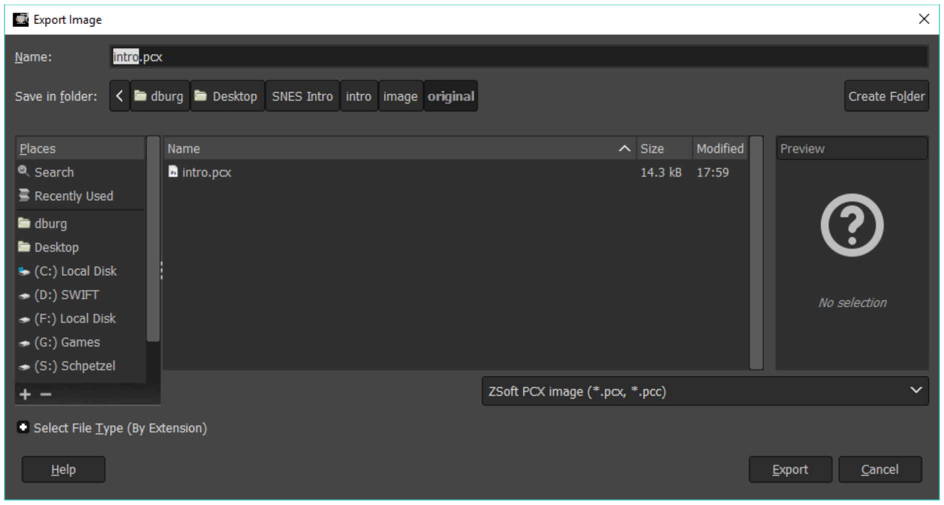
- Click "Go", make sure 256 color mode is selected and click "OK"
- After completing, an intro.gif file will be output.
- Copy the intro.gif file into the {introfiles}\image\gif\ folder. (Contains gif2sopt.exe)
- Run gif2sopt.exe by double-clicking on it.
- You will be prompted for the GIF filename, the MAP filename, the COL filename, and the SET filename. At each prompt simply type"intro" (without the quotes) and press enter.
- Three new files are generated and are available in the same folder.
- Copy the COL, MAP, and SET files into the parent directory ({introfiles}\image\)

Now you're ready for the next step. You can skip option #2 if you were successfully able to complete this step.

Option #2: Using PCX2SNES and Gimp

I'll be preparing the pcx file with Gimp, but you can do the same thing in Photoshop.

- Start Gimp and load up the intro.png example from your {introfiles}\image\original\ folder
- Click Image>Mode>Indexed...
- Set Maximum Number of Colors to 256
- Set Dithering to your desired option (I used Floyd-Steinberg) and click Convert
- Next, Click File>Export As...
- Choose ZSoft PCX image, change the name to intro.pcx and click export. (shown below)



- Copy intro.pcx into the {introfiles}\image\pcx\ folder.
- Double-click convert.bat to run the conversion.
- Three new files are generated and are available in the same folder.
- Copy the COL, MAP, and SET files into the parent directory ({introfiles}\image\)

Now you're ready for the next step.

Preparing the target ROM file

Okay, so there's some setup required to make sure this all goes off without a hitch. First, we'll be making sure we have enough room for the intro. If we need to create more space, we'll be using Lunar Expand to make the ROM the correct size. Lastly, we'll need to take note of where our intro data will be stored within the ROM before we move forward.

Checking the source files for the space required

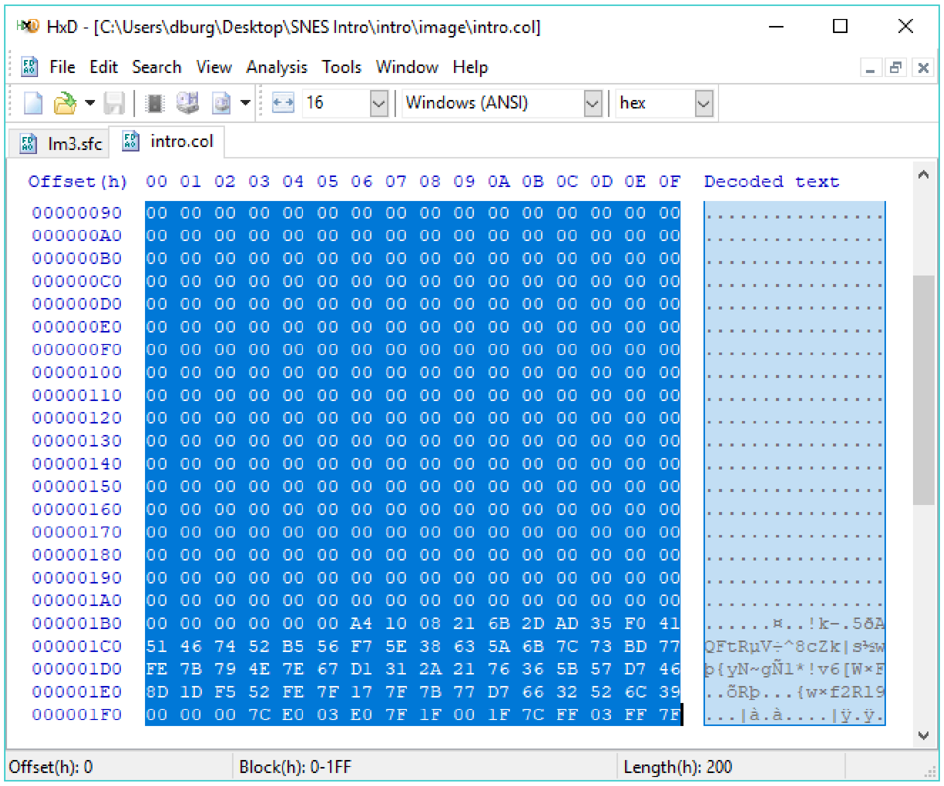
One thing that we need to do before moving forward is to determine if we've got an places inside our target ROM where we could potentially place out intro data.

The first step is opening each of the intro data files in our hex editor and checking the total number of bytes used.

In this case, my intro files take up the following space:

- 0x300 (700) bytes for the assembled code (you can check this by assembling the code to a binary file without inserting it into a ROM)
- 0x3E00 (15,872) bytes for the image data in "intro.set"
- 0x200 (512) bytes for the palette data in "intro.col"
- 0x800 (2048) bytes for the tile map data in "intro.map"

The total for all the data is approximately 0x4B00 (19,200) bytes all together. Note that I've rounded up some sizes a bit just to be safe.



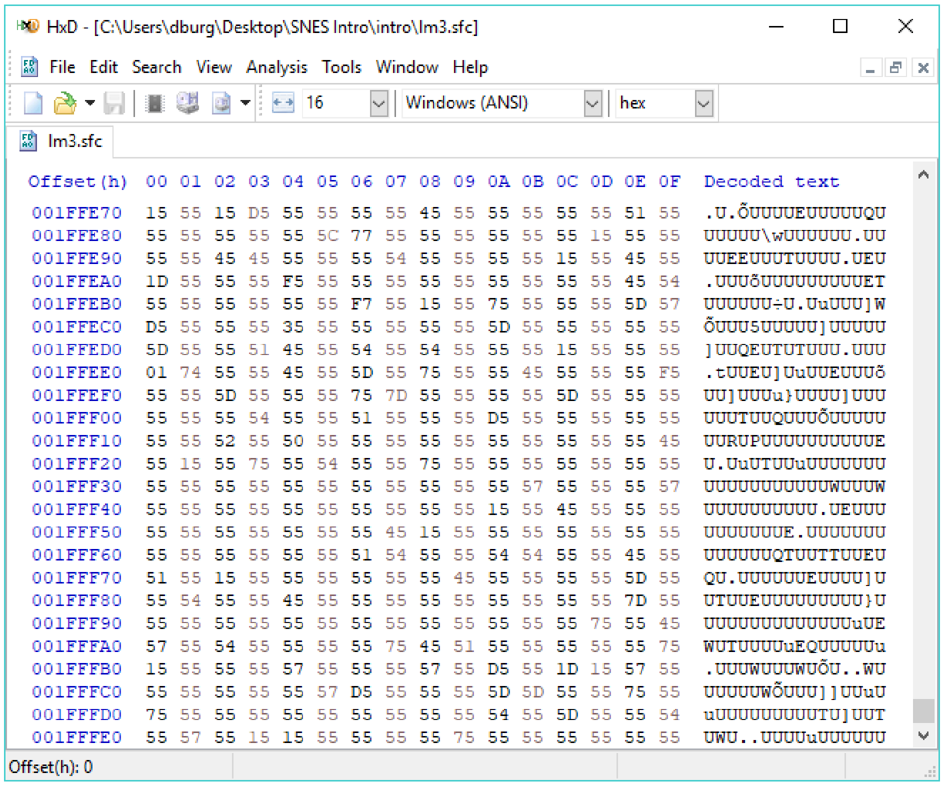
HxD - [C:\Users\dburg\Desktop\SNES Intro\intro\image\intro.col]
File Edit Search View Analysis Tools Window Help
16
Windows (ANSI)
hex
lm3.sfc
intro.col

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000110   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000120   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000130   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000140   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000150   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000160   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000170   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000180   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000190   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001B0   00 00 00 00 00 00 A4 10 08 21 6B 2D AD 35 F0 41  ......¤..!k-.5ðA
000001C0   51 46 74 52 B5 56 F7 5E 38 63 5A 6B 7C 73 BD 77  QFtRµV÷^8cZk|s½w
000001D0   FE 7B 79 4E 7E 67 D1 31 2A 21 76 36 5B 57 D7 46  þ{yN~gÑ1*!v6[W×F
000001E0   8D 1D F5 52 FE 7F 17 7F 7B 77 D7 66 32 52 6C 39  ..õRþ...{w×f2Rl9
000001F0   00 00 00 7C E0 03 E0 7F 1F 00 1F 7C FF 03 FF 7F  ...|à.à....|ÿ.ÿ.

Offset(h): 0          Block(h): 0-1FF          Length(h): 200
```

1 - Open each file and press Ctrl + A to select all bytes. At the bottom of the screen, Length(h) is shown. This value is in hexadecimal format. This is essentially the space required to insert each section of data.

Checking for Free Space in the ROM file

Now that we know how much space our intro data is going to take up, we can open up the target ROM file with the hex editor of our choice. I'll be using HxD in this example. The ROM file I'll be using is Little Master Episode 3.



HxD - [C:\Users\dburg\Desktop\SNES Intro\intro\lm3.sfc]

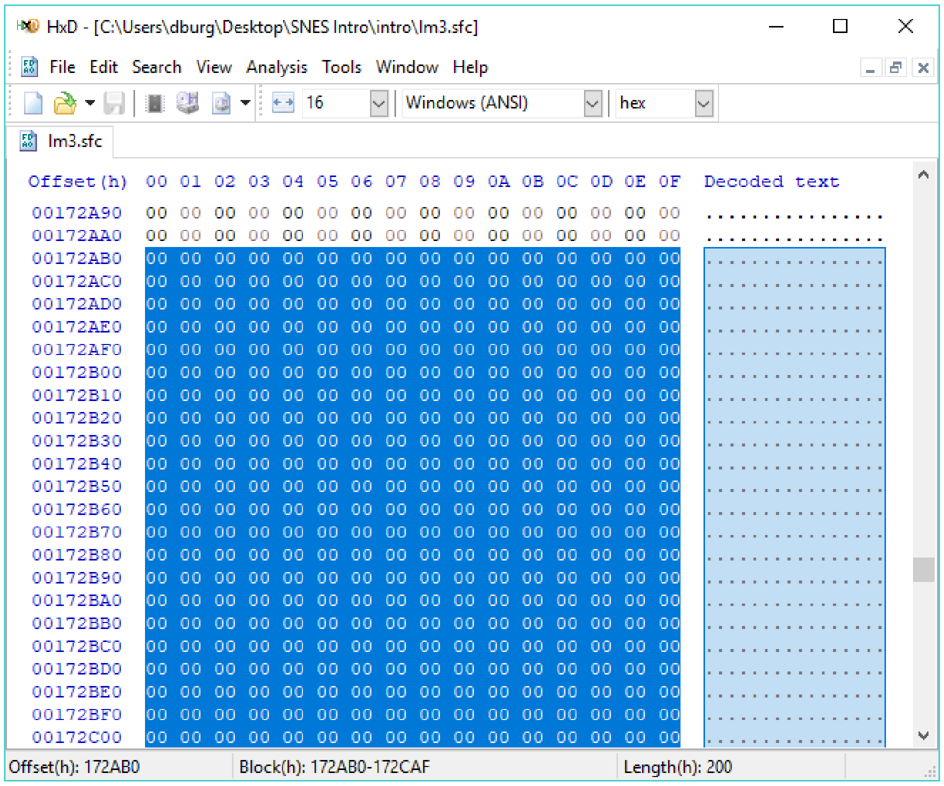
```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

001FFE70  15 55 15 D5 55 55 55 55 45 55 55 55 55 55 51 55  .U.ÕUUUUEUUUUUQU
001FFE80  55 55 55 55 55 5C 77 55 55 55 55 55 55 15 55 55  UUUUU\wUUUUUU.UU
001FFE90  55 55 45 45 55 55 55 54 55 55 55 55 15 55 45 55  UUEEUUUTUUUU.UEU
001FFEA0  1D 55 55 55 F5 55 55 55 55 55 55 55 55 55 45 54  .UUUõUUUUUUUUUET
001FFEB0  55 55 55 55 55 55 F7 55 15 55 75 55 55 55 5D 57  UUUUUU÷U.UuUUU]W
001FFEC0  D5 55 55 55 35 55 55 55 55 55 5D 55 55 55 55 55  ÕUUU5UUUUU]UUUUU
001FFED0  5D 55 55 51 45 55 54 55 54 55 55 55 15 55 55 55  ]UUQEUTUTUUU.UUU
001FFEE0  01 74 55 55 45 55 5D 55 75 55 55 45 55 55 55 F5  .tUUEU]UuUUEUUUõ
001FFEF0  55 55 5D 55 55 55 75 7D 55 55 55 55 5D 55 55 55  UU]UUUu}UUUU]UUU
001FFF00  55 55 55 54 55 55 51 55 55 55 D5 55 55 55 55 55  UUUTUUQUUUÕUUUUU
001FFF10  55 55 52 55 50 55 55 55 55 55 55 55 55 55 55 45  UURUPUUUUUUUUUUE
001FFF20  55 15 55 75 55 54 55 55 75 55 55 55 55 55 55 55  U.UuUTUUuUUUUUUU
001FFF30  55 55 55 55 55 55 55 55 55 55 55 57 55 55 55 57  UUUUUUUUUUUWUUUW
001FFF40  55 55 55 55 55 55 55 55 55 55 15 55 45 55 55 55  UUUUUUUUUU.UEUUU
001FFF50  55 55 55 55 55 55 55 45 15 55 55 55 55 55 55 55  UUUUUUUE.UUUUUUU
001FFF60  55 55 55 55 55 55 51 54 55 55 54 54 55 55 45 55  UUUUUUQTUUTTUUEU
001FFF70  51 55 15 55 55 55 55 55 55 45 55 55 55 55 5D 55  QU.UUUUUUEUUUU]U
001FFF80  55 54 55 55 45 55 55 55 55 55 55 55 55 55 7D 55  UTUUEUUUUUUUUU}U
001FFF90  55 55 55 55 55 55 55 55 55 55 55 55 55 75 55 45  UUUUUUUUUUUUUuUE
001FFFA0  57 55 54 55 55 55 55 75 45 51 55 55 55 55 55 75  WUTUUUUuEQUUUUUu
001FFFB0  15 55 55 55 57 55 55 55 57 55 D5 55 1D 15 57 55  .UUUWUUUWUÕU..WU
001FFFC0  55 55 55 55 55 57 D5 55 55 55 5D 5D 55 55 75 55  UUUUUWÕUUU]]UUuU
001FFFD0  75 55 55 55 55 55 55 55 55 55 54 55 5D 55 55 54  uUUUUUUUUUTU]UUT
001FFFE0  55 57 55 15 15 55 55 55 55 75 55 55 55 55 55 55  UWU..UUUUuUUUUUU
```

Offset(h): 0

2 - This ROM is 0x200000 bytes in length.

Now, if you are familiar with the ROM you are using, you may be able to find bits of free space throughout the ROM in which to insert the intro data.

For example, at offset 0x172AB0, you'll see the following:



```
HxD - [C:\Users\dburg\Desktop\SNES Intro\intro\lm3.sfc]
16    Windows (ANSI)    hex

Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00172A90   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172AA0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172AB0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172AC0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172AD0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172AE0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172AF0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172B00   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172B10   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172B20   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172B30   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172B40   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172B50   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172B60   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172B70   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172B80   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172B90   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172BA0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172BB0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172BC0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172BD0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172BE0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172BF0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00172C00   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Offset(h): 172AB0    Block(h): 172AB0-172CAF    Length(h): 200
```

3 - A bunch of zeros does not necessarily indicate that the space is unused, but there is a chance that it wont be a problem.

If we wanted, we could take note of this offset to use for potentially storing the palette data. In this case since there's about 0x200 bytes of (likely) "free" space here.

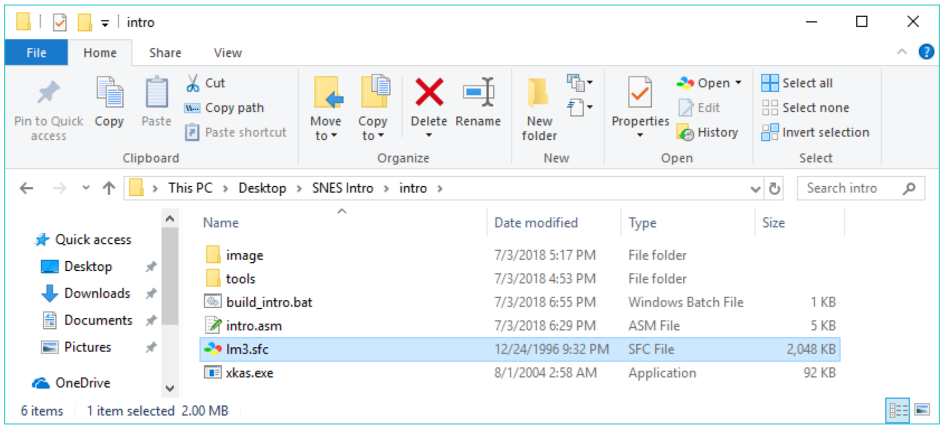
This is just an example, and it would take some familiarity with the way this area of the ROM is being used in the original code to say for sure whether placing the palette data here would actually work as intended.

If you are unsure, it's usually fastest just to expand the ROM and to place your intro data in the expanded area.

Expanding the ROM (optional)

So, you've decided to expand the usable space in the ROM file. Where do we start? While the SNES hardware is generally unconcerned about the actual amount of data on the cartridge (unless it's less than the program expects), emulators, on the other hand, tend to bomb out immediately if the ROM file is not one of the standard sizes.

We'll be using the tool, Lunar Expand to extend the amount of space available for our intro. Before we open the tool, let's check how large the ROM file we are using is to begin with.



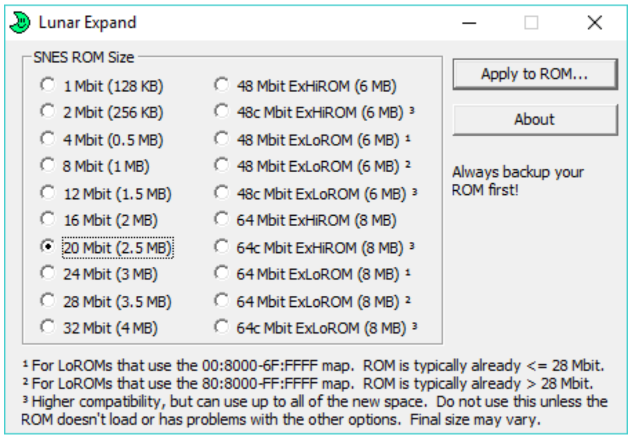
Name	Date modified	Type	Size
image	7/3/2018 5:17 PM	File folder	
tools	7/3/2018 4:53 PM	File folder	
build_intro.bat	7/3/2018 6:55 PM	Windows Batch File	1 KB
intro.asm	7/3/2018 6:29 PM	ASM File	5 KB
lm3.sfc	12/24/1996 9:32 PM	SFC File	2,048 KB
xkas.exe	8/1/2004 2:58 AM	Application	92 KB

6 items   1 item selected 2.00 MB

4 - Selecting the file in Windows Explorer will show the total size in MB at the bottom of the window.

In the example, the ROM's size is 2 Megabytes (16 Megabits, or 2048 Kilobytes). The next step is to fire up Lunar Expand, and select the final size we want for our ROM.

The next size up from 16 Megabits is 20 Megabits, so that's what we'll choose. To finish up, click "Apply to ROM...", select the ROM file and then open. Now the ROM has been expanded.



SNES ROM Size

1 Mbit (128 KB)	48 Mbit ExHiROM (6 MB)
2 Mbit (256 KB)	48c Mbit ExHiROM (6 MB) ³
4 Mbit (0.5 MB)	48 Mbit ExLoROM (6 MB) ¹
8 Mbit (1 MB)	48 Mbit ExLoROM (6 MB) ²
12 Mbit (1.5 MB)	48c Mbit ExLoROM (6 MB) ³
16 Mbit (2 MB)	64 Mbit ExHiROM (8 MB)
[✓ 20 Mbit (2.5 MB)]	64c Mbit ExHiROM (8 MB) ³
24 Mbit (3 MB)	64 Mbit ExLoROM (8 MB) ¹
28 Mbit (3.5 MB)	64 Mbit ExLoROM (8 MB) ²
32 Mbit (4 MB)	64c Mbit ExLoROM (8 MB) ³

Always backup your ROM first!

¹ For LoROMs that use the 00:8000-6F:FFFF map. ROM is typically already <= 28 Mbit.
² For LoROMs that use the 80:8000-FF:FFFF map. ROM is typically already > 28 Mbit.
³ Higher compatibility, but can use up to all of the new space. Do not use this unless the ROM doesn't load or has problems with the other options. Final size may vary.

It's a good idea to make sure the ROM is still functioning correctly after the expansion procedure and doing any more work. Fire the ROM file up in an emulator and make sure it's still running properly. Once you're sure it's working as it was before the expansion, we can move on confidently.

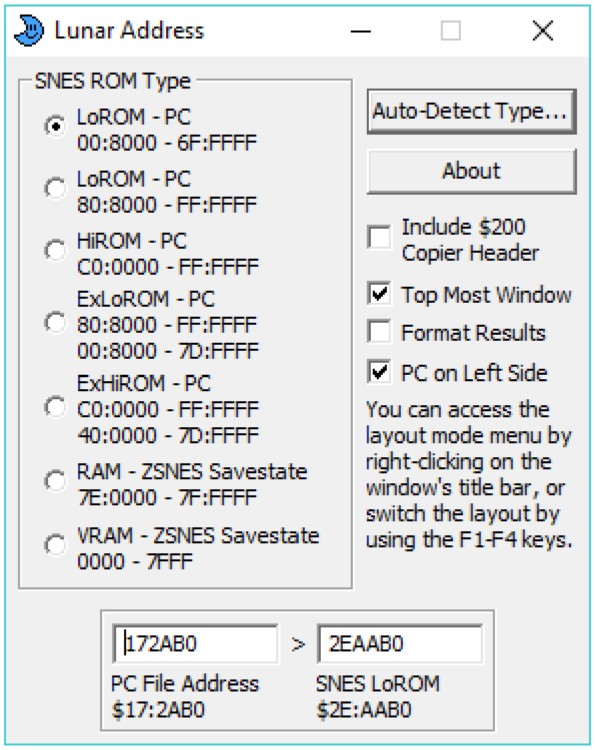
Converting PC ROM Addresses to SNES ROM Addresses

When converting PC addresses to SNES addresses or vice-versa, it's helpful to use Lunar Address. This just simplifies the address conversion and minimizes errors.

Keeping in mind any of the addresses we may have identified in previous steps, we can easily convert the PC address to something we can use in our code by using these steps:

- Load up Lunar Address
- If you know the addressing mode used by your ROM, you can select it. If unsure, you can use the auto-detect feature. Keep in mind that this feature does not always work as intended for all ROM types.

- If you've got the PC Address, type it in the PC Address side. The tool will automatically convert it to the specified SNES Addressing. The opposite is true. Typing in an address on the SNES side will generate the PC equivalent. (This always uses Hexadecimal notation.)



SNES ROM Type
- [✓ LoROM - PC 00:8000 - 6F:FFFF]
- LoROM - PC 80:8000 - FF:FFFF
- HiROM - PC C0:0000 - FF:FFFF
- ExLoROM - PC 80:8000 - FF:FFFF 00:8000 - 7D:FFFF
- ExHiROM - PC C0:0000 - FF:FFFF 40:0000 - 7D:FFFF
- RAM - ZSNES Savestate 7E:0000 - 7F:FFFF
- VRAM - ZSNES Savestate 0000 - 7FFF

- [ ] Include $200 Copier Header
- [✓] Top Most Window
- [ ] Format Results
- [✓] PC on Left Side

You can access the layout mode menu by right-clicking on the window's title bar, or switch the layout by using the F1-F4 keys.

172AB0 > 2EAAB0

PC File Address $17:2AB0

SNES LoROM $2E:AAB0

For example, in the previous section, we identified an area of free space at PC Offset 0x172AB0. To get the SNES analog of this location, type in the address without the prefix "0x" (The "0x" just signifies we're talking about a hexadecimal number.) So, "172AB0." If you are using the Little Master 3 ROM, as in the example, the correct ROM type is LoROM (the first one). This value is converted to 2EAAB0.

Bear this in mind when converting Addresses for use in the next section.

Preparing and Building the Code

Now, we're finally ready to prepare our code. We've just got to gather some information about where to hijack the existing code, and where exactly we want to place the new data for the intro within the ROM.

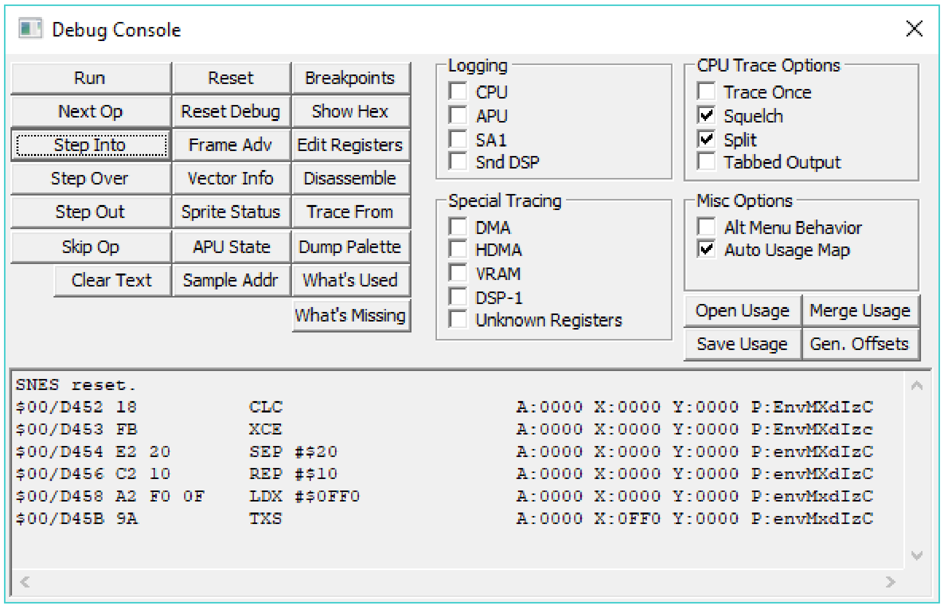
Additionally, we'll be covering basic use of the debugger, how to change variables in the code, and how to assemble our code.

Using the Debugger to find the Original Startup Code

Using a debugger might seem a bit overwhelming at first, but you'll get used to it once you start to understand how it can be utilized to help you accomplish various tasks that would be nearly impossible without it.

- Load up your debugger. In the example, we'll use Geiger's SNES9x Debugger.

- Open the ROM. In this example, I'm using Little Master Episode 3. Note that code execution is paused by default and the debugger controls are shown.



Logging: [ ] CPU [ ] APU [ ] SA1 [ ] Snd DSP
CPU Trace Options: [ ] Trace Once [✓ Squelch] [✓ Split] [ ] Tabbed Output
Special Tracing: [ ] DMA [ ] HDMA [ ] VRAM [ ] DSP-1 [ ] Unknown Registers
Misc Options: [ ] Alt Menu Behavior [✓ Auto Usage Map]

```
SNES reset.
$00/D452 18          CLC                    A:0000 X:0000 Y:0000 P:EnvMXdIzC
$00/D453 FB          XCE                    A:0000 X:0000 Y:0000 P:EnvMXdIzc
$00/D454 E2 20       SEP #$20               A:0000 X:0000 Y:0000 P:envMXdIzC
$00/D456 C2 10       REP #$10               A:0000 X:0000 Y:0000 P:envMXdIzC
$00/D458 A2 F0 0F    LDX #$0FF0             A:0000 X:0000 Y:0000 P:envMxdIzC
$00/D45B 9A          TXS                    A:0000 X:0FF0 Y:0000 P:envMxdIzC
```

5 - The SNES address of each instruction is shown on the left of the debugger output.

- Notice the SNES address that is shown next to the first instruction "CLC" reads "$00/D452." This is where the current instruction is being executed from. This is essentially the same as **$00D452,** where "00" is the bank, and "D452" is the offset within that bank. Write this down.

- Clicking on "Step Into" several times will show each instruction as it is executed and the address of execution.

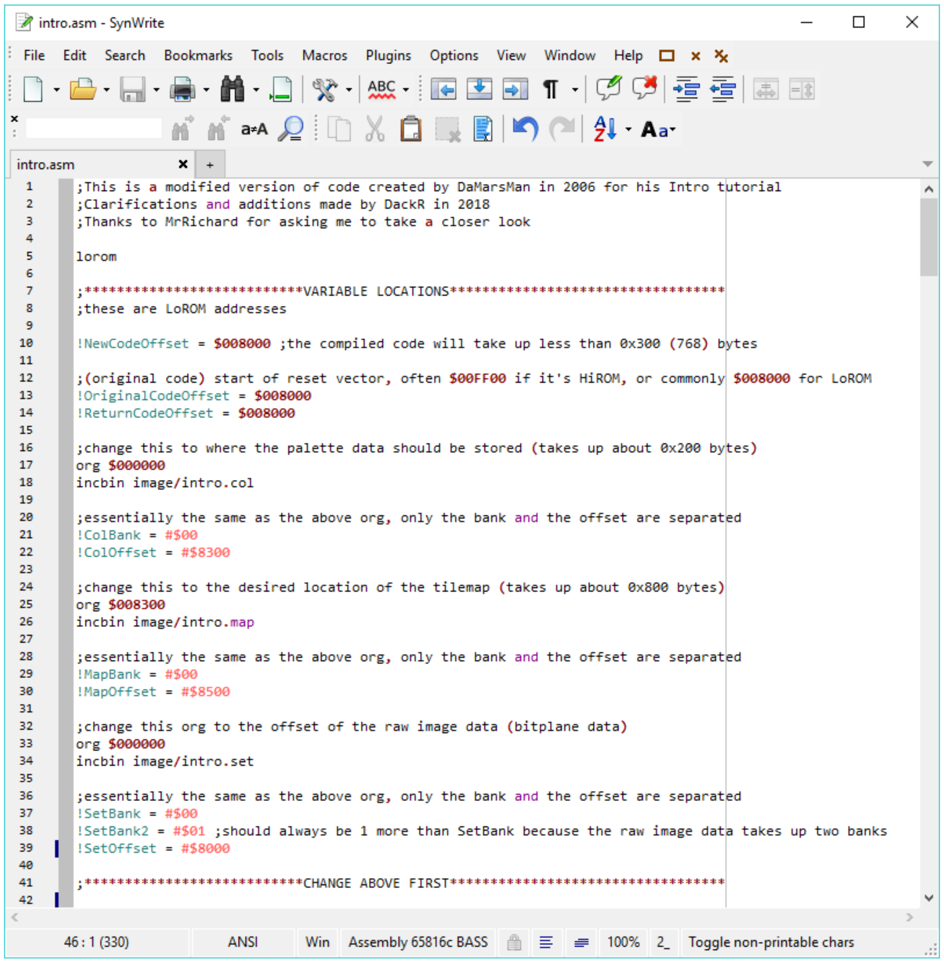
- Take note of the first few instructions. We will need them to finish our intro code.

Notice the number of bytes used for the original instructions. The actual byte representation for each instruction is shown next to the SNES address. Our hijacking code, (JMP [LONGADDRESS]) when assembled, will take up approximately 4 bytes and will be replacing the first 3 instructions in this example. That means that we need to reproduce the original 3 instructions within our code before jumping back to the original 4th instruction. I'll explain this more when the time comes.

For now, just write down the address of the first instruction (**$00D452**), the first 3 instructions (**CLC, XCE, SEP #$20**), and the address of the fourth instruction (**$00D456**). We will be using these in our code.

Check out the example code

Take a look at the intro.asm code in your favorite text editor. This example uses SynWrite because of its customizable syntax highlighting features.


intro.asm - SynWrite
File Edit Search Bookmarks Tools Macros Plugins Options View Window Help
intro.asm
46 : 1 (330) ANSI Win Assembly 65816c BASS 100% 2_ Toggle non-printable chars

```
;This is a modified version of code created by DaMarsMan in 2006 for his Intro tutorial
;Clarifications and additions made by DackR in 2018
;Thanks to MrRichard for asking me to take a closer look

lorom

;**************************VARIABLE LOCATIONS***********************************
;these are LoROM addresses

!NewCodeOffset = $008000 ;the compiled code will take up less than 0x300 (768) bytes

;(original code) start of reset vector, often $00FF00 if it's HiROM, or commonly $008000 for LoROM
!OriginalCodeOffset = $008000
!ReturnCodeOffset = $008000

;change this to where the palette data should be stored (takes up about 0x200 bytes)
org $000000
incbin image/intro.col

;essentially the same as the above org, only the bank and the offset are separated
!ColBank = #$00
!ColOffset = #$8300

;change this to the desired location of the tilemap (takes up about 0x800 bytes)
org $008300
incbin image/intro.map

;essentially the same as the above org, only the bank and the offset are separated
!MapBank = #$00
!MapOffset = #$8500

;change this org to the offset of the raw image data (bitplane data)
org $000000
incbin image/intro.set

;essentially the same as the above org, only the bank and the offset are separated
!SetBank = #$00
!SetBank2 = #$01 ;should always be 1 more than SetBank because the raw image data takes up two banks
!SetOffset = #$8000

;**************************CHANGE ABOVE FIRST***********************************
```

6 - intro.asm

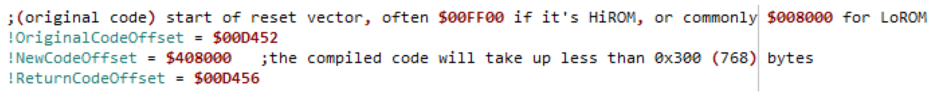
Here's a quick rundown of what the code is doing:

- The original initialization code will be hijacked by our intro code.
- A basic SNES initialization procedure is performed.
- The tilemap is DMA transferred into VRAM.
- The palette data is DMA transferred into VRAM.
- The binary image data is DMA transferred into VRAM.
- The image fades in.
- Short pause (or wait for keypress).
- The image fades out.
- The first few instructions that were overwritten by our hijack routine are executed.
- Return to original code and resume execution.

Changing Variables

The first part of the example code holds all the variables we will need to change to make the intro work properly. Let's go through the variables one at a time. I'll be showing you how to change these to make it work for Little Master 3.

- **!OriginalCodeOffset** defines the SNES address where the original code started. In an earlier section, we found this address and wrote it down. This should be **$00D452** for the Little Master 3 ROM.
- **!NewCodeOffset** defines the SNES address that will will be jumping to in order to hijack or override the original code. Go ahead and set this value to the very top of the expanded area of the ROM. In this example, we noted that this free space starts PC $200000 (don't use the PC address). If you enter this address in LunarAddress, it will convert to **$408000**. This is the value you want in this example.
- **!ReturnCodeOffset** defines the SNES address that our code will point back to after our intro code has finished execution. We wrote this address down earlier and it should be **$00D456**.



```
;(original code) start of reset vector, often $00FF00 if it's HiROM, or commonly $008000 for LoROM
!OriginalCodeOffset = $00D452
!NewCodeOffset = $408000   ;the compiled code will take up less than 0x300 (768) bytes
!ReturnCodeOffset = $00D456
```

7 - This is how the updated variables should look in this example.

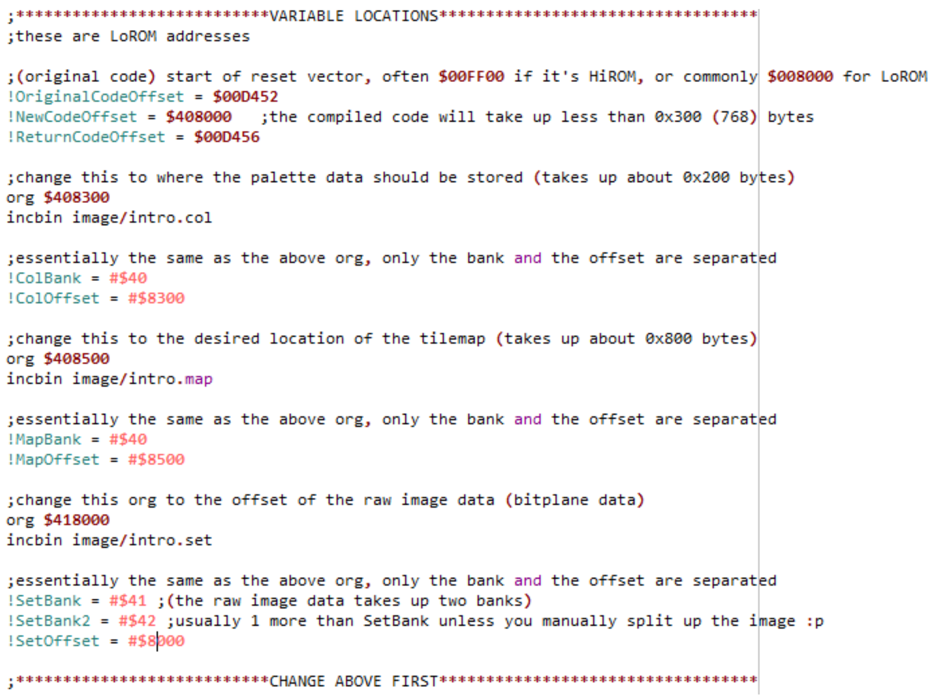
Next, we need to define where the palette data will be stored in the ROM. Since we've already taken time to determine how much space each piece of data will be taking up, this simplifies our process. Our assembled code is being inserted at offset $408000 and will take up approximately 0x300 bytes. This means that we have about 0x7CFF bytes remaining in bank 40 that can be utilized for our intro data. This should be more than enough for our palette data and tilemap data.

Let's place the palette data directly after our code at offset **$408300.** Now change the first org address to this value. Likewise, we need to change **!ColBank** to the hex value of the bank where the palette data is stored (**#$40**). **!ColOffset** should be the hex value of the offset within the bank that is home to the palette data (**#$8300**).

We will determine the location of our tilemap data in a similar way as the palette data. Just place it right after our palette data. Since the palette data only takes up about 0x200 bytes, that means the tilemap can be stored at **$408500**. Change the org address for the tilemap to this. Now we must change **!MapBank** to **#$40** and **!MapOffset** to **#$8500** just like we did for the palette variables.

Now we will determine the location of the actual image data. To make things simpler, we will be splitting up the image into 2 banks. Since we already have data at bank $40, lets use banks $41 and $42 to store our image. This translates to our image starting at **$418000**. Change the org for the intro image to this value. Now **!SetBank** becomes **#$41**, **!SetBank2** becomes **#$42**, and **!SetOffset** becomes **#$8000**.

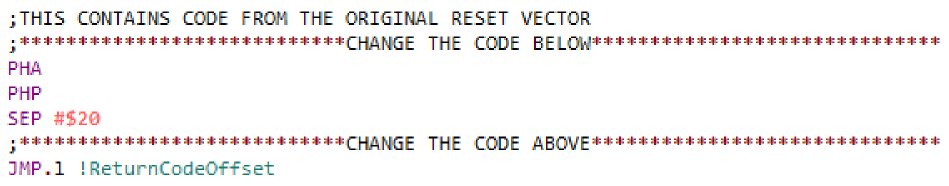
Now you should have something like this:



```
;**************************VARIABLE LOCATIONS*********************************
;these are LoROM addresses

;(original code) start of reset vector, often $00FF00 if it's HiROM, or commonly $008000 for LoROM
!OriginalCodeOffset = $00D452
!NewCodeOffset = $408000   ;the compiled code will take up less than 0x300 (768) bytes
!ReturnCodeOffset = $00D456

;change this to where the palette data should be stored (takes up about 0x200 bytes)
org $408300
incbin image/intro.col

;essentially the same as the above org, only the bank and the offset are separated
!ColBank = #$40
!ColOffset = #$8300

;change this to the desired location of the tilemap (takes up about 0x800 bytes)
org $408500
incbin image/intro.map

;essentially the same as the above org, only the bank and the offset are separated
!MapBank = #$40
!MapOffset = #$8500

;change this org to the offset of the raw image data (bitplane data)
org $418000
incbin image/intro.set

;essentially the same as the above org, only the bank and the offset are separated
!SetBank = #$41 ;(the raw image data takes up two banks)
!SetBank2 = #$42 ;usually 1 more than SetBank unless you manually split up the image :p
!SetOffset = #$8000

;**************************CHANGE ABOVE FIRST*********************************
```

Before we assemble our code, we need to add the instructions that were overwritten with the new hijacking routine introduced by our code.

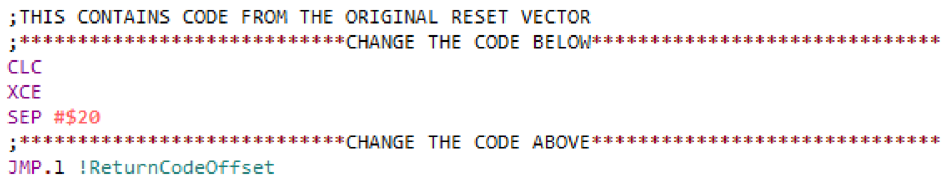
We wrote these instructions down in an earlier section. Scroll down until you see this:



```
;THIS CONTAINS CODE FROM THE ORIGINAL RESET VECTOR
;*****************************CHANGE THE CODE BELOW******************************
PHA
PHP
SEP #$20
;*****************************CHANGE THE CODE ABOVE******************************
JMP.l !ReturnCodeOffset
```

Replace the code in this section with the instructions we wrote down previously.

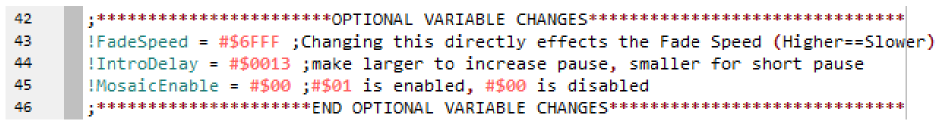
When you are finished, it should look like this:



```
;THIS CONTAINS CODE FROM THE ORIGINAL RESET VECTOR
;*****************************CHANGE THE CODE BELOW******************************
CLC
XCE
SEP #$20
;*****************************CHANGE THE CODE ABOVE******************************
JMP.l !ReturnCodeOffset
```

Optional Code Changes

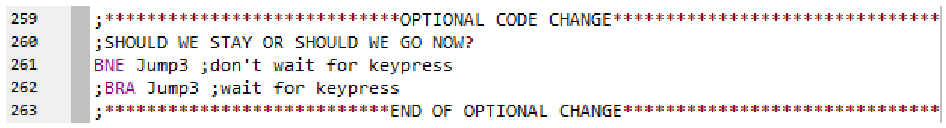
There are some optional features built into the example code that you are now able to modify or take advantage of. Three of them are controlled by variables, and one of them is controlled by a single change to code. You may wish to wait until you have successfully run xkas and tested your changes before making any optional modifications.



```
;**********************OPTIONAL VARIABLE CHANGES******************************
!FadeSpeed = #$6FFF ;Changing this directly effects the Fade Speed (Higher==Slower)
!IntroDelay = #$0013 ;make larger to increase pause, smaller for short pause
!MosaicEnable = #$00 ;#$01 is enabled, #$00 is disabled
;********************END OPTIONAL VARIABLE CHANGES****************************
```

- **!FadeSpeed** - This controls the rate that the screen fades in and out. It also changes the rate of the mosaic effect when it's enabled. A higher value means a slower effect. Only change the high byte on this value or you may get unexpected results.
- **!IntroDelay** - This changes the amount of time that the image will appear on the screen before the transition begins.
- **!MosaicEnable** - As the name suggests, this enables or disables the mosaic effect during fade-in and fade-out.

The last optional change in the example code allows you to pause the intro until a button is pressed.



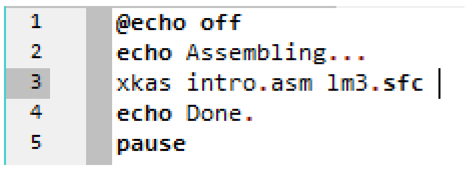
```
;****************************OPTIONAL CODE CHANGE******************************
;SHOULD WE STAY OR SHOULD WE GO NOW?
BNE Jump3 ;don't wait for keypress
;BRA Jump3 ;wait for keypress
;***************************END OF OPTIONAL CHANGE*****************************
```

In the code above, you simply need to comment out one line and remove the comment mark (; - a semicolon) in order to allow the code to wait for a button press before beginning the fade-out transition.

Build the Code with xkas

Now that we've got the code ready, and we have all of our files in the appropriate locations, we can actually run the assembler.

First, we need to make sure the batch file named "build_intro.bat" in the {introfiles} folder is configured properly. Open it up in the text editor of your choice.



```
@echo off
echo Assembling...
xkas intro.asm lm3.sfc
echo Done.
pause
```

Make sure the name of the ".asm" file is correct and make sure the ".sfc" file is the same as the ROM you are modifying. If you followed the turorial correctly, the supplied values should be correct.

Next, ensure the ROM file named in the batch file is in the same folder.

Run the batch file. It should assemble very quickly. Errors will be shown if they are detected by xkas.

Now all that is left is to test the ROM after our changes. If all of the steps were followed exactly, then the intro image should be shown as expected.

If all went as planned, you should now have an intro for the ROM of your choice. Questions? Comments? Check out the youtube video (link at the beginning of the tutorial) and leave any feedback there.