NES6502

Ing. Edgardo Gho

https://github.com/edgardogho/NES6502

Objetivo

- Comprender la arquitectura del NES
- Entender las herramientas de desarrollo
  - Compilador/Ensamblador cc65
  - Editor de rom de caracteres yychr
  - Emulador de NES Fceux
- Ejecutar el hola mundo y entender su estructura
- Modificar el hola mundo para agregar tiles y sprites
- Desarrollar un juego básico
  - No se necesita Scroll horizontal/vertical de pantalla
  - No se necesita utilizar audio
  - No se necesita modificar la rom de caracteres

Herramientas a utilizar

- Compilador / Ensamblador cc65 https://cc65.github.io/getting-started.html
- Editor de rom de caracteres yychr https://www.romhacking.net/utilities/119/
  - Puede utilizarse con wine en entornos *nix y Mac OS X
  - Se brinda como ejemplo una imagen de rom de caracteres de SMB (mario.chr)
- Emulador de NES Fceux http://www.fceux.com/web/home.html
  - La version de Windows soporta un debugger de 6502
  - Puede utilizarse con wine en entornos *nix y Mac OS X

Formato iNES de imágenes de ROM (.nes)

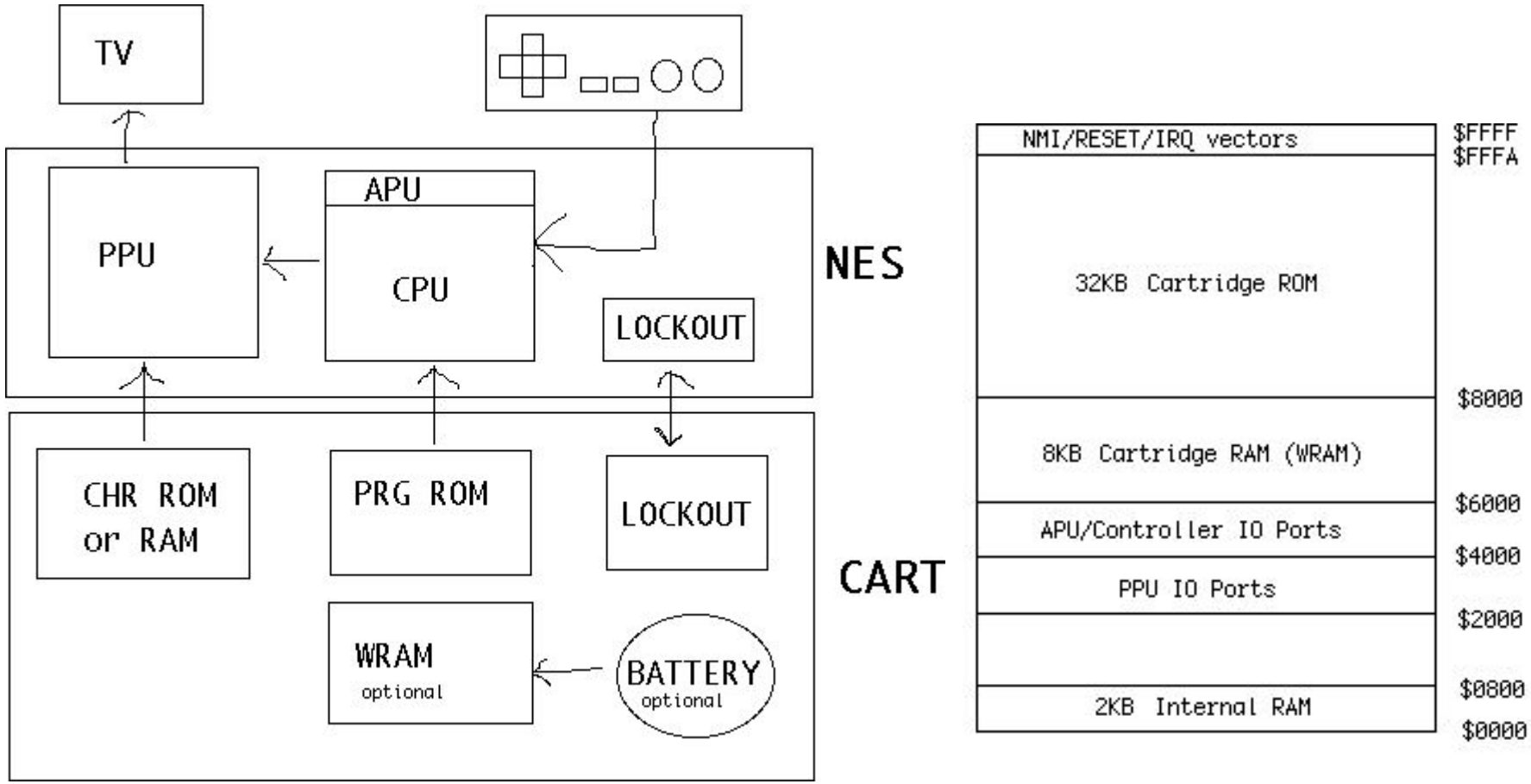
El estándar de archivos imágenes de ROM para NES es el formato iNES. Un archivo de imagen de ROM .nes posee una estructura descripta en: http://fms.komkon.org/EMUL8/NES.html#LABM

Debido a que la arquitectura del NES soporta bancos de memoria con mapeadores diversos, debe especificarse cuál de todos los mapeadores va a utilizarse. En nuestro caso utilizaremos el mapeador NROM 0, que básicamente implementa una estructura simple sin bancos de memoria.

Cada sección del código se encuentra referenciada a su posición de memoria final mediante el archivo de linker link.x . El mismo le asigna una ubicación física a las diversas secciones definidas en el código.

De esta forma podemos simular memoria ROM y RAM e incluir archivos binarios como roms de caracteres a partir de una posición de memoria dada.

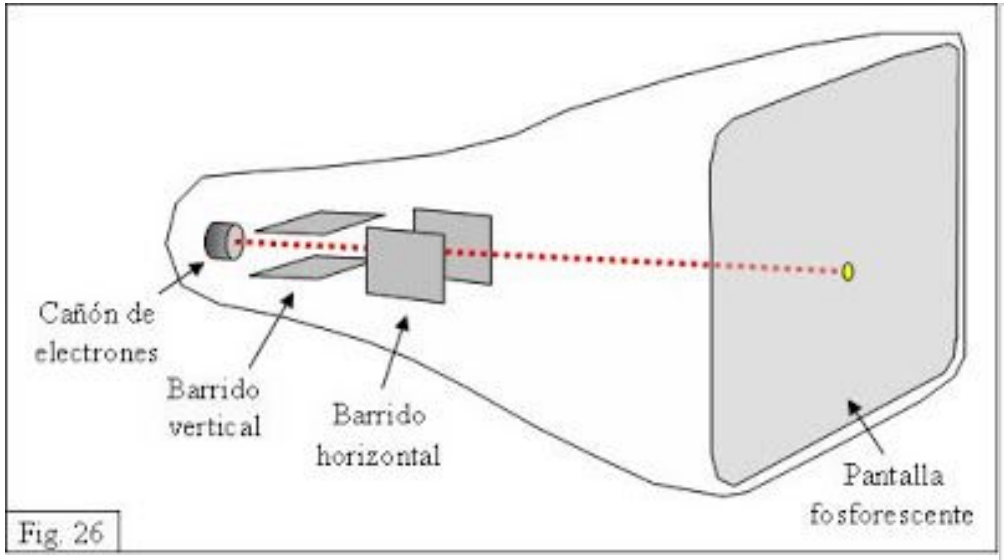
Arquitectura del NES

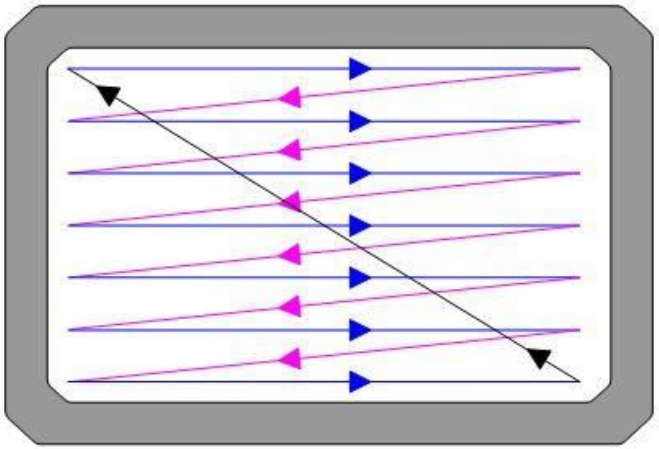


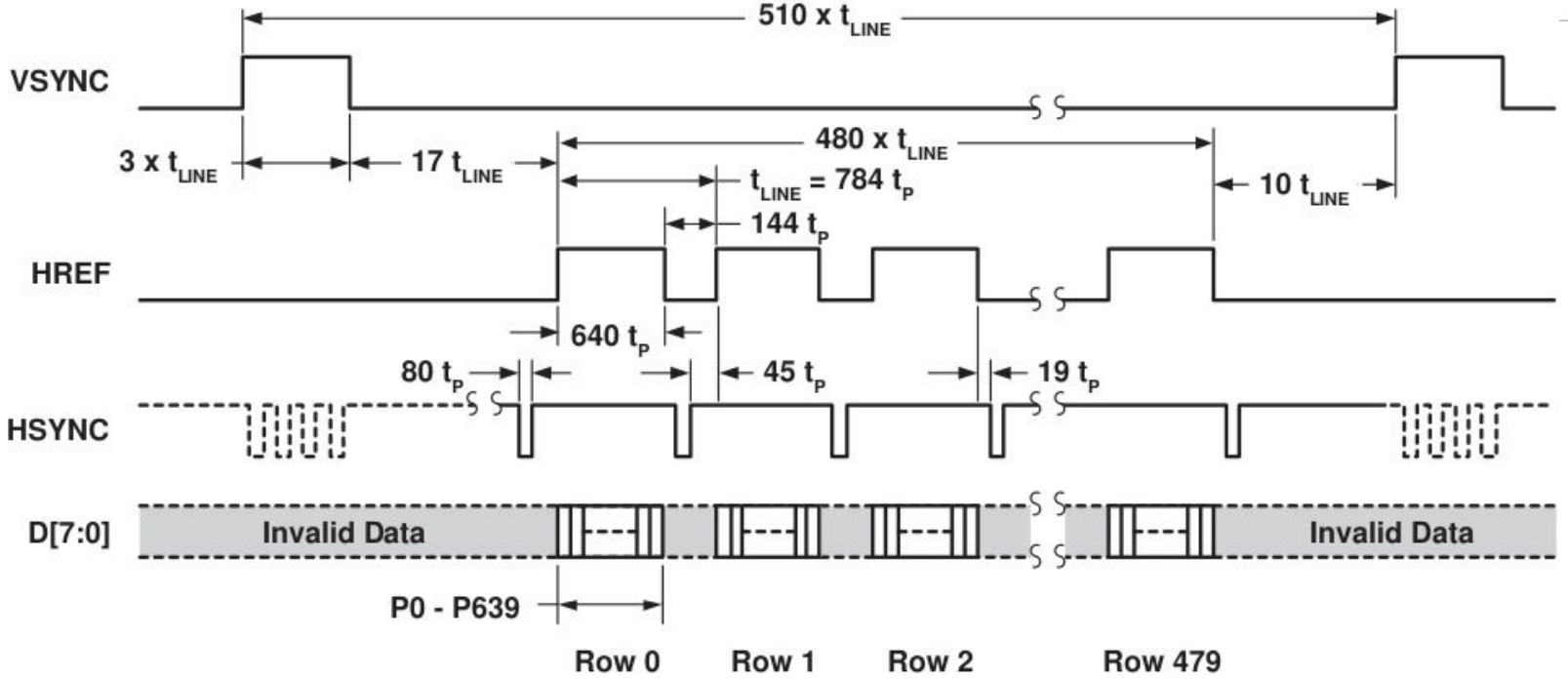
Resumen de especificaciones técnicas

- CPU: Procesador manufacturado por Ricoh de 8 bits basado en MOS Technology6502, con 4 generadores de tonos (dos cuadrados, un triángulo, un ruido), un dispositivo DAC, y Controlador DMA restringido. Contiene 2 KB de RAM interna.
    - RAM: 2 KB, con opción de utilizar una expansión si estaba presente en el cartucho.
- PPU: Procesador de vídeo Ricoh
    - Paleta: 48 colores y cinco grises en la paleta base; rojo, verde, y azul se pueden oscurecer individualmente en regiones específicas de la pantalla usando código temporizado.
    - Colores en pantalla: 52 colores en una línea de escaneo (color de fondo + 4 conjuntos de 3 colores de cuadro + 4 conjuntos de 3 colores de sprite).
    - Animaciones (sprites) apoyadas por hardware.
        - Sprites en pantalla: 64 (sin recarga en mitad de pantalla).
        - Tamaños de sprite: 8x8 u 8x16 pixeles.
    - Memoria de video: PPU conectada a 32 KB de vídeo RAM. PPU contiene 2 KB de RAM interno atribuible/de cuadro; 256 bytes de RAM de posición de sprite; 28 bytes de RAM de paleta (que permite selección de color de fondo); 8 KB de ROM/RAM de patrones de cuadros en el cartucho.
    - Resolución: 256x240 píxeles.

Barrido horizontal y vertical de TV

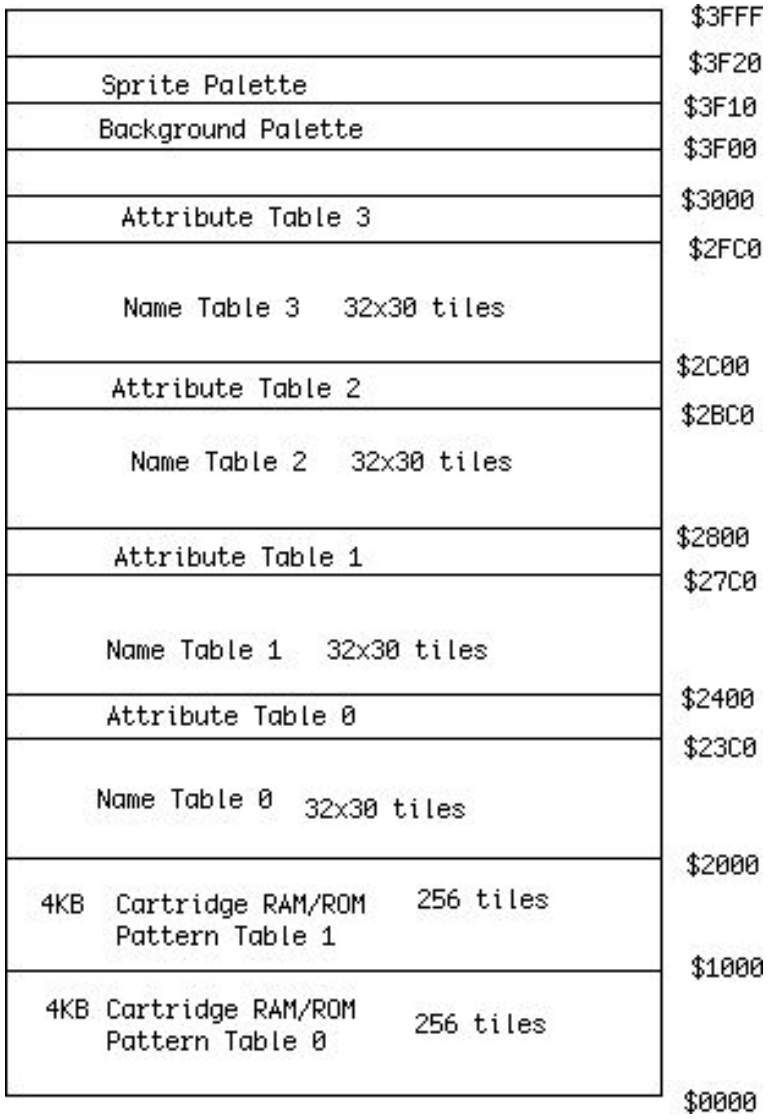






PPU (Picture Processing Unit)

Posee su propia memoria de video. Esta memoria de video se accede mediante una interface de entrada/salida de la CPU en ciertas posiciones de memoria.



El PPU dibuja dos tipos de elementos:

1. **Tiles:** Estos son grupos de 8x8 pixeles que representan el fondo de la pantalla. Siendo la resolución de 256x240 píxeles esto genera 32x30 tiles en pantalla. Los tiles se cargan en las Nametables. Cada 4 tiles se define un atributo de paleta de colores. Esto permite cargar una pantalla completa de 61440 pixeles con tan solo 1KB.
2. **Sprites:** Estos son también grupos de 8x8 píxeles que se dibujan por delante del fondo. Cada sprite tiene una coordenada X, Y, un puntero a la memoria de caracteres y un atributo que permite rotar y definir la paleta de colores del sprite.

Existe un controlador DMA para transferir bloques de RAM a VRAM.

cc65

Para ensamblar se utiliza el ca65: ca65.exe archivo.asm -o archivo.o

Una vez ensamblado, el programa objeto (archivo.o) se linkea con el linker ld65:

Ld65.exe -C link.x archivo.o -o archivo.nes

Este linker toma la configuracion del archivo final con las direcciones especificadas en link.x y genera el archivo .nes para ser utilizado por el emulador.

Para correr el programa se ejecuta fceux.exe archivo.nes