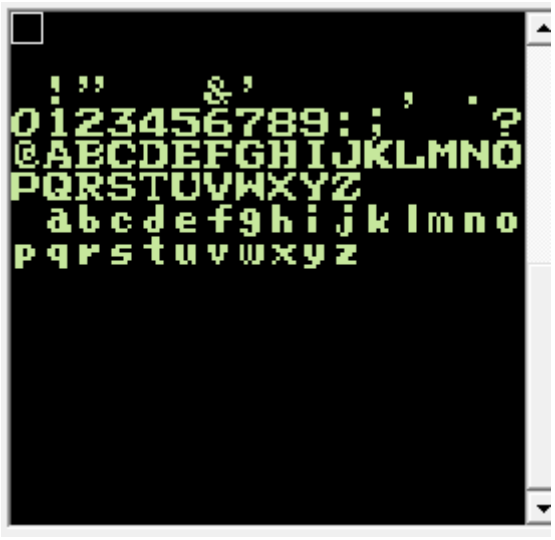
nesdoug

3. Our first program

First, we are going to print some words on the background. The NES is not like a computer, it doesn't know how to draw letters, or anything about ASCII. All it does for backgrounds is shuffle some 8×8 graphic tiles onto a background map (like puzzle pieces).

So, how do we write text on the screen? We make some graphics that look like letters, and organize them to correspond to their positions on the ASCII map. Then we can reference them in our C code using the actual letters. (Inside the reset.s file, I wrote .incbin "Alpha.chr" at the end to append the graphics to the end of the ROM).



(Any other graphics, we would reference it by using its tile #).

The start-up code is already provided, after that we jump to main (). We want to do these things…

1. Turn off the screen
2. Set a color palette
3. Write a few words to the background
4. Reset the scroll
5. Turn on the screen
6. Infinite loop

Why do we turn off the screen? The NES can only make changes to the screen either during V-blank or while the screen is off, otherwise it will draw gibberish on the screen (and probably misalign the rest of the screen). That includes changing the palette, the background tiles, or changing the sprites.

The start-up code has inserted zeros into everything, so the screen will be filled with tile #0 (in our case, blank). And the palette will be filled with gray.

To write to the screen (PPU), we first write the address (high byte, then low) to $2006. Once our starting address is set, we can start sending our data by writing to $2007. The PPU auto-increments. Each subsequent write to $2007, will go one more to the right, eventually wrapping around to the next line.

Note, the PPU can also be set to increment by 32, which is the next space directly below. We have it set (PPU control register $2000) to increment by 1, one to the right.

Sending data to a nametable will tell the PPU to draw that tile in that position. Like a puzzle piece being placed on a grid.

Try using NES screen tool, and it will tell you the current PPU address of the tile position.

We also need to send color data to the PPU…address $3f00. We only need to fill the first 4 colors of the color palette (= the first bg palette).

Writing to the PPU, also screws up the scroll position of the background, so it needs to be reset, or the text will show up on the wrong position on the screen (or perhaps, just off screen, where you can't see it). So, when we're done, we write zero to $2006 twice (and also to $2005 twice, the actual scroll registers).

Here's our code…

```
#define PPU_CTRL  *((unsigned char*)0x2000)
#define PPU_MASK  *((unsigned char*)0x2001)
#define PPU_STATUS  *((unsigned char*)0x2002)
#define SCROLL   *((unsigned char*)0x2005)
#define PPU_ADDRESS  *((unsigned char*)0x2006)
#define PPU_DATA  *((unsigned char*)0x2007)



unsigned char index;



const unsigned char TEXT[]={
"Hello World!"};



const unsigned char PALETTE[]={
0x1f, 0x00, 0x10, 0x20
}; //black, gray, lt gray, white
```

```
void main (void) {
 // turn off the screen
 PPU_CTRL = 0;
 PPU_MASK = 0;

 // load the palette
 PPU_ADDRESS = 0x3f; // set an address in the PPU of 0x3f00
 PPU_ADDRESS = 0x00;
 for(index = 0; index < sizeof(PALETTE); ++index){
  PPU_DATA = PALETTE[index];
  }

 // load the text
 PPU_ADDRESS = 0x21; // set an address in the PPU of 0x21ca
 PPU_ADDRESS = 0xca;  // about the middle of the screen
 for( index = 0; index < sizeof(TEXT); ++index ){
  PPU_DATA = TEXT[index];
  }

 // reset the scroll position
 PPU_ADDRESS = 0;
 PPU_ADDRESS = 0;
 SCROLL = 0;
 SCROLL = 0;

 // turn on screen
 PPU_CTRL = 0x90; // NMI on
 PPU_MASK = 0x1e; // screen on

 // infinite loop
 while (1);
}
```

Here's a link to the source code and graphic tiles, ready to be compiled.

http://dl.dropboxusercontent.com/s/1oxcgi4t1m4ifzj/lesson1.zip (http://dl.dropboxusercontent.com/s/1oxcgi4t1m4ifzj/lesson1.zip)

Note: since an update of cc65, I have added the line…

ONCE: load = PRG, type = ro, optional = yes;

…to all the .cfg files, inside segments{}.

You might ask, "how do we write to the background during V-Blank?" But, we'll cover that next time.

For more information about why "PPUMASK = 0x1e" turns on the screen, see the wiki…

http://wiki.nesdev.com/w/index.php/PPU_registers (http://wiki.nesdev.com/w/index.php/PPU_registers)

Final note: All the example files are 0x4000 in size. That is the smallest possible PRG ROM size. 90% of games are larger than this, and they all map to the 0x8000-0xffff section of the CPU memory. Since I am using such a small ROM, it is actually loaded to 0xc000-0xffff (and mirrored at 0x8000-0xbfff). When you make larger games, you will certainly want to change the .cfg file so that the PRG ROM starts at 0x8000 and is 0x8000 in size. AND, change the 'header' segment, to indicate 2 banks of PRG ROM.

November 17, 2015April 16, 2017 dougfraker

25 thoughts on "3. Our first program"

1. **PinkyinParis** says:

January 5, 2016 at 11:14 pm Edit
Hi, have a noob question. I get this error message:

ld65.exe: Warning: Option `-o' should precede options `-t' or `-C'
ld65.exe: Error: Object file `_afailed.o' in library `nes.lib' has wrong version

I edited the compile.bat to put the -o before the -C but (unsurprisingly) got another error message:

ld65.exe: Error: File `nes.cfg' has unknown type

Do I need an earlier / different version of cc65? I've just got the cc65-snapshot-win32 in a folder under my user folder and I put the lesson folder inside the cc65 folder. Should I have it somewhere else? Do I need to set paths or something? Is there some other thing I'm not getting?

Reply

dougfraker says:
January 6, 2016 at 2:14 am Edit
There are multiple versions of ca65, and they are not compatible with each other. Use the nes.lib that came with your version. Also, find the ld65 html file (and see if they explain how to set up the cfg file). Sorry I can't be more helpful.

2. Reply

PinkyinParis says:
January 6, 2016 at 10:35 pm Edit
Thank you. I've got it working tentatively, I've bit the bullet and read the cc65 docs, so now I'll go through this again and follow along properly. Thanks for this. This tutorial makes sense even to me 🙂

3. Reply

RG says:
February 15, 2016 at 9:02 am Edit
You say:

"Then we can write our data by writing to $2007. We have it set so that each write to $2007 will go 1 more to the right. It will keep doing that until we write a new address to $2006."

Perhaps it's just because I don't work in C all that much, but I'm a little confused by this. Why exactly does the address increment itself after each write? Does it do this by default or is it something special in the syntax?

Thanks

Reply

dougfraker says:
February 15, 2016 at 4:15 pm Edit
Good question. This is unrelated to the C language. The NES PPU works like this…
Writing to the CPU address $2000 controls the PPU. Each bit does something different, but xxxx x0xx will set it to increment the PPU by 1 automatically after every write, and xxxx x1xx will set it to increment the PPU by 32 after every write (which is the tile directly below the last one, since the screen is 32 tiles wide).

Example…Write $23 to $2006 then $00 to $2006. The first write to $2007 will store that data at PPU address $2300. If the PPU CONTROL BIT is 0, the next write to $2007 will store data at $2301. Then $2302.
If the PPU CONTROL BIT is 1, the first write to $2007 will go to $2300. The next write to $2007 will go to $2320. Then $2340. Etc. ($20 is hexadecimal for 32).
Hope that's not too confusing. See the wiki for more detailed answers…wiki.nesdev.com

4. Reply

SD says:
June 8, 2016 at 4:56 pm Edit
Hi Doug.. Please help I installed cc65, and the FCEUX emulator then I was getting the Wrong data version in 'nes.lib' error on compile, I replaced it with my version of nes.lib as u suggested.. Now I am getting an error

"ld65.exe:Error: nes.cfg(71): Attribute expected" please help resolve this

Reply

dougfraker says:
June 8, 2016 at 10:10 pm Edit
line 71 is the symbol _stack_size_, I'm not sure what attribute it's expecting…they keep changing cc65(ld65). Check with the ld65.html file that came with your version.

Reply

dougfraker says:
June 8, 2016 at 10:21 pm Edit
The only way I was able to reproduce your error…was by deleting a colon or semicolon. Are you sure you didn't accidentally change the .cfg file?

1. **SD** says:
June 9, 2016 at 2:17 pm Edit
Hey Doug.. I figured it out I had to change the code to this in line 70:
SYMBOLS {
__STACK_SIZE__: value = $0100, weak = yes;
__STACK_START__: value = $700, weak = yes;
}
It was indeed the ld65 using a new format. Thanks.

5. **DoNotWant** says:
August 9, 2016 at 8:15 am Edit
Hey, don't know if you fixed this in later tutorials, but you have to read PPU_STATUS before setting the PPU_ADDRESS.

Just doing PPU_STATUS; won't work, it will be optimized away, and since the volatile keyword isn't properly implemented in cc65, changing the PPU_STATUS definition to *((volatile unsigned char*)0x2002) will only remove the warning about the statement PPU_STATUS; having no effect.
I tried asm("lda $2002"); but that also seems to be optimized away.

Reply

dougfraker says:
August 9, 2016 at 2:53 pm Edit

Actually, as long as you only use 'pairs' of 2005 and/or 2006 writes, you only need to read from 2002 once at the beginning of the program, which I do, in the startup code.

Yes, volatile is broken in cc65. And, with -O command (see compile.bat), it will optimize out your asm statement, because it feels like you're not using the value read from 2002. You can remove the -Oi optimizations, and it will fix that issue. Or you can write a function in asm (in a separate asm module), in which the compiler never sees the actual function, so it can't optimize it out. The only thing you need in the c code is a prototype of the function and a call to the function.

6. Reply

MrEddy says:
October 19, 2016 at 8:21 pm Edit
I had to add this line in the SEGMENTS section of the nes.cfg file in order to compile :

ONCE: load = PRG, type = ro, optional = yes;

If it can help

7. Reply

Matthew C Applegate says:
November 14, 2016 at 12:12 am Edit
Hi Doug, these tutorials have been great, everything compiles things are moving, but the one problem I am having is displaying a variable to the screen. In this code the text us a "const unsigned char TEXT[]={"Hello World!"};" but is there a way of converting a int to a string in cc65 to say display a score? Or am I going about it the completely wrong way?

Thanks Matthew / Pixelh8

Reply

dougfraker says:
November 14, 2016 at 3:22 am Edit
I would ask on the nesdev forum. I personally don't use strings with cc65.

8. Reply

Maxime Trimolet says:
December 15, 2016 at 2:34 pm Edit
Hello Doug,
I have a problem here with the result. It prints "ELLO ORLD!" instead of "Hello World!". I tried to add 20h to the char value while writing to PPU_DATA, but it is still wrong: good case this time, but some letters still missing.
My work-space is the same as yours, except I'm on Linux (Fedora), so I made my Makefile and an Eclipse-CDT project. I had to add "ONCE: load = PRG, type = ro, optional = yes;" to the cfg file as @MrEddy and you suggested.

9. Reply

Cantalope says:
April 21, 2017 at 11:57 pm Edit
Hi! It seems that trying the reset.s page is nowhere to be seen on my rom and i need to see if my custom graphics works. Is there a way to do or find it another way?he reset

Reply

dougfraker says:
April 23, 2017 at 10:33 pm Edit
I don't know what you're asking.

Reply

1. **Cantalope** says:
   April 28, 2017 at 10:58 pm Edit
   My Screen is blank when copying the code to another instance.
   Do I need to mirror the graphics as well?

   dougfraker says:
   April 30, 2017 at 1:45 am Edit
   It's not clear from your comment what you are doing.

10. **Thomas** says:
    April 26, 2017 at 4:40 pm Edit
    I'm using a Mac. After downloading and installing all the necessary files to get NES programs working, I downloaded this program, copied everything in the "lesson1.c" file, pasted it on another file that worked with a previous code, which I replaced with this one, saved it, and tried to compile it using (cl65 -t nes hello-nes.c -o hello.nes) and it gives me these errors: hello-nes.c(6): Error: Include file `nes.lib' not found
    hello-nes.c(15): Warning: Unknown pragma `bss'
    hello-nes.c(26): Warning: Unknown pragma `bss'
    hello-nes.c(36): Error: Include file `PALETTE.c' not found
    hello-nes.c(37): Error: Include file `CODE.c' not found
    hello-nes.c(47): Error: Call to undefined function `All_Off'
    hello-nes.c(48): Error: Call to undefined function `Load_Palette'
    hello-nes.c(49): Error: Call to undefined function `Reset_Scroll'
    hello-nes.c(50): Error: Call to undefined function `All_On'
    Please help me on this.

    Reply

    dougfraker says:
    April 30, 2017 at 1:44 am Edit
    I don't know, I don't use a mac.

11. 太郎 **スティーブン** says:
    September 18, 2017 at 5:11 pm Edit
    MacOS Sierra here. I had to both replace line 46 of nes.cfg with
    ONCE: load = PRG, type = ro, optional = yes;

    And I had to change the SYMBOLS at the end of the file to this:

    SYMBOLS {
    __STACKSIZE__: value = $0100, weak = yes; # 1 page stack
    __STACK_START__: value = $700, weak = yes;
    }

    Reply

12.

太郎 スティーブン says:
September 18, 2017 at 5:23 pm Edit
Also if you are on a mac and also super noob ;) , you can use this compile.sh file instead of compile.bat (bat = batch file, which is for windows):

```
name="lesson1"

cc65 -Oi $name.c –add-source
ca65 reset.s
ca65 $name.s

ld65 -C nes.cfg -o $name.nes reset.o $name.o nes.lib

rm lesson1.o
rm reset.o
```

Reply

13. **Thomas** says:
April 2, 2018 at 1:23 am Edit
Hello. What's the "reset.s" for? And how does is vary by program or game?

Reply

dougfraker says:
April 2, 2018 at 12:10 pm Edit
Mostly it just initialized the NES and zeroes the RAM. But the "header" segment is here, which will change if your program increases in size or uses a more advanced Mapper.

And, the "CHARS" segment is at the bottom. The name of the CHR-ROM (graphics file) may change.

And, I use a slightly different "nmi" code in later examples…but since I started using neslib lately, that has the nmi code in neslib.s and calls reset.s "crt0.s", which is more common if you look at other people's NES code.

Reply