



UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA INFORMÁTICA

PROYECTO DE FIN DE CARRERA

**DISEÑO Y PROGRAMACIÓN DE UN EMULADOR DE LA
CONSOLA PORTÁTIL GAMEBOY**

Tutor: Jesús Carretero Pérez
Autor: Víctor García González

Diciembre, 2006

*A mi familia, por aguantarme
todo este tiempo y tener fé en mí.
A mis amigos, por apoyarme y animarme
a seguir adelante.*

Nintendo y GameBoy son marcas registradas por Nintendo Co., Ltd.

Zilog y Z80 son marcas registradas de ZiLOG Inc.

Todas las marcas registradas son propiedad de sus respectivos dueños.

ÍNDICE GENERAL

1. INTRODUCCIÓN.....	11
1.1. Organización del libro	12
2. ESTADO DEL ARTE.....	15
2.1. Emulación, Simulación y Virtualización.....	16
2.2. Legalidad sobre la emulación	17
2.3. Emuladores de consolas.....	18
2.3.1. Tercera generación	18
2.4. Técnicas de emulación	25
2.4.1. Interpretación	25
2.4.2. Recompilación estática.....	26
2.4.3. Recompilación dinámica	26
2.5. GameBoy	27
2.5.1. Historia.....	27
2.5.2. Hardware	28
2.5.3. Procesador.....	29
2.5.3.1. Instrucciones	30
2.5.3.2. Interrupciones.....	32
2.5.3.3. Registros	33
2.5.3.4. Memoria.....	34
2.5.3.5. Video.....	35
2.5.4. Accesorios.....	38
2.5.5. Juegos.....	42
3. DESCRIPCIÓN DEL SISTEMA.....	45
3.1. Visión General del Sistema	45

3.2.	Emulador de la Arquitectura.....	45
3.2.1.	Procesador.....	46
3.2.1.1.	Registros	46
3.2.2.	Instrucciones	47
3.2.2.1.	Juego de instrucciones.....	47
3.2.3.	Memoria Principal.....	48
3.3.	Emulador del hardware externo.....	48
3.3.1.	Cartuchos	49
3.3.1.1.	Memory Bank Controller	49
3.3.2.	Teclado.....	51
3.4.	Interfaz Gráfica de Usuario	51
3.4.1.	GUI	52
3.4.1.1.	Pantalla.....	52
3.4.2.	Gráficos.....	53
4.	DISEÑO ARQUITECTÓNICO	55
4.1.	Arquitectura del Sistema.....	55
4.1.1.	Arquitecturas.....	55
4.1.2.	Modelo Arquitectónico	57
4.2.	Componentes del Sistema.....	59
4.2.1.	Kernel.....	59
4.2.1.1.	Procesador.....	59
4.2.1.2.	Memoria.....	60
4.2.1.3.	Registros	61
4.2.2.	Periféricos	61
4.2.2.1.	Cartucho.....	62
4.2.2.2.	Teclado.....	62
4.2.2.3.	Debug	63
4.2.3.	Interfaz Gráfica	64

4.2.3.1.	GUI	64
4.2.3.2.	Pantalla.....	65
4.2.3.3.	Gráficos.....	65
4.3.	Diagrama de Componentes del Sistema.....	67
5.	DISEÑO DETALLADO.....	69
5.1.	Nomenclatura.....	69
5.2.	Kernel	70
5.2.1.	Kernel.CPU.....	71
5.2.2.	Kernel.Registros.....	74
5.2.3.	Kernel.Memoria	76
5.2.4.	Kernel.Constantes	77
5.2.5.	Kernel.Instruccion.....	78
5.2.6.	Kernel.InstruccionXX	79
5.3.	Periféricos.....	82
5.3.1.	Periféricos.Cartucho.....	83
5.3.2.	Periféricos.CartuchoMBC0.....	85
5.3.3.	Periféricos.CartuchoMBC1	86
5.3.4.	Periféricos.CartuchoMBC2.....	87
5.3.5.	Periféricos.CartuchoMBC3	87
5.3.6.	Periféricos.CartuchoMBC5.....	88
5.3.7.	Periféricos.Keypad.....	89
5.3.8.	Periféricos.Debug.....	90
5.4.	GUI.....	93
5.4.1.	GUI.GUI	94
5.4.2.	GUI.Pantalla.....	96
5.4.3.	GUI.Graphics	97
6.	IMPLEMENTACIÓN DEL SISTEMA.....	101

6.1.	Tecnología empleada	101
6.1.1.	Lenguaje de Programación.....	101
6.1.2.	Plataforma de Desarrollo.....	102
6.2.	Codificación.....	103
6.2.1.	Procesado gráfico.....	104
6.2.1.1.	HBlank	104
6.2.1.2.	VBlank	109
6.2.2.	Gestión de interrupciones.....	110
6.2.3.	Resumen.....	115
7.	CONCLUSIONES.....	117
7.1.	Líneas futuras.....	118
8.	PRESUPUESTO	119
9.	BIBLIOGRAFÍA.....	121
9.1.	Referencias	121
9.2.	Enlaces	122
A.	INSTRUCCIONES DE ENSAMBLADOR.....	123
B.	MANUAL DE USUARIO	137

TABLA DE ILUSTRACIONES

Imagen 1: Consola Nintendo Entertainment System.....	18
Imagen 2: Emulador NESTicle version x.xx en MS-DOS	19
Imagen 3: Emulador Nestopia v1.3.4 en MacOS X	20
Imagen 4: Consola Sega Master System II.....	20
Imagen 5: Emulador MasterGear v2.0r	21
Imagen 6: Emulador MEKA.....	21
Imagen 7: Emulador VisualBoyAdvance en Windows	22
Imagen 8: Emulador KiGB en Windows.....	23
Imagen 9: Consola portátil Sega Game Gear	23
Imagen 10: Emulador MEKA con un juego de Sega Game Gear	24
Imagen 11: Consola portátil Atari Lynx.....	24
Imagen 12: Consola portátil NintendoGameBoy	27
Imagen 13: Lateral derecho de la consola portátil GameBoy.....	28
Imagen 14: Procesador de la GameBoy.....	29
Imagen 15: Sprite de 8x8 píxeles perteneciente al juego Super Mario Land	36
Imagen 16: Composición de texto mediante sprites de 8x8 píxeles.....	36
Imagen 17: Super GameBoy.....	39
Imagen 18: GameBoy Battery Pack	39
Imagen 19: GameBoy Camera.....	40
Imagen 20: GameBoy Printer	41
Imagen 21: GameBoy Link Cable	42
Imagen 22: Interior de un cartucho de GameBoy.....	43
Imagen 23: Arquitectura de máquina virtual	56
Imagen 24: Arquitectura por Capas	57
Imagen 25: Modelo Arquitectónico del Sistema	58

Imagen 26: Patrón Command	58
Imagen 27: Diagrama de Componentes del Sistema	67
Imagen 28: Kernel.CPU.....	71
Imagen 29: Kernel.Registros	74
Imagen 30: Kernel.Memoria.....	76
Imagen 31: Kernel.Instruccion	78
Imagen 32: Perifericos.Cartucho	83
Imagen 33: Perifericos.Keypad	89
Imagen 34: Perifericos.Debug	90
Imagen 35: Gui.Gui	94
Imagen 36: Gui.Pantalla	96
Imagen 37: Gui.Graphics.....	97
Imagen 38: Pantalla principal de la aplicación	137
Imagen 39: Pantalla principal de la aplicación en funcionamiento	138
Imagen 40: Abrir un fichero o salir	138
Imagen 41: Información sobre un cartucho válido	139
Imagen 42: Opciones de control de la emulación.....	140
Imagen 43: Consola de depuración.....	142
Imagen 44: Estado de los registros y flags	142
Imagen 45: Estado de las interrupciones	142
Imagen 46: Volcado de una zona de memoria.....	143
Imagen 47: Flujo de ejecución de instrucciones.....	143

1. INTRODUCCIÓN

El objetivo del proyecto es simple: conseguir un emulador funcional de la consola portátil GameBoy de Nintendo. Un emulador es un programa creado específicamente para realizar el mismo comportamiento, tanto a nivel hardware como software, de una plataforma totalmente diferente. En este caso se trata de replicar la arquitectura y funcionamiento de dicha consola sobre la plataforma de desarrollo Mono de la compañía Ximian, capaz de funcionar sobre diferentes sistemas operativos y arquitecturas hardware.

La GameBoy fue la primera consola portátil de éxito que existió. Su primera versión apareció en el mercado en 1989 a un precio de 109 dólares norteamericanos. Debido a su enorme popularidad (casi 70 millones de unidades vendidas a lo largo de toda su historia), dio lugar a varios modelos que la han ido sucediendo a lo largo del tiempo, pero es en ese primer modelo en el que se centra este emulador.

La consola por sí sola no tenía ninguna utilidad, ya que los juegos se distribuían en pequeños cartuchos electrónicos con carcasa de plástico que se conectaban a la consola a través de un puerto de comunicaciones especial. Del mismo modo, para obtener provecho de este emulador, son necesarios volcados de los datos contenidos en dichos cartuchos a ficheros fácilmente manipulables bajo cualquier entorno computerizado.

El nombre elegido para la aplicación es CSGBE, un acrónimo que significa “C# GameBoy Emulator” o en castellano “Emulador de Gameboy en C#” (el símbolo # se escribe “sharp” en inglés). El diseño del programa no se centra en la optimización o en la compatibilidad con los programas originales de la consola, sino en realizar una arquitectura escalable, fácilmente ampliable, y especialmente intuitiva. Es por ello que el desarrollo de este emulador podrá ser ampliado y mejorado en gran medida tras la conclusión de este proyecto.

1.1. ORGANIZACIÓN DEL LIBRO

El presente documento contendrá la siguiente estructura dividida en puntos:

- **1. Introducción:** El punto actual. Realiza un pequeño resumen sobre los objetivos del proyecto.
- **2. Estado del arte:** En el segundo punto se comentan los conceptos teóricos y prácticos sobre los que se sustenta el proyecto, además del estado actual en la tecnología de la emulación de arquitecturas y videoconsolas actualmente.
- **3. Descripción del sistema:** El punto destinado a la descripción del sistema expondrá las características del sistema a construir y de cada uno de sus componentes.
- **4. Diseño arquitectónico:** En el diseño arquitectónico se realizará una descomposición modular del sistema a construir, identificando todos los integrantes de alto nivel del mismo. Se comentará el uso de patrones arquitectónicos y su aplicación al proyecto.
- **5. Diseño detallado:** En el diseño detallado se definirán en detalle cada uno de los módulos definidos en el diseño arquitectónico, identificando los datos almacenados y las funciones de cada componente. Se incluirán diagramas UML de los componentes de cada módulo.
- **6. Implementación:** El punto 6 contendrá detalles de la implementación de los puntos más delicados o representativos de la aplicación, comentándose en detalle y mostrando código real de la aplicación.
- **7. Conclusiones:** El apartado contendrá las conclusiones obtenidas tras la finalización del proyecto, así como posibles extensiones al mismo no incluidas pero que pueden ser útiles o interesantes.
- **8. Presupuesto:** El punto 8 realizará un estudio sobre el coste económico del proyecto final.
- **9. Bibliografía:** Referencias a documentos en Internet o a libros utilizados tanto para la documentación del proyecto como para su desarrollo.

- **10. Apéndices:** El último apartado contendrá información del proyecto que no se ha considerado incluir dentro de la estructura del documento. Consiste en información de referencia tal como el manual de usuario y la guía del juego de instrucciones del procesador.

2. ESTADO DEL ARTE

En el contexto de la informática, un emulador es un programa o software capaz de ejecutar programas de una plataforma distinta para la cual fueron creador originalmente, ya sea en otra arquitectura o con otro sistema operativo.

El uso más común de los emuladores es aplicado a imitar de la manera más precisa posible el funcionamiento de videoconsolas o máquinas recreativas, normalmente con cierta antigüedad, que impiden hacer funcionar el hardware o software como lo hacía originalmente. Por ejemplo, el caso de videoconsolas que ya no se pueden encontrar en el mercado porque hace mucho tiempo que se dejaron de fabricar, o en el caso de software cuando no se tiene acceso a un sistema operativo antiguo, como puede ser MS-DOS.

Los emuladores nacieron por diversos motivos:

- Como reto para los programadores, que querían hacer programas capaces de recrear o emular una determinada máquina antigua en una moderna.
- Para poder jugar y utilizar arquitecturas antiguas que no tenemos a nuestro alcance.
- Con el fin de preservar todo el legado de estas arquitecturas, de forma que cuando estas ya no estén disponibles, todavía tengamos emuladores que nos permitan ver todo su catálogo de software.
- Para desarrollar en estas consolas y ordenadores, trabajando en nuestro ordenador habitual y probándolo en un emulador. Con esto nos evitamos grabar el juego en un cartucho, disco o cinta para tener que probarlo en el sistema real.
- Para modificar los programas originales, para realizar traducciones videojuegos existentes solo en idiomas extranjeros, o reescribiendo sus diálogos, principalmente en el caso de los videojuegos de rol. Igualmente la emulación permite aplicar parches para corregir los errores de software que tenían los programas originales.

La emulación hoy en día ha alcanzado un alto nivel de madurez, y podemos encontrar programas capaces de emular una gran variedad de hardware, y normalmente

cuando más antiguo es el objeto de emulación, más precisa suele ser ésta, con contadas excepciones. Para poder maximizar el nivel de compatibilidad de un emulador, es muy recomendable tener acceso a la documentación sobre el funcionamiento de dicha plataforma, pero por desgracia eso no siempre es posible y muchas veces tiene que recurrirse a la ingeniería inversa para intentar comprender cómo funciona.

A pesar de todos los esfuerzos para emular eficientemente un sistema, un emulador siempre requerirá más potencia computacional que la plataforma original.

2.1. EMULACIÓN, SIMULACIÓN Y VIRTUALIZACIÓN

A la hora de replicar el funcionamiento de un sistema, se puede abordar desde 3 perspectivas distintas, según el interés y la finalidad que se desee.

- **Emulación:** Se parte del propio funcionamiento interno del sistema, dividiendo el sistema en partes más simples de emular, pero siempre siguiendo el esquema del sistema original.
- **Simulación:** No es necesario conocer el funcionamiento interno del sistema original, teniendo sólo en cuenta el comportamiento externo para conseguir que produzca el mismo resultado.
- **Virtualización:** Solo disponible bajo ciertas combinaciones de arquitecturas. Intenta realizar una proyección directa de todos los componentes internos del sistema directamente sobre la plataforma huésped.

Por ejemplo, un emulador de una máquina “arcade” de juegos, imitaría fielmente todo su hardware interno y necesitaría los datos del cartucho del juego para poder funcionar, sin embargo, la simulación trataría de realizar una aplicación que imitara la misma sensación del juego, mismos gráficos, mismos sonidos y misma jugabilidad, pero no necesitaría los datos originales del juego. Por contra no se podría realizar un simulador genérico de toda una consola de videojuegos.

La virtualización puede ser parcial o completa, y normalmente se intenta utilizar cuando el sistema a imitar comparte grandes similitudes con la plataforma donde se quiere realizar la virtualización. Por ejemplo, si ambas plataformas comparten la arquitectura x86 se podrían ejecutar directamente las instrucciones de ensamblador de

una manera mucho más rápida disminuyendo la sobrecarga de traducir el juego de instrucciones.

2.2. LEGALIDAD SOBRE LA EMULACIÓN

Al replicar el funcionamiento de una máquina comercial, cuyo diseño es propiedad intelectual de su fabricante, al igual que los datos de los cartuchos de los juegos, es inevitable pararse a pensar sobre los límites legales de este procedimiento.

La respuesta no es universal y depende enormemente de la legislación de cada país o incluso territorio. En muchos casos no está del todo claro, pero los principales países europeos y Estados Unidos coinciden en que la emulación de un sistema registrado está permitida siempre y cuando no se haya obtenido la información para tal objetivo de manera ilícita, aunque esto puede cambiar si todavía se vende comercialmente la consola que está siendo emulada.

Con respecto a los propios juegos, se aplican las típicas leyes de propiedad intelectual de las obras: es ilegal distribuirlos pero está permitido copiar su contenido original a otros formatos o soportes bajo el amparo del derecho a copia privada.

Un error común en este aspecto, es pensar que sobre un juego antiguo esta ley deja de aplicarse o es menos restrictiva bajo el nombre de lo que suele conocerse como “abandonware”, software o juegos en este caso de compañías o personas que ya han desaparecido o sobre los que ya no son comercializados. Esto es completamente falso. Las dos únicas formas de poder distribuir libremente un juego antiguo son: obteniendo autorización expresa de los propietarios legales, que pueden ser una empresa o los autores originales o sus herederos en caso de personas físicas, o esperando a que expiren los derechos de autor que son 75 años en Estados Unidos y 50 años en Europa desde la fecha de publicación, teniendo en cuenta que con cada reedición dichos derechos son revalidados.

2.3. EMULADORES DE CONSOLAS

Existen infinidad de consolas a lo largo de todas las generaciones, y cuanto más éxito han tenido en su época, más emuladores han surgido en torno a ellas. Normalmente los emuladores han sido creados para plataforma PC y sistema operativo Windows, aunque también son muchos los que funcionan en otros sistemas operativos. Y cada vez son más los desarrollados en entornos multiplataforma como J2SE o .NET.

Con la aparición de la actual generación de consolas, su potencia ha empezado a ser suficiente para ser comparable con un PC (e incluso superarlo) y hoy en día podemos ver casi todas las consolas antiguas emuladas sobre ellas. Por ejemplo la consola SNES en una PS2, o una N64 en una XBox.

Sin embargo, y dada la extensión del tema, sólo se va a hacer un pequeño resumen de las consolas de tercera con la aparición de los 8 bits a principios de los años 80, que es a la que pertenece la GameBoy, y sus emuladores más conocidos.

2.3.1. TERCERA GENERACIÓN

NES (NINTENDO ENTERTAINMENT SYSTEM)

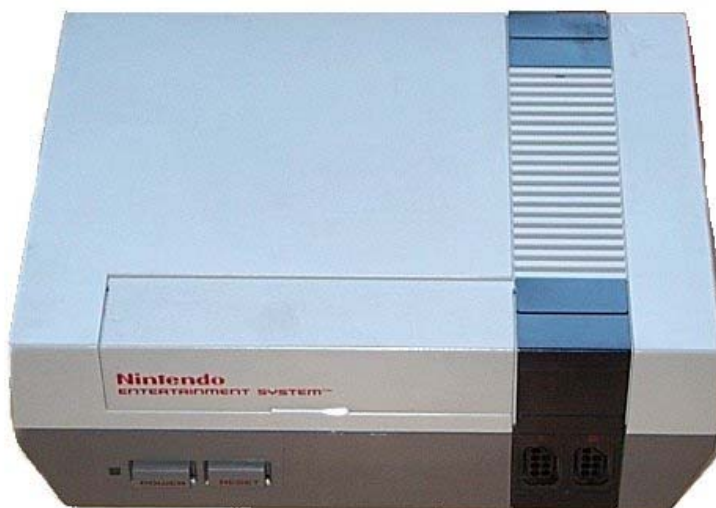


Imagen 1: Consola Nintendo Entertainment System

También conocida como Famicom. Contaba con un procesador 6502 entre 1 y 2 MHz y algo más de 4 KB de memoria base. Apareció en 1983 en Japón y en 1985-1986 en el resto del mundo. Es una de las consolas más emuladas en la actualidad, contando con casi 90 emuladores diferentes repartidos entre más de 11 plataformas o sistemas operativos diferentes, incluyendo otras consolas como PlayStation, DreamCast o portátiles como GameBoyAdvance, PocketPC, Palm o GP2X. Los más conocidos son:

- NESTicle [1]: El primero en alcanzar la compatibilidad con juegos comerciales. Apareció en 1997 para MS-DOS y Windows y fue abandonado en 1998. Incorporó algunas novedades en el mundo de la emulación como el guardado de estados, frameskip automático, juego en red y una gran optimización que le permitía funcionar en equipos antiguos.

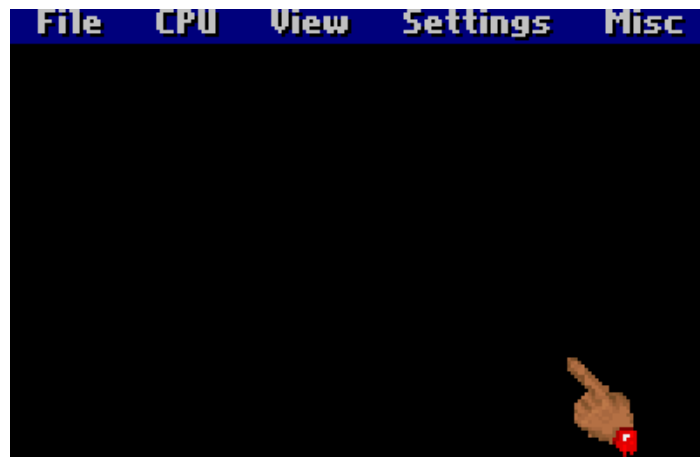


Imagen 2: Emulador NESTicle version x.xx en MS-DOS

- Nestopia [2]: Uno de los más compatibles y más precisos que existen actualmente. Funciona en plataformas Windows, Mac OS X y Linux. Su código se encuentra disponible bajo la licencia GPL.



Imagen 3: Emulador Nestopia v1.3.4 en MacOS X

SMS (SEGA MASTER SYSTEM)



Imagen 4: Consola Sega Master System II

Contó con varias revisiones de diseño aunque todas eran compatibles entre ellas. Contaba con un procesador Z80 a unos 3.5 MHz y unos 24 KB de memoria total. Fue competidora directa de la NES, y a pesar de su superioridad técnica fracasó en algunos mercados, mientras que en otros tuvo un éxito sin precedentes. Apareció entre los años 1986 y 1987. Debido a su fracaso parcial, no existen demasiados emuladores para ella. Los más destacados son:

- MasterGear [3]: Creado en 1996. Tiene un gran nivel de compatibilidad, pero lo más destacado es que debido a su diseño es altamente portable y de hecho existen versiones para todo tipo de dispositivos, como móviles Symbian, reproductores de DVD o cámaras digitales, y otras consolas como PlayStation o Nintendo64. Debido a la similitud de hardware, también emula la consola portátil Sega

GameGear. Por desgracia es comercial y de código cerrado. Su autor también ha desarrollado emuladores de otras consolas como MSX, GameBoy, GameBoyAdvance, NES u otras, todos ellos comerciales también.



Imagen 5: Emulador MasterGear v2.0r

- MEKA [4]: Emulador multi-plataforma entre las que se encuentra la SMS. También tiene una alta compatibilidad y una interfaz gráfica impecable. Iniciado en 1999, se encuentra disponible para MS-DOS, Windows y Linux. Su código fuente es libre.

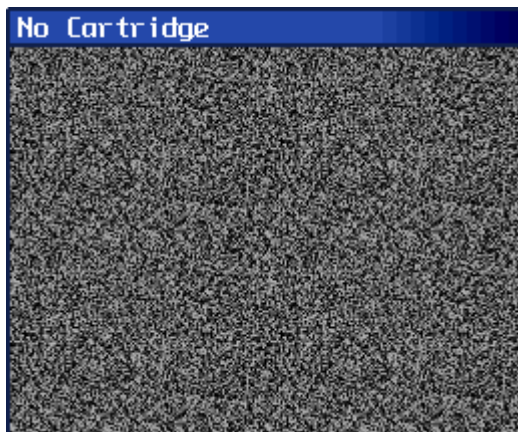


Imagen 6: Emulador MEKA

NINTENDO GAMEBOY

Apareció por primera vez en 1989 y su éxito ha sido tan grande que existen hasta 3 generaciones posteriores a día de hoy y varias variaciones del diseño original. Cuenta con un procesador Z80 ligeramente modificado a unos 4 MHz, unos 16 KB de memoria base y una pantalla de 2.6 pulgadas con 4 niveles de gris. Es sin duda la consola portátil

más emulada, con emuladores hasta en dispositivos portátiles como el iPod de Apple. Los más conocidos son:

- GnuBoy [5]: Creado en el 2000 y abandonado en el 2002. Tiene un nivel de compatibilidad cercano al 99%. Está programado en C con partes en ensamblador y con un buen diseño modular. Tiene licencia GPL y se encuentra portado a MS-DOS, Windows y Linux, aunque existen infinidad de conversiones no oficiales a todo tipo de plataformas.
- VisualBoyAdvance [6]: Orientado a la emulación de la consola GameBoy Advance, también soporta todas las versiones de Game Boy debido a la retrocompatibilidad que también tiene el hardware original. Tiene una alta compatibilidad y otras características como el guardado de estados, depurador integrado y soporte de algunos periféricos originales. Su código es GPL y gracias al uso de la librería gráfica SDL, ha podido ser portado a muchas plataformas.

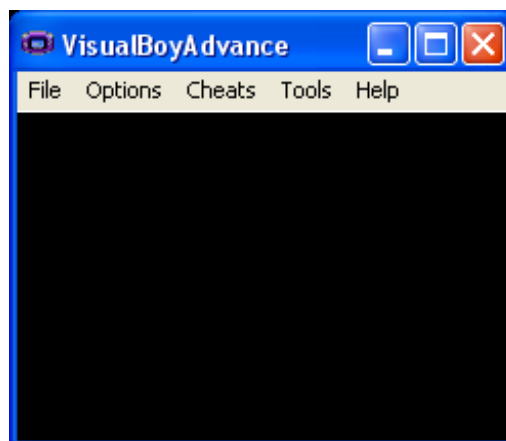


Imagen 7: Emulador VisualBoyAdvance en Windows

- KiGB [7]: El emulador con la mayor compatibilidad de todos, con soporte exclusivo de algunos efectos poco comunes usados en algunos juegos. Se encuentra disponible para Windows, Linux, MS-DOS y Mac OS. Es gratuito pero de código cerrado.



Imagen 8: Emulador KiGB en Windows

SMS (SEGA GAME GEAR)



Imagen 9: Consola portátil Sega Game Gear

Competidor directo de la GameBoy, apareció en 1990. También tenía un procesador Z80 a 3.5 MHz y 24 KB de memoria, pero donde intentó destacar fue en el apartado gráfico. Tenía una pantalla de 4.096 colores y 3.2 pulgadas. Sin embargo, lo que fue su innovación también fue su perdición, ya que con esa pantalla, el consumo de energía era enorme y con el triple de baterías que la GameBoy, ésta tenía una autonomía 5 veces inferior. Parte de su fracaso también se debió a los pocos acuerdos comerciales con los desarrolladores de juegos con respecto a su competidora Nintendo, provocando un catálogo escaso. Prácticamente todos los emuladores que dan soporte a la consola Master System, también son compatibles con la Game Gear, estando así soportada en las consolas portátiles de última generación como GP2X, Nintendo DS, o PSP (PlayStation Portable).



Imagen 10: Emulador MEKA con un juego de Sega Game Gear

ATARI LYNX



Imagen 11: Consola portátil Atari Lynx

También intentó competir con la GameBoy de Nintendo. Apareció en el año 1989 y fue la primera consola portátil de color que existió. Su procesador principal era un MOS 6502, pero contaba además con un procesador gráfico dedicado llamado “Suzy” a 16 MHz y un coprocesador matemático que podía trabajar en paralelo. Disponía de hasta 64 KB de memoria principal. Su pantalla tenía 4096 colores y 3.5 pulgadas. A pesar de su superioridad técnica con el resto de consolas portátiles de su generación, su éxito fue incluso menor a la Game Gear, de nuevo por los mismos problemas, poca autonomía, menos acuerdos incluso con los desarrolladores de software, un gran tamaño y un precio muy elevado. Debido a ello, existen muy pocos emuladores de ella, y sólo uno que merezca la pena mencionar.

- Handy [8]: Empezó en el año 1997 y tiene un nivel de compatibilidad muy bueno. Está disponible para Windows, Linux y Mac, pero no tiene disponible su código fuente.

2.4. TÉCNICAS DE EMULACIÓN

A la hora de abordar la emulación de un procesador, se puede hacer desde varias perspectivas, moviéndose en un eje entre la velocidad y la simplicidad.

2.4.1. INTERPRETACIÓN

Es la forma más sencilla e intuitiva de emular el ensamblador de un procesador. Consiste en leer y traducir el código original desde la memoria byte a byte, decodificarlo y realizar las acciones necesarias para emularlo, como los accesos a memoria o a registros de entrada y salida. Su algoritmo es algo como:

```
while(CPUactiva){  
    leerOpcode();  
    decodificarOpcode();  
    ejecutar();  
}
```

Aparte del código asociado al bucle principal de ejecución, además debe añadirse lo necesario para las funciones de depuración, sincronización e interrupciones. Para ello se basa en el cálculo del número de ciclos de reloj que consume la ejecución de cada instrucción decodificada y se realizan cálculos para averiguar los tiempos del resto de operaciones en base a esa cantidad.

El esquema es muy simple, pero tiene un inconveniente muy importante que es el rendimiento. El procesamiento del bucle principal es un cuello de botella con respecto a la velocidad y requiere una cantidad enorme de tiempo de CPU para emularla, así que es necesario un procesador mucho más rápido que la máquina que se quiere emular.

2.4.2.RECOMPILACIÓN ESTÁTICA

Esta técnica traduce el código nativo del programa de la máquina a emular a código máquina de la máquina sobre la que queremos emular. Con ésto se obtiene un ejecutable para la máquina que podrá ser lanzado sin necesidad de usar programas especiales. Por desgracia, esto no siempre es posible ya que existen muchos programas con código generado o modificado dinámicamente.

Se suele utilizar en combinación con otras técnicas para las partes conflictivas, pero casi nunca de forma exclusiva.

2.4.3.RECOMPILACIÓN DINÁMICA

La recompilación dinámica o “dynarec” en inglés, es una técnica en la que se recompilan partes del código original en tiempo real según se va ejecutando. Para evitar el problema del código generado dinámicamente, se recompilan únicamente el código entre saltos, es decir, cada vez que se produce un salto en la ejecución a una subrutina o función, se recompila sólo esa parte para posteriores llamadas y se almacena en una caché.

La principal ventaja de este sistema es que al tratar como unidad mínima una función, a la vez que se genera el nuevo código máquina, se puede optimizar para aprovechar su máximo rendimiento. Por ejemplo, si una función se encarga de copiar una cadena de texto mediante un bucle carácter a carácter, pero la máquina del emulador maneja más eficientemente la copia de cadenas enteras, entonces se puede reemplazar esa parte de código con una considerable mejora de velocidad.

Por desgracia esta técnica está limitada a la arquitectura a la que se recompila, ya que no se puede hacer un recompilador dinámico que aproveche el mejor rendimiento para cada una de ellas. Para algunas máquinas complejas y de reciente generación, es indispensable utilizar esta técnica para conseguir algo funcional.

2.5. GAMEBOY



Imagen 12: Consola portátil NintendoGameBoy

2.5.1.HISTORIA

La GameBoy es una videoconsola portátil desarrollada por Nintendo, que sacó al mercado japonés el 21 de abril de 1989 a un precio aproximado de 70 euros. Al mercado estadounidense llegó unos meses después en agosto y en Europa no se pudo ver hasta bien entrado el año 1990. Fue la primera consola portátil de éxito, y fue la predecesora de toda una gama de consolas GameBoy. Coincidiendo con el 15º aniversario de su salida al mercado en 2004, Nintendo anunció que hasta esa fecha se habían vendido casi 70 millones de unidades en todo el mundo, de las cuales 20 son sólo en el mercado japonés. Sus sucesoras la GameBoy Color y la GameBoy Advance han vendido 50 millones y 75 millones de unidades respectivamente en todo el mundo hasta esa misma fecha.

Su nombre es un juego de palabras basado en el “WalkMan” de Sony, ya que se trataba un dispositivo multimedia con la misma filosofía: poder utilizarlo en cualquier momento y lugar.

Inicialmente se distribuyó en un pack junto con el mundialmente conocido Tetris, tras un acuerdo con el autor original Alexey Pajitnov, para intentar captar la atención del público. La maniobra comercial fue un rotundo éxito de ventas.

Desde el principio, la compañía tenía en mente el objetivo de crear una consola pequeña, ligera, barata y resistente, y que también tuviera un catálogo de juegos muy variado. Aunque ha sido criticado en muchas ocasiones por su pantalla monocromo frente a las consolas de la competencia como la Sega Game Gear, el presidente de Nintendo lo defendió siempre con la excusa del coste y rendimiento, ya que una pantalla a color encarecería mucho el precio final de la consola y consumiría mucha más energía con más pilas y menos duración.

2.5.2.HARDWARE

La consola GameBoy viene empaquetada en una carcasa de plástico de dimensiones 90 milímetros de ancho, 148 milímetros de alto y 32 milímetros de fondo, siendo la consola portátil más pequeña de su generación.

El corazón de la GameBoy es un procesador basado en el Z80 de la compañía ZiLOG pero ligeramente modificado. También dispone de 8 kilobytes de memoria interna SRAM (Static random access memory) y otros 8 kilobytes para la memoria gráfica llamada VRAM.



Imagen 13: Lateral derecho de la consola portátil GameBoy

La pantalla es una LCD de 160x144 píxeles y 2.6 pulgadas de tamaño (66 milímetros) y cuenta con niveles de gris desde el verde al negro. El contraste de la pantalla es modificable a través de un regulador en lateral izquierdo de la consola. La frecuencia de su refresco horizontal es de 9198 KHz y su refresco vertical de 59.73 Hz.

Tiene 4 botones principales aparte de la cruceta direccional que permite hasta 8 direcciones diferentes con diagonales y un conmutador de encendido en la parte superior de la carcasa. Los 4 botones se llaman “A”, “B”, “SELECT” y “START”, y su función varía según el juego.

Aunque la consola viene con un único altavoz integrado, es capaz de generar sonido estéreo mediante 4 canales independientes que se pueden escuchar a través del conector jack de 3.5 milímetros para altavoces externos.

En el lado derecho se encuentra el regulador de volumen y también el conector para la conexión serie hasta 16 consolas, mediante varios cables especiales.

Los cartuchos de los juegos pueden ser de 256 kilobits, 512 kilobits, 1 Mbit, 4 Mbit y 8 Mbit. La ranura para introducirlos se encuentra en la parte superior de la consola.

Con todas estas prestaciones, sólo necesita 4 pilas de tipo AA (6 voltios) que con un consumo medio de 0.7 watios pueden llegar a durar un máximo de 35 horas en modo de espera, pero que también es bastante alto mientras se está usando si los juegos usados hacen uso de las funciones especiales de bajo consumo. Cuenta adicionalmente con un conector en el lado izquierdo para alimentación externa usando un transformador que proporcione 6 voltios DC a 250 miliamperios.

2.5.3.PROCESADOR

El procesador de la GameBoy es único, y fue creado expresamente para esta consola, por lo que es imposible encontrarlo en cualquier otra pieza de hardware. Aún así guarda muchas similitudes con otros procesadores de su época, en concreto con el 8080 de Intel y el Z80 de ZiLOG. De hecho, es más parecido al Z80 con algunas modificaciones en su juego de instrucciones.



Imagen 14: Procesador de la GameBoy

2.5.3.1. INSTRUCCIONES

Con respecto al Z80:

Se han añadido las instrucciones

Opcode	Nemónico	Descripción
E8	ADD SP, nn	Suma al registro SP un valor literal de 16 bits
22	LDI (HL), A	Guarda en la dirección de memoria apuntada por HL el valor de A e incrementa el valor del registro HL
32	LDD (HL), A	Guarda en la dirección de memoria apuntada por HL el valor de A y decrementa el valor del registro HL
2A	LDI A, (HL)	Guarda en A el valor de la dirección de memoria apuntada por HL e incrementa el valor del registro HL
3A	LDD A, (HL)	Guarda en A el valor de la dirección de memoria apuntada por HL y decrementa el valor del registro HL
F0	LD A, (0xFF00 + n)	Guarda en A el valor de la dirección de memoria 0xFF00 + un valor literal
F2	LD A, (0xFF00 + C)	Guarda en A el valor de la dirección de memoria 0xFF00 + valor del registro C
E0	LD (0xFF00 + n), A	Guarda en la dirección de memoria 0xFF00 + valor literal el contenido de A
E2	LD (0xFF00 + C), A	Guarda en la dirección de memoria 0xFF00 + valor del registro C, el contenido de A
08	LD (nn), SP	Guarda en una dirección de memoria indicada por un valor literal el contenido del registro SP
F8	LD HL, (SP + nn)	Guarda en el registro HL el contenido de la dirección de memoria apuntada por el registro SP + valor literal
10	STOP	Detiene la ejecución del procesador hasta que ocurra una interrupción. Entra en modo de bajo consumo.
CB 3x	SWAP r	Intercambia la parte alta y la parte del

		contenido de un registro.
CB 36	SWAP (HL)	Intercambia la parte alta y la parte baja de una dirección de memoria apuntada por el registro HL

Se han eliminado las instrucciones:

- Cualquier instrucción que use los registros de índice IX o IY
- Todas las instrucciones de entrada / salida
- Todas las instrucciones de intercambio (Exchange) para los registros “shadow” como F’, BC’, DE’, HL’ y demás.
- Todas las instrucciones extendidas por el opcode ED
- Todos los saltos, llamadas o retornos condicionales en base a los flags de paridad, overflow y signo.

Se han modificado las instrucciones:

Opcode anterior	Nemónico	Opcode nuevo
3A	LD A, (nnnn)	FA
32	LD (nnnn), A	EA
ED 4D	RETI	D9

Como se han eliminado todas las instrucciones relacionadas con los flags de paridad, overflow y signo, también han desaparecido esos flags. Del mismo modo, al eliminar todas las instrucciones de IN y OUT, las comunicaciones ya no se realizan en un espacio de direcciones aparte de entrada y salida, sino en registros especiales en el espacio de direcciones de memoria, normalmente entre 0xFF00 y 0xFFFF.

2.5.3.2. INTERRUPTACIONES

También el sistema de interrupciones ha sufrido modificaciones con respecto al Z80 original, dejando únicamente 5 interrupciones especiales claramente definidas:

- **V-Blank:** Ocurre unas 59.7 veces por segundo al comienzo del periodo de refresco vertical. Durante este tiempo el hardware de video no usa la memoria de video y puede ser accedida libremente. Dura aproximadamente 1.1 milisegundos.
- **LCDC Status:** Hay varios motivos para esta interrupción, controlados por el registro especial STAT (0xFF40). La más común es para indicar que una determinada línea de pantalla se va a dibujar. Se suele usar para modificar los registros de scroll horizontal y vertical dinámicamente y conseguir diversos efectos gráficos.
- **Timer Overflow:** Se dispara cuando el registro TIMA (0xFF05) desborda reiniciándose a 0.
- **Serial Transfer Completion:** Ocurre cuando una transferencia serie a través del puerto de comunicaciones ha terminado.
- **High-to-Low:** Se dispara en la transición de la pulsación de cualquiera de las teclas. Debido al efecto eléctrico de “rebote” en cualquier evento de pulsación, el software debe ser el encargado de controlar esa pequeña oscilación y contemplar que para una única pulsación, la interrupción puede ocurrir varias veces. En la práctica, muchos juegos directamente ignoran esta interrupción y realizan “polling” del estado de las teclas.

Cuando se produce una de estas interrupciones, se activa el correspondiente flag en el registro IF (0xFF0F), y si los flags IME (Interrupt Master Enable) y el correspondiente bit del flag IE (Interrupt Enable) están activos, se desactiva el IME, se inserta el contador de programa en la pila y se salta a la dirección de comienzo de la rutina de tratamiento de la interrupción. El estado del flag IME se puede controlar manualmente con las instrucciones EI (Enable Interrupt) y DI (Disable Interrupt). El regreso de la rutina de tratamiento se puede hacer mediante la instrucción RET o RETI,

pero esta última además vuelve a activar el IME. Si hay varias interrupciones pendientes, se invocan las rutinas en orden según su prioridad.

2.5.3.3. REGISTROS

Cuenta con 6 registros de 16 bits que pueden ser accedidos individualmente o únicamente a su parte alta o baja de 8 bits directamente por hardware.

- AF: Compuesto por el acumulador A y los flags de estado F
- BC: Compuesto por los registros B y C de 8 bits
- DE: Compuesto por los registros D y E de 8 bits
- HL: Compuesto por los registros H y L de 8 bits
- SP: Puntero de pila
- PC: Contador de programa

El juego de instrucciones contiene algunas operaciones sobre los de 8 bits, mientras que otras lo hacen sobre los de 16 bits, pero se tratan de los mismos registros. Por ejemplo, al modificar el valor del registro B también lo hará la parte alta del registro BC y viceversa.

Los flags de operación se ven reflejados en la parte baja del registro AF, usado normalmente para guardar fácilmente el contexto, ya que no se puede acceder directamente a ellos como un registro separado de 8 bits. Los flags disponibles son 4:

- Z (Zero): Se activa cuando el resultado de la operación ha sido 0
- C (Carry): Se activa cuando el resultado de una suma desborda un registro o es negativo cuando se produce una resta. También se activa con las operaciones de rotación si el bit desplazado es 1.
- N (Add/Sub): Indica si la operación ha sido una suma o una resta. Sólo se usa este flag para la conversión a BCD por la instrucción DAA.
- H (Half-Carry): Indica si en el resultado de la operación ha habido acarreo desde los 4 bits menos significativos del registro de 8 bits. Solo usado por la instrucción DAA para la conversión a BCD.

2.5.3.4. MEMORIA

El espacio de la memoria es de 65536 direcciones, es decir, desde 0 hasta 0xFFFF, pero no todo es memoria física, ya que en ciertos rangos se encuentra proyectado el cartucho del juego y los registros de entrada / salida. La distribución de la memoria es la siguiente:

Rango de direcciones	Descripción
0000-3FFF	ROM Bank #0 del cartucho (16 kB)
4000-7FFF	ROM Bank #1-#n intercambiable del cartucho (16 kB)
8000-9FFF	VRAM o Video RAM (8 kB)
A000-BFFF	RAM externa del cartucho (8 kB)
C000-DFFF	RAM interna o WRAM (Work RAM) (8 kB)
E000-FDFF	Eco de la WRAM (no usada)
FE00-FE9F	OAM (Object Attribute Memory) de los sprites
FEA0-FEFF	No disponible
FF00-FF7F	Puertos de entrada / salida
FF80-FFFE	HRAM (High RAM)
FFFF	Registro IE (Interrupt Enable)

La mayoría de las direcciones de los puertos de entrada / salida contienen un valor por defecto por hardware al arrancar.

2.5.3.5. VIDEO

El buffer de pantalla de la GameBoy es de 256x256 píxeles o de 32x32 “tiles”. Un “tile”, o también llamado baldosa o azulejo, no es más que una pequeña imagen de 8x8 o 8x16 píxeles que forma el elemento gráfico más pequeño que se puede manipular en la consola. Cada píxel ocupa 2 bits que proporciona los 4 niveles de colores disponibles, por lo que un tile de 8x8 ocupa 16 bytes en memoria. Cada tile se encuentra almacenado línea a línea de 2 bytes cada una, por lo que el color de un píxel no viene indicado por dos bits consecutivos. Por ejemplo, supongamos un tile así donde el color viene indicado por un número decimal del 0 al 3:

```
.33333..
22...22.
11...11.
2222222.
33...33.
22...22.
11...11.
.....
```

Su codificación sería:

```
.33333.. -> 01111100 -> 0x7C
          01111100 -> 0x7C
22...22. -> 00000000 -> 0x00
          11000110 -> 0xC6
11...11. -> 11000110 -> 0xC6
          00000000 -> 0x00
2222222. -> 00000000 -> 0x00
          11111110 -> 0xFE
33...33. -> 11000110 -> 0xC6
          11000110 -> 0xC6
22...22. -> 00000000 -> 0x00
          11000110 -> 0xC6
11...11. -> 11000110 -> 0xC6
          00000000 -> 0x00
..... -> 00000000 -> 0x00
          00000000 -> 0x00
```

Como sólo se pueden mostrar en pantalla 160x144 píxeles, un par de registros se encargan de especificar el desplazamiento horizontal y vertical sobre la imagen global que debe mostrarse.

La composición final que se ve en la pantalla es una mezcla de 3 procesados gráficos: un fondo de base, una ventana encima del fondo, y una serie de sprites que pueden ir delante o detrás según su prioridad.

El fondo se encuentra definido en un área de la VRAM. Cada tile a mostrar viene especificado por un identificador de un byte en una matriz de 32 filas y 32 columnas. Los datos de los propios tiles se pueden encontrar en dos zonas distintas de la memoria. En una de ellas los tiles están numerados como un byte sin signo de 0 a 255, mientras que en la otra tienen signo en un rango de -128 a 127. El banco de tiles a usar viene especificado en uno de los registros de control.

Del mismo modo, también hay dos matrices de fondos, aunque sólo se puede mostrar uno de ellos. También se selecciona el mapa de tiles a usar mediante un registro de control.

Por encima del fondo, también se encuentra una “ventana”, que no es más que otro fondo superpuesto pero que puede ser móvil. Su posición se controla mediante dos registros de control especificando su posición en los ejes X e Y. Tanto el fondo como esta “ventana” comparten los datos de los tiles y los mapas, pero se pueden activar o desactivar por separado. Combinando estos registros de control con la interrupción LCDC se pueden conseguir efectos gráficos complejos y muy vistosos.

El límite técnico es de 40 sprites como máximo, ya sean de 8x8 o 8x16 píxeles, y por una limitación de hardware, solo se pueden mostrar hasta 10 en una sola línea. Los sprites son almacenados y tratados igual que los tiles, pero se encuentran almacenados en otra región de memoria, y la lista de sprites está en la zona OAM. Cuando dos sprites en diferente posición horizontal se solapan, aquel con menor coordenada X será el de mayor prioridad, pero si se encuentran en la misma posición X, tendrá más prioridad el que se encuentre antes en la lista de sprites.



Imagen 15: Sprite de 8x8 píxeles perteneciente al juego Super Mario Land



Imagen 16: Composición de texto mediante sprites de 8x8 píxeles

Debido al límite de 10 sprites por línea, en caso de haber más de 10, aquellos con menor prioridad según las reglas escritas anteriormente, no serán mostrados, así que se aconseja normalmente moverlos a una zona no visible de la pantalla para que no afecten.

La tabla OAM de sprites guarda la siguiente información por cada sprite:

- Posición X en la pantalla
- Posición Y de la pantalla
- Número de sprite (0-255)
- Prioridad: Si se encuentra desactivado, el sprite se mostrará por encima del fondo y de la ventana, pero si está activado, sólo se verá sobre el color 0 del fondo o pantalla.
- Inversión vertical: Indica si el sprite está invertido verticalmente
- Inversión horizontal: Indica si el sprite está invertido horizontalmente
- Paleta: Los colores de un sprite pueden proceder de dos paletas de colores distintas distintas, mientras que el fondo y ventana comparten la misma. Cada paleta sólo ocupa un byte y codifica la intensidad con 2 bits por cada identificador de color.

El estado del controlador LCD, y por tanto del proceso gráfico es cíclico y cambia entre 4 estados posibles:

- Modo 0: Durante el refresco horizontal (H-Blank) la CPU puede acceder a la VRAM.
- Modo 1: Durante el refresco vertical (V-Blank) la CPU puede acceder a la VRAM.
- Modo 2: Cuando se está usando la tabla OAM la CPU no puede acceder a ella.
- Modo 3: Cuando se está usando tanto la OAM como la VRAM, la CPU no puede acceder a ninguna de las dos.

La secuencia en el tiempo es típicamente así:

Modo 0	000__000__000__000__000__000__000__
Modo 1	1111111111111111__
Modo 2	__2__2__2__2__2__2__2__
Modo 3	__33__33__33__33__33__33__3

El modo 0 dura unos 205 ciclos (48 microsegundos), el modo 2 unos 80 ciclos (19 microsegundos) y el modo 3 casi 173 ciclos (41 microsegundos). Un bucle entre estos 3 modos tarda unos 456 ciclos (109 microsegundos), sólo interrumpido por el modo 1 del refresco vertical cada 16.6 milisegundos y dura 4560 ciclos de procesador. Una secuencia completa por los 4 modos sólo ocurre una vez cada 70224 ciclos.

Por último cabe destacar que el procesador cuenta con un mecanismo de transferencia de memoria por DMA, especialmente diseñado para el apartado gráfico. La transferencia comienza al escribir la dirección de inicio múltiplo de 0x100 en un registro especial (0xFF46), y copia una sección de 140 bytes desde cualquier parte de la ROM o RAM a la tabla OAM. El proceso tarda unos 160 microsegundos y mientras tanto, la CPU no puede acceder a ninguna parte de la memoria excepto a la zona de memoria alta o HRAM situada entre las direcciones 0xFF80 y 0xFFFE. Es por ello que normalmente la rutina que invoca la transferencia y espera a que termine debe encontrarse en esa zona de memoria.

2.5.4.ACESORIOS

A lo largo de la existencia de la GameBoy, se fabricaron varios periféricos para dotar de nuevas funcionalidades al hardware original.

SUPER GAME BOY

Aunque se trate de un accesorio de SNES (Super Nintendo Entertainment System) de la siguiente generación, está estrechamente ligado con la GameBoy. Se trataba de un accesorio que permitía jugar a los juegos de la consola portátil en la SNES. Realmente se trataba de una consola GameBoy camuflada, que sólo se comunicaba con la SNES para la entrada/salida y el pintado gráfico.

Como la resolución de la GameBoy es muy inferior a la ofrecida por la SNES, este accesorio añadía unos bordes predefinidos y animados hasta rellenar la pantalla. Sin

embargo, muchos fueron los juegos que incorporaban sus propios bordes de colores que solo se podían ver usando este accesorio y no en la propia GameBoy. Algunos incluso tenían funciones extras en este modo, como sonido de mayor calidad, soporte para dos jugadores e incluso alguno con calidad de 16 bits que permitía una mayor resolución sin usar los llamados “bordes”.

Debido a la simplicidad del sistema, muchos son los emuladores que emulan el borde de aquellos juegos que lo soportan.

Salió en el año 1994 por unos 60 dólares (la SNES apareció por primera vez en 1990 en Japón, 1991 en Estados Unidos y 1992 en Europa).



Imagen 17: Super GameBoy

GAMEBOY BATTERY PACK

Energía adicional para aumentar la autonomía de la consola. Tenía 2 baterías de níquel-cadmio que permitían poder jugar durante unas 4 o 5 horas más. Se podían cargar directamente y duraban unas 1000 recargas antes de que perdieran efectividad. Su principal problema era el peso y el tamaño de los conectores que sobresalían.

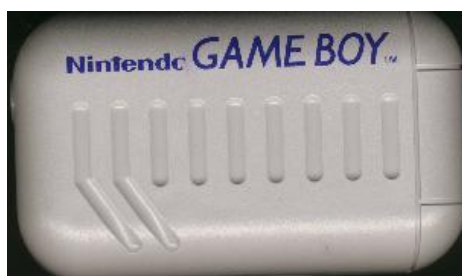


Imagen 18: GameBoy Battery Pack

GAMEBOY CAMERA

Salió en el año 1998 y permitía tomar fotos a una resolución de 128x123 píxeles en blanco y negro. Su principal uso era para imprimir directamente a la impresora, por lo que inicialmente se vendían juntos. Ya no se fabrica y ningún emulador existente lo soporta, por lo que no es capaz de emular aquellas partes de los juegos que hicieran uso de él. Sin embargo, existe un adaptador capaz de conectarla a un PC convencional y mediante un programa especial capturar fotografías y almacenarlas en formato digital.



Imagen 19: GameBoy Camera

GAMEBOY PRINTER

Salió al mercado a la par que el accesorio de la cámara. Se trataba de una impresora térmica que sin hacer uso de ninguna tinta y con un papel especial era capaz de imprimir imágenes y texto en blanco y negro. Aparte del uso directo con la cámara, varios fueron los juegos que la usaban para imprimir puntuaciones o imágenes de logros conseguidos. Usaba 6 pilas de tipo AA y existen adaptadores para poder conectarla al puerto paralelo de un PC convencional.

Algunos emuladores como el KiGB simulan su funcionamiento y permiten exportar la imagen que se quiere imprimir a un fichero de imagen digital.



Imagen 20: GameBoy Printer

GAMEBOY LINK CABLE

El GameBoy Link Cable fue el accesorio más vendido y más conocido de todos. Su éxito se debió principalmente por la popularidad de los juegos de la saga Pokemon, que permitía interactuar entre varias versiones del juego, y sobre todo porque había algunos objetivos que solo se podían alcanzar usando dicho cable.

La conexión es simplemente una transferencia serie usando un par de registros de entrada salida, uno para el control de la conexión y otro para recibir o enviar la información. Sólo se podía transmitir un byte cada vez. Debido a la simplicidad del protocolo, ha sido ampliamente implementado por muchos emuladores, con la intención

de permitir conexiones online por TCP/IP. Por desgracia, la velocidad de la transferencia es tan alta, que únicamente se consigue una velocidad aceptable mediante conectividad en el mismo equipo, o mediante algunas optimizaciones de buffer y caché, en red local.

Algunos emuladores, especialmente los primeros que implementaron esta característica, y debido al principal problema de la velocidad de transferencia, hicieron que la transferencia sólo fuera posible en la misma máquina, pero en lugar de usar conectividad IP, directamente transfería la información entre zonas de memoria compartida de los procesos del emulador. Por lo tanto era necesario iniciar una instancia del proceso del emulador por cada consola.



Imagen 21: GameBoy Link Cable

2.5.5.JUEGOS

Los juegos van almacenados en cartuchos estéticamente iguales, pero realmente existen infinidad de tipos y algunos pueden contener algo de memoria estática. El tipo de cartucho viene identificado por el valor en la posición 0x147 del primer banco del juego, común a todos ellos y donde se almacena la metainformación del cartucho. Para todos ellos, el tamaño concreto de la ROM y la RAM viene indicado por dos valores de la cabecera, la 0x148 y 0x149 respectivamente. El tamaño de la ROM oscila entre 256 kilobits (32 kilobytes o 2 bancos) hasta 12 megabits (1.5 megabytes o 96 bancos), siendo cada banco de 16 kilobytes. El tamaño de la RAM oscila entre 0 hasta 1 megabit (128 kilobytes o 16 bancos) siendo cada banco de 8 kilobytes.

Existen otros campos de interés en la cabecera del juego, como la comprobación de integridad (el hardware original la ignoraba), el título interno del juego en 16

caracteres ASCII, indicadores de si esta diseñado para GameBoy original, Color o Super GameBoy, y el punto de entrada de ejecución de 4 bytes de longitud.



Imagen 22: Interior de un cartucho de GameBoy

Se distinguen 5 tipos de cartucho principales:

ROM ONLY

Como su propio nombre indica solo tiene ROM de 32 kilobytes (256 kilobits) de tamaño fijo y nada de RAM interna.

MBC1 (MEMORY BANK CONTROLLER 1)

Tiene dos modos de funcionamiento en combinaciones de ROM/RAM: 16 megabits de ROM y 8 kilobits de RAM o 4 megabits de ROM y 32 kilobits de RAM, siendo el primero el modo por defecto. Se puede cambiar el modo al escribir un valor en una región de memoria de sólo lectura.

Según el modo de funcionamiento, la forma de seleccionar un banco de ROM o de RAM cambia.

MBC2 (MEMORY BANK CONTROLLER 2)

Funciona de manera similar al tipo MBC1 pero puede trabajar con ROM de hasta 2 megabits. Sólo tiene 256 bytes (2048 bits) de memoria RAM integrada, que se

puede activar o desactivar escribiendo un valor 1 o 0 en una determinada región de memoria.

Como puede contener hasta 15 bancos de ROM pero solo se puede proyectar uno cada vez (el banco 0 siempre es fijo), se selecciona el que se quiere usar escribiendo el número de banco en una determinada región de memoria.

MBC3 (MEMORY BANK CONTROLLER 3)

Funciona de manera similar al tipo MBC1, excepto que puede seleccionar el número de banco ROM a usar de hasta 2 megabits con una sola selección, no escribiendo primero la parte baja y luego la alta.

También dispone de un RTC (Real Time Clock) que solo se encuentra en este tipo de cartucho. Los juegos pueden hacer uso de él de manera opcional. Toda la saga Pokemon lo utiliza para conocer la hora real de juego (se establece la hora al crear una partida) y lo usa para los eventos de día y noche.

MBC5 (MEMORY BANK CONTROLLER 5)

Este tipo de cartucho fue el primero con el que se aseguraba que funcionaba en la nueva GameBoy Color, aunque luego en la práctica todos los tipos de cartuchos eran compatibles con ella.

Es similar al MBC3, pero sin RTC y pudiendo direccionar hasta 64 megabits de ROM y 1 megabit de RAM. Para seleccionar un banco de ROM es necesario seleccionar primero la parte alta y luego la baja del número de banco, mientras que el banco de RAM se selecciona con una única petición. Éste también es el primer tipo de cartucho que permite seleccionar el banco 0 de ROM a pesar de estar implícito y siempre presente.

Hubo varias variaciones del MBC5, entre las que estaban los cartuchos con vibrador (Rumble Carts) que solo podían tener 256 kilobits de RAM, ya que el resto se usa como región para controlar el estado del motor. También hubo unos pocos cartuchos de este tipo que contenían un controlador de infrarrojos (HuC1).

3. DESCRIPCIÓN DEL SISTEMA

En el apartado actual se indicarán las capacidades que deberá tener el sistema a construir. Se realizará una sencilla descomposición en subsistemas de funcionamiento, indicada por cada uno sus características, funcionalidades y restricciones.

3.1. VISIÓN GENERAL DEL SISTEMA

El emulador de la consola portátil GameBoy es una aplicación cuyo objetivo es conseguir una reproducción fiel del funcionamiento del hardware original. Además de la emulación del procesador y de los periféricos principales, la aplicación también permitirá realizar tareas de depuración, así como un seguimiento de la ejecución del software emulador y la posibilidad de modificar ciertos valores del hardware en tiempo de ejecución.

El sistema estará dividido en las siguientes partes:

- Emulador de la arquitectura donde se encuentra el procesador principal
- Emulador del hardware externo relacionado directamente con la arquitectura
- Interfaz gráfica de usuario

Las siguientes secciones describirán en detalle cada una de estas partes.

3.2. EMULADOR DE LA ARQUITECTURA

Como se ha comentado previamente, el emulador depende del subsistema encargado de emular el hardware principal donde se encuentra el procesador y el corazón de la aplicación. La arquitectura empleada para ello seguirá el modelo de Von Neumann, por lo que estará compuesta por un procesador, un sistema de memoria principal y un conjunto de sistemas de entrada/salida.

3.2.1.PROCESADOR

A continuación se detallan las capacidades del procesador emulado por la aplicación, el cual se pretende que sea lo más próximo al procesador real de la consola GameBoy.

3.2.1.1.REGISTROS

Hace uso de 5 registros de 8 bits y 3 registros de 16 bits:

- **A:** Registro acumulador, donde normalmente se escribe el resultado de la mayoría de las operaciones
- **B:** Registro B de propósito general de 8 bits
- **C:** Registro C de propósito general de 8 bits
- **D:** Registro D de propósito general de 8 bits
- **E:** Registro E de propósito general de 8 bits
- **HL:** Registro HL de propósito general de 16 bits
- **PC:** Registro que contiene la dirección de memoria de la instrucción en ejecución
- **SP:** Registro que contiene el punto a la cima de la pila

A pesar de esta distinción, también es posible acceder a estos registros en otros modos de 8 o 16 bits, de esta forma existen los registros virtuales:

- **H:** Registro con los 8 bits más significativos del registro HL
- **L:** Registro con los 8 bits menos significativos del registro HL
- **BC:** Registro compuesto por los registros B y C
- **DE:** Registro compuesto por los registros D y E
- **AF:** Registro compuesto por el acumulador y estado global de los flags de ejecución.

Debido al uso de los flags como un registro más en el registro virtual AF, también es posible acceder a ellos de manera independiente a través del registro F. Su representación en un registro de 8 bits es la siguiente:

Flag	Máscara
Z	1000 0000
N	0100 0000
H	0010 0000
C	0001 0000

3.2.2.INSTRUCCIONES

Una GameBoy original cuenta con exactamente 500 instrucciones de ensamblador de longitud variable: 1, 2 o hasta 3 bytes si se trata de una instrucción extendida.

Las instrucciones básicas pueden tomar ninguno, uno o dos parámetros para sus operaciones, por ejemplo una instrucción de carga de un número de 2 bytes en un registro de 16 bits.

El límite físico de 255 instrucciones (número máximo de combinaciones con 8 bits) se rompe mediante el uso de instrucciones extendidas, esto son instrucciones que comparten un prefijo. En este caso concreto, el opcode 0xCB indica que el siguiente byte determina de qué instrucción se trata, permitiendo así que por cada prefijo pueda haber 255 instrucciones adicionales, pero a costa de rendimiento y velocidad, ya que necesita un ciclo adicional de lectura para que el procesador pueda decodificar la instrucción que debe ejecutar.

3.2.2.1.JUEGO DE INSTRUCCIONES

Muchas instrucciones tienen implícito en el propio código de la instrucción información sobre literales o registros, es por ello que una operación de suma entre dos registros tiene distinto identificador u opcode según los registros afectados aunque se trate de la misma operación.

En el proceso de emulación se puede tomar ventaja de estas similitudes de operaciones, y reutilizar las mismas funciones para todo un subconjunto de instrucciones. Es decir, se puede implementar la función de suma de tal manera que pueda ser utilizada por todas las operaciones de suma existentes.

De esta forma, y aunque existan 500 instrucciones distintas, el emulador puede implementarlas todas con únicamente 100 funciones que implementen de forma genérica las operaciones del procesador.

El juego de instrucciones se encuentra en detalle en el anexo A de este documento.

3.2.3.MEMORIA PRINCIPAL

El sistema contará con un conjunto de memoria principal representando en un array unidimensional de bytes. El tamaño total será el máximo del espacio direccionable, es decir, 65536 posiciones, aunque no todo se use como memoria efectiva.

La sección desde 0 hasta 0x8000 será delegada al subsistema de cartuchos, ya que realmente se trata de una proyección de éstos sobre el espacio de direcciones, al igual que el rango de 0xA000 a 0xC000 que contiene la zona de RAM integrada también en los cartuchos.

También se tratará de forma especial la zona del espacio de memoria destinada a los puertos de entrada / salida entre las direcciones 0xFF00 y 0xFF4C, de manera que cuando se escriba o se lea un valor de ese intervalo, se realicen operaciones especiales específicas para cada puerto. Por ejemplo, en una petición de transferencia DMA que se escribe la dirección de inicio en la dirección 0xFF46 o cuando se solicita el estado de las teclas pulsadas en la dirección 0xFF00.

El resto de las zonas del espacio de direccionamiento, como la VRAM, WRAM, HRAM o la tabla OAM se acceden directamente sobre la memoria física.

3.3. EMULADOR DEL HARDWARE EXTERNO

Para que el emulador pueda usarse de manera útil, debe emular también el hardware “externo” de la consola, es decir, ciertas partes que aunque no estén directamente relacionadas con el procesador, son necesarias para que el emulador tenga sentido, como son los cartuchos donde se encuentran almacenados los juegos y los botones de interacción con el usuario.

3.3.1.CARTUCHOS

El cartucho es la interfaz de comunicación del emulador con el hardware removible. Es el encargado de gestionar el acceso a los datos persistentes y código del juego, y de la memoria no volátil que pueda tener.

Cada cartucho se adapta a las necesidades de almacenamiento y prestaciones del juego que necesita, dentro unos límites y modelos predefinidos. Para que el emulador pueda conocer las necesidades de cada uno de los cartuchos, debe interpretarlas de la cabecera de metainformación que se encuentra en el primer bloque de ROM. El primer bloque o banco de datos es común a todos los cartuchos y a todos los modelos, por lo que en primer lugar sólo se leen esos primeros 16 kB.

De la cabecera se extrae información necesaria como el número de bloques de ROM de 16 kB (posición 0x148), el número de bloques de RAM de 8 kB (posición 0x149) y el nombre interno del programa (16 caracteres desde 0x134 a 0x142). En base al número de bloques se crean los arrays unidimensionales adecuados calculando su tamaño real.

También se procesa el “checksum” o comprobación de integridad del cartucho, que aunque el hardware original lo ignora, siempre es aconsejable para tener en cuenta si posibles problemas en la emulación pueden tener base en errores en los datos del cartucho. El checksum se calcula sumando todos los bytes de la ROM excepto las posiciones 0x14E y 0x14F, y truncando el resultado a 16 bits. Si el resultado es el mismo que el almacenado en los dos valores evitados, la comprobación es satisfactoria.

Una vez creada la base común, el cartucho se especializa en el modelo concreto especificado por la posición 0x147. Existen más de 30 tipos de cartucho distintos entre todas las combinaciones de modelos de memoria, selección de bancos y prestaciones, aunque se pueden agrupar en 6 grupos principales en la implementación genérica del emulador. Son los llamados MBC o Memory Bank Controllers.

3.3.1.1.MEMORY BANK CONTROLLER

Los Memory Bank Controller o MBC son las grandes categorías donde se pueden clasificar los cartuchos de la GameBoy. Están numerados según su orden cronológico de aparición. Dentro de cada uno de ellos existen varios submodelos.

ROM ONLY o MBC0

Solo tiene uno o dos bancos de ROM no intercambiables, por lo que permite ninguna escritura para selección o para RAM. Es el más sencillo de todos.

MBC1

Es de los más complejos, ya que tiene dos modos completamente distintos de direccionamiento. En uno de los modos se puede direccionar mucha más ROM a costa de un solo banco de RAM y por ello la selección de un banco de datos se realiza mediante la escritura de la parte más significativa y la menos significativa del número de banco por separado en dos regiones de la zona del cartucho. En el otro modo puede haber varios bancos de RAM pero a costa de menor tamaño de ROM, pudiendo seleccionarse bancos para ambas memorias con una sola selección en dos regiones del cartucho.

MBC2

Es también muy sencillo, ya que solo permite selección de banco de ROM y la activación o desactivación de su único banco de RAM.

MBC3

Tiene mecanismo de selección de banco de ROM, banco de RAM y para activar o desactivar la RAM por completo. En esta categoría existe uno de los submodelos más conocidos que contiene el RTC o Real Time Clock, que a costa de poder seleccionar menos bancos de RAM, utiliza parte de esos bits para gestionar el reloj interno pudiendo modificar el día, horas, minutos y segundos.

MBC4

Escasamente utilizado. Casi no existe documentación sobre él y muchos emuladores existentes lo han ignorado.

MBC5

Es una combinación del MBC3 y MBC1. Tiene mecanismo de selección de banco de ROM, banco de RAM y para activar y desactivar la RAM, pero además puede direccionar una cantidad enorme de ROM y la selección del banco puede hacerse dividiéndolo en parte baja y parte alta.

Fue de los últimos en aparecer y es el que prácticamente utilizan todos los juegos de la sucesora de la GameBoy en color.

3.3.2. TECLADO

Los 8 botones de la GameBoy sólo tienen dos estados: presionados o no presionados, aunque puede haber varios a la vez en el mismo estado. El estado de los botones se divide en una matriz de 2x4, de modo que sólo se pueden consultar 4 simultáneamente y no los 8 a la vez.

En un grupo se encuentran los 4 botones direccionales: arriba, abajo, izquierda y derecha, y en el otro el resto: A, B, Start y Select.

Para consultar el estado, el software escribe en un registro de entrada / salida seleccionando el grupo de botones a consultar. En el mismo registro se devuelve el resultado, siendo un bit para cada botón, en lógica negativa, 1 desactivado y 0 pulsado.

Debido al efecto de “rebote” del hardware original de la GameBoy, prácticamente ningún juego hace uso de la interrupción de “High-to-Low” de pulsación de tecla, y realiza “polling” o consulta periódica del estado de los botones que le interesan esperando un tiempo a que se estabilice, así que en el desarrollo del emulador, en cuanto se realiza la petición de consulta, se devuelve la respuesta aunque el software aún espere unos cuantos ciclos a leerla e interpretarla.

3.4. INTERFAZ GRÁFICA DE USUARIO

Debido a la naturaleza de la aplicación, una interfaz gráfica cobra un papel tan importante como el resto de componentes del sistema. La interfaz gráfica deberá:

- Gestionar el dibujado de la pantalla física de la consola
- Controlar diversas opciones del proceso de emulación y de visualización
- Interpretar la memoria gráfica o VRAM para realizar el dibujado

A continuación se detallan cada una de estas funcionalidades representado en varios subcomponentes.

3.4.1.GUI

El GUI (Graphic User Interface) o Interfaz Gráfica de Usuario permite al usuario interactuar directamente con la aplicación. Debe contener un área destinada a mostrar el dibujado de la pantalla física y ser lo más sencilla posible. Debe tener mecanismos para:

- Cargar una ROM desde un fichero mostrando una interfaz para seleccionarlo desde su ubicación en el sistema de archivos.
- Iniciar o reiniciar el proceso de emulación
- Pausar o reanudar el proceso de emulación
- Detener el proceso de emulación
- Cambiar el tamaño de la pantalla emulada. Debido a la baja resolución del hardware original y del gran tamaño de las pantallas actuales, puede ser interesante aplicar un factor de multiplicación sobre la imagen original.
- Acceder a una consola de depuración del proceso de emulación.

3.4.1.1.PANTALLA

La pantalla es una representación directa del hardware original de la GameBoy, pero manteniendo la cohesión como un elemento más de la interfaz gráfica.

Su funcionamiento se basa en una imagen de mapa de bits, sobre la que se va dibujando cada uno de los píxeles desde el procesador gráfico. Dependiendo de la escala seleccionada desde la interfaz, un solo píxel puede convertirse en varios. Por este motivo, cuanto mayor sea el factor de multiplicación, mayor será la carga computacional de la emulación.

Sólo bajo demanda, por la interrupción vertical de la consola o por petición externa del sistema de ventanas del sistema operativo, se redibujará sobre la pantalla un frame completo.

El color del píxel a dibujar se escoge desde una paleta interna de 4 colores a partir del identificador de color extraído del procesado gráfico. Los colores escogidos son: blanco, gris claro, gris oscuro y negro, ordenados de mayor a menos claridad.

3.4.2.GRÁFICOS

La gestión de gráficos es la encargada de interpretar correctamente la memoria gráfica (VRAM) adecuadamente y convertirla en algo visible para la pantalla.

Cada frame dibujado se compone de exactamente 3 capas: fondo, ventana y sprites, pero el proceso se divide en líneas según el refresco horizontal del hardware original, ya que jugando con la línea exacta a dibujar y la interrupción LCDC, algunos juegos realizan ciertos efectos gráficos. Aunque la pantalla virtual es de 255x255 píxeles, únicamente se realiza el procesado de las 144 visibles demarcadas por los registros de scroll horizontal y vertical.

A continuación se detalla el procesado de cada una de las capas:

El fondo: Primero se obtienen las direcciones del mapa y la ubicación de los sprites entre las combinaciones posibles, así como el desplazamiento de scroll marcado en los registros especiales. Se obtiene el identificador de tile del mapa correspondiente a cada una de las posiciones horizontales de la línea, y para cada una de ellas se obtiene los datos del tile en esa posición. Se descodifica el color usando la paleta de colores y se procede a dibujar.

La ventana: El proceso es muy similar al fondo, con la única diferencia de que la ubicación de la ventana viene dada por otro par de registros especiales diferentes.

Los sprites: La interpretación de los sprites es ligeramente más complicada, ya que contienen diversos metadatos asociados a cada uno de ellos. Además, a nivel global, los sprites pueden ser de dos tamaños: 8x8 píxeles o 8x16. Se comprueban los sprites existentes en la línea horizontal que se va a dibujar y por cada uno de ellos se obtiene su posición en la pantalla, el identificador de sprite y sus atributos. Los atributos especifican la paleta de colores a utilizar, si el sprite está invertido horizontal o verticalmente y si tiene prioridad sobre la capa de fondo o no. Basándose en el tamaño del sprite, su posición original, y las condiciones de inversión, se calcula la posición en la pantalla y se dibuja.

Cada una de las capas se puede desactivar individualmente, y solo se procede a su dibujo si ha sido activada.

4. DISEÑO ARQUITECTÓNICO

Como se ha comentado en las secciones anteriores, el emulador de GameBoy está dividido en distintos subsistemas según su funcionalidad. En el presente apartado se definirán con mayor precisión los distintos componentes del sistema, analizándose los módulos que componen cada uno de ellos y las relaciones existentes entre los distintos componentes. Además de lo anterior, se analizará el modelo arquitectónico empleado en el diseño de la aplicación, comentando los motivos por los que se ha seleccionado la arquitectura entre las distintas posibilidades.

4.1. ARQUITECTURA DEL SISTEMA

En el diseño de aplicaciones informáticas es común el uso de soluciones genéricas a distintos problemas tipo sujetas a cierto número de restricciones y objetivos. Cuando los patrones anteriores se refieren a un nivel muy alto de la fase de diseño se los denomina patrones arquitectónicos o arquitectura software. Resulta esencial escoger una arquitectura software que se adapte a los requisitos y objetivos de la aplicación, aportando soluciones a los problemas típicos presentes en el diseño de un sistema con las características deseadas, y obteniéndose una solución más elegante, carente de errores y fácil de mantener.

De entre todos los modelos arquitectónicos disponibles, no es obligatorio el uso de uno de ellos, ni tampoco se restringe el uso a uno solo, pudiéndose escoger la incorporación de distintas arquitecturas a un único sistema.

En esta aplicación no se ha utilizado un único modelo arquitectónico y de manera pura, sino que se ha optado por basarse en algunos para crear un híbrido que se ajuste mejor a la funcionalidad deseada.

4.1.1. ARQUITECTURAS

Por una parte se dispone de una arquitectura de máquina virtual que se encargará de interpretar y ejecutar el lenguaje de ensamblador, por lo que es necesario construir un intérprete que comprenda dichas instrucciones.

En esa arquitectura se distinguen dos partes claramente diferenciadas:

Cliente de máquina virtual: El cliente de la máquina virtual debe leer y preprocesar los programas del lenguaje interpretado, así como reconocer la sintaxis, para posteriormente cargarlo y ejecutarlo en la máquina virtual.

Máquina virtual: La máquina virtual es la parte del sistema encargada de ejecutar el código cargado que indique el cliente de máquina virtual. También deberá devolver el resultado de la ejecución si hubiera alguno.

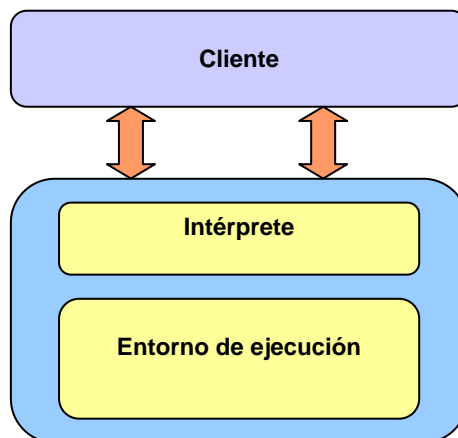


Imagen 23: Arquitectura de máquina virtual

Por otro lado también se dispone de una arquitectura por capas, en la que la aplicación se divide en varios niveles o capas. Cada capa tiene una función específica y únicamente se comunicará con sus capas adyacentes mediante interfaces muy definidas.

El uso de esta arquitectura suele simplificar en gran medida el diseño y ayuda a dividir la aplicación en subsistemas independientes y reutilizables.

Se distinguen las siguientes capas en esta arquitectura:

Interfaz gráfica de usuario: Es la capa más elevada y con la que interactúa el usuario. Es la encargada de transmitir los eventos y peticiones externas al resto de la aplicación pasando por su capa inferior y de reflejar los cambios en el sistema usando los componentes propios de la interfaz

Control de Eventos: Capa de control que recoge, procesa y transforma las peticiones de la interfaz dadas por el usuario a llamadas propias de la aplicación que

realmente contiene la funcionalidad. Sirve de acoplamiento entre la interfaz gráfica y la lógica de negocio.

Lógica de negocio: Está formada por el núcleo de la aplicación y realiza todas las operaciones solicitadas por el usuario y el procesamiento interno. Debido a su tamaño y complejidad, se suele dividir en subsistemas más pequeños y modulares.

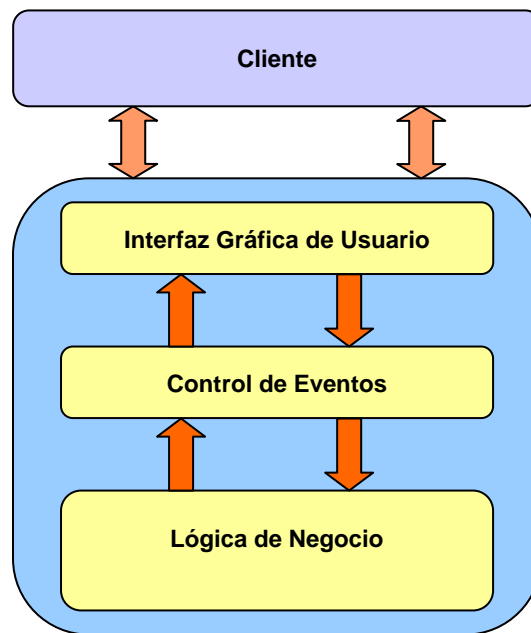


Imagen 24: Arquitectura por Capas

4.1.2.MODELO ARQUITECTÓNICO

Basándose en las arquitecturas expuestas en el punto anterior, es necesario unificarlas ahora para conseguir un diseño homogéneo y consistente bajo un único modelo arquitectónico.

La arquitectura final será similar a la arquitectura por capas, con ligeras modificaciones, y con la particularidad de que la capa de la lógica de negocio contendrá, entre otras cosas, una versión de la arquitectura de máquina virtual dentro del subcomponente del procesador emulado.

El esquema sería:

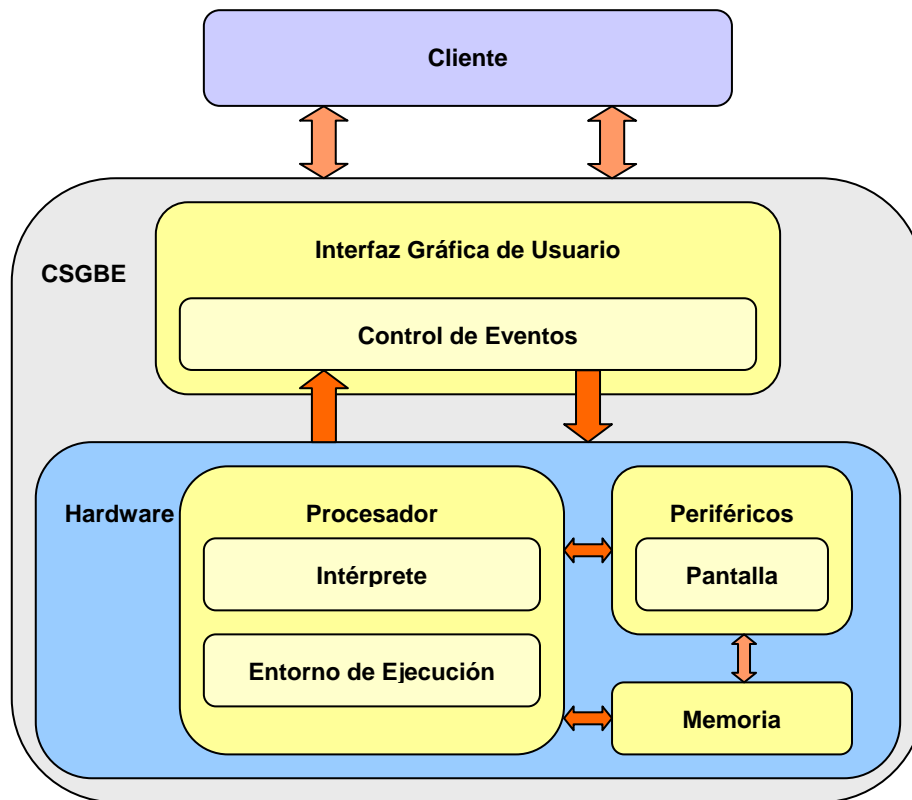


Imagen 25: Modelo Arquitectónico del Sistema

La arquitectura de cliente / servidor utilizada para el componente del Procesador también hará uso de patrones de diseño, concretamente el patrón Comando o “Command”.

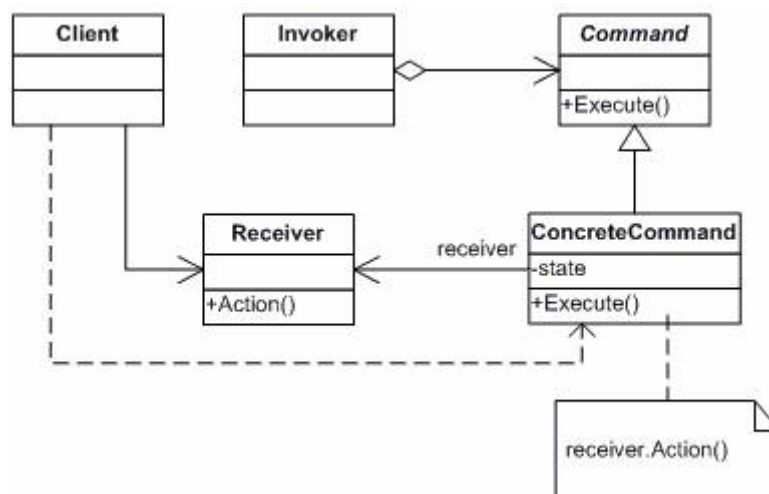


Imagen 26: Patrón Command

Se contará con una interfaz común a todas las instrucciones de ensamblador que será capaz de ejecutar el procesador. El procesador creará una instancia concreta de

cada instrucción bajo la misma interfaz usando la característica del polimorfismo de los lenguajes orientados a objetos. Todas las instrucciones actuarán sobre los mismo receptores que son la memoria y los registros, ambos pertenecientes al procesador.

4.2. COMPONENTES DEL SISTEMA

El sistema estará compuesto por un conjunto de componentes. Cada componente desempeñará una función bien delimitada y se comunicará con otros componentes cuando sea necesario. La división en componentes persigue aumentar la cohesión dentro de cada componente y reducir el acoplamiento entre módulos, lo que simplifica enormemente el diseño a alto nivel de la aplicación. Lo anterior se debe a que las funciones que se comuniquen mucho entre sí tienden a tener a realizar la misma funcionalidad común, por lo que estarán contenidas en un mismo componente, mientras que las funciones que realicen funcionalidades distintas presumiblemente se comunicarán menos entre sí.

Los componentes de la aplicación estarán compuestos por módulos, y estos a su vez en submódulos. Al igual que los componentes, los módulos persiguen agrupar las funciones del sistema según su funcionalidad, disminuyendo el acoplamiento dentro de cada componente y simplificando su diseño.

En los siguientes apartados se definirán los componentes que forman el sistema, y la descomposición de cada uno en módulos.

4.2.1.KERNEL

El componente Kernel será el encargado de emular el hardware principal de la consola, compuesto por el procesador, sus registros y la memoria principal, por lo que existirá un subcomponente por cada uno de esos elementos.

4.2.1.1.PROCESADOR

El módulo Procesador consiste en una unidad de control encargada de coordinar todo el funcionamiento interno, y el juego de instrucciones emulado.

El procedimiento de funcionamiento es de lectura de la posición del contador de programa, decodificación de la instrucción y por último su ejecución. A este proceso se le añade también la gestión de las interrupciones, tanto la emulación del hardware que las produce como la llamada a su rutina de tratamiento.

INTERFAZ

La interfaz externa del módulo Procesador ofrecerá:

- **Resetear:** reinicia todo el procesador a su estado inicial, afectando también a los registros y a ciertas posiciones de memoria.
- **Iniciar:** Comienza el proceso de emulación deteniéndose sólo en caso de fallo o si entra en modo depuración.
- **Asignar punto de interrupción:** Añade o elimina una dirección de memoria en la que debe producirse un punto de interrupción. Cuando vaya a ejecutar dicha dirección de memoria entrará automáticamente en modo depuración.

Y para cada instrucción descodificada y emulada:

- **Ejecutar:** Realiza todas las operaciones asociadas a la instrucción afectado a los registros o la memoria si procede.

COMUNICACIONES

El procesador se comunicará con los módulos Registro, Memoria y Debug del componente Periféricos. También se comunicará intensivamente con las instrucciones de este mismo módulo.

4.2.1.2.MEMORIA

El módulo de Memoria principal será el encargado de mantener y gestionar el acceso a la memoria física emulada, proporcionando operaciones que permitan el acceso y la escritura de forma segura por las instrucciones y el procesador.

También será el encargado de gestionar los registros especiales de entrada / salida, ya sea emulando directamente su comportamiento, o delegando en otros componentes o módulos.

INTERFAZ

Contará con los siguientes procedimientos públicos:

- Leer: Lee un valor o dato de una posición de memoria indicada
- Escribir: Escribe un dato en una posición de memoria determinada

COMUNICACIONES

Se comunicará con el módulo Teclado del componente Periféricos para la gestión del registro especial de entrada / salida asociado al funcionamiento de los botones de la consola emulada.

4.2.1.3. REGISTROS

El módulo de Registros será el encargado de almacenar el valor de todos los registros hardware emulados y el estado de todos los flags, proporcionando adicionalmente varios métodos de acceso y modificación.

INTERFAZ

Ofrecerá las siguientes funciones públicas:

- **Obtener Registro:** Devuelve el valor de un registro determinado
- **Asignar Registro:** Escribe un valor dado en un registro concreto
- **Obtener Flag:** Devuelve el estado de uno de los flags de ejecución
- **Asignar Flag:** Cambia el estado de uno de los flags

COMUNICACIONES

No depende ni invoca a ningún otro componente del sistema. Será utilizado por el módulo Procesador y sus instrucciones.

4.2.2. PERIFÉRICOS

El componente Periféricos es el encargado de emular todo aquel hardware que no se considera principal o del “núcleo” de la consola, lo cual no quiere decir que no sea imprescindible. Se trata del hardware más externo visualmente en la consola.

4.2.2.1. CARTUCHO

El cartucho proporciona el acceso a los datos de la ROM y la RAM no volátil. Se trata de un conjunto de clases por cada tipo de cartucho soportado que gestiona de manera independiente la lectura y escritura de los datos y el intercambio de los bancos de memoria. El procesamiento del tipo de cartucho, tamaño de datos y demás parámetros que puedan afectar al cartucho se realiza de manera automática por la clase primigenia.

INTERFAZ

Todos los cartuchos ofrecen las mismas funciones al resto del hardware:

- **Obtener información:** Devuelve información acerca de las características del cartucho en cuanto a tamaño de ROM y RAM si tiene, y realiza la comprobación de integridad de sus datos.
- **Leer:** Lee un dato de una posición del cartucho. Según la dirección de lectura se distingue entre la ROM y la RAM
- **Escribir:** Escribe un dato en una posición especificada del cartucho. Si se escribe sobre la zona de ROM interpreta la codificación de selección de bancos u otras prestaciones adicionales que pueda tener cada tipo de cartucho.

COMUNICACIONES

Este módulo no invoca a ningún otro, y sólo es utilizado por el módulo de Memoria para emular la proyección de los datos del cartucho sobre unas regiones de la memoria principal.

4.2.2.2. TECLADO

El módulo de Teclado gestiona y mantiene toda la información asociada a los eventos de los botones de la consola, directamente proyectados sobre algunas teclas del teclado del ordenador sobre el que está funcionando el emulador.

Cada uno de los 4 direcciones de cursos y 4 botones de acción estará asociado a una única tecla, de modo que cada vez que se pulse o se libere se actualizará el estado de dicho botón.

INTERFAZ

El teclado ofrece operaciones para manipular los botones:

- **Modificar estado:** Cambia el estado de un botón emulado, es decir, invierte su situación actual de pulsado a liberado o viceversa.
- **Actualizar estado:** Para que los cambios efectuados sobre los botones sean visibles para el componente Kernel, es necesario actualizar un registro especial de la memoria bajo demanda.

COMUNICACIONES

Este módulo no realiza ninguna llamada a otros módulos o componentes, y es invocado por el componente de Interfaz Gráfica para delegar la gestión de las teclas pulsadas y por el módulo de Memoria para actualizar el estado virtual de los botones de la consola.

4.2.2.3.DEBUG

El módulo de Debug no representa a ningún elemento físico de la consola original ni emula ninguna parte de hardware, pero proporciona una interfaz en modo texto para acceder y manipular de la forma más sencilla posible el estado de los componentes hardware que intervienen en el proceso de emulación: Procesados, Memoria, Registros y Teclado. El acceso al cartucho viene implícito a través del módulo de Memoria principal.

Las opciones que ofrece la consola de depuración y explicaciones acerca de su uso, se encuentran en el apartado de “Manual de Usuario”.

INTERFAZ

Sólo proporciona un método público reseñable:

- **Iniciar Consola de depuración:** Inicia la interfaz en modo texto para manipular la consola de depuración y estará activa hasta que se indique lo contrario.

COMUNICACIONES

Realiza llamadas a los principales componentes del sistema: Procesador, Memoria, Registros y Teclado. Es invocado desde la interfaz gráfica.

4.2.3.INTERFAZ GRÁFICA

El componente Interfaz Gráfica será el encargado de mostrar al usuario una interfaz de usuario sencilla y lo más completa posible del sistema, ofreciéndole los controles necesarios para el control de la emulación y la visualización de la misma.

4.2.3.1.GUI

La Interfaz Gráfica de Usuario (o GUI en inglés) estará compuesta por una ventana cuyo elemento central será la pantalla emulada y una serie de menús para realizar las acciones. La interacción del teclado será directa sobre la ventana que delegará sobre el componente Teclado.

Las opciones disponibles para el usuario son:

- **Cargar cartucho:** Permite seleccionar un fichero que contiene los datos del cartucho a emular
- **Iniciar emulación:** Inicia el proceso de emulación. Requiere que previamente se haya cargado algún cartucho.
- **Detener / Pausar / Reanudar emulación:** Detiene, pausa o reanuda el proceso de emulación. Requiere que la emulación haya sido previamente iniciada.
- **Cambiar escala:** Permite modificar el factor de multiplicación de la pantalla emulada para aumentar o disminuir su tamaño. La resolución seguirá siendo la misma original de la consola.
- **Entrar en modo depuración:** Inicia la consola de depuración del sistema donde se podrán modificar diferentes aspectos del proceso de emulación en tiempo real. Requiere que se haya cargado previamente algún cartucho.

INTERFAZ

No ofrece ningún método público ya que ningún componente del sistema debe modificar la interfaz.

COMUNICACIONES

El usuario será el que interactúe directamente con la interfaz gráfica. Desde aquí se podrán invocar operaciones sobre el Procesador, Cartucho y Teclado.

4.2.3.2. PANTALLA

La Pantalla es el elemento gráfico principal mostrado en la interfaz gráfica de usuario. Debido a que se trata de un elemento crítico, y dotado de cierta independencia del resto de la interfaz, se le considera lo suficientemente importante para tratarlo como un módulo.

Es la encargada de visualizar y mantener la imagen estática producida por la emulación y actualizarla convenientemente para proporcionar la sensación de movimiento e interactividad a los juegos emulados.

INTERFAZ

Proporciona los siguientes métodos públicos:

- **Dibujar Píxel:** Dibuja un píxel en una posición determinada de la pantalla y del color dado. Utiliza una paleta de colores integrada para descodificar el color a usar. El píxel dibujado o modificado no se ve reflejado inmediatamente en la interfaz gráfica, sino que se almacena en un buffer interno.
- **Limpiar Pantalla:** Pinta todos los píxeles de la pantalla de un mismo color, eliminando cualquier imagen previa que hubiera.
- **Actualizar Pantalla:** Realiza el refresco del buffer de la pantalla y la presenta al usuario a través de la interfaz de usuario.

COMUNICACIONES

La Pantalla es utilizada por el módulo de Gráficos, que es el único que ha descodificado la memoria gráfica y puede dibujar los píxeles correctamente, y por el GUI para integrarlo con el resto de componentes de la interfaz.

No realiza ninguna llamada externa a otro módulo.

4.2.3.3. GRÁFICOS

La gestión de gráficos es, junto al Procesador, el elemento más crítico y más complejo de todo el sistema. Su misión es descodificar la memoria gráfica (VRAM) generada por el proceso de emulación y traducirlo a píxeles visibles a través de la Pantalla.

La decodificación e interpretación se realiza bajo demanda y línea a línea por el sistema de interrupciones del procesador. Por cada línea deben interpretarse las tres capas por las que está compuesta el procesado gráfico: fondo, ventana y sprites.

La pantalla original de la consola cuenta con 144 líneas horizontales y 160 píxeles en cada una, haciendo un total de 23.040 píxeles interpretados individualmente por cada frame. Esto quiere decir que si se quiere conseguir una frecuencia de refresco de la pantalla cercana al hardware original de 50 hercios, será necesario interpretar más de 1.150.000 píxeles por segundo, lo cual nos da una idea aproximada de la criticidad de este módulo.

INTERFAZ

Los métodos públicos que pueden ser accedidos por otros módulos son:

- **Interrupción horizontal:** Decodifica e interpreta las tres capas de una determinada línea de la pantalla.
- **Interrupción vertical:** Refresca la Pantalla con las líneas interpretadas hasta ese momento.

COMUNICACIONES

El módulo de Gráficos interacciones directamente con el módulo de la Pantalla, y es accedido a su vez por el módulo del Procesador, desde la gestión de las interrupciones.

4.3. DIAGRAMA DE COMPONENTES DEL SISTEMA

A continuación se muestra el diagrama final con los componentes y módulos que forman el diseño arquitectónico de la aplicación.

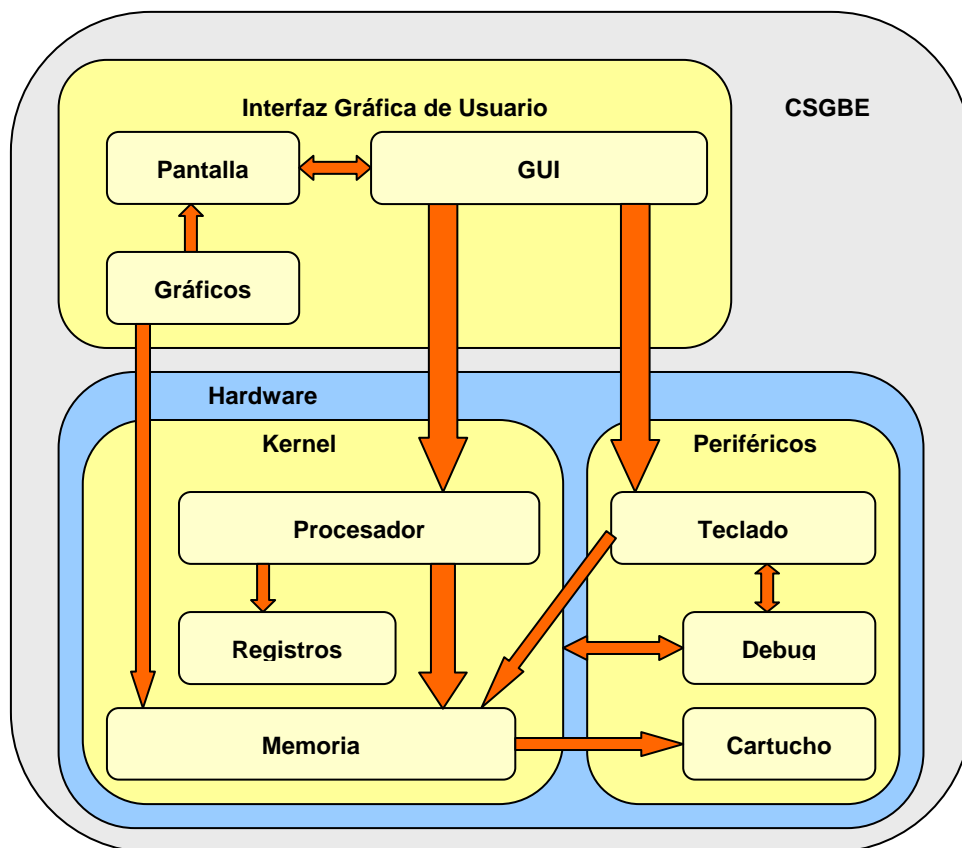


Imagen 27: Diagrama de Componentes del Sistema

5. DISEÑO DETALLADO

En el siguiente apartado se realizará un análisis detallado de la composición y funcionamiento de cada uno de los componentes que forman la aplicación. Por cada componente o módulo se indicarán las clases que lo componen, y por cada clase se presentará la información almacenada por la misma (atributos) y las operaciones públicas ofrecidas (funciones). Por cada elemento se indicará el propósito del mismo, además de la labor realizada en el caso de las funciones. La inmensa mayoría de las funciones serán de una complejidad muy reducida, por lo que se evitará entrar en detalles sobre su funcionamiento interno, lo cual se reservará a las funciones de mayor importancia y complejidad.

5.1. NOMENCLATURA

Cada elemento de diseño de la aplicación tendrá un identificador único. Dicho código se obtendrá de la unión del código del elemento padre del actual en la jerarquía de diseño y el nombre del propio elemento, separados por un punto.

Ejemplo:

El componente “Componente1” tiene un subordinado llamado “Modulo1”.

La nomenclatura final de dicho módulo será “Componente1.Modulo1”.

5.2. *KERNEL*

PROPÓSITO

El componente Kernel será el encargado de emular el hardware principal del sistema. Consiste en un procesador con su juego de instrucciones, la memoria principal, los registros y una serie de constantes globales a la emulación de toda la arquitectura.

TIPO

Paquete

SUBORDINADOS

Son subordinados de este paquete los siguientes módulos:

- Kernel.CPU
- Kernel.Registros
- Kernel.Memoria
- Kernel.Constantes
- Kernel.Instruccion
- Kernel.InstruccionXX

Siendo Kernel.InstruccionXX una agrupación semántica de todas las clases que contienen el juego de instrucciones completo del procesador.

DATOS

No procede.

INTERFACES

El componente Kernel se comunicará únicamente con el módulo Cartucho del paquete Periféricos desde el módulo de Memoria.

5.2.1.KERNEL.CPU

kernel::CPU
-_MHZ : double -_instruccionesProcesadas : int -_tiempo : double -_ciclos : int -_debug : bool -_bp : int -_error : bool -_memoria : Memoria -_registros : Registros -_graphics : Graphics
+CPU(entrada memoria : Memoria, entrada velocidad : double) +reset() +toggleBreakpoint(entrada pc : int) +iniciar() +procesarInstrucciones(entrada cantidad : int) -dispararInterrupciones() -comprobarInterrupciones() -procesarInstruccion() -procesarInstruccionCB() : Instruccion

Imagen 28: Kernel.CPU

PROPÓSITO

La clase CPU será la encargada de emular el procesador Z80 ligeramente modificado que lleva incorporado la consola GameBoy. Tiene gestión de interrupciones y la decodificación, interpretación y ejecución de las instrucciones de ensamblador.

TIPO

Clase

SUBORDINADOS

Ninguno

DATOS

Visibilidad	Nombre	Tipo	Descripción
Privada	_MHZ	Double	Velocidad del procesador en MHz
Privada	_instruccionesProcesadas	Int	Cantidad de instrucciones emuladas desde el último reinicio.
Privada	_tiempo	Double	Microsegundos que debería haber tardado la emulación.
Privada	_ciclos	Int[]	Array con el numero de ciclos transcurridos desde cada ultima interrupción.
Privada	_bp	ArrayList	Lista de puntos de interrupción de direcciones de memoria para el debug.
Privada	_debug	Bool	Indica si esta ejecutando el procesador en modo de depuración.
Privada	_error	Bool	Indica si ha ocurrido algún error grave e irreparable en la emulación.
Privada	_memoria	Kernel.Memoria	Memoria principal.
Privada	_registros	Kernel.Registros	Registros internos de la CPU.
Privada	_graphics	Gui.Graphics	Sistema de gestión gráfica.

INTERFACES

Signatura	public CPU(Memoria memoria, double velocidad)
Descripción	Constructor del procesador. Referencia la memoria y crea los registros y el gestor gráfico. Inicializa los atributos y la memoria.

Signatura	public void reset()
Descripción	Asigna un valor por defecto a los registros y a ciertas direcciones de memoria. Equivale a un reinicio físico de la consola.

Signatura	public void toggleBreakpoint(int pc)
Descripción	Crea o elimina un punto de interrupción en una dirección de memoria.

Signatura	public void procesarInstrucciones(int cantidad)
Descripción	Ejecuta un número determinado de instrucciones según el flujo de ejecución.

Signatura	public void iniciar()
Descripción	Inicia el proceso de emulación y solo se detiene cuando encuentre un error.

Signatura	private void dispararInterrupciones()
Descripción	Activa las interrupciones adecuadas y simula su funcionamiento por hardware.

Signatura	private void comprobarInterrupciones()
Descripción	Inicia las rutinas de tratamiento de cada una de las interrupciones que estén activas

Signatura	private void procesarInstruccion()
Descripción	Ejecuta una instrucción del contador de programa

Signatura	private Instruccion procesarInstruccionCB(int parm1, int parm2)
Descripción	Decodifica una instrucción extendida por el opCode CB

5.2.2.KERNEL.REGISTROS

kernel::Registros
-A : int -B : int -C : int -D : int -E : int -SP : int -PC : int -HL : int -flagZ : bool -flagH : bool -flagN : bool -flagC : bool -IME : bool
+getReg(entrada registro : string) : int +setReg(entrada registro : string, entrada valor : int) +getFlag(entrada flag : string) : bool +setFlag(entrada flag : string, entrada valor : bool) +getFlag() +setFlag(entrada flags : int)

Imagen 29: Kernel.Registros

PROPÓSITO

La clase registros representa a los registros internos que tiene a disposición el procesador y a los que accede a través del juego de instrucciones.

TIPO

Clase

SUBORDINADOS

Ninguno

DATOS

Visibilidad	Nombre	Tipo	Descripción
Private	A	int	Registro A de 8 bits
Private	B	int	Registro B de 8 bits
Private	C	int	Registro C de 8 bits
Private	D	int	Registro D de 8 bits
Private	E	int	Registro E de 8 bits
Private	SP	int	Registro SP de 16 bits (puntero de pila)
Private	PC	int	Registro PC de 16 bits (contador de programa)

Private	HL	int	Registro HL de 16 bits
Private	Flag_Z	bool	Flag de cero
Private	Flag_H	bool	Flag de Half-Carry
Private	Flag_N	bool	Flag de Suma/Resta
Private	Flag_C	bool	Flag de acarreo
Private	IME	bool	Interrupt Master Enable

Aparte de estos atributos, se dispone de unas propiedades públicas para poder acceder a ellos y a versiones modificadas de los registros, por ejemplo el registro BC, que simplemente es una unión virtual de los registros de 8 bits B y C.

Las propiedades en el lenguaje C# son unos elementos que están entre los atributos y los métodos. Se podrían considerar como funciones especiales que únicamente se pueden utilizar para leer o escribir en atributos determinados, o se podrían ver como atributos con algunos cálculos intermedios.

INTERFACES

Signatura	public int getReg(string registro)
Descripción	Obtiene el valor de un registro a partir de su nombre

Signatura	public void setReg(string registro, int valor)
Descripción	Asigna un valor a un registro identificado por su nombre

Signatura	public bool getFlag(string flag)
Descripción	Obtiene el estado de un flag por su nombre

Signatura	public void setFlag(string flag, bool valor)
Descripción	Asigna un nuevo estado a un flag identificado por su nombre

Signatura	public int getFlag()
Descripción	Obtiene el valor del registro con todos los flags

Signatura	public void setFlag(int flags)
Descripción	Asigna un nuevo estado a todos los flags a partir de su equivalente numérico.

5.2.3.KERNEL.MEMORIA

kernel::Memoria
_ram : byte _cartucho : Cartucho _lecturas : int _escrituras : int +Memoria(entrada tamanyo : int, entrada cartucho : object) +leer(entrada direccion : int) : byte +escribir(entrada valor : int, entrada direccion : int) -escribirO(entrada valor : int, entrada direccion : int)

Imagen 30: Kernel.Memoria

PROPÓSITO

Clase encargada de simular el comportamiento del mapa de memoria, así como de la memoria física y de las direcciones de entrada / salida.

TIPO

Clase

SUBORDINADOS

Ninguno

DATOS

Visibilidad	Nombre	Tipo	Descripción
Private	_ram	Byte[]	Array con toda la memoria principal.
Private	_cartucho	Cartucho	Cartucho cargado
Private	_lecturas	int	Numero de lecturas realizadas.
Private	_escrituras	Int	Numero de escrituras realizadas.

INTERFACES

Signatura	public Memoria(int tamanyo, Cartucho cartucho)
Descripción	Crea e inicializa la memoria física.

Signatura	public byte leer(int direccion)
Descripción	Lee una posición de memoria

Signatura	public void escribir(int valor, int direccion)
Descripción	Escribe un valor en una dirección de memoria. Esta función solo debe ser accedida por las instrucciones, el resto de objetos como periféricos de pantalla o teclado deberán acceder directamente a la memoria sin pasar por estas funciones porque podrían producir un bucle infinito

Signatura	private void escribirIO(int valor, int direccion)
Descripción	Escribe y realiza un tratamiento especial en las direcciones de entrada / salida.

5.2.4.KERNEL.CONSTANTES

PROPÓSITO

Clase que contiene valores constantes usadas en múltiples partes de la aplicación y directamente relacionadas con el proceso de emulación.

TIPO

Clase

SUBORDINADOS

Ninguno.

DATOS

Debido a la enorme cantidad de constantes usadas, no resulta significativo enumerarlas una a una ya que quedarían poco claras.

Las constantes usadas pueden dividirse en tres grupos según su utilidad:

- **Direcciones de memoria:** El valor de la dirección de memoria de un registro especial de entrada salida, como pueden ser el control del temporizador o el estado de los botones.

- **Tiempo:** Valores numéricos que indican la frecuencia de un suceso o el tiempo entre dos eventos. Por ejemplo el número de ciclos que dura una interrupción vertical, el número de instrucciones medio entre dos interrupciones o la velocidad del procesador.
- **Valores:** Constantes numéricas no significativas usadas en el sistema, como el identificador numérico de cada botón emulado, el tamaño del espacio de direcciones o el número de bit de cada interrupción en el registro de interrupciones.

INTERFACES

Ninguna

5.2.5.KERNEL.INSTRUCCION

kernel:: <i>Instruccion</i>
#_nombre : string #_longitud : int #_duracion : int #_registros : Registros #_memoria : Memoria
+Instruccion(entrada registros : Registros, entrada memoria : Memoria) +ejecutar() : int

Imagen 31: Kernel.Instruccion

PROPÓSITO

Interfaz que define las operaciones básicas que debe proporcionar una instrucción de ensamblador para ser ejecutada en el procesador.

TIPO

Clase

SUBORDINADOS

Ninguno

DATOS

Visibilidad	Nombre	Tipo	Descripción
Protected	_nombre	String	Nombre mnemónico de la instrucción.
Protected	_longitud	Int	Longitud en bytes de la instrucción contando sus parámetros.
Protected	_duracion	Int	Duración teórica de ejecución en ciclos.
Protected	_registros	Kernel.Registros	Registros de la CPU.
Protected	_memoria	Kernel.Memoria	Referencia a la memoria

INTERFACES

Signatura	public Instruccion(Registros registros, Memoria memoria)
Descripción	Construye una instrucción guardando las referencias a los registros y a la memoria.

Signatura	public abstract int ejecutar()
Descripción	Ejecuta la instrucción.

5.2.6.KERNEL.INSTRUCCIONXX

Esta clase representa a todas las implementaciones concretas de la interfaz Instrucción. Son 99 clases agrupadas en 9 categorías que implementan el total de las 499 instrucciones diferentes.

Su nomenclatura tiene un significado semántico sobre las instrucciones reales que implementa. Una R indica un registro de 8 bits, RR un registro de 16 bits, RADR indica la dirección de memoria almacenada en un registro de 16 bits y N indica un valor literal

Instrucciones aritméticas:

InstruccionDEC_R	InstruccionDEC_RADR	InstruccionDEC_RR
InstruccionINC_R	InstruccionINC_RADR	InstruccionINC_RR
InstruccionADD_R_R	InstruccionADD_R_RADR	InstruccionADD_RR_RR
InstruccionADD_R_NN	InstruccionADD_R_R	InstruccionCP_R
InstrucciónCP_RADR	InstruccionCP_N	InstruccionSBC_R_R
InstruccionSUB_R_R	InstruccionSUB_R_RADR	InstruccionSBC_R_N
InstruccionADC_R_R	InstruccionADC_R_RADR	InstruccionADC_R_N
InstruccionSUB_R_N	InstruccionSBC_R_RADR	InstruccionDAA

Instrucciones de bit:

InstruccionBIT_R	InstruccionBIT_RADR	InstruccionSET_R
InstruccionSET_RADR	InstruccionRES_R	InstruccionRES_RADR
InstruccionSWAP_R	InstruccionSWAP_RADR	

Instrucciones de complemento:

InstruccionCPL	InstruccionSCF	InstruccionCCF
----------------	----------------	----------------

Instrucciones de interrupciones:

InstruccionEI	InstruccionHALT	InstruccionSTOP
InstruccionDI		

Instrucciones de salto: CC0 indica una condición no cumplida, es decir, un flag no activo y CC1 condición cierta

InstruccionJP_ADR	InstruccionJP_RADR	InstruccionJP_CC0_ADR
InstruccionJP_CC1_ADR	InstruccionJR_N	InstruccionJR_CC0_N
InstruccionJR_CC1_N	InstruccionCALL_ADR	InstruccionCALL_CC0_ADR
InstruccionPUSH_RR	InstruccionPOP_RR	InstruccionCALL_CC1_ADR
InstruccionRET_ADR	InstruccionRETI	InstruccionRET_CC0_ADR
InstruccionINT	InstruccionRST	InstruccionRET_CC1_ADR

Instrucciones de carga: DR indica una dirección de memoria desplazada por el valor de un registro de 8 bits, y SPD hace referencia al puntero de pila también desplazado una cantidad dada por un literal.

InstruccionLD_R_R	InstruccionLD_DD_NN	InstruccionLD_R_N
InstruccionLD_ADR_R	InstruccionLD_ADR_RR	InstruccionLDD_RADR_R
InstruccionLD_R_DADR	InstruccionLD_R_DR	InstruccionLDI_RADR_RR
InstruccionLD_DADR_R	InstruccionLD_DR_R	InstruccionLDD_R_RADR
InstruccionLD_RADR_R	InstruccionLD_R_ADR	InstruccionLDI_R_RADR
InstruccionLD_R_ADR	InstruccionLD_R_RADR	InstruccionLD_RADR_N
InstruccionLD_R_SPD		

Instrucciones lógicas:

InstruccionXOR_R	InstruccionXOR_RADR	InstruccionXOR_N
InstruccionOR_R_R	InstruccionOR_R_ADR	InstruccionOR_N
InstruccionAND_R_R	InstruccionAND_R_N	InstruccionAND_RADR

Instrucciones de desplazamiento: R indica desplazamiento a derechas, L a izquierdas y una C adicional indica que tiene en cuenta el flag de acarreo.

InstruccionRLC	InstruccionRLC_RADR	InstruccionSLA
InstruccionSLA_RADR	InstruccionSRA	InstruccionSRA_RADR
InstruccionSRL	InstruccionSRL_RADR	InstruccionRL
InstruccionRL_RADR	InstruccionRRC	InstruccionRRC_RADR
InstruccionRR	InstruccionRR_RADR	

Adicionalmente existe la instrucción NOP cuyo contenido está vacío.

PROPÓSITO

Implementación de una o varias instrucciones de ensamblador.

TIPO

Clase

SUBORDINADOS

Ninguno

5.3. PERIFERICOS

PROPÓSITO

El componente Periféricos engloba la emulación de todo aquel hardware no considerado como parte del núcleo de la consola, es decir, el hardware visible de la consola original como pueden ser los botones o los cartuchos que contienen los juegos.

TIPO

Paquete

SUBORDINADOS

Son subordinados los módulos:

- Perifericos.Keypad

- Perifericos.Cartucho
- Perifericos.CartuchoMBC0
- Perifericos.CartuchoMBC1
- Perifericos.CartuchoMBC2
- Perifericos.CartuchoMBC3
- Perifericos.CartuchoMBC5
- Perifericos.Debug

DATOS

No procede.

INTERFACES

El componente Perifericos se comunicará con el módulo Memoria del Kernel a través del Keypad, y con todos los módulos del componente Kernel desde el módulo Debug.

5.3.1.PERIFERICOS.CARTUCHO

perifericos::Cartucho
#_nombreFichero : string #_nombre : string #_romBloques : int #_ramBloques : int #_rom : byte #_ram : byte
+Cartucho(entrada nombreFichero : string) -cargar() +info() : string +getTipo() : string +leer(entrada direccion : int) : byte +escribir(entrada valor : int, entrada direccion : int) -checksum() : bool +cargarCartucho(entrada nombreFichero : string) : Cartucho

Imagen 32: Perifericos.Cartucho

PROPÓSITO

Representa un cartucho de GameBoy con su ROM y opcionalmente RAM interna.

TIPO

Clase.

SUBORDINADOS

Son subordinados de esta clase las siguientes clases que heredan:

- Perifericos.CartuchoMBC0
- Perifericos.CartuchoMBC1
- Perifericos.CartuchoMBC2
- Perifericos.CartuchoMBC3
- Perifericos.CartuchoMBC5

DATOS

Visibilidad	Nombre	Tipo	Descripción
Protected	_nombreFichero	String	Ruta al fichero que contiene la ROM.
Protected	_nombre	String	Nombre interno de la ROM.
Protected	_romBloques	Int	Numero de bloques de programa (16 Kb cada uno).
Protected	_ramBloques	Int	Numero de bloques de RAM internos del cartucho (8 Kb cada uno). Puede ser 0 si no tiene.
Protected	_rom	Byte[]	ROM.
Protected	_ram	Byte[]	RAM. Puede ser un array vacío si no tiene.

INTERFACES

Signatura	public Cartucho(string nombreFichero)
Descripción	Creador de cartuchos. Carga el contenido de la ROM en memoria.

Signatura	private void cargar()
Descripción	Carga en memoria toda la ROM del cartucho y la RAM si tiene.

Signatura	public string info()
Descripción	Devuelve información sobre el cartucho cargado como el tipo, tamaño de la ROM, tamaño de la RAM, nombre interno, y resultado del test de checksum.

Signatura	public abstract string getTipo()
Descripción	Obtiene el nombre del tipo de cartucho.

Signatura	public abstract byte leer(int direccion)
Descripción	Lee un byte de una dirección.

Signatura	public abstract void escribir(int valor, int direccion)
Descripción	Escribe un byte en una dirección.

Signatura	private bool checksum()
Descripción	Comprueba la integridad de la ROM mediante una operación de checksum.

Signatura	public static Cartucho cargarCartucho(string nombreFichero)
Descripción	Crea un tipo específico de cartucho.

5.3.2.PERIFERICOS.CARTUCHOMBC0

PROPÓSITO

Representa un cartucho de tipo MBC0 (solo ROM sin RAM).

TIPO

Clase.

SUBORDINADOS

Ninguno.

DATOS

Visibilidad	Nombre	Tipo	Descripción
Private	_tipoCartucho	String	Definición del tipo de cartucho.

INTERFACES

Implementaciones de los métodos abstractos de la clase Perifericos.Cartucho.

5.3.3.PERIFERICOS.CARTUCHOMBC1

PROPÓSITO

Representa un cartucho de tipo MBC1.

TIPO

Clase.

SUBORDINADOS

Ninguno.

DATOS

Visibilidad	Nombre	Tipo	Descripción
Private	_nombreCartucho	String	Definición del tipo de cartucho
Private	_ramEnabled	Bool	Determina si la zona de RAM esta habilitada.
Private	_romPage	Int	Banco de ROM que se encuentra actualmente proyectado.
private	_ramPage	Int	Banco de RAM que se encuentra actualmente proyectado.

INTERFACES

Implementaciones de los métodos abstractos de la clase Perifericos.Cartucho.

5.3.4.PERIFERICOS.CARTUCHOMBC2

PROPÓSITO

Representa un cartucho de tipo MBC2 (solo 1 banco de RAM).

TIPO

Clase.

SUBORDINADOS

Ninguno.

DATOS

Visibilidad	Nombre	Tipo	Descripción
Private	_nombreCartucho	String	Definición del tipo de cartucho
Private	_ramEnabled	Bool	Determina si la zona de RAM esta habilitada.
private	_romPage	int	Banco de ROM que se encuentra actualmente proyectado.

INTERFACES

Implementaciones de los métodos abstractos de la clase Perifericos.Cartucho.

5.3.5.PERIFERICOS.CARTUCHOMBC3

PROPÓSITO

Representa un cartucho de tipo MBC3

TIPO

Clase.

SUBORDINADOS

Ninguno.

DATOS

Visibilidad	Nombre	Tipo	Descripción
Private	_nombreCartucho	String	Definición del tipo de cartucho
Private	_ramEnabled	Bool	Determina si la zona de RAM esta habilitada.
Private	_romPage	Int	Banco de ROM que se encuentra actualmente proyectado.
private	_ramPage	Int	Banco de RAM que se encuentra actualmente proyectado.

INTERFACES

Implementaciones de los métodos abstractos de la clase Perifericos.Cartucho.

5.3.6.PERIFERICOS.CARTUCHOMBC5**PROPÓSITO**

Representa un cartucho de tipo MBC5.

TIPO

Clase.

SUBORDINADOS

Ninguno.

DATOS

Visibilidad	Nombre	Tipo	Descripción
Private	_nombreCartucho	String	Definición del tipo de cartucho
Private	_ramEnabled	Bool	Determina si la zona de RAM esta habilitada.
Private	_romPage	Int	Banco de ROM que se encuentra actualmente proyectado.
private	_ramPage	Int	Banco de RAM que se encuentra actualmente proyectado.

INTERFACES

Implementaciones de los métodos abstractos de la clase Perifericos.Cartucho.

5.3.7.PERIFERICOS.KEYPAD

perifericos::Keypad
- _teclas : bool
+toggleTecla(entrada tecla : int)
+teclaPulsada(entrada tecla : object)
+teclaLiberada(entrada tecla : object)
+actualizar(entrada memoria : Memoria)

Imagen 33: Perifericos.Keypad

PROPÓSITO

Gestiona la pulsación de las teclas de la consola emulada

TIPO

Clase.

SUBORDINADOS

Ninguno.

DATOS

Visibilidad	Nombre	Tipo	Descripción
Private	_teclas	Bool[]	Estado actual de las teclas.

INTERFACES

Signatura	public static void toggleTecla(int tecla)
Descripción	Invierte el estado de una tecla.

Signatura	public static void teclaPulsada(Key tecla)
Descripción	Registra la pulsación de una tecla.

Signatura	public static void teclaLiberada(Key tecla)
Descripción	Registra la liberación de una tecla pulsada previamente.

Signatura	public static void actualizar(Memoria memoria)
Descripción	Actualiza la dirección de memoria adecuada según la matriz de teclas solicitada con el estado actual de las teclas.

5.3.8.PERIFERICOS.DEBUG

periféricos::Debug
-hexChars : string = 0123456789ABCDEF
+WriteLine(entrada cadena : string)
+WriteLine()
+Write(entrada cadena : string)
+Write(entrada cadena : char)
+iniciarContador() : object
+detenerContador(entrada time : object) : double
+hexByte(entrada b : int) : string
+hexWord(entrada w : int) : string
+byteHex(entrada hex : string) : int
+wordHex(entrada hex : string) : int
-hexDump(entrada memoria : object, entrada direccion : int, entrada longitud : int)
-imprimirRegistros(entrada registros : Registros)
-imprimirInterrupciones(entrada registros : Registros, entrada memoria : Memoria)
-asignarInterrupcion(entrada memoria : Memoria, entrada interrupcion : string, entrada valor : string)
-toggleTecla(entrada memoria : Memoria, entrada tecla : string)
+ConsolaDepuracion(entrada cpu : CPU)
-mostrarAyudaConsolaDepuracion()

Imagen 34: Perifericos.Debug

PROPÓSITO

Clase con funciones auxiliares y funciones de depuración para el proceso de emulación.

TIPO

Clase.

SUBORDINADOS

Ninguno.

DATOS

Ninguno

INTERFACES

Signatura	public static void WriteLine(string cadena)
Descripción	Imprime una línea por la salida estándar.

Signatura	public static void WriteLine()
Descripción	Imprime un salto de línea por la salida estándar.

Signatura	public static void Write(string cadena)
Descripción	Imprime texto en la línea actual de la salida estándar.

Signatura	public static void Write(char caracter)
Descripción	Imprime un carácter en la línea actual de la salida estándar

Signatura	public static DateTime iniciarContador()
Descripción	Inicia un contador con la fecha actual.

Signatura	public static double detenerContador(DateTime time)
Descripción	Detiene un contador y devuelve el tiempo en segundos transcurridos desde que se inició.

Signatura	public static string hexByte(int b)
Descripción	Convierte un byte en su equivalente en caracteres hexadecimales.

Signatura	public static string hexWord(int w)
Descripción	Convierte una palabra de dos bytes en su equivalente en caracteres hexadecimales.

Signatura	<code>public static int byteHex(string hex)</code>
Descripción	Convierte un byte expresado en caracteres hexadecimales a su equivalente numérico.

Signatura	<code>public static int wordHex(string hex)</code>
Descripción	Convierte una palabra de cuatro caracteres hexadecimales en su equivalente numérico.

Signatura	<code>private static void hexDump(kernel.Memoria memoria, int direccion, int longitud)</code>
Descripción	Imprime una zona de memoria en un formato presentable.

Signatura	<code>private static void imprimirRegistros(kernel.Registros registros)</code>
Descripción	Imprime el valor de todos los registros.

Signatura	<code>private static void imprimirInterrupciones(kernel.Registros registros, kernel.Memoria memoria)</code>
Descripción	Imprime el estado de las interrupciones y del IME (Interrupt Master Enable).

Signatura	<code>private static void asignarInterrupcion(kernel.Memoria memoria, string interrupcion, string valor)</code>
Descripción	Activa o desactiva una interrupción identificada por nombre.

Signatura	<code>private static void toggleTecla(kernel.Memoria memoria, string tecla)</code>
Descripción	Cambia el estado de una tecla de la consola identificada por nombre.

Signatura	<code>public static void ConsolaDepuracion(kernel.CPU cpu)</code>
Descripción	Procesa las peticiones de la consola de depuración.

Signatura	private static void mostrarAyudaConsolaDepuracion()
Descripción	Muestra las opciones posibles de la consola de depuración.

5.4. GUI

PROPÓSITO

El componente GUI es el encargado de toda la parte visual, tanto de la interfaz de usuario como del procesado gráfico del proceso de emulación que se verá reflejado sobre parte de la interfaz.

TIPO

Paquete.

SUBORDINADOS

Son subordinados las clases:

- GUI.GUI
- GUI.Graphics
- GUI.Pantalla

DATOS

No procede.

INTERFACES

El componente GUI se comunica con el módulo Kernel.Procesador para controlar el flujo de la emulación, con el módulo Kernel.Keypad sobre el que delega para la gestión de eventos del teclado, y con el módulo Kernel.Memoria desde el módulo GUI.Graphics para la interpretación de la memoria gráfica.

5.4.1.GUI.GUI

gui::gui
-_consola : object
-_pantalla : Pantalla
+GUI()
+GUI(entrada rom : string)
-iniciarGUI()
-CambiarZoom(entrada zoom : int)
-Debug(entrada o : object, entrada args : object)
-Open_File(entrada o : object, entrada args : object)
-Key_Pressed(entrada o : object, entrada args : object)
-Key_Released(entrada o : object, entrada args : object)

Imagen 35: Gui.Gui

PROPÓSITO

Implementa y gestiona la interfaz gráfica de la aplicación. Permite al usuario controlar el estado del emulado y ver la pantalla de la consola emulada. También proporciona la posibilidad de iniciar la consola de depuración.

TIPO

Clase.

SUBORDINADOS

DATOS

Visibilidad	Nombre	Tipo	Descripción
private	_consola	GB	Representa la consola con su procesador y memoria dentro de un hilo de ejecución.
private	_pantalla	GUI.Pantalla	Pantalla.
Private	_lastDir	string	Dirección en el sistema de archivos al directorio que contiene el último cartucho abierto.

Adicionalmente existen varios atributos que representan cada uno de los elementos de la interfaz de usuario, como la barra de menú con sus opciones o la barra de estado, pero debido a que no afectan directamente al propósito de la aplicación ni a la emulación en sí misma, serán omitidos.

INTERFACES

Signatura	public GUI()
Descripción	Constructor de la interfaz gráfica.

Signatura	public GUI(String rom)
Descripción	Crea una interfaz gráfica con un cartucho cargado inicialmente y comienza inmediatamente su emulación.

Signatura	private void iniciarGUI()
Descripción	Crea e inicializa todos los objetos de la interfaz grafica.

Signatura	static void CambiarZoom(int zoom)
Descripción	Gestiona el evento de cambio de zoom de la pantalla. Debe eliminar la pantalla actual y crear una nueva con el nuevo factor de multiplicación.

Signatura	static void Debug (object o, EventArgs args)
Descripción	Inicia el modo de depuración de la consola. Requiere que haya un cartucho cargado previamente. Comienza el proceso de emulación si no estaba iniciado anteriormente.

Signatura	static void Open_File (object o, EventArgs args)
Descripción	Abre un cuadro de diálogo para seleccionar un fichero de cartucho y recuerda el último directorio accedido.

Signatura	static void Key_Pressed (object o, KeyPressEventArgs args)
Descripción	Evento de pulsación de tecla. Delega en el módulo de Keypad.

Signatura	static void Key_Released (object o, KeyReleaseEventArgs args)
Descripción	Evento de liberación de tecla. Delega en el módulo de Keypad.

5.4.2.GUI.PANTALLA

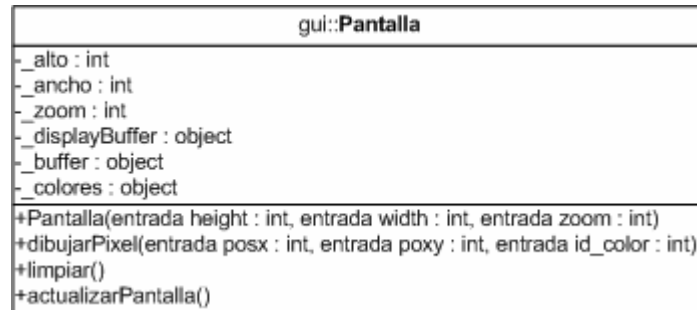


Imagen 36: Gui.Pantalla

PROPÓSITO

Elemento gráfico que permite el dibujo mediante píxeles. Depende de la plataforma gráfica GTK#.

TIPO

Clase.

SUBORDINADOS

Ninguno.

DATOS

Visibilidad	Nombre	Tipo	Descripción
Private	_alto	Int	Alto en píxeles de la pantalla
Private	_ancho	Int	Ancho en píxeles de la pantalla
Private	_zoom	Int	Zoom de la pantalla
Private	_displayBuffer	Bitmap	Imagen con el frame actual de la pantalla
Private	_buffer	Drawing.Graphics	Buffer de dibujo para acceder a la imagen
Private	_colores	Color[]	Paleta de colores usados

INTERFACES

Signatura	public Pantalla(int height, int width, int zoom)
Descripción	Crea una pantalla con una alto, ancho y ampliación determinados.

Signatura	public void dibujarPixel(int posX, int posY, int id_color)
Descripción	Dibuja un píxel de un color en una posición concreta de la pantalla.

Signatura	public void limpiar()
Descripción	Limpia la pantalla de dibujo a negro.

Signatura	public void actualizarPantalla()
Descripción	Actualiza la pantalla dibujando el frame que tiene actualmente en el buffer.

5.4.3.GUI.GRAPHICS

gui::Graphics
- _pantalla : Pantalla - _memoria : Memoria - _frames : int +Graphics(entrada memoria : object) +vblank() +hblank() -getIdColor(entrada x : int, entrada y : int) : int -id2color(entrada id : int, entrada direccion : int) : int -actualizarBG(entrada scanLine : int) -actualizarVentana(entrada scanLine : int) -actualizarSprites(entrada scanLine : int) -getIdTile(entrada xTile : int, entrada yTile : int, entrada mapAddress : int, entrada tileAddress : int) : int -getTile(entrada idTile : int, entrada tileAddress : int, entrada bitX : int, entrada bitY : int) : int

Imagen 37: Gui.Graphics

PROPÓSITO

El componente Graphics es el elemento clave del componente GUI, creando el nexo de unión entre la emulación y la interfaz gráfica. Se encarga de interpretar la zona de memoria gráfica.

TIPO

Clase.

SUBORDINADOS

Ninguno.

DATOS

Visibilidad	Nombre	Tipo	Descripción
Private	_pantalla	GUI.Pantalla	Pantalla sobre la que realizar el dibujado.
Private	_memoria	Kernel.Memoria	Memoria a interpretar gráficamente.
Private	_frames	int	Cantidad de frames renderizados hasta el momento.

INTERFACES

Signatura	public Graphics(Memoria memoria)
Descripción	Construye el intérprete gráfico en base a una memoria.

Signatura	public void vblank()
Descripción	Interrupción vertical. Solicita a la pantalla un refresco total de la imagen.

Signatura	public void hblank()
Descripción	Interrupción horizontal. Actualiza el fondo, ventana y sprites de la línea actual de dibujo

Signatura	private int getIdColor(int x, int y)
Descripción	Obtiene el identificador de color del fondo/ventana en una posición concreta.

Signatura	private int id2color(int id, int direccion)
Descripción	Transforma un identificador de color en su color real usando una paleta de colores.

Signatura	private void actualizarBG(int scanLine)
Descripción	Actualiza el fondo de la pantalla de una línea concreta.

Signatura	private void actualizarVentana(int scanLine)
Descripción	Actualiza la ventana de una línea concreta.

Signatura	private void actualizarSprites(int scanLine)
Descripción	Actualiza los sprites que aparecen en una línea de la pantalla.

Signatura	private int getIdTile(int xTile, int yTile, int mapAddress, int tileAddress)
Descripción	Obtiene el numero de tile correspondiente a una posición de la pantalla.

Signatura	private int getTile(int idTile, int tileAddress, int bitX, int bitY)
Descripción	Obtiene el valor del color de un píxel de un tile en concreto.

6. IMPLEMENTACIÓN DEL SISTEMA

En la sección de implementación se indicarán los elementos tecnológicos empleados en la creación del sistema, como el lenguaje de programación o la herramienta de desarrollo. También se incluirá y comentará una sección de código representativa de cada uno de los módulos de la aplicación, ya sea por su importancia o su complejidad.

6.1. TECNOLOGÍA EMPLEADA

6.1.1. LENGUAJE DE PROGRAMACIÓN

La totalidad de la implementación del sistema se realizará en el lenguaje orientado a objetos C#, en la versión 1.0. El lenguaje anterior resulta muy cómodo para la realización de emuladores y simulaciones, ya que la tecnología de orientación a objetos permite simular el comportamiento de entidades reales, como los elementos hardware de forma sencilla, aunque con una pequeña sobrecarga al tratarse de un lenguaje de tan alto nivel y siendo inicialmente interpretado.

Otro factor importante en la elección del lenguaje, además de la enorme proyección del mismo, es la posibilidad de ejecutar los programas implementados en varias plataformas, usando una implementación distinta en cada una pero con un enorme nivel de compatibilidad entre ellas. La implementación Mono de Ximian Inc. (ahora absorbida por Novell) funciona en plataformas Windows, GNU/Linux y MacOS, y tiene soporte para prácticamente la totalidad de la especificación del lenguaje y de las librerías de la implementación .NET de Microsoft bajo Windows, a excepción de la interfaz gráfica System.Windows.Forms (SWF).

Para la interfaz gráfica, dado que se persigue la mayor compatibilidad posible con las plataformas existentes, se ha optado por las librerías gráficas GTK+, desarrolladas por la fundación GNOME e impulsadas también por Ximian Inc. Para su uso en lenguaje C# existen unos “bindings” o envoltorios de dicha librerías llamados GTK#. Estas librerías pueden ser usadas tanto por la plataforma .NET como Mono y también se encuentran disponibles en los principales sistemas operativos.

Finalmente se ha optado por usar la implementación de Mono principalmente bajo la plataforma GNU/Linux en combinación con las librerías gráficas GTK#.

6.1.2.PLATAFORMA DE DESARROLLO

Tras la elección del lenguaje de programación con el que se codificará la aplicación, resulta importante, dada la magnitud del sistema a construir, escoger una plataforma de desarrollo que facilite la gestión de la totalidad del proceso de implementación de la aplicación. Al contrario de lo que se pueda pensar, no siempre la utilización de un IDE gráfico potente y completo ayuda a la implementación, aunque se trate de un proyecto de gran envergadura, sino que lo más eficiente siempre resultar ser con lo que el desarrollador se sienta más cómodo y maneje todos sus aspectos con la mayor soltura.

Es por ello que se ha optado por el uso del editor de consola VIM, que junto con las múltiples pestañas que permite un simulador de terminal de consola como Gnome-Terminal, y varios scripts de Bash y Makefile, proporcionan una plataforma de desarrollo completa, muy personalizable y fácilmente modificable.

En el script de compilación de tipo Makefile se ha incluido también un apartado para soportar un sistema de backups o copias de seguridad online. Bajo demanda permite realizar un comprimido de todo el proyecto y enviarlo a un servidor remoto mediante el protocolo SSH.

El contenido del fichero Makefile es el siguiente:

En primer lugar se declaran variables comunes al script, tales como referencias a las librerías externas que usa la aplicación, los nombres de los programas a usar, directorios o ficheros de salida, parámetros de configuración en tiempo de compilación y por último una cadena de texto que contiene la fecha y hora de ejecución del script.

```
CSC= mcs
EXE= mono
BIN= bin
ASS= -pkg:gtk-sharp-2.0 -r:System.Drawing.dll -pkg:gtk-dotnet-2.0
RES= -resource:gb.ico,gb_icon
DOC= csgbe.xml
OPTS= -win32icon:gb.ico -doc:${DOC}
FECHA=`date +%Y_%m_%d_%H.%M`
```

El objetivo por defecto consiste en limpiar la compilación anterior, compilar de nuevo toda la aplicación y ejecutarla.

```
all: clean compilar run

compilar:
    @echo Compiling...
    @${CSC} -unsafe -out:${BIN}/csgbe.exe ${ASS} ${RES} -
recurse:*.cs ${OPTS}

clean:
    @echo Cleaning...
    @rm -rf ${BIN}/*.exe
    @rm -rf ${DOC}

run:
    ${EXE} ${BIN}/csgbe.exe
```

Para crear ficheros comprimidos con su fecha que sirven para las copias de seguridad. Esto permite poder revertir cambios u hojear versiones antiguas en busca de código ya eliminado.

```
zip: clean
    @echo Backup [$(FECHA)]
    @cd .. && rm -f csgbe_$(FECHA).zip && zip -9 -r
csgbe_$(FECHA).zip csgbe
```

El backup consiste simplemente en una conexión SSH hacia un servidor predeterminado.

```
sync: zip
    @echo Connecting...
    @scp -P 6112 ../csgbe_$(FECHA).zip
victor@denibol.com:Proyectos/csgbe/
```

6.2. CODIFICACIÓN

En el presente apartado se comentarán secciones de código claves o críticas en los distintos componentes de todo el sistema, que también se suelen corresponder con los de complejidad elevada.

6.2.1.PROCESADO GRÁFICO

Uno de los puntos más complejos y críticos del emulador, después del procesador, es el procesado gráfico. Es el encargado de interpretar correctamente la zona de memoria gráfica o VRAM y convertirla en píxeles de una pantalla virtual también emulada.

El procesado gráfico se divide en dos fases, la interrupción horizontal o HBlank y la interrupción vertical o VBlank.

6.2.1.1.HBLANK

La interrupción horizontal o HBlank es la encargada de procesar una determinada línea de la pantalla. Para averiguar la línea se lee del registro especial LCD_Y_LOC que la gestión de interrupciones del procesador ya se encarga de modificar.

El procesado gráfico se descompone en tres capas, y por lo tanto también se han creado sendas funciones.

```
public void hblank(){  
    int scanLine = (_memoria.leer(LCD_Y_LOC) & 0xFF);  
  
    actualizarBG(scanLine);  
    actualizarVentana(scanLine);  
    actualizarSprites(scanLine);  
}
```

La primera capa es la del fondo. Comprueba si la línea que tiene que dibujar es visible y si además la capa está activada desde el registro especial LCD_CTRL. Obtiene las direcciones del mapa de tiles y los propios tiles que debe utilizar y los valores de desplazamiento horizontal y vertical sobre el fondo de 256x256.

A continuación recorre toda la línea obteniendo el tile a utilizar desde el mapa en coordenadas X e Y, y por cada uno obtiene los píxeles que se corresponden de ese tile sobre esa línea.

Tan solo falta dibujar el color de ese píxel en la pantalla usando la paleta adecuada.


```
private void actualizarBG(int scanLine){
    // Solo dibuja las lineas visibles (0-144)
    if (((_memoria.leer(LCD_CTRL) & 0x01) != 0)
    && scanLine < 144){
        // BG Tile Map Display Select
        int mapAddress = (_memoria.leer(LCD_CTRL)
        & 0x08) != 0 ? 0x9C00 : 0x9800;
        // BG & Window Tile Data Select
        int tileAddress = (_memoria.leer(LCD_CTRL)
        & 0x10) != 0 ? 0x8000 : 0x8800;
        int scrollX = _memoria.leer(LCD_SCROLL_X);
        int scrollY = _memoria.leer(LCD_SCROLL_Y);

        // La linea tiene 160 pixeles de ancho
        for(int x = 0; x < 160; x++){
            if ((scrollY + scanLine) > 255)
                scrollY -= 255;
            if ((scrollX + x) > 255)
                scrollX -= 255;

            // Tile
            int xTile = (scrollX + x) >> 3;
            int yTile = (scrollY + scanLine) >> 3;
            // Pixel dentro del tile
            int bitX = (scrollX + x) & 0x07;
            int bitY = (scrollY + scanLine) & 0x07;

            int idTile = getIdTile(
                xTile,
                yTile,
                mapAddress,
                tileAddress);
            int tile = getTile(
                idTile,
                tileAddress,
                bitX,
                bitY);
            _pantalla.dibujarPixel(
                x,
                scanLine,
                id2color(tile, LCD_BACK_PALETTE));
        }
    }
}
```

El procesado de la capa de ventana es análogo al del fondo con una única excepción. En lugar de utilizar las variables de desplazamiento horizontal y vertical por “scroll”, utiliza unos registros especiales que indican la esquina inferior izquierda en la que debe situarse la ventana.

```
private void actualizarVentana(int scanLine){
    int winY = _memoria.leer(LCD_WIN_Y);

    if ((_memoria.leer(LCD_CTRL) & 0x20) != 0)
        && winY <= scanLine){

        int winX = _memoria.leer(LCD_WIN_X) - 7;
        int mapAddress = (_memoria.leer(LCD_CTRL)
            & 0x40) != 0 ? 0x9C00 : 0x9800;
        int tileAddress = (_memoria.leer(LCD_CTRL)
            & 0x10) != 0 ? 0x8000 : 0x8800;

        for(int wx = 0; wx < (160 - winX); wx++){
            int xTile = wx >> 3;
            int yTile = (scanLine - winY) >> 3;

            int bitX = wx & 0x07;
            int bitY = (scanLine - winY) & 0x07;

            int idTile = getIdTile(
                xTile,
                yTile,
                mapAddress,
                tileAddress);
            int tile = getTile(
                idTile,
                tileAddress,
                bitX,
                bitY);
            if ((wx + winX) < 160
                && (wx + winX) >= 0)
                _pantalla.dibujarPixel(
                    wx + winX,
                    scanLine,
                    id2color(tile,
                        LCD_BACK_PALETTE)
                );
        }
    }
}
```

La capa con los sprites es la más compleja de las tres, por la cantidad de casos posibles que pueden darse. Aunque físicamente en la consola original es imposible que haya más de 10 sprites por línea de pantalla, en la emulación se contemplan los 40 posibles que puede haber en total en el dibujado de cada línea.

Por cada sprite hay una entrada en la tabla OAM que empieza en la dirección 0xFE00 y que ocupa 4 bytes. Los dos primeros especifican la posición de la esquina superior izquierda del sprite sobre la pantalla en coordenadas X e Y, siendo el origen de coordenadas la esquina superior izquierda de la pantalla. El tercer byte contiene el identificador de sprite que indica la dirección donde se encuentran los datos del sprite, y por último un byte para marcar una serie de atributos individuales.

Uno de los atributos más problemáticos es el de prioridad. Indica si el sprite tiene prioridad sobre la capa que contiene el fondo (funcionamiento por defecto), o si por el contrario se encuentra por detrás. En ese último caso el sprite debe dibujarse sólo por encima del color 0 de la paleta del fondo, que no tiene porqué corresponderse con el color negro, es decir, que debe comprobarse el color del fondo o ventana en ese píxel, y en caso de que no sea 0, dicho píxel no se dibujará porque estaría por detrás.

Únicamente se llega a dibujar un sprite si su capa está activa y si se encuentra visible en la pantalla, bajo cualquier circunstancia.

```
private void actualizarSprites(int scanLine){
    if ((_memoria.leer(LCD_CTRL) & 0x02) != 0){
        // Los sprites pueden ser de 8x8 o 8x16
        pixeles
        int spriteSize = (_memoria.leer(LCD_CTRL)
            & 0x04) != 0 ? 16 : 8;
        // El sprite 0 es el de menor prioridad
        for (int i = 39; i >= 0; i--){
            int spriteY = _memoria.leer(0xFE00 +
                (i * 4));
            int spriteX = _memoria.leer(0xFE01 +
                (i * 4));

            if((spriteY <= scanLine + 16)
                && (spriteY > scanLine +
                    (16 - spriteSize))){
                int tileNum = _memoria.leer(0xFE02
                    + (i * 4));
                int attributes = _memoria.leer(
                    0xFE03 + (i * 4));

                // Paleta a utilizar
                bool pal =
                    (attributes & 0x10) == 0x10;
                // Inversion horizontal
                bool flipX =
                    (attributes & 0x20) == 0x20;
                // Inversion vertical
                bool flipY =
                    (attributes & 0x40) == 0x40;
                // Prioridad sobre el fondo
                bool priority =
                    (attributes & 0x80) == 0x80;
```

```

// Todos los sprites tiene
// 8 pixeles de ancho
for (int j = 0; j < 8; j++){

    int posX = flipX ?
        spriteX - 1 - j :
        spriteX + j - 8;
    int posY = flipY ?
        spriteSize - (scanLine
        - spriteY + 17) : scanLine -
        spriteY + 16;
    int tile = getTile(
        tileNum,
        0x8000,
        j,
        posY);
    if (posX >= 0
        && tile != 0
        && (!priority ||
        (priority &&
        getIdColor(posX, scanLine)
        == 0)))
        _pantalla.dibujarPixel(
            posX,
            scanLine,
            id2color(tile, pal ?
            LCD_SPR1_PALETTE :
            LCD_SPR0_PALETTE));
}
}
}
}
}

```

Para obtener la dirección de los tiles, las funciones de procesado de las tres capas se basan en esta función auxiliar que convierte unas coordenadas en el identificador de tile, usado más adelante para localizar los datos del tile según esa posición.

Los mapas de tiles son virtualmente de 32x32, pero en memoria se encuentran almacenados linealmente, las filas son consecutivas. Por ello, para calcular la posición en el mapa se multiplica la fila por 32 sumado a la columna para conocerla.

Dependiendo de la dirección de memoria a la que pertenezca dicho identificador de tile, debe tratarse como un número con signo o sin signo. Los tiles situados en la dirección 0x8000 están numerados del -127 al 128.

```
private int getIdTile(int xTile, int yTile,
    int mapAddress, int tileAddress){
    int idTile = _memoria.leer(mapAddress
        + (yTile << 5) + xTile);
    if (tileAddress != 0x8000) idTile ^= 0x80;
    return idTile;
}
```

Una vez que se ha obtenido el identificador de tile, las funciones de procesado de las tres capas se apoyan en esta función para extraer los datos de cada tile.

Cada tile contiene exactamente 64 píxeles de datos, ordenados en 8 filas de 8 píxeles cada una. Cada píxel está compuesto por 2 bits, pero tal y como se ilustró en el apartado 2.3.2.5 Vídeo, ambos bits se encuentran en filas diferentes y por tanto no consecutivos.

```
private int getTile(int idTile, int tileAddress, int bitX,
    int bitY){
    // El color de un pixel esta compuesto por dos bits
    // (4 colores, blanco, negro y dos niveles de gris)
    int a = ((_memoria.leer(tileAddress + 1 + (bitY <<
1)
        + (idTile << 4)) >> (7 - bitX)) & 0x01) << 1;
    int b = (_memoria.leer(tileAddress + (bitY << 1)
        + (idTile << 4)) >> (7 - bitX)) & 0x01;
    return a | b;
}
```

6.2.1.2. VBLANK

La interrupción vertical o VBlank no tiene ninguna complicación conceptualmente, pero en ella radica uno de los pilares de la arquitectura del emulador. Debido a la decisión de utilizar las librerías GTK#, existen varios problemas al ser combinadas con el uso de varios hilos de ejecución, como es el caso de este sistema.

El emulador tiene dos hilos de ejecución explícitos: la interfaz gráfica en GTK# por un lado, y el proceso de emulación por otro. El nexo de unión entre ambos hilos se encuentra en esta función. Los desarrolladores de la librería GTK advierten que no es aconsejable realizar llamadas al hilo de ejecución que contiene la interfaz gráfica sin

algún mecanismo de sincronización, porque los resultados pueden ser impredecibles. Por fortuna, han proporcionado varios métodos y manuales que ayudan a resolver este problema. Tal y como comentan en http://www.monoproject.com/Responsive_Applications, hay varios mecanismos para afrontar esta situación, pero se escogió el siguiente por su simpleza y su claridad para ser entendido por alguien que no ha utilizado estas librerías nunca.

En el bucle principal de ejecución de la interfaz gráfica contenido en la llamada `Gtk.Application.Run()`, existe una cola de eventos pendientes por procesar. Mediante la función `Gtk.Application.Invoke()` se inserta en dicha cola una notificación para que ejecute una determinada función (sobre la que delega) cuando procese dicha cola en el flujo normal de ejecución de ese hilo. De esta forma, aunque no se puede asegurar ni controlar el momento de ejecución de dicha solicitud, se consigue que la aplicación sea más segura y estable con la mínima complejidad delegando la gestión en las propias librerías gráficas.

```
public void vblank(){
    Gtk.Application.Invoke(
        delegate {
            _pantalla.actualizarPantalla();
        }
    );
    _frames++;
}
```

6.2.2.GESTIÓN DE INTERRUPCIONES

Dentro del módulo del procesador, la parte más compleja es la emulación de las interrupciones hardware. Es necesario emular el temporizador, el comportamiento del registro DIV, la interrupción LCDC o de coincidencia de línea y la interrupción vertical VBlank.

Cada interrupción tiene una frecuencia de ocurrencia predeterminada, o configurable en el caso del temporizador, por lo que, aunque tiene que comprobarse su estado tras la ejecución de cada instrucción de procesador, no todas son emuladas en cada momento. Para conseguir la mayor precisión, se almacena el número de ciclos transcurridos desde la última interrupción para cada una de ellas, y dado que se conoce la duración exacta en ciclos de cada instrucción ejecutada, se puede mantener y

controlar en todo momento cual debería ser el momento exacto en el que se debe emular cada interrupción.

El caso de la interrupción Timer o temporizador es un poco especial, ya que su frecuencia no es constante y puede variar entre cuatro posibles valores. El registro especial de entrada / salida `TIMER_CRTL`, indica si el temporizador está activo y cual es su frecuencia en ese momento, es decir, cuantos ciclos de procesador deben transcurrir cada vez para activarse. En cada activación, el contador del temporizador es incrementado en uno, y cuando desborda o alcanza el valor 255, se activa la interrupción para que la parte adecuada del sistema sepa que debe invocar a la rutina de tratamiento de la interrupción, y se vuelve a reiniciar dicho contador.

```
private void dispararInterrupciones(){
    // Interrupcion TIMER
    if ((_memoria.leer(TIMER_CRTL) & 0x04) != 0){
        int timer_max = 0;
        // Velocidad del temporizador
        switch(_memoria.leer(TIMER_CRTL) & 0x03){
            case 0: timer_max = CYCLES_TIMER_MODE0;
                    break;
            case 1: timer_max = CYCLES_TIMER_MODE1;
                    break;
            case 2: timer_max = CYCLES_TIMER_MODE2;
                    break;
            case 3: timer_max = CYCLES_TIMER_MODE3;
                    break;
        }
        if (_ciclos[1] > timer_max){
            _ciclos[1] = 0;
            _memoria.escribir(
                _memoria.leer(TIMER_COUNT) + 1,
                TIMER_COUNT);
            // Si desborda se activa la
            // interrupcion y se reinicia
            // el contador
            if (_memoria.leer(TIMER_COUNT) == 0xFF){
                _memoria.escribir(
                    _memoria.leer(TIMER_RELOAD),
                    TIMER_COUNT);
                _memoria.escribir(
                    _memoria.leer(INT_FLAG) |
                    INT_TIMER, INT_FLAG);
            }
        }
    }
}
```

El registro `DIV` no es una interrupción propiamente dicha, pero su comportamiento es muy similar, por lo que se introduce su emulación junto con el resto. Es parecido al temporizador pero con una única frecuencia y cuando desborda no

provoca ninguna reacción en el sistema y debe ser el propio juego o software el que consulte su valor.

```
// Registro DIV
if (_ciclos[0] > CYCLES_DIV){
    _memoria.escribir(
        _memoria.leer(DIV_CNTR) + 1, DIV_CNTR);
    _ciclos[0] = 0;
}
```

La interrupción LCDC, abreviatura de “Line CoinciDenCe”, se activa cuando la línea actual que se va a dibujar en la pantalla coincide con la línea especificada en el registro LCD_Y_COMP. En cada activación se aumenta en uno la línea actual en el registro especial LCD_Y_LOC, posteriormente usado en el procesado gráfico, y adicionalmente, si la línea es la misma que la que el software ha configurado, se activa la interrupción.

```
// Interrupcion LCDC
if (_ciclos[2] > CYCLES_LCD_MODEL){
    _ciclos[2] = 0;
    // Aumento de linea de dibujo
    if (_memoria.leer(LCD_Y_LOC) == 0x99)
        _memoria.escribir(0, LCD_Y_LOC);
    else _memoria.escribir(
        _memoria.leer(LCD_Y_LOC) + 1, LCD_Y_LOC);

    // Comparacion de linea
    if (_memoria.leer(LCD_Y_LOC) ==
        _memoria.leer(LCD_Y_COMP)){
        _memoria.escribir(
            _memoria.leer(LCD_STAT) | 0x04,
            LCD_STAT);
        if ((_memoria.leer(LCD_STAT) & 0x40) > 0)
            _memoria.escribir(
                _memoria.leer(INT_FLAG) |
                INT_LCDC, INT_FLAG);
    } else _memoria.escribir(
        _memoria.leer(LCD_STAT) & 0xFB,
        LCD_STAT);
}
```

La interrupción vertical o VBlank se produce sólo bajo ciertas circunstancias especiales. Basándose en el diagrama de estados del hardware de vídeo detallado en el apartado “2.3.2.5 Video”, desde aquí se emula dicho comportamiento. Mientras que los estados 2 y 0 son puramente transitorios ya que no es necesario emular nada adicional al

acceder a la HRAM y VRAM respectivamente, durante el estado 3 se invoca el procesado gráfico de la línea actual, y durante el estado 1 se invoca el refresco total de la pantalla.

En cada uno de los estados se realizan los cambios necesarios para transitar al siguiente estado, y también activan o desactivan las interrupciones VBLANK y LCDC en consecuencia.

```

        if (_memoria.leer(LCD_Y_LOC) < 144){
            // Modo 10 (Cuando se esta accediendo
            // entre 0xFE00 y 0xFE9F)
            if (_ciclos[2] < CYCLES_LCD_MODE2 &&
                (_memoria.leer(LCD_STAT) & 0x03) != 0x02){
                _memoria.escribir(
                    (_memoria.leer(LCD_STAT) & 0xFC) |
                    0x02, LCD_STAT);
                if ((_memoria.leer(LCD_STAT) & 0x20) > 0)
                    _memoria.escribir(
                        _memoria.leer(INT_FLAG) | INT_LCDC,
                        INT_FLAG);
            }
            // Modo 11
        }else if(_ciclos[2] >= CYCLES_LCD_MODE2 &&
            _ciclos[2] < CYCLES_LCD_MODE3 &&
            (_memoria.leer(LCD_STAT) & 0x03) != 0x03){
            // Se dibujan las primeras 144 lineas
            // cuando se ha dejado de escribir en
            // la zona grafica de memoria
            _graphics.hblank();
            _memoria.escribir(
                (_memoria.leer(LCD_STAT) & 0xFC) |
                0x03, LCD_STAT);
            // Modo 00 (Durante el HBLANK, la CPU
            // puede acceder a la display RAM entre
            // 0x8000 y 0x9FFF)
        }else if(_ciclos[2] >= CYCLES_LCD_MODE3 &&
            (_memoria.leer(LCD_STAT) & 0x03) != 0){
            _memoria.escribir(
                _memoria.leer(LCD_STAT) & 0xFC,
                LCD_STAT);
            if ((_memoria.leer(LCD_STAT) & 0x08) > 0)
                _memoria.escribir(
                    _memoria.leer(INT_FLAG) |
                    INT_LCDC, INT_FLAG);
        }
        // Modo 01 (Periodo VBLANK, la CPU puede
        // acceder a la display RAM
        // entre 0x8000 y 0x9FFF)
    }else if ((_memoria.leer(LCD_Y_LOC) >= 144) &&
        (_memoria.leer(LCD_STAT) & 0x03) != 0x01){
        // Refresco vertical
        _graphics.vblank();
        _memoria.escribir(
            (_memoria.leer(LCD_STAT) & 0xFC) | 0x01,
            LCD_STAT);
        if ((_memoria.leer(LCD_STAT) & 0x10) > 0)
            _memoria.escribir(
                _memoria.leer(INT_FLAG) | INT_LCDC,
                INT_FLAG);
        _memoria.escribir(
            _memoria.leer(INT_FLAG) | INT_VBLANK,
            INT_FLAG);
    }
}

```

6.2.3.RESUMEN

La implementación del proyecto ha concluido con los siguientes números:

Número de clases: 115 (en 31 ficheros)

Número de líneas de código: 6998

Tamaño código: 380 KB

Tamaño documentación: 150 KB (XML)

7. CONCLUSIONES

Tras la realización del proyecto, se considera que se han conseguido los objetivos propuestos en su comienzo, consiguiendo una aplicación que ofrece:

- Emulación de una arquitectura hardware funcional
 - Se ha implementado un procesador de 8 bits basado en el Z80 con 5 registros de 8 bits y 3 registros de 16 bits.
 - Emulación del juego completo de 499 instrucciones.
 - Cuenta con un espacio de memoria de 2^{16} direcciones.
 - Dispone de registros de entrada / salida compartiendo espacio de direcciones con la memoria.
- Emulación de la consola portátil Game Boy
 - Carga de cartuchos de juegos comerciales.
 - Pantalla con cuatro niveles de gris.
 - Botones
- Control de la emulación
 - Control sobre el flujo de ejecución.
 - Escalado de la imagen.
 - Visualización de la emulación del sistema
 - Depuración y manipulación del estado del hardware emulado

En resumen, se ha conseguido obtener una aplicación funcional, que aunque dista mucho de la calidad de los emuladores existentes, sirve para comprender el funcionamiento de una arquitectura hardware sencilla bajo una aplicación comercial de la misma.

7.1. LÍNEAS FUTURAS

El proyecto, a pesar de mostrar resultados funcionales, aún se puede ampliar más, tanto en el aspecto de la optimización, como en el de la compatibilidad y añadiendo nuevas funcionalidades. Por ejemplo:

- **Añadir soporte a más tipos de cartuchos.** Existieron muchos tipos de cartuchos donde se alojaban los juegos comerciales, y en este proyecto sólo se han implementado los más comunes y que se usaron para la mayoría de ellos, pero se puede ampliar la compatibilidad soportando más versiones.
- **Sonido.** Soporte para el sonido estéreo que ofrecía la consola original. Es necesario implementar un par de generadores de onda cuadrada y senoide e interpretar correctamente los registros de salida adecuados. La sincronización es vital para el sonido.
- **Optimización de la emulación:** Debido al uso de la técnica de interpretación para conseguir la emulación, el proceso de emulación resultante resulta bastante lento y pesado para equipos poco potentes. Se puede optimizar en gran medida aplicando algunos cambios a la técnica actual como una caché de instrucciones descodificadas o utilizando otra librería de dibujo gráfico que sería fácilmente sustituible.
- **Soporte para la consola GameBoy Color:** Aunque se pueda pensar lo contrario, la revisión de esta consola años más tarde para dotarla de más potencia y de una pantalla de colores no dista mucho del diseño original sobre el que se ha basado este proyecto. A pesar de ello sería necesario realizar pequeñas modificaciones en prácticamente todos los módulos de este emulador, pero conservando el diseño arquitectónico y tan solo ampliando funcionalidad reutilizando todo el desarrollo ya existente.
- **Soporte para otros periféricos:** Gran parte del éxito que tuvo la consola en su época fue también debido a la gran cantidad de periféricos que se crearon para ella. Sería posible ampliar la funcionalidad del emulador añadiendo soporte para algunos de ellos comentado en el apartado de “Estado del Arte” como el accesorio de Super GameBoy o el cable serie para la comunicación entre varias consolas.

8. PRESUPUESTO

A la hora de elaborar el presupuesto del trabajo a realizar, se identificará cada una de las fases en las que está dividido el proyecto, de forma que se puedan analizar cada una de ellas de forma independiente para obtener el total.

Para la realización del proyecto se aplicará un ciclo de vida clásico en el que se distinguirán las fases de análisis, diseño, implementación, pruebas y documentación.

Para cada fase se tendrá en consideración dos factores, por una parte los recursos humanos y por otra los recursos materiales. Estos últimos incluyen el material fungible y los equipos informáticos empleados en la elaboración del proyecto.

RECURSOS HUMANOS

La siguiente tabla muestra el coste total de los recursos humanos empleados en la elaboración del proyecto. En cada una de las semanas empleadas se considerará que se ha empleado una media de cuatro horas al día durante seis días a la semana a la elaboración del proyecto, por lo que contabilizará como 24 horas de trabajo. Cada una de las fases, además, repercutirá en un coste por hora distinto, el cual dependerá del perfil mínimo necesario para la elaboración de cada trabajo.

Fase	Precio/Hora (€)	Semanas	Horas	Precio Fase (€)
Análisis	45	5	120	5.400 €
Diseño	45	2	48	2.160 €
Implementación	35	15	360	12.600 €
Pruebas	35	4	96	3.360 €
Documentación	40	4	96	3.840 €
Total	27.360 €			

Tabla 1: Recursos Humanos

RECURSOS MATERIALES

La elaboración del proyecto ha requerido de multitud de recursos materiales, tanto vienen físicos, como material fungible o servicios. La siguiente tabla muestra la relación recursos materiales empleados durante la realización del proyecto.

Elemento	Cantidad	Precio	Total
Ordenador portátil	1	900 €	900 €
Conexión de a Internet de banda ancha (cantidad en meses)	8	40 €	320 €
Cartuchos de tinta para impresora	3	25 €	75 €
Impresión encuadernada con tapas duras del proyecto	3	40 €	120 €
Impresión encuadernada en espirar del proyecto	4	30 €	120 €
Material de oficina			20 €
Total		1.555 €	

Tabla 2: Recursos Materiales

Es complicado asignar los recursos materiales a las fases del proyecto, por lo que se considerará que han sido empleados de forma uniforme durante la duración del proyecto.

COSTE TOTAL DEL PROYECTO

Teniendo en cuenta los datos anteriores, el coste total del proyecto se obtendrá de la suma del coste total de los recursos humanos y el coste total de los recursos materiales.

Por lo que el coste total del proyecto será de **28.915 €** IVA no incluido.

9. BIBLIOGRAFÍA

- **Programación de un emulador de la consola de videojuegos Super Nintendo sobre un PC**

Carlos Alberto Lozano Baidés

Proyecto de fin de carrera de Ingeniería técnica en informática de gestión, Julio de 1997

- **Z80 Assembly Language Programming Manual**

ZiLOG, Abril de 1980

- **El lenguaje de programación C#**

José Antonio González Seco

<http://www.josanguapo.com>, 2001

- **Sistemas Operativos: Una visión aplicada**

Jesús Carretero, Félix García, Pedro de Miguel y Fernando Pérez

McGraw-Hill, 2001

9.1. REFERENCIAS

- **Everything You Always Wanted To Know About GAMEBOY**

<http://www.devrs.com/gb/files/gbspec.txt>

- **Using the Gameboy skeleton for serious business**

<http://verhoeven272.nl/fruttenboel/Gameboy/GBmain.html>

- **Nintendo Company FAQs: Emulators, ROMs and Intellectual Property**

<http://www.nintendo.com/corp/faqs/legal.html>

- **Magazine ZX: Instrucción a la emulación**

<http://www.speccy.org/magazinezx/index.php?revista=2&seccion=6>

- **Wikipedia: Zilog Z80**

http://en.wikipedia.org/wiki/Zilog_Z80

- **Otaku No Gameboy Crib Sheet**

<http://otakunozoku.com/Articles/Development/GBCribsheet/GBCribsheet.html>

- **Z80 Official Support Page**

<http://www.z80.info/>

- **U.S. Patent: Software implementation of a handheld video game hardware platform (Nintendo Co., Ltd.)**

<http://www.google.com/patents?vid=USPAT6672963&id=HrYCAAAAEB AJ>

- **Proyecto Mono**

<http://www.mono-project.com>

- **GTK# Bindings**

<http://gtk-sharp.sourceforge.net/>

9.2. ENLACES

1. NESTicle: <http://bloodlust.zophar.net/NESticle/nes.html>
2. Nestopia: <http://nestopia.sourceforge.net/>
3. MasterGear: <http://fms.komkon.org/MG/>
4. MEKA: <http://www.smspower.org/meka/>
5. Gnuboy: <http://en.wikipedia.org/wiki/Gnuboy>
6. VisualBoyAdvance: <http://vba.ngemu.com/>
7. KiGB: <http://kigb.emuunlim.com/>
8. Handy: <http://homepage.ntlworld.com/dystopia/>

A. INSTRUCCIONES DE ENSAMBLADOR

A continuación se listan todas las instrucciones de ensamblador usadas por la consola detalladas. Para cada una se muestra su opCode en hexadecimal, su mnemónico, los flags que modifica, su tamaño en bytes y su duración total en ticks de reloj, que es diferente al número de ciclos.

Para los flags se muestra el identificador del flag afectado, que según la instrucción lo hará de una manera u otra, una asignación a 0 indica que lo desactiva incondicionalmente y una asignación a 1 que lo activa incondicionalmente. Si un flag no aparece significa que la ejecución de esa instrucción no lo modifica y mantiene su estado anterior.

- **Instrucciones de carga de 8 bits**

OpCode	Mnemónico	Flags	Tamaño	Duración
0x7F	LD a, a	Ninguno	1	4
0x78	LD a, b			
0x79	LD a, c			
0x7A	LD a, d			
0x7B	LD a, e			
0x7C	LD a, h			
0x7D	LD a, l			
0x47	LD b, a	Ninguno	1	4
0x40	LD b, b			
0x41	LD b, c			
0x42	LD b, d			
0x43	LD b, e			
0x44	LD b, h			
0x45	LD b, l			
0x4F	LD c, a	Ninguno	1	4
0x48	LD c, b			

OpCode	Mnemónico	Flags	Tamaño	Duración
0x49	LD c, c			
0x4A	LD c, d			
0x4B	LD c, e			
0x4C	LD c, h			
0x4D	LD c, l			
0x57	LD d, a	Ninguno	1	4
0x50	LD d, b			
0x51	LD d, c			
0x52	LD d, d			
0x53	LD d, e			
0x54	LD d, h			
0x55	LD d, l			
0x5F	LD e, a	Ninguno	1	4
0x58	LD e, b			
0x59	LD e, c			
0x5A	LD e, d			
0x5B	LD e, e			
0x5C	LD e, h			
0x5D	LD e, l			
0x67	LD h, a	Ninguno	1	4
0x60	LD h, b			
0x61	LD h, c			
0x62	LD h, d			
0x63	LD h, e			
0x64	LD h, h			
0x65	LD h, l			
0x6F	LD l, a	Ninguno	1	4
0x68	LD l, b			

OpCode	Mnemónico	Flags	Tamaño	Duración
0x69	LD l, c			
0x6A	LD l, d			
0x6B	LD l, e			
0x6C	LD l, h			
0x6D	LD l, l			
0x3E	LD a, n	Ninguno	2	7
0x06	LD b, n			
0x0E	LD c, n			
0x16	LD d, n			
0x1E	LD e, n			
0x26	LD h, n			
0x2E	LD l, n			
0x7E	LD a, (hl)	Ninguno	1	7
0x46	LD b, (hl)			
0x4E	LD c, (hl)			
0x56	LD d, (hl)			
0x5E	LD e, (hl)			
0x66	LD h, (hl)			
0x6E	LD l, (hl)			
0x77	LD (hl), a	Ninguno	1	7
0x70	LD (hl), b			
0x71	LD (hl), c			
0x72	LD (hl), d			
0x73	LD (hl), e			
0x74	LD (hl), h			
0x75	LD (hl), l			
0x36	LD (hl), n	Ninguno	2	10
0x0A	LD a, (bc)	Ninguno	1	7

OpCode	Mnemónico	Flags	Tamaño	Duración
0x1A	LD a, (de)			
0x02	LD (bc), a	Ninguno	1	7
0x12	LD (de), a			
0xFA	LD a, (nn)	Ninguno	3	13
0xEA	LD (nn), a	Ninguno	3	13
0xF0	LD a, (0xFF00 + nn)	Ninguno	2	15
0xF2	LD a, (0xFF00 + c)	Ninguno	1	15
0xE0	LD (0xFF00 + nn), a	Ninguno	2	15
0xE2	LD (0xFF00 + c), a	Ninguno	1	15
0x22	LDI (hl), a	Ninguno	1	15
0x32	LDD (hl), a	Ninguno	1	15
0x2A	LDI a, (hl)	Ninguno	1	15
0x3A	LDD a, (hl)	Ninguno	1	15

• Instrucciones de carga de 16 bits

OpCode	Mnemónico	Flags	Tamaño	Duración
0x01	LD bc, nn	Ninguno	3	10
0x11	LD de, nn			
0x21	LD hl, nn			
0x31	LD sp, nn			
0xF9	LD sp, hl	Ninguno	1	6
0x08	LD (nn), sp	Ninguno	3	13
0xF8	LD hl, (sp+nn)	C	2	10
0xF5	PUSH af	Ninguno	1	11
0xC5	PUSH bc			
0xD5	PUSH de			
0xE5	PUSH hl			
0xF1	POP af	Ninguno	1	10

OpCode	Mnemónico	Flags	Tamaño	Duración
0xC1	POP bc			
0xD1	POP de			
0xE1	POP hl			

• Instrucciones aritméticas y lógicas de 8 bits

OpCode	Mnemónico	Flags	Tamaño	Duración
0x87	ADD a, a	Z, H, C, N = 0	1	4
0x80	ADD a, b			
0x81	ADD a, c			
0x82	ADD a, d			
0x83	ADD a, e			
0x84	ADD a, h			
0x85	ADD a, l			
0xC6	ADD a, n	Z, H, C, N = 0	2	7
0x86	ADD a, (hl)	Z, H, C, N = 0	1	7
0x8F	ADC a, a	Z, H, C, N = 0	1	4
0x88	ADC a, b			
0x89	ADC a, c			
0x8A	ADC a, d			
0x8B	ADC a, e			
0x8C	ADC a, h			
0x8D	ADC a, l			
0xCE	ADC a, n	Z, H, C, N = 0	2	7
0x8E	ADC a, (hl)	Z, H, C, N = 0	1	7
0x97	SUB a	Z, H, C, N = 1	1	4
0x90	SUB b			
0x91	SUB c			
0x92	SUB d			

OpCode	Mnemónico	Flags	Tamaño	Duración
0x93	SUB e			
0x94	SUB h			
0x95	SUB l			
0xD6	SUB n	Z, H, C, N = 1	2	7
0x96	SUB (hl)	Z, H, C, N = 1	1	7
0x9F	SBC a, a	Z, H, C, N = 1	1	4
0x98	SBC a, b			
0x99	SBC a, c			
0x9A	SBC a, d			
0x9B	SBC a, e			
0x9C	SBC a, h			
0x9D	SBC a, l			
0xDE	SBC a, n	Z, H, C, N = 1	2	7
0x9E	SBC a, (hl)	Z, H, C, N = 1	1	7
0xA7	AND a	Z, H = 1, N = 0, C = 0	1	4
0xA0	AND b			
0xA1	AND c			
0xA2	AND d			
0xA3	AND e			
0xA4	AND h			
0xA5	AND l			
0xE6	AND n	Z, H = 1, N = 0, C = 0	2	7
0xA6	AND (hl)	Z, H = 1, N = 0, C = 0	1	7
0xB7	OR a	Z, H = 0, N = 0, C = 0	1	4
0xB0	OR b			
0xB1	OR c			
0xB2	OR d			
0xB3	OR e			

OpCode	Mnemónico	Flags	Tamaño	Duración
0xB4	OR h			
0xB5	OR l			
0xF6	OR n	Z, H = 0, N = 0, C = 0	2	7
0xB6	OR (hl)	Z, H = 0, N = 0, C = 0	1	7
0xAF	XOR a	Z, H = 0, N = 0, C = 0	1	4
0xA8	XOR b			
0xA9	XOR c			
0xAA	XOR d			
0xAB	XOR e			
0xAC	XOR h			
0xAD	XOR l			
0xEE	XOR n	Z, H = 0, N = 0, C = 0	2	7
0xAE	XOR (hl)	Z, H = 0, N = 0, C = 0	1	7
0xBF	CP a	Z, H, C, N = 1	1	4
0xB8	CP b			
0xB9	CP c			
0xBA	CP d			
0xBB	CP e			
0xBC	CP h			
0xBD	CP l			
0xFE	CP n	Z, H, C, N = 1	2	7
0xBE	CP (hl)	Z, H, C, N = 1	1	7
0x3C	INC a	Z, H, N = 0	1	4
0x04	INC b			
0x0C	INC c			
0x14	INC d			
0x1C	INC e			
0x24	INC h			

OpCode	Mnemónico	Flags	Tamaño	Duración
0x2C	INC l			
0x34	INC (hl)	Z, H, N = 0	1	11
0x3D	DEC a	Z, H, N = 1	1	4
0x05	DEC b			
0x0D	DEC c			
0x15	DEC d			
0x1D	DEC e			
0x25	DEC h			
0x2D	DEC l			
0x35	DEC (hl)	Z, H, N = 1	1	11

• **Propósito general aritmético y control de la CPU**

OpCode	Mnemónico	Flags	Tamaño	Duración
0x27	DAA	Z, H, C	1	4
0x2F	CPL	H = 1, N = 1	1	4
0x3F	CCF	H, C, N = 0	1	4
0x37	SCF	H = 0, N = 0, C = 1	1	4
0x00	NOP	Ninguno	1	4
0x76	HALT	Ninguno	1	4
0xF3	DI	Ninguno	1	4
0xFB	EI	Ninguno	1	4
0x10	STOP	Ninguno	2	4

• **Instrucciones aritméticas de 16 bits**

OpCode	Mnemónico	Flags	Tamaño	Duración
0x09	ADD hl, bc	H, C, N = 0	1	11
0x19	ADD hl, de			

OpCode	Mnemónico	Flags	Tamaño	Duración
0x29	ADD hl, hl			
0x39	ADD hl, sp			
0xE8	ADD sp, n	H, C, N = 0	2	7
0x03	INC bc	Ninguno	1	6
0x13	INC de			
0x23	INC hl			
0x33	INC sp			
0x0B	DEC bc	Ninguno	1	6
0x1B	DEC de			
0x2B	DEC hl			
0x3B	DEC sp			

• Instrucciones de rotación y desplazamiento

OpCode	Mnemónico	Flags	Tamaño	Duración
0x07	RLCA	C, H = 0, N = 0	1	4
0x17	RLA	C, H = 0, N = 0	1	4
0x0F	RRCA	C, H = 0, N = 0	1	4
0x1F	RRA	C, H = 0, N = 0	1	4
0xCB 0x07	RLC a	Z, C, H = 0, N = 0	2	8
0xCB 0x00	RLC b			
0xCB 0x01	RLC c			
0xCB 0x02	RLC d			
0xCB 0x03	RLC e			
0xCB 0x04	RLC h			
0xCB 0x05	RLC l			
0xCB 0x06	RLC (hl)	Z, C, H = 0, N = 0	2	15
0xCB 0x17	RL a	Z, C, H = 0, N = 0	2	8
0xCB 0x10	RL b			

OpCode	Mnemónico	Flags	Tamaño	Duración
0xCB 0x11	RL c			
0xCB 0x12	RL d			
0xCB 0x13	RL e			
0xCB 0x14	RL h			
0xCB 0x15	RL l			
0xCB 0x16	RL (hl)	Z, C, H = 0, N = 0	2	15
0xCB 0x0F	RRC a	Z, C, H = 0, N = 0	2	8
0xCB 0x08	RRC b			
0xCB 0x09	RRC c			
0xCB 0x0A	RRC d			
0xCB 0x0B	RRC e			
0xCB 0x0C	RRC h			
0xCB 0x0D	RRC l			
0xCB 0x0E	RRC (hl)	Z, C, H = 0, N = 0	2	15
0xCB 0x0F	RR a	Z, C, H = 0, N = 0	2	8
0xCB 0x08	RR b			
0xCB 0x09	RR c			
0xCB 0x0A	RR d			
0xCB 0x0B	RR e			
0xCB 0x0C	RR h			
0xCB 0x0D	RR l			
0xCB 0x0E	RR (hl)	Z, C, H = 0, N = 0	2	15
0xCB 0x27	SLA a	Z, C, H = 0, N = 0	2	8
0xCB 0x20	SLA b			
0xCB 0x21	SLA c			
0xCB 0x22	SLA d			
0xCB 0x23	SLA e			
0xCB 0x24	SLA h			

OpCode	Mnemónico	Flags	Tamaño	Duración
0xCB 0x25	SLA l			
0xCB 0x26	SLA (hl)	Z, C, H = 0, N = 0	2	15
0xCB 0x2F	SRA a	Z, C, H = 0, N = 0	2	8
0xCB 0x28	SRA b			
0xCB 0x29	SRA c			
0xCB 0x2A	SRA d			
0xCB 0x2B	SRA e			
0xCB 0x2C	SRA h			
0xCB 0x2D	SRA l			
0xCB 0x2E	SRA (hl)	Z, C, H = 0, N = 0	2	15
0xCB 0x3F	SRL a	Z, C, H = 0, N = 0	2	8
0xCB 0x38	SRL b			
0xCB 0x39	SRL c			
0xCB 0x3A	SRL d			
0xCB 0x3B	SRL e			
0xCB 0x3C	SRL h			
0xCB 0x3D	SRL l			
0xCB 0x3E	SRL (hl)	Z, C, H = 0, N = 0	2	15

• Instrucciones salto

OpCode	Mnemónico	Flags	Tamaño	Duración
0xC3	JP nn	Ninguno	3	10
0xCA	JP z, nn	Ninguno	3	10
0xC2	JP NZ, nn			
0xDA	JP C, nn			
0xD2	JP NC, nn			
0x18	JR e	Ninguno	2	12
0x38	JR C, e	Ninguno	2	7-12

OpCode	Mnemónico	Flags	Tamaño	Duración
0x30	JR NC, e			
0x28	JR Z, e			
0x20	JR NZ, e			
0xE9	JP (hl)	Ninguno	1	4

• Instrucciones de llamada y retorno

OpCode	Mnemónico	Flags	Tamaño	Duración
0xCD	CALL nn	Ninguno	3	17
0xCC	CALL Z, nn	Ninguno	3	10-17
0xC4	CALL NZ, nn			
0xDC	CALL C, nn			
0xD4	CALL NC, nn			
0xC9	RET	Ninguno	1	10
0xC8	RET Z	Ninguno	1	5-11
0xC0	RET NZ			
0xD8	RET C			
0xD0	RET NC			
0xD9	RETI	Ninguno	1	10
0xC7	RST 0x00	Ninguno	1	11
0xCF	RST 0x08			
0xD7	RST 0x10			
0xDF	RST 0x18			
0xE7	RST 0x20			
0xEF	RST 0x28			
0xF7	RST 0x30			
0xFF	RST 0x38			

- **Instrucciones bit, set y test**

OpCode	Mnemónico	Flags	Tamaño	Duración
0xCB 0x40 – 0x7F	BIT b, r	Z, H = 1, N = 0	2	8
	BIT b, (hl)	Z, H = 1, N = 0	2	12
0xCB 0xC0 – 0xFF	SET b, r	Ninguno	2	8
	SET b, (hl)	Ninguno	2	15
0xCB 0x80 – 0xBF	RES b, r	Ninguno	2	8
	RES b, (hl)	Ninguno	2	15
0xCB 0x30 – 0x37	SWAP r	Z, H = 0, N = 0	2	8
0xCB 0x36	SWAP (hl)	Z, H = 0, N = 0	2	16

B. MANUAL DE USUARIO

El manual de usuario describe las funcionalidades del sistema a las que puede acceder el usuario, así como una breve descripción de cómo poder realizar cada una de ellas. Se hará uso de capturas de pantalla para facilitar la explicación.

APARIENCIA GENERAL DEL SISTEMA

Al arrancar la aplicación se muestra la principal ventana de ejecución, con los menús de control y la pantalla emulada, que por defecto se encuentra vacía.

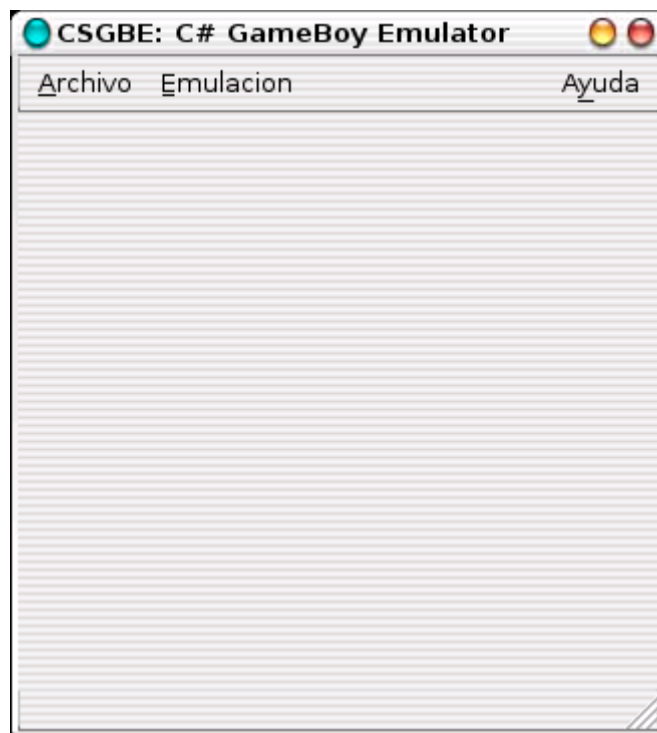


Imagen 38: Pantalla principal de la aplicación

Una vez que se ha cargado un fichero de ROM y se inicia la emulación, se mostrará en el centro de la ventana la pantalla emulada, y en la parte inferior se muestra el nombre interno de la aplicación que se está ejecutando.



Imagen 39: Pantalla principal de la aplicación en funcionamiento

FUNCIONALIDADES DEL SISTEMA

- Cargar un fichero o salir de la aplicación:

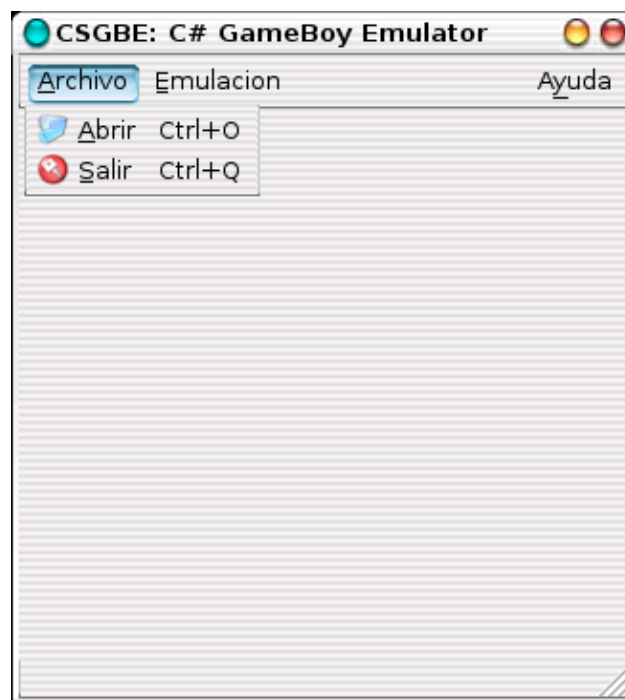
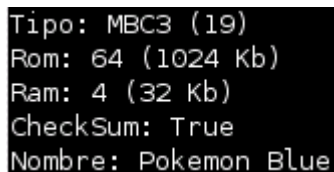


Imagen 40: Abrir un fichero o salir

Seleccionando la opción “Abrir” o usando la combinación rápida de teclas “Control + O”, se mostrará un cuadro de diálogo donde se podrá explorar el sistema de archivos local en busca de ficheros de ROM. Los ficheros que contengan juegos de GameBoy, independientemente del tipo de cartucho que contengan, deben tener la extensión “gb” para que la aplicación los reconozca.

Cada vez que se carga un fichero se muestra por el terminal desde donde se inició el emulador, un mensaje de error descriptivo si algo falló, o información acerca del cartucho:



```
Tipo: MBC3 (19)
Rom: 64 (1024 Kb)
Ram: 4 (32 Kb)
Checksum: True
Nombre: Pokemon Blue
```

Imagen 41: Información sobre un cartucho válido

Se muestra el tipo de cartucho y su código identificativo entre paréntesis, el número de bloques de ROM y RAM así como su tamaño en kilobytes, el resultado de la comprobación de integridad o checksum de los datos del fichero y el nombre interno del cartucho.

Durante el proceso de emulación se puede abrir otro fichero en cualquier momento sin necesidad de salir de la aplicación, lo que provocará que el hardware emulado se reinicie y cargue el nuevo fichero como si fuera la primera vez.

La opción “Salir” o con la combinación de teclas “Control + Q”, cierra la aplicación y aborta abruptamente cualquier proceso de emulación que hubiera en ese momento.

- Control de la emulación



Imagen 42: Opciones de control de la emulación

Una vez se ha cargado con éxito un fichero que contenga una ROM válida de GameBoy, las opciones del submenú “Emulación” pasan a estar disponibles.

- **Zoom:** Permite cambiar el tamaño de la pantalla emulada. 1x es la resolución original de la GameBoy de 160x144 píxeles, 2x sería el doble de esa resolución y así sucesivamente. El tamaño por defecto es 2x. Es importante saber que cuanto mayor sea el zoom más lento irá el emulador.
- **Iniciar:** Inicia el proceso de emulación de un fichero recién cargado, o si ya se había iniciado, realiza un reinicio desde 0 del mismo cartucho.
- **Reanudar:** Permite reanudar la ejecución en el mismo sitio después de haber sido pausada. Solo se activa esta opción si se ha realizado la pausa previamente.
- **Pausa:** Pausa temporalmente el proceso de emulación. Mientras dure la pausa, el emulador entra en modo de espera y no consumirá tiempo de procesador

de la máquina donde se está ejecutando. Tampoco se refrescará la pantalla emulada.

- **Parar:** Detiene la ejecución permanentemente sin que se pueda reanudar en el mismo punto. Sólo mediante la opción “Iniciar” se puede iniciar de nuevo.
- **Debug:** Inicia el modo de depuración de la consola. Cuando se selecciona, la ejecución se detiene y desde el terminal donde se inició el emulador se mostrarán las opciones de depuración.

- Consola de depuración

```

Iniciando Consola de depuracion
Opciones:
?           Mostrar esta ayuda
r           Mostrar registros y flags
i           Mostrar interrupciones
m dir long  Muestra 'long' bytes de memoria desde 'dir'
e len       Ejecuta 'len' instrucciones a partir del PC actual
x           Reinicia la consola
a reg valor Asigna 'valor' al registro 'reg'
f flag      Cambia el estado del flag 'flag' (Z, N, H, C)
w dir valor Escribe 'valor' en la direccion 'dir' de memoria
n int l|0   Activa o desactiva la interrupcion 'int' (vblank, lcdc, timer, serialtx, key)
k key       Cambia el estado de la tecla 'key' (down, up, left, right, start, select, a, b)
b dir       Inserta o elimina un punto de interrupcion en la direccion 'dir'
q           Salir de la consola de depuracion
<CTRL> + c  Salir de CSGBE
CSGBE>

```

Imagen 43: Consola de depuración

En cuanto se entra en el modo de depuración se muestra una lista de opciones disponibles y el código que debe introducirse para ejecutar cada uno de ellos.

- **?**: Muestra el mismo mensaje de ayuda que al iniciar la consola de depuración
- **r**: Muestra el estado actual de los registros internos del procesador y de los flags de ejecución. Todos los valores se presentan en hexadecimal.

```

CSGBE> r
PC = 1F58      SP = FFFE
A = 00 B = 00 C = 13 D = 00 E = D8 H = 01 L = 4D
Z = True      N = False      H = False      C = False

```

Imagen 44: Estado de los registros y flags

- **i**: Muestra el estado de las interrupciones del sistema y del IME (Interrupt Master Enable)

```

CSGBE> i
IME = False
VBLANK = False  LCDC = False  TIMER = False  SERIALTX = False  KEY = False

```

Imagen 45: Estado de las interrupciones

- **m**: Muestra los datos contenidos en un rango del espacio de direcciones. En caso de tratarse de una región intercambiable como los bancos de ROM, se muestra los datos del que esté en ese momento proyectado. Recibe dos parámetros, la dirección de inicio en hexadecimal, y la cantidad de bytes a mostrar desde esa dirección.

```

CSGBE> m 0 500
0000  FF 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00  .....
0010  FF 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00  .....
0020  FF 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00  .....
0030  FF 00 00 00 00 00 00 00 00 FF 00 00 00 00 00 00 00  .....
0040  C3 24 20 00 00 00 00 00 00 FF 00 00 00 00 00 00 00  .$. ....
0050  C3 06 23 00 00 00 00 00 00 C3 25 21 00 00 00 00 00  ..#.....%!....
0060  D9 AF E0 0F F0 FF 47 CB 87 E0 FF F0 44 FE 91 20  ....G....D..
0070  FA F0 40 E6 7F E0 40 78 E0 FF C9 F0 40 CB FF E0  ..@. ,. @x....@...
0080  40 C9 AF 21 00 C3 06 A0 22 05 20 FC C9 3E A0 21  @..!....". ..>.!
0090  00 C3 11 04 00 06 28 77 19 05 20 FB C9 EA E9 CE  ....(w.. ....
00A0  F0 B8 F5 FA E9 CE E0 B8 EA 00 20 CD B5 00 F1 E0  ....
00B0  B8 EA 00 20 C9 2A 12 13 0B 79 B0 20 F8 C9 00 00  ... *....y. ....
00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0100  00 C3 50 01 CE ED 66 66 CC 0D 00 0B 03 73 00 83  ..P...ff.....s..
0110  00 0C 00 0D 00 08 11 1F 88 89 00 0E DC CC 6E E6  .........n.
0120  DD DD D9 99 BB BB 67 63 6E 0E EC CC DD DC 99 9F  ....gcn.....
0130  BB B9 33 3E 50 6F 6B 65 6D 6F 6E 20 42 6C 75 65  ..3>Pokemon Blue
0140  00 00 00 00 30 31 03 13 05 03 01 33 00 B3 9E 0A  ....01.....3....
0150  FE 11 28 03 AF 18 02 3E 00 EA 1A CF C3 54 1F 3E  ..(....>.....T.>
0160  20 0E 00 E0 00 F0 00 F0 00 F0 00 F0 00 F0 00 F0  .....
0170  00 2F E6 0F CB 37 47 3E 10 E0 00 F0 00 F0 00 F0  ./...7G>.....
0180  00 F0 00 F0 00 F0 00 F0 00 F0 00 F0 00 F0 00 2F  .....
0190  E6 0F B0 E0 F8 3E 30 E0 00 C9 F0 B8 F5 3E 03 E0  ....>0.....>..
01A0  B8 EA 00 20 CD 00 40 F1 E0 B8 EA 00 20 C9 A1 42  ... ..@..... ..B
01B0  57 43 54 45 4E 47 00 40 98 49 00 40 A7 4B 00 40  WCTENG.@.I.@.K.@
01C0  1E 49 A4 49 A4 49 C3 40 00 40 E6 41 90 43 81 45  .I.I.I.@.@.A.C.E
01D0  00 40 00 40 2D 41 86 46 D4 42 BE 44 6D 46 0C 48  .@.@-A.F.B.DmF.H
01E0  99 49 2C 49 DA 4A 20 4B 38 4C 78 4E F1 40 FF 4F  .I,I.J K8LxN. @.0
CSGBE>

```

Imagen 46: Volcado de una zona de memoria

- **e:** Ejecuta un número determinado de instrucciones de ensamblador a partir de la dirección almacenada en el contador de programa (registro PC). Recibe sólo como parámetro el número de instrucciones a ejecutar, incluyendo las rutinas de tratamiento de interrupción.

```

CSGBE> e 10
#31 0069: E0 LD 0xFF(255), A Z:True N:False H:False C:False A:00 B:00 C:13 D:00 E:D8 H:01 L:4D
#32 006B: F0 LD A, 0xFF(68) Z:True N:False H:False C:False A:00 B:00 C:13 D:00 E:D8 H:01 L:4D
#33 006D: FE CP A, 145 Z:False N:True H:True C:True A:00 B:00 C:13 D:00 E:D8 H:01 L:4D
#34 006F: 20 JR NZ, -6 Z:False N:True H:True C:True A:00 B:00 C:13 D:00 E:D8 H:01 L:4D
#35 006B: F0 LD A, 0xFF(68) Z:False N:True H:True C:True A:00 B:00 C:13 D:00 E:D8 H:01 L:4D
#36 006D: FE CP A, 145 Z:False N:True H:True C:True A:00 B:00 C:13 D:00 E:D8 H:01 L:4D
#37 006F: 20 JR NZ, -6 Z:False N:True H:True C:True A:00 B:00 C:13 D:00 E:D8 H:01 L:4D
#38 006B: F0 LD A, 0xFF(68) Z:False N:True H:True C:True A:00 B:00 C:13 D:00 E:D8 H:01 L:4D
#39 006D: FE CP A, 145 Z:False N:True H:True C:True A:00 B:00 C:13 D:00 E:D8 H:01 L:4D
#40 006F: 20 JR NZ, -6 Z:False N:True H:True C:True A:00 B:00 C:13 D:00 E:D8 H:01 L:4D
CSGBE>

```

Imagen 47: Flujo de ejecución de instrucciones

Cada línea se corresponde a una instrucción ejecutada

- Número de instrucción empezando en 0 desde el inicio de la emulación

- Dirección de memoria donde se encuentra la instrucción, que se corresponde con el contador de programa (PC)
 - opCode o identificador hexadecimal de la instrucción
 - Mnemónico decodificado de la instrucción
 - Estado de los flags después de ejecutar la instrucción
 - Contenido de los registros después de ejecutar la instrucción
- **x**: Reinicia la consola. Tiene el mismo efecto que la opción del menú “Iniciar” de la interfaz gráfica
 - **a**: Asigna un valor a un registro. Toma dos parámetros, el nuevo valor en hexadecimal, y el nombre del registro. El nombre del registro puede ser cualquiera entre A, B, C, D, E, H, L, PC, SP, HL, BC, DE y AF, sabiendo que algunos de ellos pueden ser virtuales como el caso de BC que es lo mismo que B en la parte alta y C en la baja.
 - **f**: Cambia el estado de uno de los flags. Dada la naturaleza binaria de los flags, sólo es necesario indicar el nombre del flag y automáticamente se invertirá su estado. El nombre del flag puede ser cualquiera entre Z, N, H y C.
 - **w**: Escribe un valor en hexadecimal sobre una dirección de memoria también en hexadecimal. Se aplican las mismas operaciones que una instrucción normal en el caso de acceder a una dirección especial de entrada / salida o a una región de intercambio de banco en un cartucho.
 - **n**: Activa o desactiva el estado de una interrupción. Recibe como parámetro el nuevo estado (1 para activar, 0 para desactivar) y el nombre de la interrupción. El nombre de la interrupción puede ser cualquiera entre vblank, lcdc, timer, serialtx y key, con independencia de mayúsculas o minúsculas.
 - **k**: Cambia el estado de pulsación de una de los botones de la consola. Dada la naturaleza binaria de los botones, sólo es necesario indicar el nombre del botón y automáticamente se invertirá su estado. El nombre del botón puede ser cualquiera entre A, B, Select, Start, Up, Down, Left y Right.
 - **b**: Inserta o elimina un punto de interrupción en el flujo de ejecución cuando se alcance una dirección determinada. Si se vuelve a crear un punto de interrupción

en la misma dirección en la que se insertó uno, éste se desactivará. Cuando el flujo de ejecución llegue a una dirección en la que haya un punto de interrupción, saltará automáticamente la consola de depuración, tanto si se encuentra fuera de la consola, o dentro ejecutando un número determinado de instrucciones de manera controlada.

- **q**: Sale de la consola de depuración y retorna a la ejecución normal desde la ventana principal de la interfaz gráfica. Los puntos de interrupción que se hayan marcado se siguen conservando igualmente.
- **Control + C**: Aborta de manera abrupta la aplicación y la emulación. No es aconsejable utilizar esta opción salvo casos especiales.