

TUTORIAL:
PROGRAMACIÓN DE CONSOLAS DE 8 y 16 BITS, PARA NOVATOS
1ª PARTE : INTRODUCCIÓN

Por David Senabre, Julio 2005

BREVE INTRODUCCIÓN

Seguro que muchos hemos jugado con una NES, o una Megadrive, y hemos pasado momentos divertidos, o tensos, o emocionantes (y también aburridos. De todo hay en la vida). Pero tal vez muy pocos se han parado a pensar de donde venía todo lo que estaban viendo, o en definitiva, cómo se hacía para crear esos juegos, animarlos y darles realismo. Con este documento espero arrojar un poco de luz a aquellos que se planteen estas preguntas y sean novatos en el tema.

Hay documentos muy buenos y completos sobre esto, pero que a muchos, que no tienen conocimientos en la materia, pueden parecerles impenetrables. Mi objetivo es acercar este "misterio" a aquellos que ven esto como algo desconocido.

Eso sí, hay un mínimo. Quién desee iniciarse en la programación de consolas de 8 y 16 bits, deberá tener conocimientos básicos de informática y programación, al menos de alto nivel, es decir, que conozca algún lenguaje de programación (C/C++, Pascal, Delphi, Java,...), aunque no es del todo imprescindible. Y sería MUY útil saber algo de ensamblador. Pero si no sabes ensamblador no te frustres, ese es un obstáculo salvable. De hecho, con suficientes ganas, cualquier puede aprender, por poco que sepa.

¿QUE ES UN PROGRAMA?

Básicamente, y como muchos ya sabréis, un programa es un "algo" que se ejecuta en una máquina, como por ejemplo un ordenador, y que hace "algo". Un ejemplo es el navegador que usas para ver páginas Web. Un programa en definitiva es una secuencia de órdenes que una máquina tiene que cumplir, y no tiene porque ser un ordenador. Por ejemplo una lavadora tiene diferentes programas de lavado. Así, ejecutar un programa es hacer que una máquina ejecute una serie de acciones, una tras otra.

En el caso que nos interesa, el de un ordenador, o una consola, un programa es algo que funciona en esa máquina y que hace alguna función, interactuando con el usuario, y habitualmente mostrando unos resultados por una pantalla.

¿QUÉ ES UN VIDEOJUEGO?

Un videojuego es simplemente un programa que está hecho con el objetivo de entretener. Tal y como se conocen, tienen un apartado gráfico especialmente cuidado, que otros programas no necesariamente deberían tener. Por ejemplo, muchos programas muestran básicamente texto, y unos pocos colores y figuras geométricas sencillas, y con ello pueden realizar tareas complejas, o muy complejas, mientras que de un videojuego esperamos unos gráficos, unos personajes y unas ciertas exigencias de colores, y sonoras, por ejemplo. No quiero decir que un juego necesariamente deba ser gráfico, ya que se podrían hacer (y existen) juegos que sólo muestren texto, y sean grandes juegos, y muy divertidos. Pero es por dar una idea, y una diferencia sencilla de entender entre un programa cotidiano, y un videojuego, que no es otra cosa que un programa, pero orientado al ocio.

¿CÓMO SE PROGRAMA UN PROGRAMA?

Existen formas de decirle a una máquina qué secuencia de órdenes debe ejecutar. Es decir, de "meterle" un programa.

En la actualidad todo se hace a través de lenguajes de programación de alto nivel, es decir, para quien no lo sepa, un lenguaje relativamente complejo que sirve para hacer programas. Y cada vez más se usan herramientas que automatizan y facilitan la programación, ya que cada vez las aplicaciones y los juegos son más y más complejos.

¿Y CÓMO SE PROGRAMA UN JUEGO?

El caso de las consolas de 8 y 16 bits es distinto. No se usan lenguajes de alto nivel, porque para poder ser ejecutados, deben traducirse a código máquina. Un ordenador, o consola, sólo entiende código máquina. Los lenguajes de programación son formas de comunicarnos de manera sencilla con ella, pero deben ser compilados, es decir, traducidos a código máquina, usando un compilador. Este proceso hace que el programa traducido sea un poco, o mucho menos eficiente que si se hubiera hecho directamente en código máquina. Y como las consolas de 8 y 16 bits no son lo que se dice muy potentes, este desperdicio de poder las penalizaría mucho, no siendo capaces de exprimir todo su potencial, e incluso, no siendo capaces de ejecutar ningún juego a una velocidad decente (como sería el caso de la NES).

Por ello, se deben programar instrucción por instrucción. Pero como el código máquina es una sucesión de "unos" y "ceros", lo único que entiende un ordenador, se usan lenguajes ensamblador, que se trata de dar a cada sucesión de "unos" y "ceros" que representa una instrucción, un nombre abreviado, llamado nemotécnico. Cada nemotécnico se traducirá a su secuencia de "unos" y "ceros" por un ensamblador.

Así, obtendremos todo el poder de nuestra máquina. En ensamblador se puede hacer literalmente todo lo que la máquina sea capaz de hacer. Pero tiene un lado oscuro...

¿POR QUÉ NO SIEMPRE SE PROGRAMA EN ENSAMBLADOR?

La contrapartida del ensamblador es que es mucho más difícil de desarrollar aplicaciones grandes, de mantener el código organizado, de entenderlo, y de depurarlo. Y cuando digo mucho más difícil, digo MUCHO. Por ello, hoy en día se hacen en ensamblador únicamente pequeños códigos que requieran ser ejecutados muy rápidamente, o deban ser muy optimizados, por ejemplo. También se usa para programar pequeños microcontroladores, como los PIC, que son pequeños ordenadores de potencia muy limitada, metidos en un chip pequeño, y que se usa con frecuencia en circuitos electrónicos. Y por su puesto se usa el ensamblador en todas las consolas de 8 y 16 bits que yo conozco.

¿CÓMO PUEDO EMPEZAR A PROGRAMAR UNA CONSOLA DE 8 o 16 BITS?

En primer lugar es necesario saber algo de informática y programación. Hay que conocer conceptos como CPU, RAM, ROM, etc. Y luego aprender ensamblador. Pero como ensamblador no es más que código máquina, hay un ensamblador diferente para cada CPU, y para programar una consola, necesitas saber el ensamblador de su CPU.

Para la NES, el ensamblador del 6502.

Para la SNES, el ensamblador del 65c816.

Para la Master System, el ensamblador del Zilog Z80.

Para la Megadrive, el ensamblador del Motorola 68000.

Puede parecer mucho más caótico o difícil de lo que parece el que cada consola use una CPU distinta, y por tanto un ensamblador propio. Pero

aprender un nuevo ensamblador cuando ya se conoce otro es una tarea muy sencilla en comparación con aprender por primera vez un ensamblador. Es como programar. Lo difícil no es aprender un nuevo lenguaje. Si ya sabes programar, aprendes rápido. Gente profana en este tema no parece entenderlo bien. Pero lo difícil es pensar de la manera adecuada, y abstraer. Aprender un nuevo lenguaje, o ensamblador, es como aprender a hablar otro idioma. Pero aprender a programar es aprender a hablar.

En mi opinión, y con la experiencia que tengo, el más sencillo de estos 4 ensambladores, es el del 6502. Pero la consola no es sólo una CPU, sino que tiene otros chips que apoyan a la CPU principal, y que realizan tareas de video, audio, etc. Para programar una consola es necesario conocer sus especificaciones técnicas, su arquitectura, sus componentes y cómo funcionan, cómo comunicarse con ellos, el mapa de memoria, etc. Esta información se halla en los llamados documentos técnicos, que se pueden hallar en Internet, y que han sido escritos por aficionados o almas caritativas que han decidido compartir con otros lo que saben. En mi Web encontraras recopilaciones de estos documentos, suficiente como para empezar a programar una consola.

Una vez que conoces algo del ensamblador de la consola, conoces su arquitectura interna, y tienes un programa para ensamblar, puedes empezar a programar algo fácil. Te será de mucha ayuda alguna demo muy sencilla hecha por alguien. Tanto los programas para ensamblar (ensambladores) como alguna demo la podrás hallar en mi Web.

¿CÓMO PUEDO PROBAR LAS ROMS QUE HAGA?

Lo más cómo es hacerlo en un emulador.

Pero para quién tenga ganas y posibilidad de hacerlo, se puede probar en una consola real si se dispone o se fabrica un cartucho regrabable de desarrollo.

Eso sí, es mucho más rápido probar en un emulador mientras estamos trabajando en la ROM, porque hay que hacer muchas pruebas y depurar bastante, y es muy tedioso grabar un cartucho cada vez que quieras probar un pequeño cambio o tantear si averiguas donde hay un puñetero bug...

```
/******\  
|¿CONCEPTO BÁSICOS? ¿PRIMEROS PASOS?|  
\\*****/
```

Hay ciertas cosas que se deben conocer bien para que esto de programar una consola sea posible. Intentaré ir introduciendo los conceptos necesarios. Primeramente, una consola es un ordenador, diseñado y optimizado para el juego. Como ordenador, consta de varias partes fundamentales, como son la CPU, la memoria y el sistema de entrada salida.

MEMORIA

La memoria es un almacén de datos temporales, que se usan en la ejecución de un programa, o en nuestro caso, de un juego. Ha de verse como un conjunto de posiciones que almacenan un byte cada una. Si la memoria es de 2Kb, significa que tiene capacidad para 2048 bytes. Es decir, tiene 2048 posiciones. Para escribir o leer en la memoria, hay que dar la dirección. El primer byte va a la dirección 0, el segundo a la dirección 1, y así hasta la dirección 2047. Normalmente estas direcciones se dan en hexadecimal, por comodidad. Para esta memoria de 2kb, las direcciones irían de \$000 a \$7FF (se suele escribir \$000-7FF)

Hay diversas memorias en una consola, habitualmente:
La RAM principal.
La VRAM (para tareas de video).
La ROM (donde están almacenados los juegos).

La CPU de la consola maneja las memorias RAM y ROM, al igual que los chips específicos que la ayudan en sus cálculos, de los que iré hablando. A cada cosa se le asigna un rango de direcciones. Así, por poner un ejemplo sencillo, si una CPU pudiera dar direcciones desde \$0000 hasta \$1FFF, el rango \$0000-\$07F puede referirse a una memoria, y \$0800-\$1FFF a otra. Esto crea distintas zonas, que lógicamente no debe solaparse. Cada zona se reserva para un dispositivo.

MAPA DE MEMORIA

Es una tabla que recoge qué significa para la consola cada dirección que da la CPU. Como a la CPU van conectados diferentes dispositivos, habrá que saber como referirse a cada uno de ellos, por ejemplo, a la RAM, a la ROM de los cartuchos, o al chip de video o audio. Cada CPU puede dar un máximo posible de direcciones. Cuantas más direcciones sea capaz de dar, más memoria podrá manejar.

Por ejemplo, las CPU de la NES o la Master System pueden manejar 64Kb (es decir, direcciones \$0000-\$FFFF). Eso quiere decir que se podría usar, por ejemplo, los primeros 32Kb (0-31) para referirse a 32Kb de RAM, y los segundos 32Kb (32-63), para referirse a 32Kb del cartucho. Pero si se hace esto no quedan direcciones para comunicarse con el resto de chips de la consola. En realidad, ni la NES ni la Master System tienen un mapa de memoria tan sencillo (puedes ver cual tienen en mis documentos "Lo más básico sobre...", parte 1 y parte 2).

Básicamente, lo que se hace es dedicar un conjunto de direcciones para acceder a la RAM, otro para la ROM del cartucho, otro para **puertos**, y a veces otro para memorias de expansión o cualquier otro propósito. Y este mapa de memoria varía de una consola a otra, claro. Ahora habrá que ver que es eso de puertos.

PUERTOS

Así se llama a una dirección de la CPU que no se refiere a memoria, ni RAM, ni ROM, sino que se refiere a un dispositivo, por ejemplo, a otro chip de la consola, como puede ser el procesador de video.

Un puerto o registro, es algo en lo que se puede escribir y leer información, y que suele tener muy poca capacidad. Habitualmente 8, 16 o 32 bits (o sea, 1, 2 o 4 bytes). ¿Para qué puede servir entonces? Pues bien, al escribir a ciertos puertos, se puede mandar datos a un chip específico, y al leer de ciertos puertos, recibirlos de ellos. Es la manera de comunicarse con el "mundo exterior", y por eso se llaman puertos o registros de I/O (entrada/salida).

Por ejemplo, para cargar los gráficos de un juego, éstos deben almacenarse en la VRAM, o RAM de video. Esta RAM es como la RAM principal, pero no es la misma, tiene distinto objetivo. Está exclusivamente dedicada a lo relacionado con los gráficos y las tareas de video. No puede ser accedida directamente por la CPU, sino que debe ser accedida a través del procesador de video de la consola. ¿Y cómo? Pues muy fácil, claro. Hay unos puertos dedicados a comunicarse con el chip de video para encargarle cosas, y usando alguno de esos puertos, se le puede ordenar que escriba o lea información en la VRAM.

Para escribir o leer un puerto necesitas saber su dirección, como si se tratara de un acceso a memoria RAM. Imagina un puerto como un cajón, donde dejas información que otro dispositivo puede ver, y donde él te puede dejar cosas para que tú las leas.

Puedes imaginar un puerto de 8 bits como:

```

-----
| b | b | b | b | b | b | b | b |
-----
d7 d6 d5 d4 d3 d2 d1 d0

```

Donde las "b" significan bit. Los 8 bits del puerto forman un byte. Los bits se numeran desde el menos significativo (d0) al más significativo (d7). Por tanto, si escribimos \$47 (01000111 en binario) en el puerto \$4016, éste tendría esta información.

\$4016

```

-----
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
-----

```

Entonces podríamos decir que su bit d6 está a 1, y su bit d3 está a 0, y así sucesivamente. Esto es importante porque a veces cada bit del registro sirve para indicar una cosa al dispositivo.

APARTADO GRÁFICO

Planos de scroll

Como es lógico, el mostrar gráficos por pantalla es algo imprescindible. Pero, ¿cómo se hace?

En líneas generales, lo básico es cargar los tiles que se necesiten en la VRAM, en un orden conocido, y decirle al chip de video lo que queremos que muestre en pantalla. Para esto último, existe alguna tabla (depende de la consola) que representa la pantalla, y dando valores a cada elemento de esta tabla, conseguimos que muestre uno u otro tile en esa posición. Si la pantalla es de 32x30 tiles (ancho x alto), como en el caso de la NES, (cada tile ocupa 8x8 pixels, y la pantalla tiene 256x240 pixels), la tabla tendrá 32x30 = 960 bytes. Un byte para cada tile. Y si queremos que se muestre en la esquina superior izquierda el tile de posición 7, habrá que escribir un 7 en el principio de la tabla. Esta tabla se llama **Name Table** en el caso de la NES.

Para que resulte ilustrativo, supongamos una hipotética consola cuya pantalla esté compuesta únicamente de 5x5 tiles. Una pantalla podría tener este aspecto.

0	0	1	1	3
2	1	0	0	0
0	0	0	0	0
0	0	1	1	1
0	0	0	0	0

El número haría referencia al tile que se mostraría, el 0 para el primero, el 1 para el segundo, el 2 para el tercero, etc.

Recuerda que todo lo que se almacena en una memoria lo hace de forma secuencial, es decir, un byte tras otro, luego no se puede meter la tabla con anchura y altura, sino un elemento tras otro, y al llegar al final de una fila, el primer elemento de la siguiente fila va seguido en la memoria. La tabla entonces debería almacenar la siguiente información.

Tabla:

0	0	1	1	3	2	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Cada consola tiene sus peculiaridades. Tanto la NES como la Master System sólo tienen un plano de scroll, pero la Megadrive y la SNES tienen varios, luego habrá una tabla para cada plano de scroll. La NES, por ejemplo, tiene 4 tablas, aunque sólo usa 2, y de esas 2, una va a continuación de la otra.

Sprites

Una vez puesto el fondo, se necesitan objetos que interactúen en el mundillo que hemos creado. En un juego, por una parte, están los escenarios, las plataformas, etc. Que forman parte de los planos de scroll, constituidos por tiles, y que sirven para representar los niveles, los mapas, o lo que sea, en definitiva, el mundo que se representa en el juego.

Y por otra parte están los sprites, que son los objetos móviles que "viven" en ese mundo, tanto los enemigos, como el propio personaje que controla el jugador, o los ítems. En definitiva, los sprites son los objetos que interactúan en los juegos.

Al igual que antes, ahora debemos almacenar la información de los sprites en una zona de memoria dedicada a eso. En la NES se llama SPR-RAM (sprite RAM) y es parte de la VRAM.

Habitualmente, se dedican un par de bytes para cada sprite, uno para la posición X en la pantalla, otro para la posición Y, otro para el tile que se debe mostrar y otro para atributos (como mirroring, color, ...), por ejemplo.

Cada consola tiene un número máximo de sprite que puede manejar simultáneamente. Y esta limitación la impone el tamaño de la memoria de sprites. La SNES, por ejemplo, puede manejar 128 sprites, mientras que la NES sólo 64.

Colores y Paleta

Cada consola es capaz de mostrar un cierto número de colores distintos. A más colores, más variedad de tonos. Pero otra cosa es cuántos de esos colores puede mostrarlos a la vez en la pantalla. Lo normal es que sólo sea capaz de usar un grupo de ellos, no todos. A este grupo se le llama paleta. En cada momento, el programador debe cargar los colores que quiera usar en la paleta. A pesar de que la consola pueda mostrar más colores, sólo los que estén en la paleta pueden usarse en ese momento. Es como un pintor. Los colores que es capaz de usar serán los que sea capaz de mezclar. Pero los que pueda usar en un momento dado, serán los que tiene en su paleta, es decir, lo que ya ha mezclado.

Por ejemplo, la NES puede mostrar 52 colores, pero su paleta es de 16. La Master System puede mostrar 64 colores, pero su paleta es de 16.

Realmente estas 2 consolas tiene 2 paletas. Una para los sprites, y otra para los fondos. Cada una de 16 colores. Pero en el caso de la NES, hay 4 colores de cada paleta que no se usan realmente, porque representan el color transparente. Luego cada paleta permite almacenar 12 colores. En total, la paleta de sprites y de fondos permiten mostrar 24 de los 52 colores posibles.

En la Master System sólo hay un color que se usa como transparente en cada paleta. Luego cada una almacena 15 colores, y en total permite mostrar simultáneamente 30 de los 64 colores posibles.

Ya sabes, tienes que distinguir entre máximos colores posibles, y paleta de colores (que siempre es más reducida, y son los que se pueden mostrar al a vez).

Un ejemplo más moderno sería la Nintendo DS. Aunque esta consola tiene varios modos gráficos, algunos son parecidos a los que usan las consolas clásicas (modos tile). Normalmente, cuando usa esos modos, la DS puede mostrar 32768 colores, pero su paleta es de 256. Lo mismo se aplica para la GameBoy Advance, aunque parezca mentira (la mejora de la DS frente a la Advance es, hasta donde se, potencia, memoria, y modos 3D). Pero más sorprendente es que la GameBoy Color tiene también 32768 colores en su abanico, pero con una paleta mucho más reducida, 56 como máximo.

ORIENTACIÓN Y CONCLUSIÓN

Eso es todo lo que quería contar en este documento, por ahora. Si quieres avanzar tendrá que hacerse con la documentación necesaria, de la consola elegida, y aprender ensamblador, al igual que otros conceptos de informática, y empezar a hacer maravillas para nuestras queridas 8 y 16 bits :)

Después de un tiempo, he decidido escribir una serie de documentos en clave de tutorial, para servir como breve pero completa introducción a la programación práctica. Así que ya sabes donde empezar.

He escrito algunas demos especialmente pensadas en el aprendizaje, que pueden seguirse línea por línea, y muestran aspectos básicos de la programación de la máquina. Recomiendo empezar por la NES, aunque si te interesa algún sistema en concreto, no dudes ir a por él. Te motivará más. La NES es la más sencilla para mí. Por tanto es un buen sistema para empezar si no se tiene claro.

Respecto a las demos de otros autores, debes saber que algunas son muy complejas para un principiante, así que no intentes entenderlo todo. Podría hastiarte. Puedes usarlas como apoyo, o para entrenarte en entender un programa escrito por otro, y averiguar que está haciendo en rasgos generales.

La primera consola que aprendí a programar fue la NES, aunque ya tenía experiencia en programación de alto nivel (Pascal, Basic y C), e incluso algo de ensamblador del x86 (PC). Sino es tu caso, paciencia, se puede conseguir.

Suerte :)