

Vircon32

CONSOLA VIRTUAL DE 32 BITS



Especificación del sistema

Parte 2: Arquitectura de la consola

Documento con fecha 2024.01.07

Escrito por Carra

¿Qué es esto?

Este documento es la parte número 2 de la especificación del sistema Vircon32. Esta serie de documentos define el sistema Vircon32, y provee una especificación completa que describe en detalle sus características y comportamiento.

El principal objetivo de esta especificación es definir un estándar de lo que es un sistema Vircon32, y cómo debe implementarse un sistema de juego para que se considere conforme a él. Además, al ser Vircon32 es un sistema virtual, un importante objetivo adicional de estos documentos es proporcionar a cualquiera el conocimiento para crear sus propias implementaciones de Vircon32.

Sobre Vircon32

El proyecto Vircon32 fue creado de forma independiente por Carra. El sistema Vircon32 y su material asociado (incluyendo documentos, software, código fuente, arte y cualquier otro elemento relacionado) son propiedad del autor original.

Vircon32 es un proyecto libre y de código abierto en un esfuerzo por promover que cualquiera pueda jugar a la consola y crear software para ella. Para obtener información más detallada al respecto, se recomienda consultar los textos de licencia incluidos en cada uno de los programas disponibles.

Sobre este documento

Este documento se proporciona bajo la Licencia de Atribución Creative Commons 4.0 (CC BY 4.0). Puede leerse el texto completo de la licencia en el sitio web de Creative Commons: <https://creativecommons.org/licenses/by/4.0/>

Índice

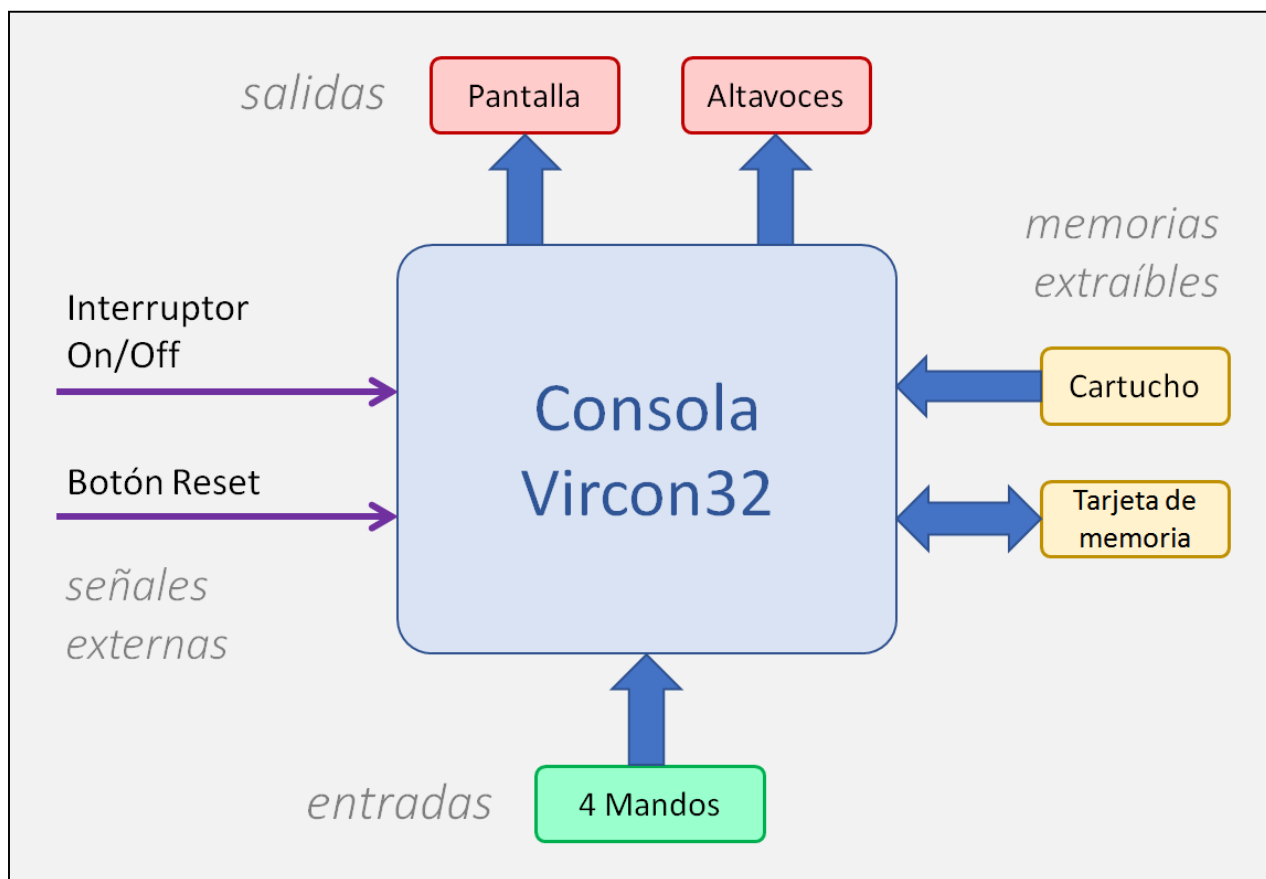
La parte 2 de la especificación comienza identificando todos los componentes de la consola y las tareas que tienen asignadas. A continuación, explica las formas en que los distintos chips pueden comunicarse e interactuar de modo que se integren para trabajar juntos como una consola.

También se tratan aquí algunos mecanismos y propiedades de los datos que afectan al funcionamiento de todos los componentes de la consola.

1 El interior de la consola	3
2 Control del tiempo	6
3 Buses de comunicaciones	7
4 Bus de memoria	8
5 Bus de control	9
6 Comandos de los chips	11
7 Formatos de datos internos	11
8 Orden de los bytes	13
9 Errores hardware	14

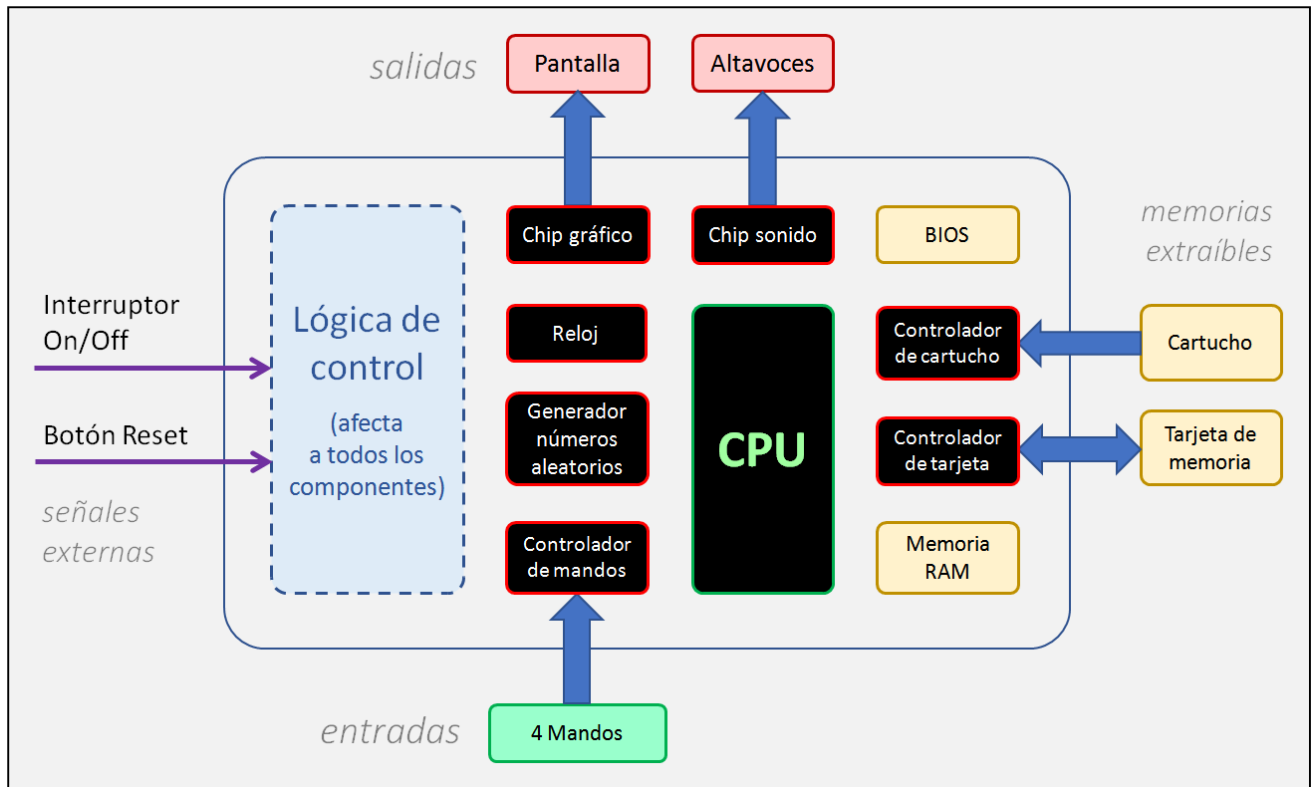
1 El interior de la consola

Tras leer las secciones de introducción de la parte 1, ya podemos deducir que la arquitectura básica de Vircon32 a nivel de sistema sigue este diagrama:



El diagrama utiliza un sencillo código de colores para identificar cada tipo de componente externo que puede conectarse a la consola: hay dispositivos de entrada y salida, y dispositivos que conectan memoria externa. Cualquiera de ellos puede estar conectado a la consola o estar ausente, ya que todos ellos son opcionales. Cuando no estén presentes, la consola aún funcionará, aunque obviamente carecerá de sus funciones añadidas.

La parte 1 ya dio una visión general de todos esos dispositivos externos, así que ahora omitiremos temporalmente el entorno de la consola y echaremos un vistazo a la propia consola. Para pasar al siguiente nivel de detalle podemos tomar ese mismo diagrama anterior y ampliarlo de modo que la consola ya no sea una caja negra. En este segundo diagrama podemos distinguir sus componentes internos.



Por claridad, este diagrama omite los buses de datos que comunican estos chips internos de la consola (que se presentan en secciones posteriores). El bloque "Lógica de control", es una forma más simple de tener en cuenta mecanismos que abarcan toda la consola. En este caso esos mecanismos son señales de control que reciben todos los componentes de la consola, y transmiten eventos relacionados con la alimentación, reset y temporización.

1.1 Chips de la consola

Los componentes principales de la consola son los distintos chips responsables de realizar sus funciones. Los demás componentes de la consola son secundarios y están presentes sobre todo para dar soporte a estos chips. Los chips de la consola Vircon32 son:

- **Procesador (CPU):** Es el chip principal y el que ejecuta el programa. El procesador ejecuta las instrucciones del programa una a una e interactúa con los otros chips cuando es necesario.
- **Reloj:** Controla cuándo debe producirse cada ciclo de ejecución de la CPU. Además, 60 veces por segundo, inicia un nuevo frame. Esto provoca el refresco de la pantalla, pero también controla algunas funciones de otros chips.
- **Controlador de mandos:** Detecta cuáles de los 4 mandos están conectados en cada momento. Permite a los programas leer el estado de sus crucetas y botones.

- **Controlador de cartucho:** Detecta cuándo se conecta un cartucho y puede proporcionar información básica sobre su contenido. A través de él, la CPU puede leer la memoria de programa del cartucho.
- **Controlador de tarjeta:** Permite saber si hay una tarjeta de memoria conectada. Cuando está presente permite a la CPU acceder a la memoria de esa tarjeta.
- **Chip gráfico (GPU):** Este chip accede a las imágenes presentes en la ROM de vídeo del cartucho y las usa para dibujar en pantalla. Puede aplicarles algunos efectos como rotación y escalado.
- **Chip de sonido (SPU):** La SPU tiene acceso directo a los sonidos de la ROM de audio del cartucho. Puede reproducir hasta 16 de ellos a la vez y aplicarles algunos efectos como cambios de velocidad y bucles.
- **Generator de números aleatorios:** Usa un algoritmo para producir números pseudoaleatorios cada vez que lo pedimos, lo que permite simular aleatoriedad.
- **Memoria RAM:** Es la memoria de trabajo del programa que se ejecuta en la CPU. Su tamaño es de 16MB o, en palabras de 32 bits, 4MW.
- **BIOS:** Es una memoria de sólo lectura, similar en estructura a un cartucho. Contiene un software interno que controla el arranque de la consola y reacciona ante posibles errores de hardware. También incluye una fuente de texto para permitir la escritura en pantalla.

1.2 Buses de comunicaciones

Los chips de la consola descritos no pueden funcionar de forma aislada: deben poder comunicarse e interactuar. En el diagrama detallado de la arquitectura se han omitido intencionadamente los 2 buses de que conectan los diferentes chips, que son:

- **Bus de memoria:** Este bus permite a los diferentes dispositivos hacer accesible su memoria interna a la CPU. Une todos sus rangos de direcciones locales en un mapa de direcciones global que puede leerse y escribirse de la misma manera.
- **Bus de control:** El propósito de este bus es permitir que la CPU acceda a un conjunto de "puertos de control" expuestos por cada chip. Leyendo y escribiendo en esos puertos, el programa en ejecución puede controlar las funciones de esos chips.

Estos buses de comunicaciones y sus configuraciones de conexión se describirán en detalle en el capítulo 3.

1.3 Señales de control

Hay algunos eventos globales a nivel de consola que deben estar disponibles para todos los chips, para asegurar que todos ellos puedan reaccionar al evento cuando sea necesario. Para esto la consola usa las siguientes señales de control:

- **Encendido:** No es realmente una señal en sí. Una implementación por hardware deberá transmitir la corriente a todos los componentes, controlándola con un botón o interruptor. En los emuladores esto puede reducirse a simples métodos On/Off.
- **Reset:** La señal de Reset puede ser activada directamente por el usuario, pero también ocurre cada vez que se enciende la consola. Se transmite a todos los componentes para que vuelvan a un estado inicial conocido.
- **Nuevo frame:** Esta señal se activa cada vez que se refresca la señal de vídeo de Vircon32, lo que ocurre exactamente a 60 fps. Debido a la forma en que Vircon32 maneja la temporización (ver siguiente sección), esto también debe transmitirse a algunos chips para que puedan tomar acciones.
- **Nuevo ciclo:** Los ciclos son la unidad de tiempo mínima dentro de Vircon32. La señal de ciclo está pensada para actuar como señal de reloj para la CPU y otros componentes. Los ciclos de Vircon32 ocurren a una velocidad de 15 MHz.

2 Control del tiempo

Todo el funcionamiento del sistema Vircon32 sigue un esquema de temporización global. El temporizador es responsable de producir las señales de "nuevo frame" a 60 Hz. Luego, dentro de cada frame, producirá 250.000 señales de "nuevo ciclo". Esto da como resultado 15.000.000 ciclos por segundo.

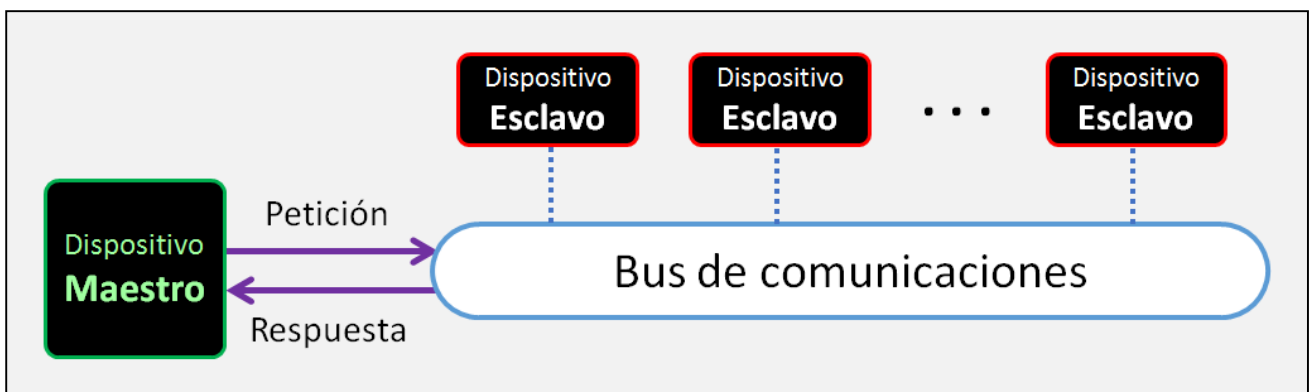
Toda acción realizada por los componentes de Vircon32 ocurre siempre (directa o indirectamente) como reacción a una señal del Temporizador. Ya que todos los componentes comparten estas mismas señales, la temporización global se simplifica enormemente porque no es necesaria la coordinación entre componentes.

2.1 Temporización dentro de un frame

Un sistema Vircon32 debe garantizar que los frames comienzan exactamente a intervalos de 60 Hz. Sin embargo, los ciclos de un frame no tienen que ocurrir a intervalos regulares en tiempo real. Cada implementación puede elegir cómo organizar los ciclos en un frame, mientras preserve el orden y los 250.000 ciclos sucedan antes del siguiente frame. Es válido, por ejemplo, ejecutar todos los ciclos muy deprisa y luego esperar. Por ello, los programas no pueden contar los ciclos de un frame para medir el tiempo con precisión.

3 Buses de comunicaciones

Los chips de la consola Vircon32 usan buses como mecanismo de comunicación. Estos buses siguen un modelo maestro-esclavo, donde existe un único dispositivo maestro que controla la comunicación y envía las peticiones. El resto de componentes conectados (los esclavos) se comportan de forma pasiva y se limitan a responder a las peticiones que reciben. En ambos buses de consola la CPU controla la comunicación (es el maestro), y la misión del resto de componentes conectados al bus es dar servicio al procesador.

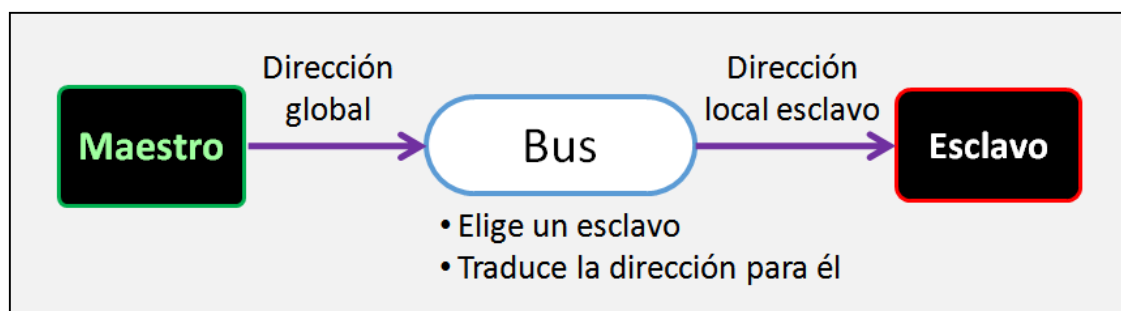


3.1 Espacio de direcciones

Cada esclavo conectado expone una serie de registros de 32 bits, que se ponen a disposición del dispositivo maestro. Cada uno de esos registros se identifica dentro del dispositivo que los aloja con un valor entero (su dirección local).

Cuando hay varios esclavos conectados, el bus se encarga de traducir entre direcciones locales para cada esclavo en particular, y direcciones globales que el maestro puede utilizar como parte de un espacio de direcciones global unificado.

Para hacer una petición, el maestro del bus debe seleccionar un registro de destino proporcionando al bus la dirección de destino deseada. A continuación, el bus elegirá el dispositivo esclavo correcto al que debe enviarse la petición, y realizará el mismo proceso a la inversa cuando dicho esclavo envíe su respuesta.



3.2 Mecanismo de petición

En ambos buses, como maestro, la CPU puede hacer los mismos 2 tipos de peticiones:

- **Petición de lectura:**

La CPU solicita leer el valor almacenado en una dirección global específica. Cuando la petición tiene éxito, se proporciona el valor pedido.

- **Petición de escritura:**

La CPU solicita escribir un valor dado en una dirección global específica. Cuando la solicitud tiene éxito, el valor se escribe en la dirección pedida.

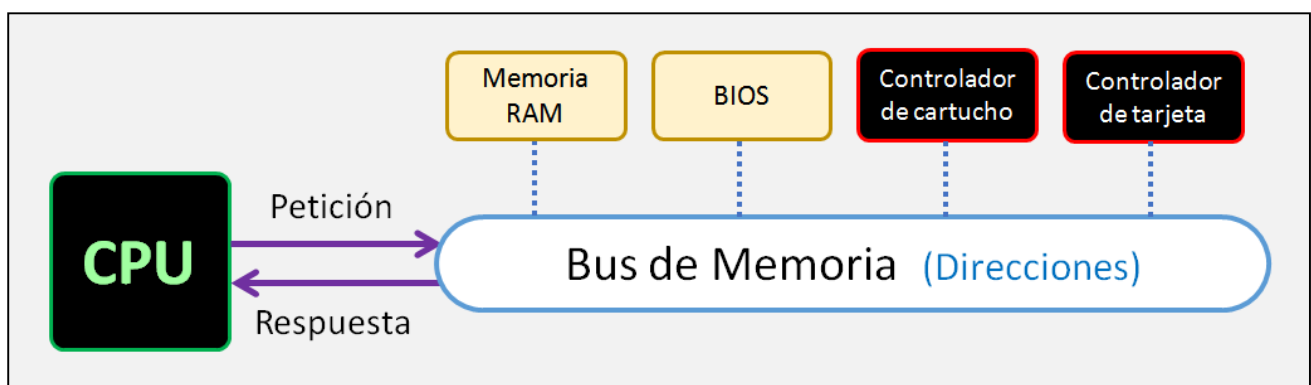
En ambos casos la respuesta a la CPU debe incluir un valor booleano que indique si la petición ha tenido éxito o ha fallado.

Una petición puede fallar por dos razones. La primera es que la petición puede referirse a una dirección inexistente (que no está expuesta por ninguno de los dispositivos conectados actualmente). Si la dirección existe, la segunda causa de fallo es que un dispositivo esclavo no pueda realizar esa petición en concreto. Un ejemplo típico es intentar escribir en una memoria de sólo lectura, como un cartucho.

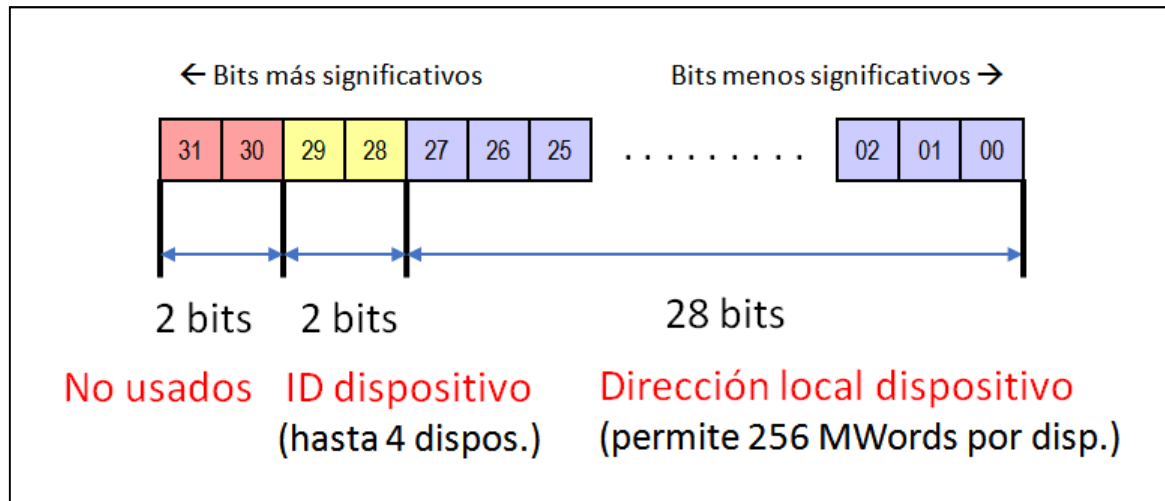
4 Bus de memoria

Este es el bus al que se conectarán todos los dispositivos que tengan memoria (ya sea RAM o ROM). Al conectarse a este bus, su espacio de memoria recibe un rango de direcciones global y su contenido se vuelve accesible para la CPU. Las peticiones de la CPU para leer o escribir una dirección en este bus, cuando tienen éxito, tienen el efecto de leer o escribir esa palabra en particular en la memoria del dispositivo destino.

El esquema de conexiones para el bus de memoria se muestra en este diagrama. Nótese que, aunque los 4 esclavos están siempre conectados, puede no haber memoria accesible en los chips controladores si el cartucho o la tarjeta de memoria no están presentes.



El bus de memoria usa direcciones de 32 bits, y para convertir entre espacios de direcciones globales y locales divide la dirección en campos como se muestra aquí. Los 2 bits superiores (no usados) se ignoran y se tratan como si su valor fuera siempre 00.



Los 4 dispositivos conectados se asignan a direcciones de memoria con los siguientes IDs:

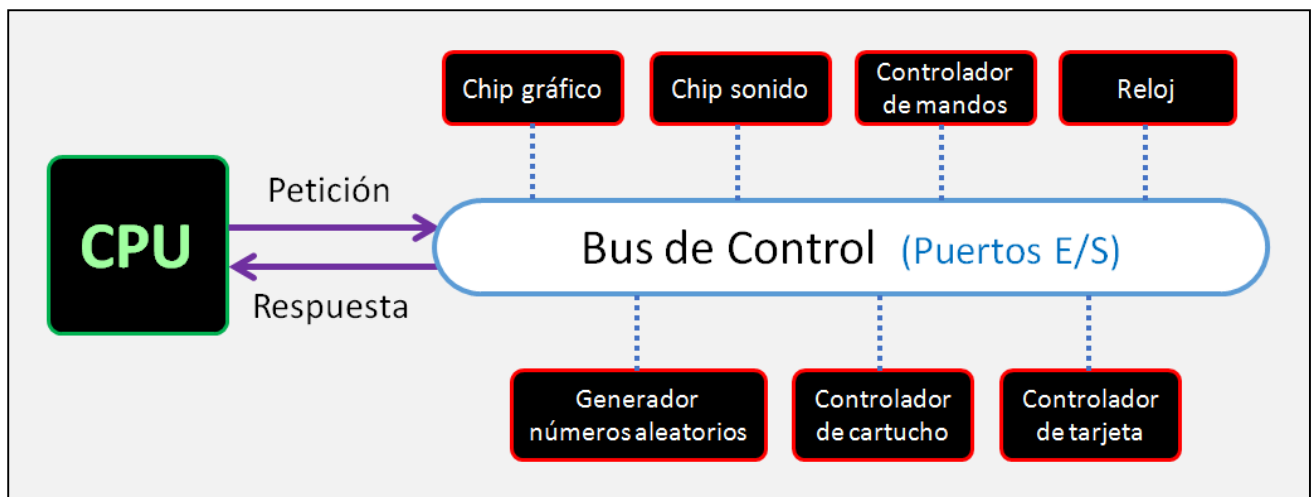
ID dispositivo	Dispositivo conectado
0	Memoria RAM
1	ROM de programa BIOS
2	ROM de programa cartucho
3	RAM de tarjeta de memoria

5 Bus de control

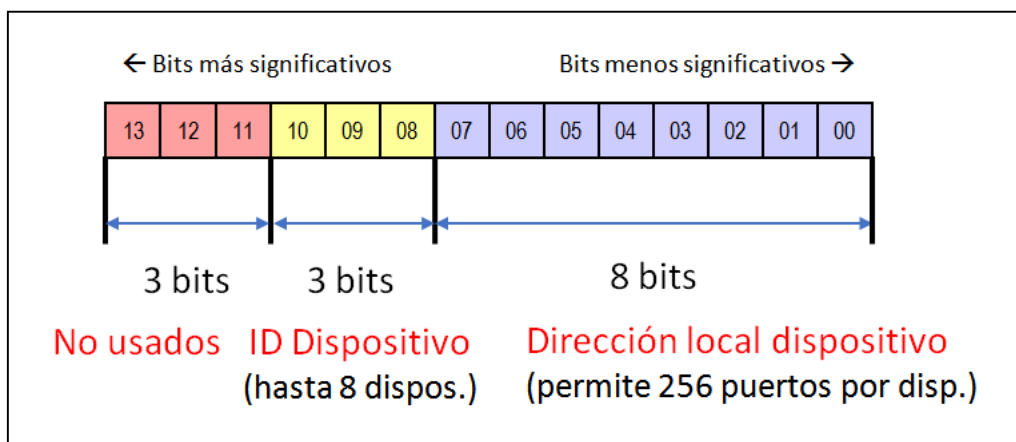
Este bus permite al procesador acceder a una serie de puertos de E/S en cada uno de los chips conectados. Cada uno de estos puertos es un registro de control de una sola palabra que expone el chip. Al leer y escribir en estos puertos, la CPU puede pedir operaciones a los diferentes chips de la consola y recibir información sobre su estado.

Cuando la CPU lee o escribe con éxito un puerto de E/S, los efectos en general pueden ser muy diferentes del resultado esperable de sólo "leer o escribir un valor". Los puertos pueden tener efectos secundarios cuando son leídos o escritos. Algunas peticiones de escritura pueden incluso no escribir ningún valor. Para conocer el comportamiento de un puerto específico, véase la lista de puertos de control en la especificación de cada chip.

Las conexiones del bus de control son las que muestra este diagrama. En este caso, como todos los chips están siempre conectados, sus puertos de control son siempre accesibles.



El bus de control usa direcciones de sólo 14 bits porque se extraen de instrucciones del programa (véase su formato en el capítulo 7). Para convertir entre espacios de direcciones globales y locales se divide la dirección en campos como se muestra aquí. Los 3 bits superiores (no usados) se ignoran y se tratan como si su valor fuera siempre 000.



Los 7 dispositivos conectados se asignan a los 8 IDs disponibles como indica esta tabla:

ID dispositivo	Dispositivo conectado
0	Temporizador
1	Generator núm. aleatorios
2	Chip gráfico (GPU)
3	Chip de sonido (SPU)
4	Controlador de mandos
5	Controlador de cartucho
6	Controlador de tarjeta
7	(no hay dispositivo conectado)

6 Comandos de los chips

Como se dijo antes, al ejecutar el programa la CPU necesitará usar el bus de control para invocar funciones de otros chips. El mecanismo que usan los chips de la consola para permitirlo es exponer un puerto de comando. En él la CPU puede escribir ciertos códigos de comandos para provocar acciones específicas de ese chip.

Comandos con parámetros

Algunas funciones de los chips necesitan que se les indiquen ciertos parámetros. Por ejemplo, para dibujar una región de textura en pantalla, se necesita saber qué región dibujar y en qué punto colocarla. La lógica para usar ese comando podría ser ésta:

```
GPU.DrawRegion( Region, X, Y )
```

Pero invocar una función del chip escribiendo en un puerto no permite que el comando reciba parámetros, por lo que no se puede usar este mecanismo. En su lugar, cada chip contará con unas variables internas que controlan los comandos invocados. Esas variables se exponen a través de puertos para que puedan ser leídas o modificadas. Así, el proceso para dibujar una región seguiría realmente un proceso como éste:

```
GPU.SelectedRegion = Region
GPU.DrawingPointX = X
GPU.DrawingPointY = Y
GPU.DrawRegion( )
```

Primero la CPU ajustará las variables necesarias, y sólo entonces escribirá en el puerto de comandos del chip. Esto asegurará que las funciones del chip hagan lo deseado.

7 Formatos de datos internos

Todos los datos procesados por componentes de la consola son siempre palabras de 32 bits. Sin embargo, los mismos 32 bits se interpretarán de diferentes formas por cada componente dependiendo del contexto. En esta sección describiremos los formatos binarios disponibles y sus interpretaciones dentro de un sistema Vircon32:

7.1 Entero

La palabra se interpreta como un entero de 32 bits con signo. Utiliza la notación de complemento a dos. Equivale al tipo de datos “int32_t” de C.

Obsérvese que éste es el único tipo de dato disponible para enteros: no existen enteros sin signo ni variantes para distintos tamaños.

7.2 Booleano

La palabra se interpreta primero como un número entero, como se vio antes. Luego este número se toma como un valor booleano verdadero/falso. Si el valor entero es 0 (es decir, todos los bits son 0), se interpreta como falso. Cualquier otro valor se interpreta como verdadero. Esto equivale al tipo de datos "bool" de C.

Al producir un booleano, Vircon32 siempre usa valor 1 para “verdadero” y 0 para “falso”.

7.3 Float

La palabra se interpreta como un número en coma flotante de 32 bits. Sigue la notación exponencial habitual definida por IEEE 754. Equivale al tipo de datos "float" de C.

Nótese que los floats pueden usar valores especiales como NaN (not a number), infinitos e indeterminados. Los sistemas Vircon32 no están obligados a implementar ningún tipo de soporte o detección de errores para estos valores no numéricos, por lo que su uso en un programa dará lugar a un comportamiento indeterminado.

7.4 Binario

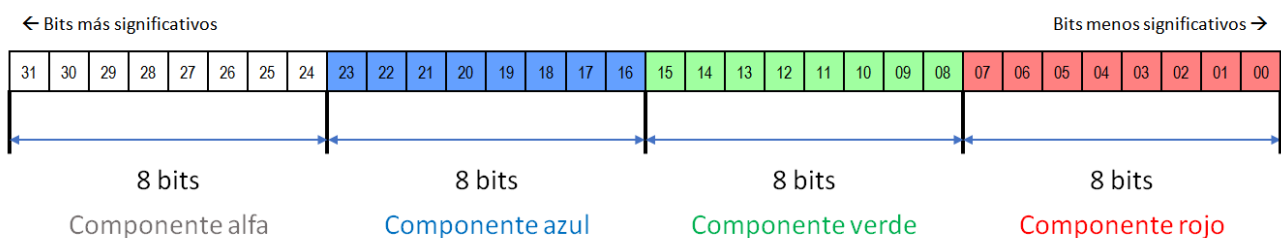
Se considera que la palabra es sólo una secuencia de 32 bits, independientes entre sí y sin ningún significado adicional como grupo.

Esta interpretación sólo suele utilizarse en operaciones binarias como AND/OR. Esto es equivalente a la forma en que C trata los enteros en tales operaciones.

7.5 Color de GPU

Si la GPU usa una palabra para producir un color en pantalla, interpretará esa palabra como un conjunto de 4 campos de 8 bits que representan los componentes de color en el espacio RGBA. Esto equivale al formato RGBA8 utilizado por muchas tarjetas gráficas.

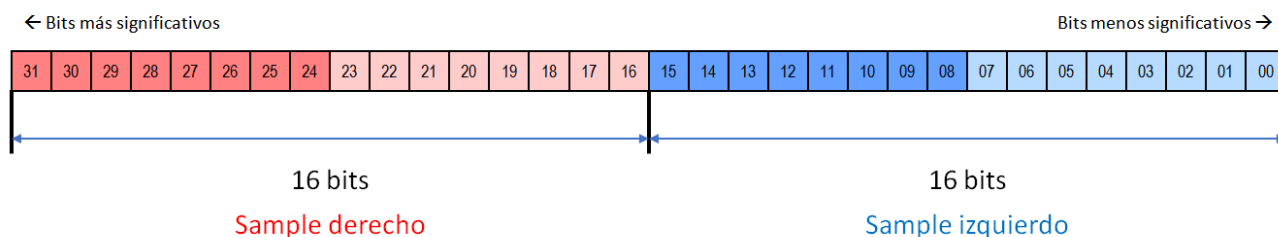
En algunas situaciones el componente alfa puede no ser aplicable, y en esos casos el valor para el campo Alfa simplemente se ignorará, y se tratará como “opacidad total”.



7.6 Sample de SPU

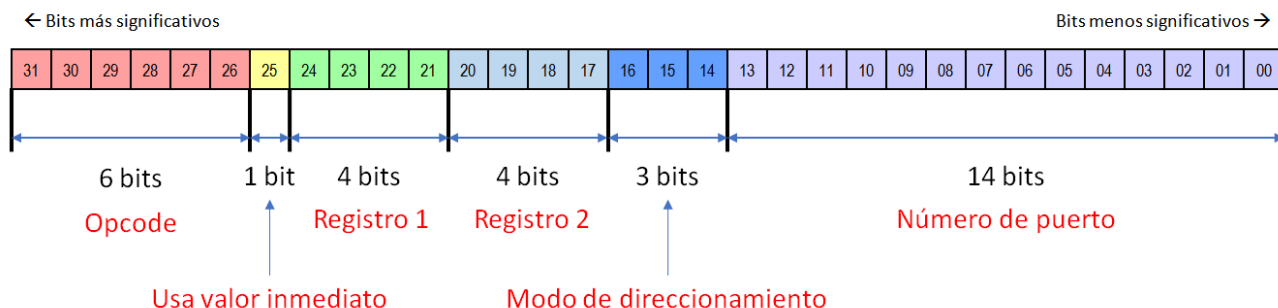
Si la SPU usa una palabra para emitir un sample de sonido, la interpretará como un par de enteros con signo de 16 bits: uno para el altavoz izquierdo y otro para el derecho. Los dos juntos forman un único sample estéreo de 32 bits.

En este diagrama, los bits más oscuros de cada sample son el byte más significativo dentro de ese entero de 16 bits. Este es el formato de sample usado en los archivos WAV.



7.7 Instrucción de CPU

Si la CPU intenta ejecutar una palabra como una instrucción, la interpretará como un código de instrucción ("Opcode") con una serie de campos complementarios. Entre esos campos se encuentra el "Número de puerto" que usa el bus de control.



El uso e interpretación de los demás campos se cubrirá en la especificación de la CPU.

8 Orden de los bytes

En todo el sistema Vircon32, el tamaño mínimo (y único) de los datos no es el byte, sino la palabra de 32 bits. Lo mismo se aplica al almacenamiento en memoria: las direcciones se expresan como offsets en palabras y no se puede acceder a bytes individuales.

Como las palabras no se toman como una secuencia de bytes, en la mayoría de casos no es necesario aplicar el concepto de "endianness". Sin embargo, la mayoría de los ordenadores almacenan y acceden a los datos en bytes. Por esto, es posible que algunos

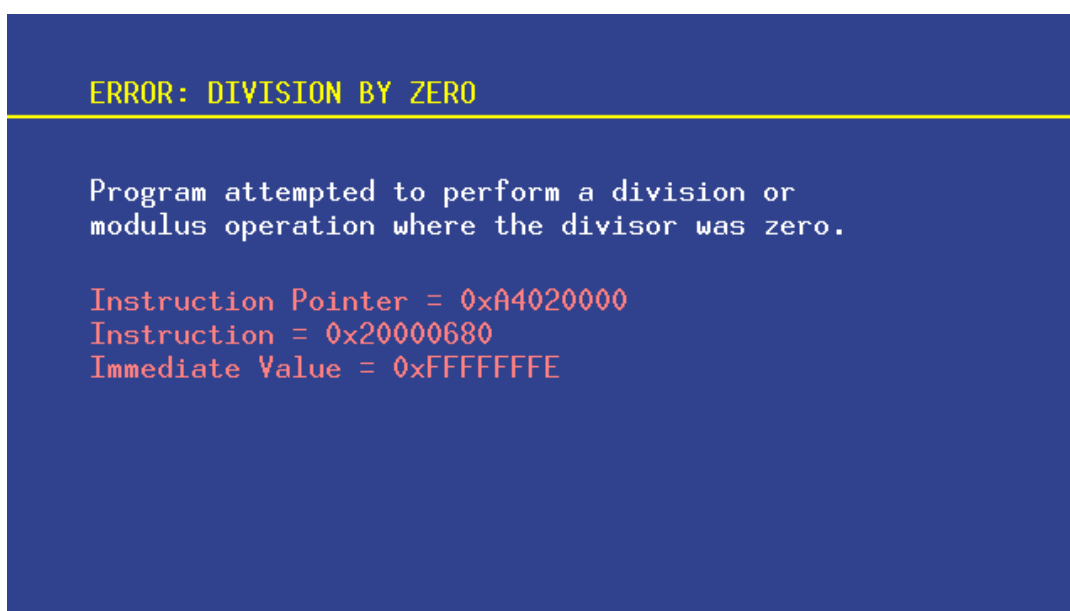
detalles de implementación deban tener en cuenta el orden de bytes para asegurar la compatibilidad binaria.

Por esta razón, el sistema Vircon32 está definido para representar palabras usando un sistema **little-endian**. Ésta es la representación que se usa en la mayoría de sistemas informáticos actuales.

9 Errores hardware

Mientras se ejecuta un programa, es posible que a algunos componentes del hardware se les pida realizar operaciones que no pueden completar. Ejemplos comunes son el acceso a memoria que no existe, y errores matemáticos como dividir por cero. Cuando la consola detecta alguna de estas situaciones dispara un error de hardware. Cuando se dispara un error de hardware, la consola detendrá la ejecución del programa y transferirá el control a las rutinas de error de la BIOS para gestionar dicho error.

Los gestores de error de la BIOS son muy simples, ya que sólo se requiere que muestren información básica sobre el error y detengan toda ejecución. En esta captura se puede ver un ejemplo de cómo se presenta un error en pantalla:



```
ERROR: DIVISION BY ZERO

Program attempted to perform a division or
modulus operation where the divisor was zero.

Instruction Pointer = 0xA4020000
Instruction = 0x20000680
Immediate Value = 0xFFFFFFFF
```

Todos los errores de hardware se producen al procesar instrucciones de la CPU, por lo que la lista de errores y su procesamiento específico se tratarán en la parte 3 de la especificación, dedicada a la CPU.

(Fin de la partie 2)