WedNESDay

Make Classic Games
&
Emiliollbb

wedNESday

- Qué son los wedNESday
  - NES
  - Herramientas de desarrollo (ASM, C, Gráficos, Trackers).
  - Creación de juegos para NES.
  - Estudio de los componentes de la NES.
  - También probaremos algunos de los juegos
- Todos los miércoles de 17:30 a 18:30.

¿Qué haremos en esta sesión?

- ¿Qué es la NES?¿Por qué?
- ¿Cómo? ¿ASM?
- Ejemplo de juego desarrollado en NES
  - Inicialización de la consola
  - Interrupciones
  - Fondos
  - Sprites

¿POR QUÉ LA NES? ¿QUÉ TIENE DE ESPECIAL?

Más de 40 años después de su lanzamiento, “La Nintendo” sigue siendo divertida de jugar y programar.

¿POR QUÉ LA NES? ¿QUÉ TIENE DE ESPECIAL?

- Diseñada para ser un juguete muy barato de fabricar, permitió a los niños de los 80 y 90 tener su propia recreativa en casa sin tener que echar monedas
- Suficientemente potente para ejecutar juegos interesantes, y a la vez suficientemente sencilla para ser fácil de programar
- Su diseño impone una estética de imágenes y sonidos únicos que han creado escuela

Masayuki Uemura (上村雅之, Uemura Masayuki; 20 June 1943 – 6 December 2021)

“Un día de 1971, los jefes de Masayuki le enviaron a preguntar a Nintendo si querían comprar semiconductores. Se citó con Gunpei Yokoi y su equipo, y estos pensaron que estas células podrían ser usadas para hacer nuevos juguetes.[5]

Tras esto, a Masayuki Uemura le comunicaron que le iban a trasladar a otra ciudad, algo a lo que Masayuki se negó, por lo que dejó su trabajo en Sharp y rápidamente fue contratado por Gunpei Yokoi y su equipo de Nintendo.

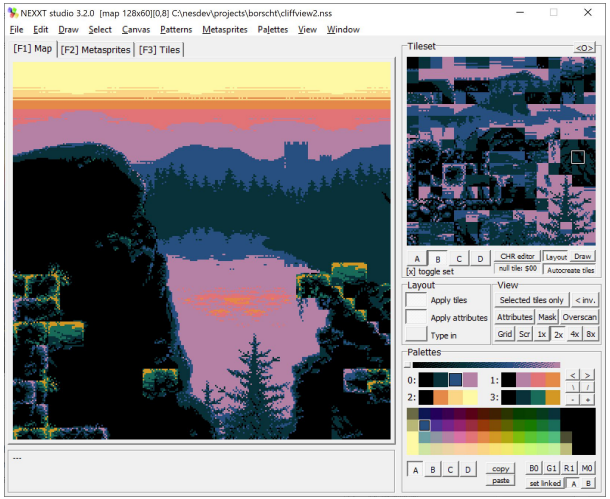
Trabajando para Nintendo, Masayuki se sentía como cuando era un niño (por eso de hacer juguetes), y se sintió sorprendido que todos esos hombres tan serios hiciesen juguetes” – Extracto de Wikipedia

NES (Nintendo Entertainment System)

- CPU Ricoh 2A03 (MOS 65C02) a 1,79Mhz (NTSC)/1,66Mhz (PAL).
- PPU (Picture Processor Unit) 256x240 de resolución y 25 colores simultáneos (paleta de 40 colores) y 64 sprites por pantalla.
- 2KB de RAM y 2KB de VRAM (+256 bytes para oam).
- 4 generadores de tono.
- 2 Controladores, puerto de expansión y slot de cartuchos.
- Cartuchos (ROM) de 60/72 pines.

¿Que necesitamos?

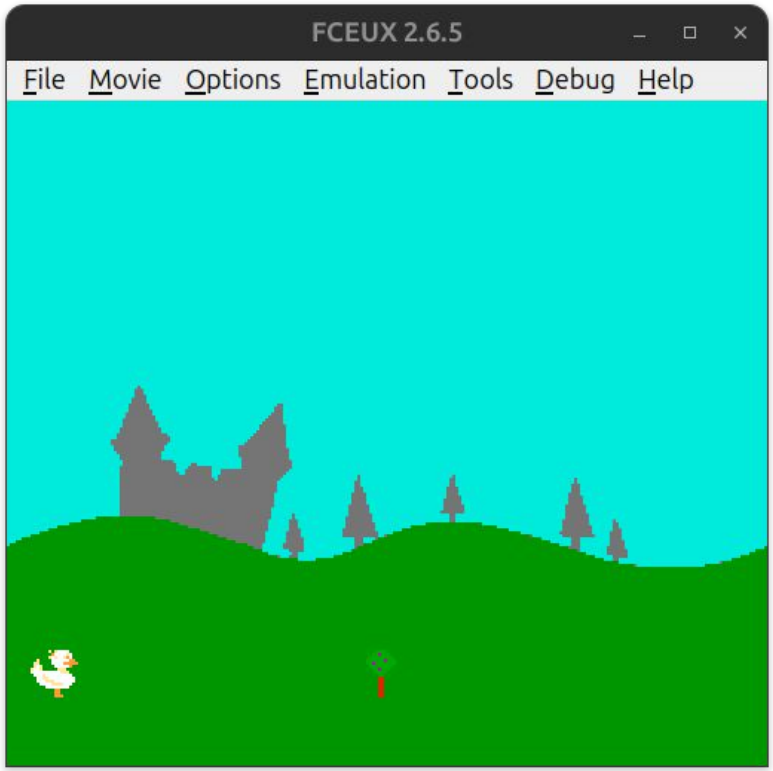
- Un editor de Texto
- Un “Ensamblador” / Compilador.
- Editor Gráfico.
- Tracker.



Análisis de un Juego de NES (MS Patman Redux)

Analizaremos el juego MS Patman Redux para entender cómo se debe desarrollar un juego.

Veremos tanto las herramientas utilizadas como se ha organizado el código del juego.



Arquitectura NES

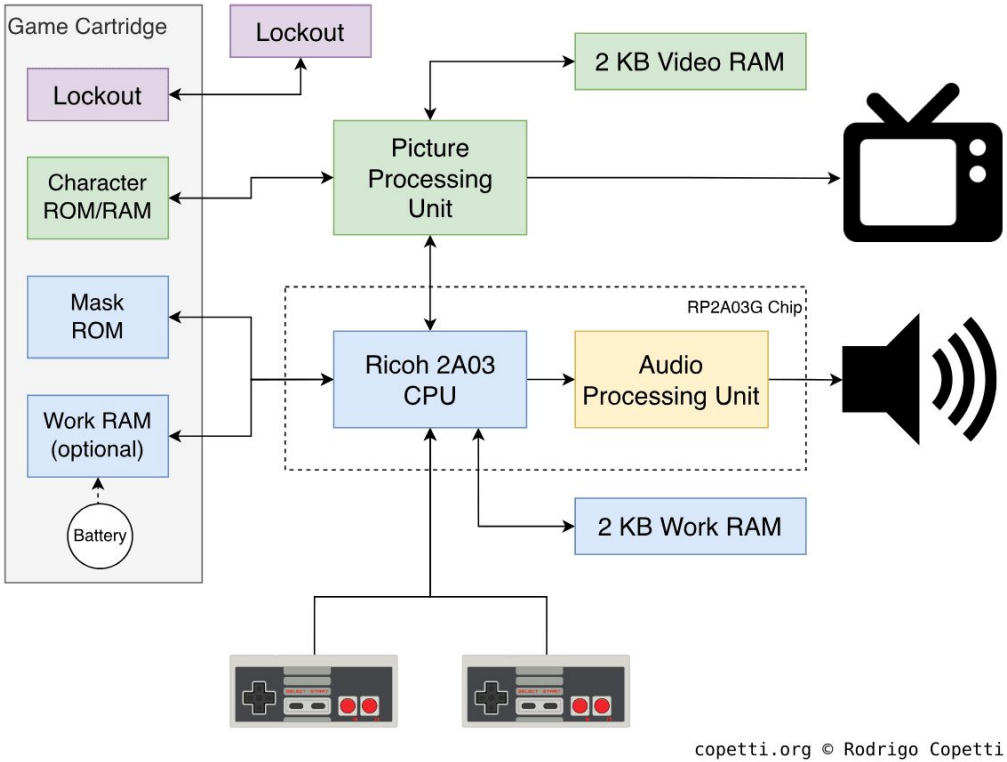
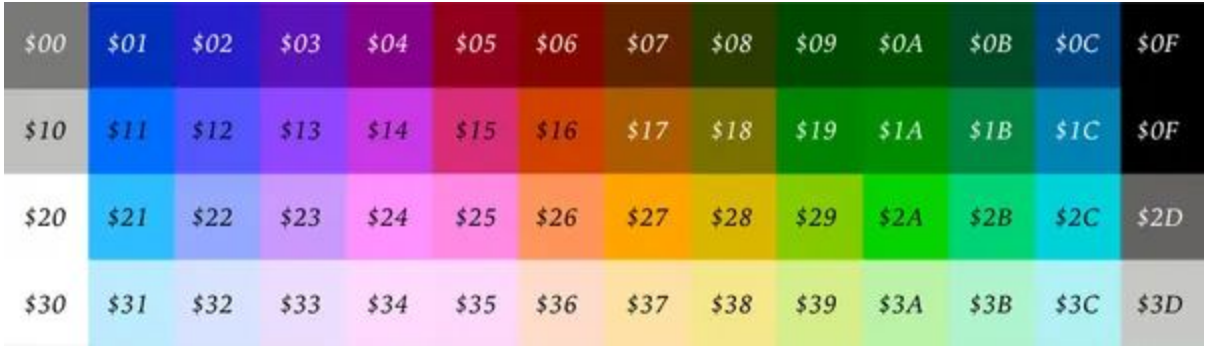


Tabla de colores



La NES sólo puede mostrar 55 colores, programados internamente en la PPU con los códigos mostrados en la tabla.

Capacidades gráficas

La PPU de la NES puede dibujar:

- Fondos
  - 2 pantallas de 256x240 pixels en disposición horizontal ó vertical
  - Scroll x,y a nivel de pixel
- Sprites
  - 64 sprites
  - Máximo 8 sprites por scanline

Tiles

- Todos los gráficos de la NES se basan en tiles de 8x8 pixels
- En el cartucho hay 2 bancos de 256 tiles cada uno
  - Lo habitual es usar un banco para fondos y otro para sprites
  - Alternativamente se puede usar el mismo banco para fondos y sprites
- Cada fondo está compuesto por 32 tiles de ancho y 30 tiles de alto
  - Existen 4 fondos, 2 de ellos mirror
- Cada sprite está compuesto por 1 tile
  - Alternativamente se puede configurar la PPU para usar sprites dobles (1 tile de ancho y 2 de alto)

Paletas

- La NES utiliza 8 paletas de 4 colores cada una
  - 4 paletas para fondos
    - Cada cuadrado de 2x2 tiles usa la misma paleta
  - 4 paletas para sprites
    - Cada sprite puede usar una de las 4 paletas disponibles
  - El primer color de cada una de las 8 paletas debe ser el mismo, y es considerado color de fondo / transparencia
  - Las paletas se encuentran en la dirección $3F00 de la RAM de la PPU

Un poco de informática básica

- Una memoria la podemos imaginar como una tabla bidimensional (como una estantería) en la que guardar datos
  - El espacio de memoria del 6502 nos podemos imaginar una estantería, con 256 baldas, numeradas de $00 a $FF, dónde en cada balda tiene hueco para 256 números, también etiquetados de $00 a $FF.
  - No todas las baldas son memoria RAM, algunas baldas nos permiten interactuar con la PPU, o leer los mandos.

Un poco de ensamblador 6502

- Un registro es “una cajita” dentro de la cpu en la que podemos guardar un número de 8 bits (%00000000 - %11111111), ($00, $FF) (0,255)
- La CPU 6502 tiene un registro principal llamado A (Acumulador)
- Para cargar un valor, lo haremos con:
  - LDA #16 ; En decimal
  - LDA #$F0 ; También lo podemos hacer en Hexadecimal, que será lo habitual
  - LDA #%00010000 ; Algunas veces será conveniente hacerlo en binario
- La CPU sólo entiende binario, en todos los casos anteriores, el ensamblador generará el mismo código binario para la CPU, el poder usar distintas notaciones es una comodidad que nos ofrece el ensamblador.

Un poquito más. Es sencillo, lo prometo.

- Para escribir el valor que tenemos guardado en “la cajita A”, usaremos:
  - STA $2002
- No es necesario aprender de memoria las posiciones de memoria. Podemos usar constantes:
  - STA PPUADDR

Ejemplo de Ensamblador

Veamos el siguiente ejemplo:

```
LDA #2
STA $20
LDX $20
```

Probemoslo en un simulador:

https://tony-cruise.github.io/6502Simulator.html

Primer ejemplo: Cargar colores en las paletas

```
1   LDA PPUSTATUS
2   LDA #$3F
3   STA PPUADDR
4   LDA #$00
5   STA PPUADDR
6   ; Enviar los codigos de color a la PPU
7   LDA #$01 ; Azul
8   STA PPUDATA
9   LDA #$14 ; Violeta
10  STA PPUDATA
11  LDA #$16 ; Rojo
12  STA PPUDATA
13  LDA #$29 ; Verde
14  STA PPUDATA
```

Para enviar datos a la PPU, primero indicamos a qué dirección de memoria queremos mandarlos, y luego los vamos mandando byte a byte. En este ejemplo enviamos los 4 colores de la primera paleta de fondos.

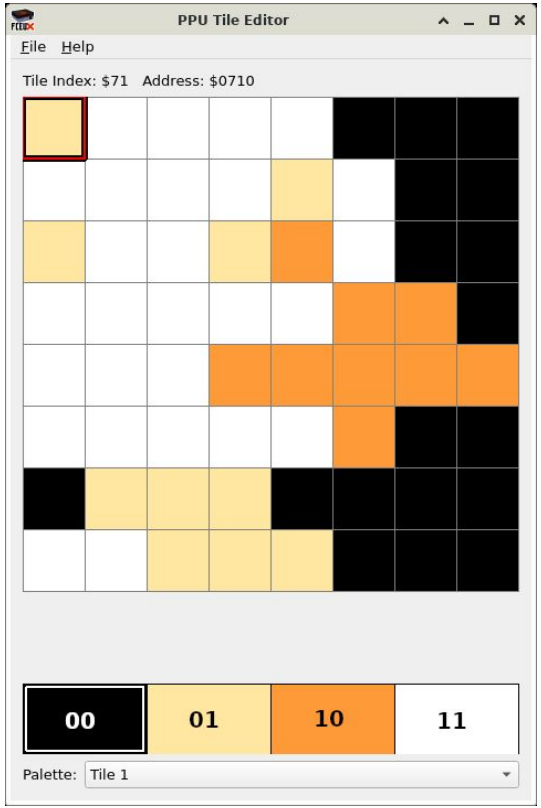
Primer ejemplo: Cargar las paletas

Veamos un ejemplo para cargar las paletas en la NES:

https://github.com/makeclassicgames/wedNESdaysExamples/blob/main/01_paletas.s

Tiles y Sprites

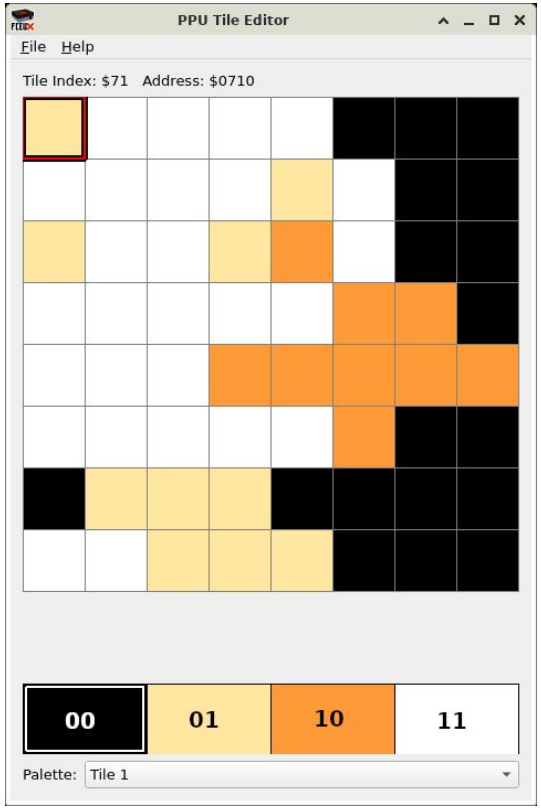
Creando Tiles



Los tiles los podemos crear con cualquier herramienta que nos sea cómoda, siendo la más habitual NEXXT Studio.

https://frankengraphics.itch.io/nexxt

Creando Tiles

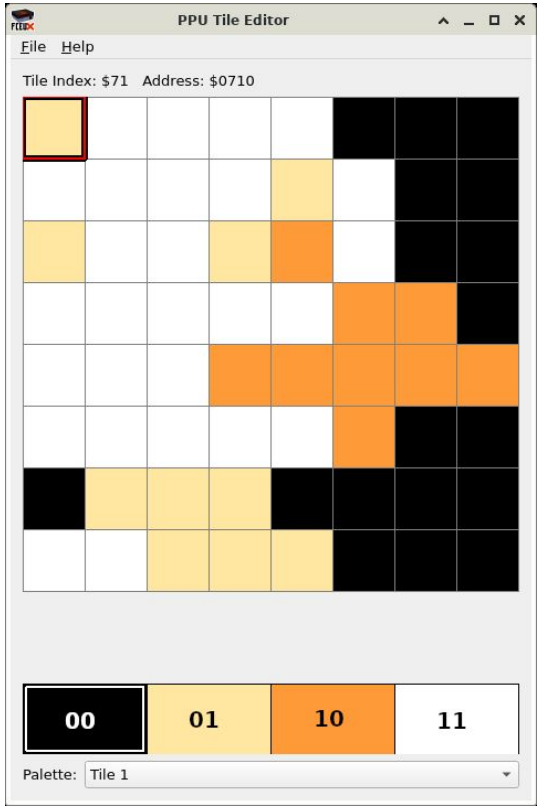


El formato de las tiles en NES es muy sencillo.

1. Numeramos los píxeles usando los índices de la paleta en binario

01 11 11 11 11 00 00 00
11 11 11 11 01 11 00 00
01 11 11 01 10 11 00 00
11 11 11 11 11 10 10 00
11 11 11 10 10 10 10 10
11 11 11 11 11 10 00 00
00 01 01 01 00 00 00 00
11 11 01 01 01 00 00 00

Creando Tiles

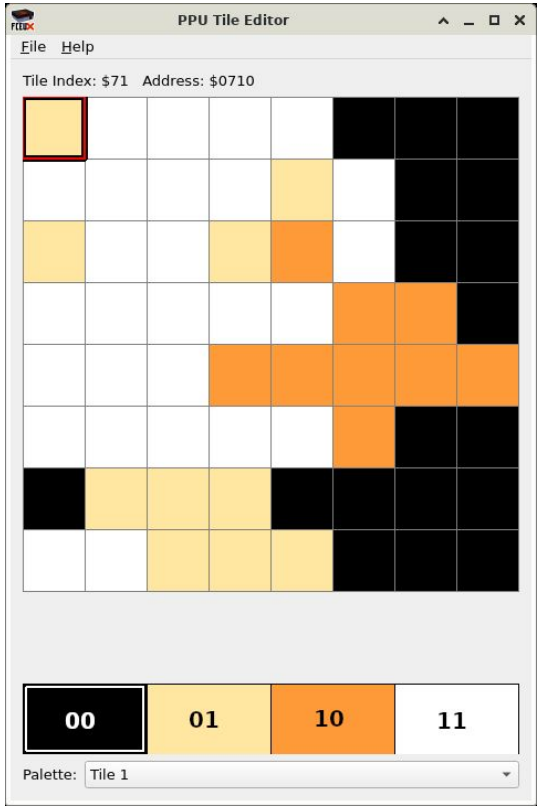


2. Por cada fila, formamos un byte con los bits menos significativos, obteniendo los primeros 8 bytes del tile.

01 11 11 11 11 00 00 00	11111000	F8
11 11 11 11 01 11 00 00	11111100	FC
01 11 11 01 10 11 00 00	11110100	F4
11 11 11 11 11 10 10 00	11111000	F8
11 11 11 10 10 10 10 10	11100000	E0
11 11 11 11 11 10 00 00	11111000	F8
00 01 01 01 00 00 00 00	01110000	70
11 11 01 01 01 00 00 00	11111000	F8

F8 FC F4 F8 E0 F8 70 F8

Creando Tiles



3. Hacemos lo mismo con los bits más significativos, obteniendo los 8 bytes restantes.

01 11 11 11 11 00 00 00	01111000	78
11 11 11 11 01 11 00 00	11110100	F4
01 11 11 01 10 11 00 00	01101100	6C
11 11 11 11 11 10 10 00	11111110	FE
11 11 11 10 10 10 10 10	11111111	FF
11 11 11 11 11 10 00 00	11111100	FC
00 01 01 01 00 00 00 00	00000000	00
11 11 01 01 01 00 00 00	11000000	C0

F8 FC F4 F8 E0 F8 70 F8 78 F4 6C FE FF FC 00 C0

Sprites

Un Sprite es un elemento del juego que “puede” tener un comportamiento o simplemente está ahí.

La NES tiene capacidad de trabajar con sprites de forma separada de los fondos (utilizan también Tiles).

Los sprites son los personajes que se mueven sobre el fondo

Las características de los sprites, vienen definidas en una zona de la memoria especial llamada OAM (Object Attribute Memory) de 256 bytes de capacidad.

Sprites

Un sprite, no es más que una referencia a un Tile con algunos atributos extra.

Cada Sprite necesita de 4 bytes y representa un Tile de 8x8 píxeles. Si necesitamos representar Sprites más grandes, necesitaremos combinar diferentes Sprites.

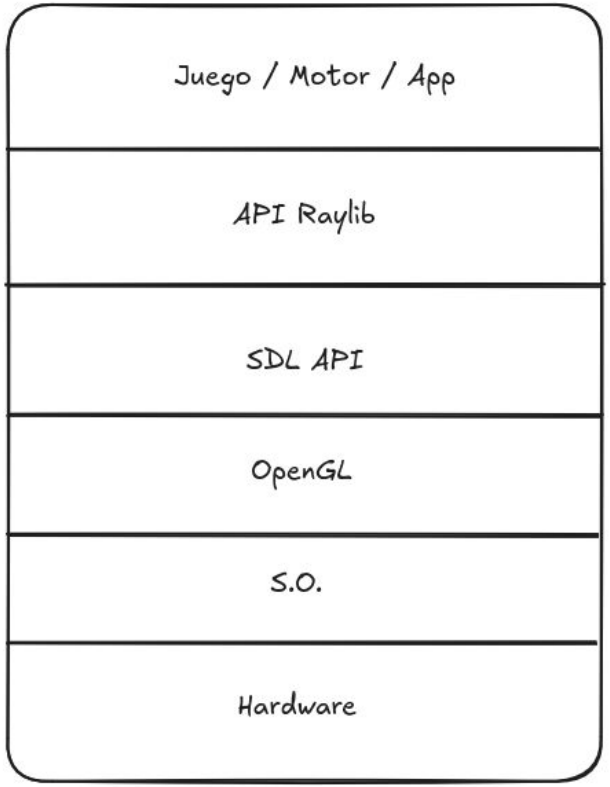
La NES, tiene capacidad para 64 Sprites (256/4).

OAM Memory

- Los sprites se definen en la memoria OAM, ubicada en la PPU.
- A diferencia del resto de memoria, es memoria DINÁMICA, lo que nos obliga a refrescarla continuamente
- La forma habitual de trabajar con ella, es crear una caché en la memoria principal de la CPU, en la página $02 ($0200 - $02FF), y en cada VBlank, enviar esta página a la OAM.
- La NES cuenta con DMA (Direct Memory Access) para esta operación.

Un pequeño inciso

- ¿Por qué necesitamos conocer tanto detalle del hardware?
- En un sistema moderno, tenemos numerosas capas de abstracción que nos permiten trabajar con una API de alto nivel que es independiente del hardware. Pensemos en los ejemplos de Raylib del canal. El mismo código hecho en C usando las mismas funciones sirve tanto para un PC x86, como un móvil ARM, o una Gamecube. Esto es porque programamos sobre la api de Raylib. Si nos limitamos a PC, la api de Raylib está sobre SDL, que a su vez está sobre OpenGL, que a su vez está sobre el SO. Todas estas capas nos facilitan el trabajo alejándonos de las peculiaridades del hardware
- En la NES No tenemos NADA, ni siquiera un firmware con procedimientos almacenados. Tenemos que programar directamente sobre el hardware.

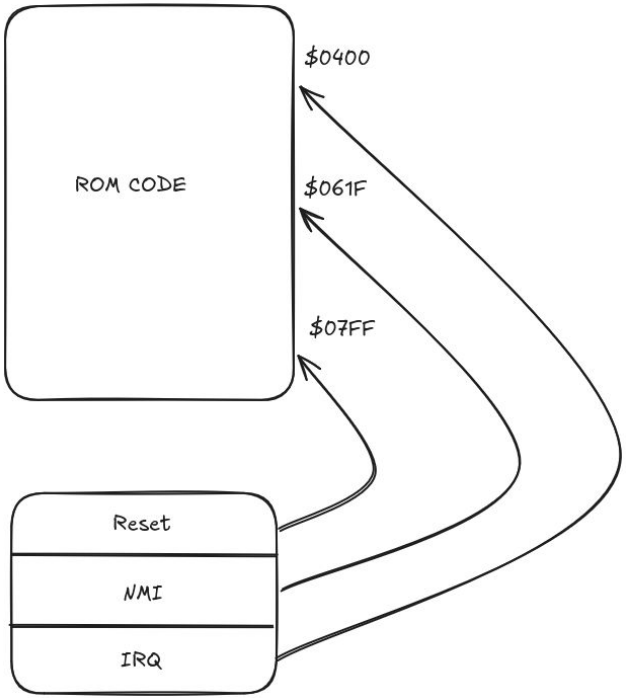


Un poco más de 6502. Vectores e Interrupciones

- El procesador 6502 cuenta con 2 interrupciones, llamadas IRQ (Hardware) y NMI (VBlank).
- Cuando llega una señal eléctrica a la entrada de alguna de las interrupciones, el procesador interrumpe el trabajo en curso, y atiende la interrupción. Una vez terminada la interrupción, continúa con lo que estaba haciendo
- En la NES, la interrupción NMI está conectada a la PPU, y ésta la activa al iniciar el VBlank, es decir, al terminar de pintar cada frame.
- La actualización gráfica del juego que hacemos en la PPU la tenemos que hacer durante el VBlank, es decir, en la interrupción NMI
- De igual forma, usaremos la interrupción NMI para actualizar la OAM

Un poco más de 6502. Vectores e Interrupciones

- Cuando se enciende el 6502, ¿Cómo sabe por donde tiene que empezar a ejecutar código? Podríamos pensar que por el principio, ¿Pero cuál es el principio? En cada arquitectura, la ROM con el código va a estar en una dirección de memoria distinta.
- Al final de la ROM, hay 3 vectores de 2 bytes (6 bytes en total) llamados vectores que indican los puntos de comienzo de las ejecuciones, uno para el código principal, otro para la interrupción IRQ y otro para la interrupción NMI.
- Al encender el 6502 (o al pulsar Reset), el procesador va a la dirección de memoria $FFFC y $FFFD, y empieza la ejecución por la dirección que haya allí escrita. Por ejemplo: $FFFC=$00 y $FFFD=$08, la ejecución empieza en $0800. (6502 es little endian)



Sprites

Veamos como se define un Sprite:

- **Byte 0**: Posición Y en pantalla.
- **Byte 1**: Índice del Tile de la VRAM.
- **Byte 2**: Atributos; define una serie de atributos del sprite.
- **Byte 3**: Posición X en pantalla.

Veamos los atributos (byte 2) que se definen a nivel de bit:

- **Bit 0-1**: Número de paleta a utilizar (4 para sprites).
- **Bit 5**: Indica si el Sprite está delante o detrás del fondo.
- **Bit 6**: Indica si está volteado horizontalmente.
- **Bit 7**: Indica si está volteado verticalmente.

Sprites

Veamos un ejemplo de uso de Sprites:

https://github.com/makeclassicgames/wedNESdaysExamples

Memoria y Código

Estructura de un Juego

Cuando creamos un juego, necesitamos saber de que “partes” se componen:

- ROM de código (Alojado en el cartucho).
- ROM(RAM) de patrones (Alojado en el cartucho).
- RAM disponible (2KB).
- RAM Extra (Alojado en cartucho).

NOTA: También está disponible la RAM de Vídeo (VRAM) para almacenar lo que se está mostrando por pantalla…

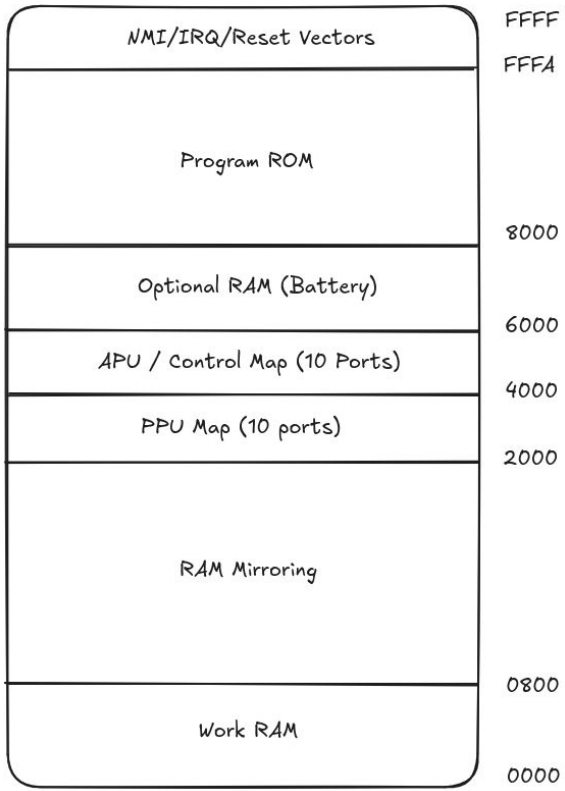
Memoria en la NES. CPU

La NES tiene diferentes memorias disponibles; algunas de solo lectura (ROM) y otras de acceso aleatorio (RAM).

Pero… otro de los aspectos a tener en cuenta, es ¿Cómo puedo hablar con el hardware?

Para ello, se utilizan direcciones de memoria “especiales” que llamamos *Puertos* (Ports).

Al conjunto de memoria y sus direcciones lo llamamos *Mapa de memoria*.

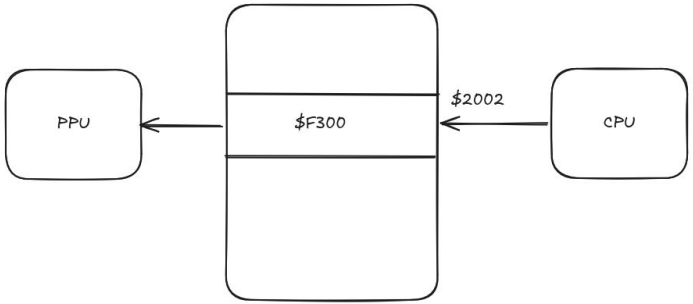


¿Hablar con el Hardware?

Cuando se quiere enviar o recibir información desde el hardware, se utilizan direcciones de memoria prefijadas.

Estas direcciones de memoria especiales se les llama Puertos y permiten comunicarse con diferentes aspectos del hardware.

Por ejemplo la PPU tiene diferentes puertos para comunicarse con la CPU.



Segmentos de memoria

Como hemos podido ver en el mapa de memoria, existen diferentes zonas; que podemos establecer en nuestro código ensamblador; a estos fragmentos, se les llama segmentos.

- ZP :Zero Page.
- RO: ROM de programa.
- TILES: ROM de Patrones.
- HEADER: Cabecera.
- CODE: ROM de programa para nuestro código.
- VECTORS: vectores de interrupción (NMI, IRQ, Reset).

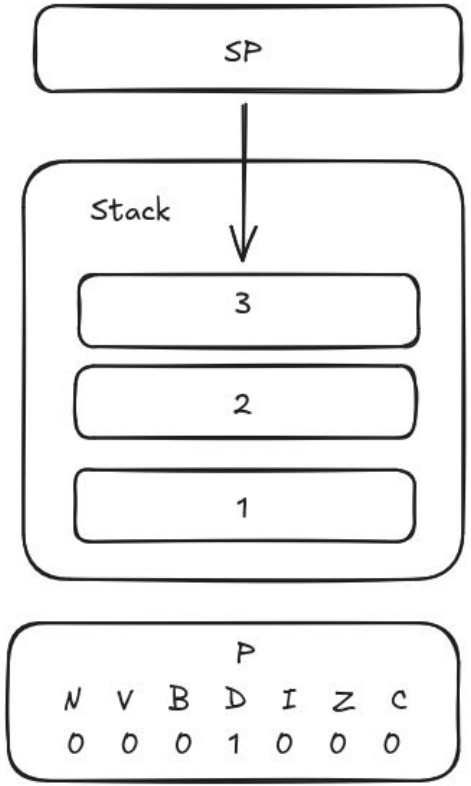
NOTA: Estos segmentos se pueden configurar en nuestro programa ensamblador, o establecerlos nosotros mismos en el código.

Zero Page

- 256 primeras posiciones de memoria ($0000 - $00-FF)
- Se referencian con un sólo byte ($00 - $FF)
- Acceso más rápido (1 ciclo menos)
- Permite usar punteros a memoria

Registro de Estado (P)

- NV - - DIZC
- Negative, oVerflow, Decimal, Interrupt disable, Zero, Carry
- Cuando hacemos una operación, el registro de estado se actualiza con el resultado de la operación (para las operaciones que aplican)
- El registro de estado, se utiliza para hacer los saltos condicionales



Stack

- La página $01 ($0100 - $01FF) está reservada por la cpu para el uso de la pila (Stack)
- Capacidad de 256 bytes
- La pila se usa para guardar el estado actual al llamar a funciones (y poder restaurarlo después).
- El procesador tiene un registro especial SP (Stack Pointer) que apunta a la dirección del elemento superior de la pila.
- Se puede usar la pila para guardar el Acumulador (A)
    - PHA, PLA

Saltos condicionales

- BPL (Branch on PLus)
- BMI (Branch on MInus)
- BVC (Branch on oVerflow Clear)
- BVS (Branch on oVerflow Set)
- BCC (Branch on Carry Clear)
- BCS (Branch on Carry Set)
- BNE (Branch on Not Equal)
- BEQ (Branch on EQual)

Comparadores

Podemos comparar valores alojados en los diferentes registros de la CPU que activan o desactivan los bits del registro de estado.

- CMP: Compara el registro A con un valor.
- CPX: Compara el registro X con un valor.
- CPY: Compara el registro Y con un valor.

A / X / Y >= 0	BPL (Branch on Plus)
A / X / Y < 0	BMI (Branch on MInus)
A / X / Y < valor	BCC (Branch on Carry Clear)
A / X / Y >= valor	BCS (Branch on Carry Set)
A / X / Y != valor	BNE (Branch on Not Equal)
A / X / Y == valor	BEQ (Branch on EQual)

Ejemplo de bucle

Veamos un ejemplo de bucle:

```
      LDX #10 ; cargamos 10 en X.
loop: CPX #0 ; Se compara X con 0.
      BEQ exit_loop ; Si son iguales, se salta fuera del bucle.
      DEX ; Decrementamos 1 en X.
      jmp loop ; salta la ejecución a la posición loop.
exit_loop: LDA #5
```

https://tony-cruise.github.io/6502Simulator.html

Ejemplo optimizado

```
LDX #10
bucle:
DEX
BPL bucle
LDA #5
```

Memoria en la NES. PPU

- La PPU también tiene su propio espacio de memoria
- Tenemos los dos bancos de tiles ubicados en el cartucho
- Tenemos la memoria de video, ubicada en la consola, compuesta por las paletas, y los mapas de tiles (Name tables).
- Las Attribute Table son las referencias a las paletas de los tiles del Attribute Table

Referencias

- https://famicom.party/book/
- https://www.copetti.org/writings/consoles/nes/
- https://nerdy-nights.nes.science/
- https://www.nesdev.org/wiki/Nesdev_Wiki
- https://kh-labs.org/concepts/xa65/
- https://cc65.github.io/
- https://makeclassicgames.dev/pdfs/6502.pdf
- https://makeclassicgames.dev/pdfs/asmnes.pdf