Programación en Ensamblador para NES

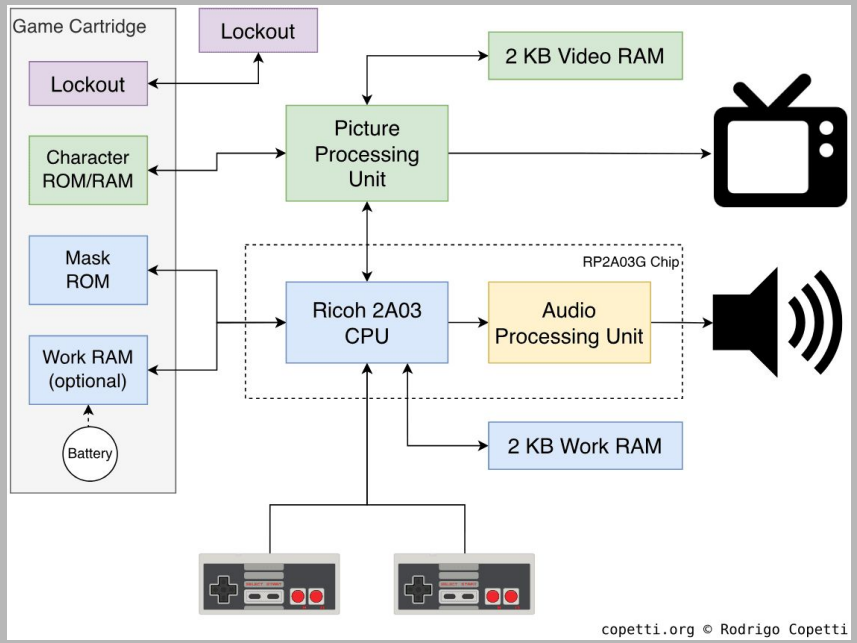
Make Classic Games

1. Introducción

Vamos a comenzar un proyecto de creación de un juego para NES; pero antes de comenzar, tenemos que saber Qué es la NES y cómo funciona.

En esta pequeña presentación, vamos a comenzar con que hace falta para comenzar un proyecto y cómo crear un juego realizandolo en NES.

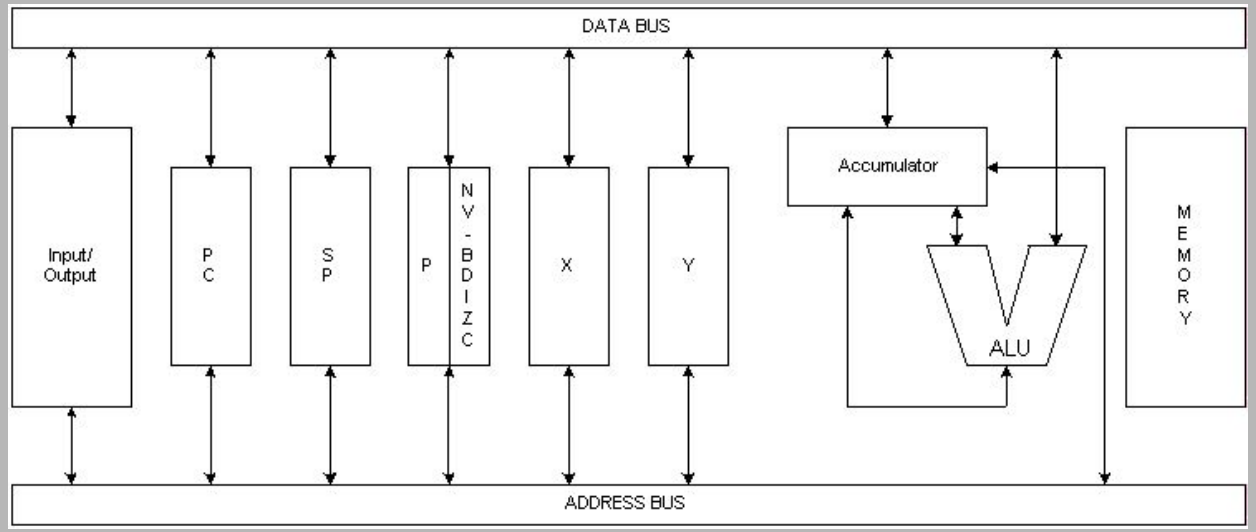
2. Arquitectura de la NES



Fuente: [Copetty.org]

3. El procesador 6502

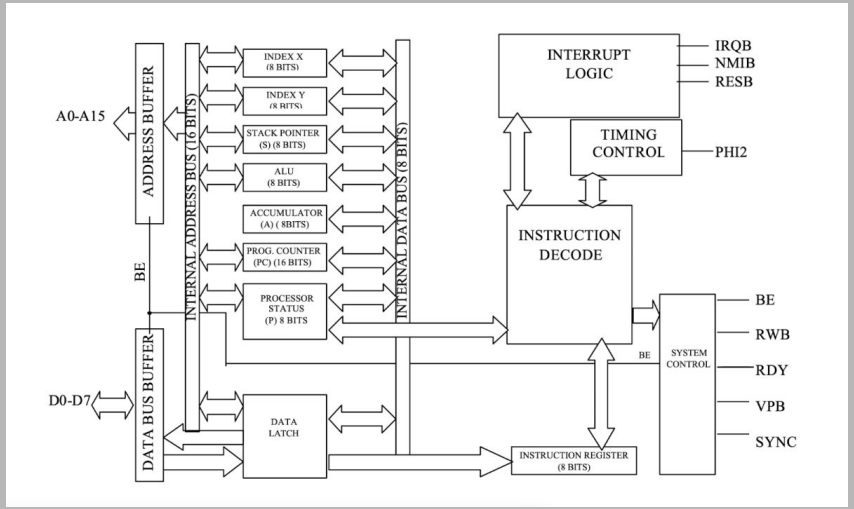
El procesador MOS 6502, es un procesador de 8 bits con capacidad de trabajar con direcciones de 16 bits. Veamos un diagrama:



3. El procesador 6502

Alguno de los elementos que tiene el 6502 son:

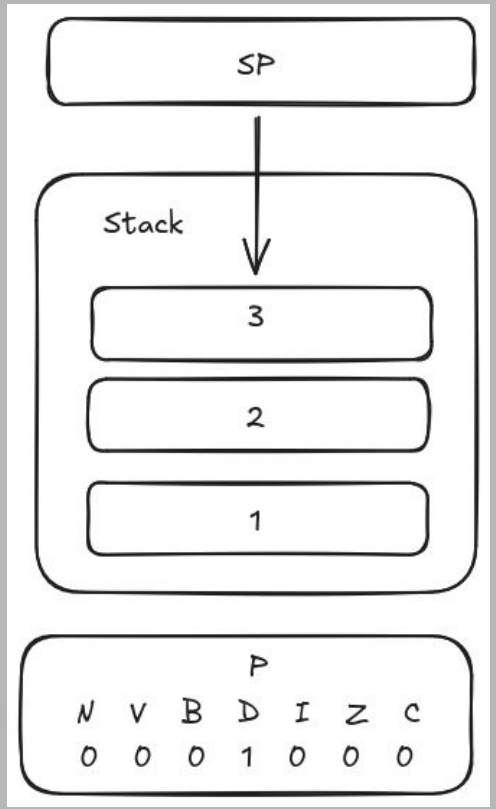
- **3 Registros:**
  - A (Acumulador)
  - X (Registro índice X)
  - Y (Registro índice Y)
- **Registros de estado y de programa**
  - P (Registro de estado)
  - PC: Contador de programa (16 bits)
  - SP: Puntero de Pila
- **ALU para trabajar con operaciones de 8 bits.**
- **Bus de Direcciones de 16 Bits**
- **Bus de Datos de 8 bits**



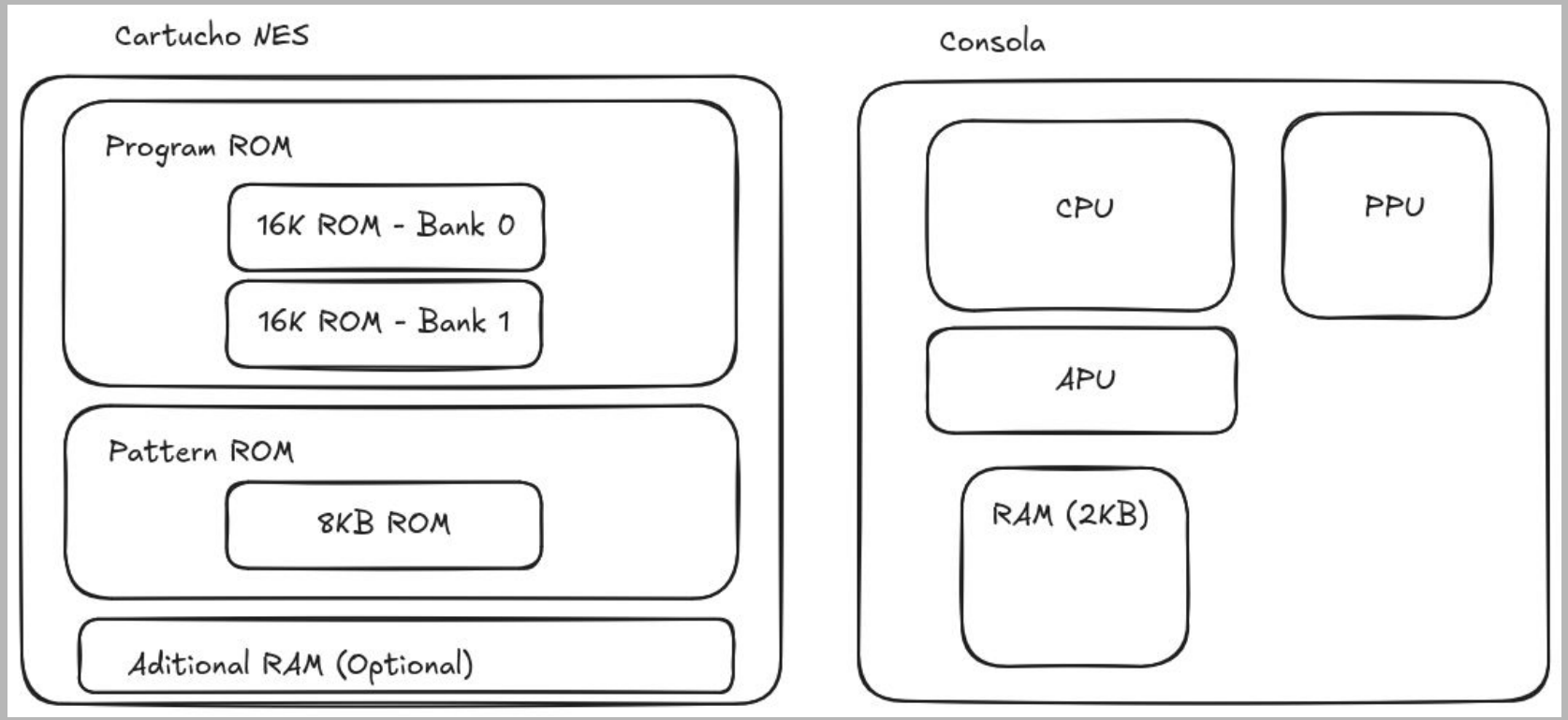
3. El procesador 6502

Hay varios registros especiales que vamos a comentar a continuación:

- Contador de programa: Indica la siguiente instrucción a ejecutar.
- Puntero a pila: permite almacenar la dirección de retorno y almacenar datos en dicha pila (por ejemplo estado de los registros).
- Registro de estado (P): Muestra el estado de las operaciones del procesador.
    - N: Negativo
    - V: OverFlow
    - B: break
    - D: Decimal
    - I: Interrupción
    - Z: Zero
    - C: Acarreo



4. Estructura de un Juego de NES



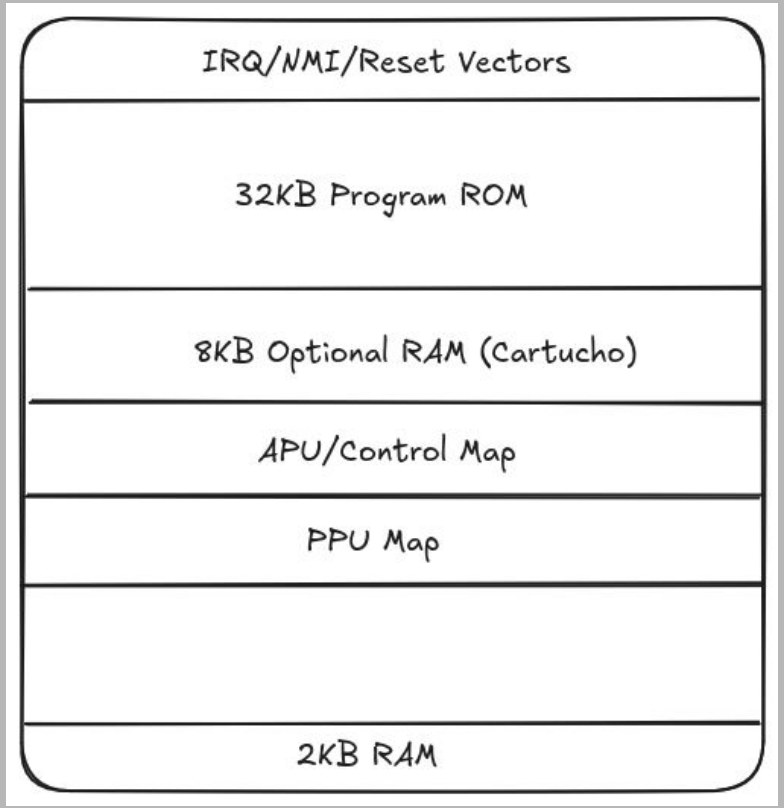
4. Estructura de un Juego de NES

Hay que tener en cuenta que nuestro juego necesitará:

- ROM de programa (alojado en el cartucho). Contiene el código del programa.
- ROM de Patrones (alojado en el cartucho). Contiene los patrones o gráficos del programa.
- RAM de la consola. Que guardará las variables y datos del programa en ejecución.
- RAM Adicional. Algunos cartuchos incluyen RAM adicional.

Todos estos elementos, nos van a permitir crear nuestro juego. Pero es importante definir todas las zonas que usaremos en el juego para gestionarlos correctamente. Para ello, vamos a ver el mapa de memoria de la NES.

5. Mapa de Memoria de la NES



5. Mapa de Memoria de la NES

Zonas lógicas de la memoria

- ZP (Zero page): Primeros 256 bytes de la memoria.
- OAM: Sección de la RAM que almacena una copia de la tabla de definición de sprites. (256 bytes)
- RAM: Resto de datos de la RAM.
- HDR: Header ROM. Contiene la cabecera de la ROM. 16 bytes
- PRG: ROM Program. Contiene el código de nuestro programa (32Kb).
- CHR; Tiles ROM. Contiene los gráficos de nuestro programa (8Kb).

5. Mapa de Memoria de la NES

Segmentos

Los siguientes segmentos se usarán en nuestro ensamblador.

- ZP: Zero Page
- OAM: tabla de Sprites
- BSS: Información para las paletas
- HEADER: Cabecera
- CODE: Código del programa
- RODATA: Información almacenada en nuestra ROM
- VECTORS: Almacena los vectores de interrupción o reset.
- TILES: Almacenará la información de los Tiles o gráficos.

6. Herramientas de Desarrollo

Veamos qué herramientas de Desarrollo necesitaremos:

- Un editor de código: Visual Studio Code con la extensión CC65 Macro Assembler
- Un programa ensamblador: Utilizaremos el compilador/ensamblador CC65.
- Un Emulador para NES: Utilizaremos Mesen
- Un editor de Tiles para NES. Utilizaremos NEXXT (Solo Windows)

7. Ensamblador para 6502

Puedes encontrar información sobre las instrucciones de ensamblador para 6502 en la siguiente presentación:

Presentación ensamblador 6502

Recuerda que utilizaremos diferentes instrucciones para crear nuestro juego y que explicaremos ahora solo algunas instrucciones. Para ayudarnos a comprender mejor las instrucciones, usaremos el siguiente simulador online:

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

7. Comenzar Proyecto

Vamos a comenzar nuestro proyecto pero... por dónde empezamos?

- Crear nuestras funciones y macros
- Variables y constantes necesarias (ZP, puertos,etc...).
- Inicializar la consola:
  - Configurar interrupciones
  - Inicio PPU
  - Inicio OAM
- Preparar fondos
- Iniciar Sprites
- Leer mandos

7. Comenzar Proyecto (Funciones)

En muchas ocasiones, necesitaremos crear funciones o subprogramas para poder reutilizar código y ser llamada desde cualquier parte del código. Una llamada a una función almacena en el puntero a pila la dirección de retorno para continuar. Veamos un ejemplo:

```
.proc sumy2
    ldy #2 ; Almacenamos el valor 2 en el registro Y
    sty $0300 ; Guardamos el valor de Y en la dirección de memoria $0300
    clc ; Limpiamos el flag de acarreo antes de la suma
    adc $0300 ; Sumamos el valor en $0300 (2) al acumulador (0)
    sta $0301 ; Guardamos el resultado de la suma en la dirección de memoria $0301
    rts ; retornamos de la subrutina
.endproc
```

```
lda #3
jsr sumy2
```

7. Comenzar Proyecto (Funciones)

En ocasiones, no es necesario crear funciones sino macros; las macros son “funciones” que son manejadas en tiempo de ensamblado; es decir que no se realizan llamadas sino que el propio ensamblador copia su contenido donde se llame.

```
.macro SUBTRACT_A_FROM_B dest, src
    LDA \src
    SEC             ; Set carry for subtraction
    SBC \dest
    STA \dest
.endmacro
```

```
SUBTRACT_A_FROM_B $05, $03
```

7. Comenzar Proyecto (Variables y constantes)

Otro aspecto importante, es donde almacenar los datos. Normalmente se utilizan variables en la ZP (Zero Page) o en cualquier otro lugar disponible.

También es importante tener en cuenta las direcciones especiales (o puertos) de los diferentes dispositivos; para ayudarnos a recordarlas, se utilizan las llamadas constantes.

Variables

```
.segment "ZEROPAGE"

nmi_ready:      .res 1
gamepad:        .res 1

d_x:            .res 1
d_y:            .res 1
```

Constantes

```
; Joystick/Controller values
JOYPAD1 = $4016 ; Joypad 1 (Read/Write)
JOYPAD2 = $4017 ; Joypad 2 (Read/Write)

; Gamepad bit values
PAD_A      = $01
PAD_B      = $02
PAD_SELECT = $04
PAD_START  = $08
PAD_U      = $10
PAD_D      = $20
PAD_L      = $40
PAD_R      = $80
```

8. Inicializar Consola

Antes de hacer cualquier acción de nuestro juego, necesitaremos inicializar todos los elementos del hardware y los recursos que vamos a utilizar (nametables, sprites, paletas, interrupciones,...). Es por ello que veremos algunas de estas acciones:

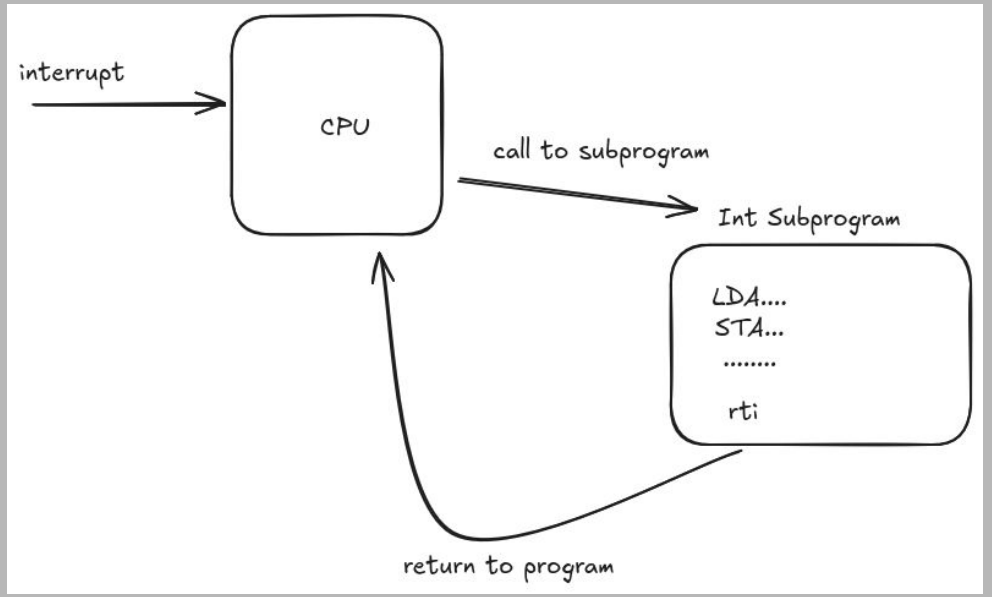
- Inicializar subprogramas para interrupción
- inicializar fondos
- inicializar paletas
- inicializar sprites

8. Inicializar Consola

Interrupciones (vectores)

Antes de continuar, vamos a ver qué son las interrupciones y sus rutinas.

Una interrupción, es una señal que recibe el procesador y que al recibirla, ejecutara un subprograma llamada “rutina de interrupción”.



8. Inicializar Consola

Interrupciones (vectores)

NES, tiene un espacio de memoria para almacenar las direcciones de los manejadores de estas interrupciones. a este espacio o segmento se le llama *vectors*; y normalmente establece 3 manejadores:

- *IRQ*: Interrupción por hardware. Establece la subrutina que se llamará cuando un hardware externo mande información.
- *NMI*: Interrupción lanzada cuando se pinta la pantalla (vBlank).
- *Reset*: Vuelve al punto inicial de la consola.

8. Inicializar Consola

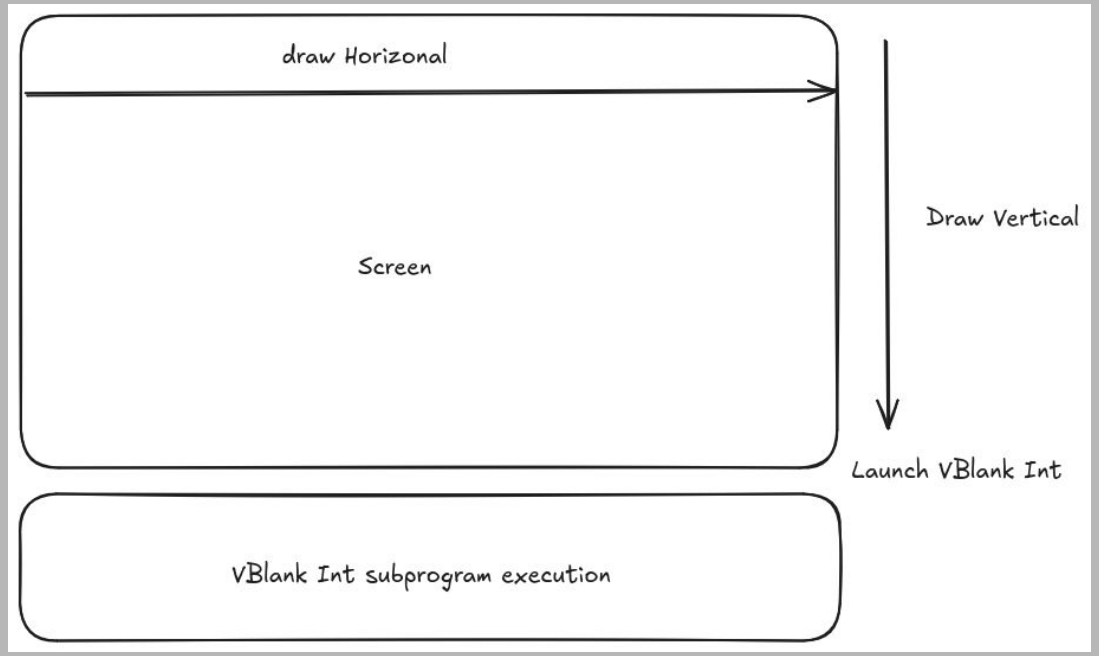
Interrupciones (vectores)

NES, tiene un espacio de memoria para almacenar las direcciones de los manejadores de estas interrupciones. a este espacio o segmento se le llama *vectors*; y normalmente establece 3 manejadores:

- *IRQ*: Interrupción por hardware. Establece la subrutina que se llamará cuando un hardware externo mande información.
- ***NMI*: Interrupción lanzada cuando se pinta la pantalla (vBlank).**
- *Reset*: Vuelve al punto inicial de la consola.

8. Inicializar Consola

Veamos cómo funciona la interrupción NMI (VBlank).



8. Inicializar Consola

Por lo tanto en el tiempo que tarda en reiniciar el pintado de pantalla, podemos:

- Copiar los registros a la Pila
- Actualizar los Sprites
- Actualizar los fondos e información del juego.
- Reiniciar los valores de la pila a los registros.

8. Inicializar la consola

Actualización de PPU

Es importante conocer cómo funciona el chip PPU (Picture Processor Unit); dentro de la NES. La forma que tiene para comunicarse la CPU con el PPU, es a través de puertos (Direcciones de memoria).

Habrás podido ver que hay varias direcciones de memoria:

```
; Define PPU Registers
PPU_CONTROL = $2000 ; PPU Control Register 1 (Write)
PPU_MASK = $2001 ; PPU Control Register 2 (Write)
PPU_STATUS = $2002; PPU Status Register (Read)
PPU_SPRRAM_ADDRESS = $2003 ; PPU SPR-RAM Address Register (Write)
PPU_SPRRAM_IO = $2004 ; PPU SPR-RAM I/O Register (Write)
PPU_VRAM_ADDRESS1 = $2005 ; PPU VRAM Address Register 1 (Write)
PPU_VRAM_ADDRESS2 = $2006 ; PPU VRAM Address Register 2 (Write)
PPU_VRAM_IO = $2007 ; VRAM I/O Register (Read/Write)
SPRITE_DMA = $4014 ; Sprite DMA Register
```

8. Inicializar la consola

Inicialización y Gestión de Sprites

Existe una zona de memoria llamada Oam(Object Attribute Memory) que nos va a permitir almacenar información para los Sprites. Normalmente se establece en una zona de memoria de 256 bytes; para cada sprite se almacena:

- Posición Y en pantalla (1 byte).
- Índice del tile (Patrón) a utilizar (1 byte).
- Atributos (1 byte): dependiendo del bit de este byte:
  - 0-1: La paleta a utilizar (4 disponibles).
  - 5: Indica si el Sprite se muestra delante o detrás del fondo actual.
  - 6: Indica si está volteado horizontalmente.
  - 7: Indica si está volteado verticalmente.
- Posición X en la pantalla(1 byte).

9. Referencias

- Classic Game Programming on the NES - Tony Cruise Ed. Manning.
- CC65: https://cc65.github.io/
- NEXXT Studio: https://frankengraphics.itch.io/nexxt
- Mesen: https://www.mesen.ca/