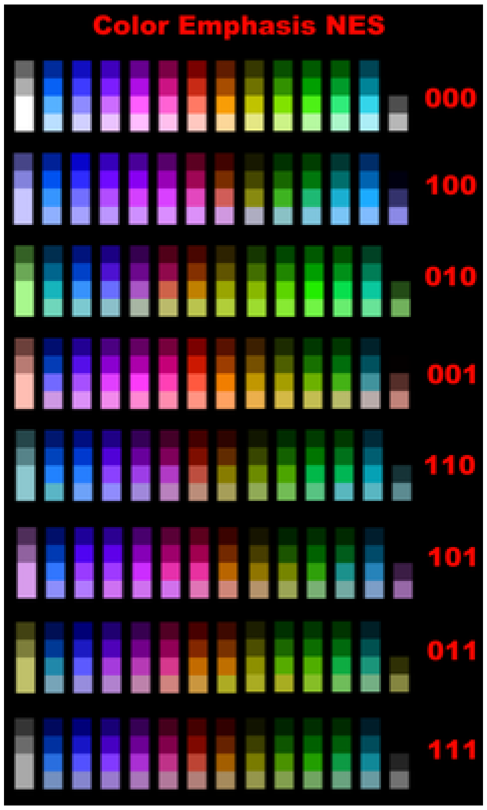
22.More

I wanted to make sure I covered everything. I barely mentioned mappers. If you want to make a game bigger than $8000 bytes, you would have to use a better mapper than NROM256. That would allow you to switch PRG-ROM banks in-game, and/or CHR-ROM banks. CC65/CA65 is the perfect tool for setting up the different banks. I wish I could provide source code for this, but I mostly work with small games. There are dozens of mappers. I think MMC3 is a good one. It allows both CHR and PRG swapping, and has a scanline counter.

How to get a game on a cart. You can get a usb / flash cart, such as the Everdrive or the PowerPak. You could also burn your own EPROM and solder them into an actual cartridge, but that's a little bit above my skill level at the moment.

I skipped over Color Emphasis bits of register $2001. The 3 high bits are B, G, R…and they emphasize those colors by slightly darkening the other colors. Set all the bits will darken the whole screen. Here's what the palette looks like under various color emphasis bits (Using an Emulator)…



Additional cc65 features.

Ullrich von Bassewitz has apparently added the entire C standard library, and some other functions. It seems you can multiply and divide numbers, it's just very slow (perhaps a table of presolved answers would be faster).

If you #include "..\include\stdlib.h"

It looks like you can use calloc, malloc, free, and realloc. (I had a hard time testing these…I had to doubly define __STACK_SIZE__ and __STACKSIZE__ in the .cfg file. It seems to me like one of those is a typo, but I've seen it both ways.) You will also have to define the HEAP to use these functions…here's a link that talks about setting up the HEAP…

http://www.cc65.org/mailarchive/2009-05/6581.html (http://www.cc65.org/mailarchive/2009-05/6581.html)

I highly recommend that you DON'T use any heap functions. They produce the slowest code. And, C code on the NES is already notoriously slow.

Also interesting is rand (Random Numbers), srand (to seed the random number). And qsort (QuickSort).

If you #include "..\include\cc65.h"

You can use cc65_sin and cc65_cos (Sine and Cosine).

And if you #include "..\include\zlib.h"

You can use the built in decompression library. (I haven't tested it).

————————————————

I also never covered the IRQ. It works like NMI. It's an interrupt that stops the normal code, and jumps to the code (defined by the IRQ vector, at FFFE-FFFF). There are 3 main ways to encounter an IRQ…

1. if the code encounters a BRK instruction (opcode #00)
2. music channel IRQ is turned on, and DMC sample ends
3. mapper generated IRQ, like MMC3's scanline counter

Only the mapper generated ones seem useful to me, as this could reliably change many PPU settings midframe with great precision…or swap tiles midframe, so you can have different tiles at the top of the screen from the bottom. Etc.

————————-

One last note. Some emulators cut off the top and bottom 8 pixels. The NES produces the full 240 pixel high image, but due to the way old TVs projected their image, they would tend to cut off 8-16 pixels off the top and bottom of the screen. Newer TVs might not cut any pixels off the top or bottom. Long story short, don't put anything important on the top or bottom 16 pixels of the screen.

And that's all I can think of right now. Go make some games. I bet you can make something better than I just did.

…

Oh, I thought of one more important thing I forgot to mention. Reading from the PPU works the same as writing to the PPU. Write the upper PPU address to $2006, lower PPU address to $2006, then READ from $2007 (LDA $2007). But, one important detail. The first read from the PPU is garbage*. The next read will be from the address you specified. The next after that will be one to the RIGHT. Etc. I didn't design the NES, so you'd have to ask the engineers why this is so. And, like writing, you can only read from the PPU during V-blank or while rendering is OFF.

*This is true everywhere except for the palettes…PPU addresses $3F00-$3FFF. The first read will be true.

December 27, 2015February 18, 2016 dougfraker