Arquitectura de Computadores

4. La Memoria

1. Jerarquía de la memoria
2. La memoria principal
3. Interconexión CPU-Memoria

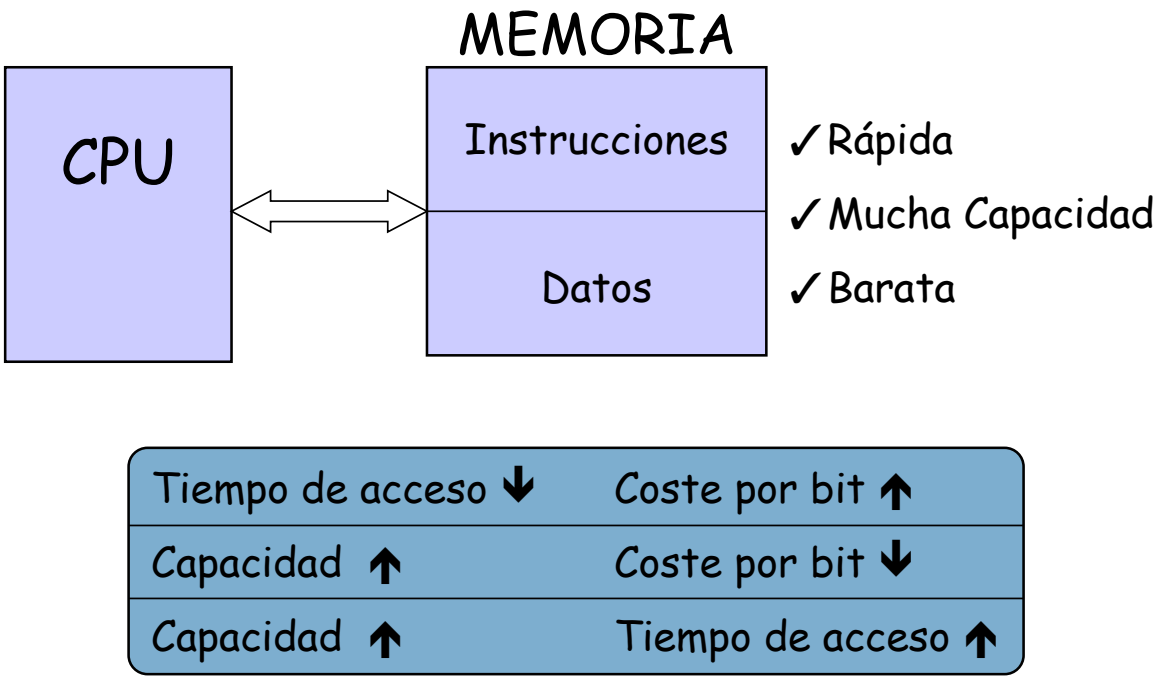
Continuando con la descripción de los componentes básicos de un ordenador, le toca el turno a la memoria.

Comenzaremos mostrando la jerarquía de la memoria tal como se suele estructurar en los ordenadores actuales, para, a continuación, pasar a describir con detalle las memorias de semiconductores (o memoria principal), comentando sus características, tipos y organización, así como la descripción de las patillas o *pines* que suelen presentar para su interconexión.

Seguidamente ofreceremos una visión completa de la interconexión entre un procesador MC68000 y algunos módulos de memoria.

Más adelante, se estudiará también la memoria caché, una memoria pequeña y muy rápida que se ubica entre la memoria principal y la CPU, pero esto será en otro capítulo posterior.

Jerarquía de Memoria



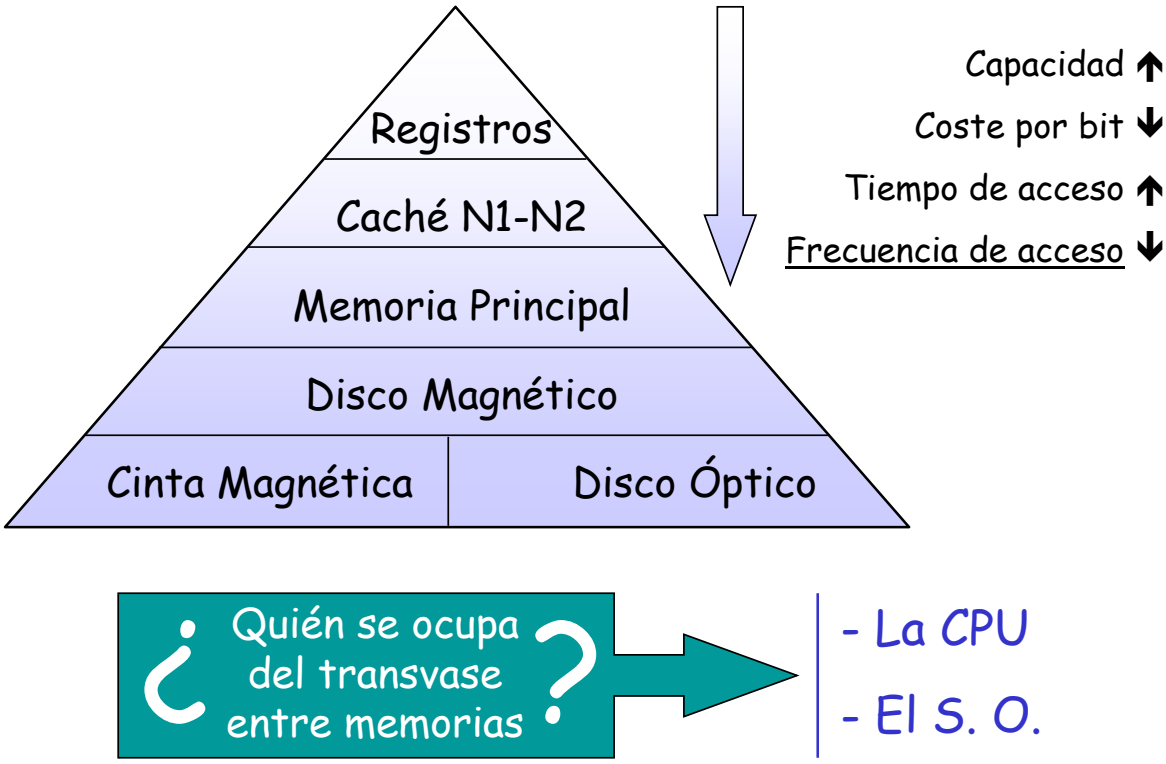
La memoria es la parte del ordenador en la que se guardan o almacenan los programas (las instrucciones y los datos). Sin una memoria de la que los procesadores leyeran o escribieran la información, no habría ordenadores digitales de programa almacenado (como son todos los actuales desde el EDVAC en 1945).

Por una parte tenemos que la velocidad de ejecución de los programas es muy dependiente de la velocidad a la que se pueden transferir los datos entre la CPU y la memoria. Por otra parte, también es importante disponer de una gran cantidad de memoria, para facilitar la ejecución de programas que son grandes o que trabajan con una gran cantidad de datos.

Por esto, idealmente, la memoria debería ser rápida, grande y barata. Como cabría esperar, hay un compromiso entre estas tres características de la memoria que mantienen las siguientes relaciones:

- A menor tiempo de acceso → mayor coste por bit.
- A mayor capacidad → menor coste por bit.
- A mayor capacidad → mayor tiempo de acceso.

...Jerarquía de Memoria



De cara al diseñador el dilema está claro; le gustaría disponer de una tecnología de memoria que le proporcionara una gran capacidad, tanto porque se necesita cantidad de memoria, como porque el coste por bit es pequeño. Sin embargo, para conseguir buen rendimiento y velocidad se necesitan memorias de rápido acceso, que son de poca capacidad y más caras.

La pirámide del gráfico superior, está construida por diversos tipos de memoria, de tal manera que a medida que se va de arriba hacia abajo, sucede lo siguiente:

- Disminuye el coste por bit
- Aumenta la capacidad
- Aumenta el tiempo de acceso
- Disminuye la frecuencia de acceso a la memoria desde la CPU

La clave de la solución está en este último punto: la decreciente frecuencia de acceso. Esto simplemente quiere decir que no se accede a todos los datos con la misma frecuencia; obviamente se accede más a los datos del programa en ejecución que a los de uno que no se ejecuta desde hace un año; y de igual manera, en un momento dado se accede más a los datos de una expresión que se está evaluando en ese preciso instante que a otros datos del programa.

En lugar de decidirnos por un único tipo o tecnología de memoria, lo que hay que hacer es construir una estructura con una **jerarquía de memoria** en la que haya diversas tecnologías, de tal manera que los pocos datos con los que se está ejecutando la instrucción en curso están en los registros de la CPU; los datos de las últimas instrucciones, en la memoria caché; el resto de los datos del programa en ejecución estarán repartidos entre memoria principal y secundaria de rápido acceso (discos magnéticos); los programas o datos que no se ejecutan asiduamente se guardan en memoria secundaria rápida y en memorias secundarias masivas de mayor tiempo de acceso, como la cinta magnética y el disco óptico.

La CPU y el sistema operativo se encargarán de ir llevando y trayendo los datos de las memorias lentas a las rápidas y viceversa, a medida que se vayan referenciando los distintos datos o programas.

Tipos de Memoria

<table>
<tr><th>Tipo Memoria</th><th>Propósito</th><th>Borrado</th><th>Modo de Escritura</th><th>Volatilidad</th></tr>
<tr><td>Random Access Memory (RAM)</td><td>Memoria Lec/Esc</td><td>Eléctrico a nivel byte</td><td>Eléctricamente</td><td>Volátil</td></tr>
<tr><td>Read Only Memory (ROM)</td><td rowspan="2">Memoria de solo Lectura</td><td rowspan="2">No Posible</td><td>Máscaras</td><td rowspan="6">No Volátil</td></tr>
<tr><td>Programmable ROM (PROM)</td><td rowspan="5">Eléctricamente</td></tr>
<tr><td>Erasable PROM (EPROM)</td><td rowspan="3">Memoria sobre todo de Lectura</td><td>Luz UV a nivel chip</td></tr>
<tr><td>Memoria Flash</td><td>Eléctrico a nivel bloque</td></tr>
<tr><td>Electrically Erasable PROM (EEPROM)</td><td>Eléctrico a nivel byte</td></tr>
</table>

Ya sabemos que la memoria es la parte del ordenador en la que se guardan o almacenan los programas y los datos. Aunque la CPU dispone de una memoria interna (sus registros), ésta es demasiado pequeña como para albergar programas completos, pues está pensada para albergar solamente la instrucción a ejecutar, sus operandos y poco más. Por eso es necesario disponer de un sistema de memoria externa suficientemente grande. Para ello se cuenta con la Memoria Principal, que hoy día es totalmente electrónica y está construida a base de semiconductores.

Como ya sabemos, las memorias de semiconductores están formadas por una serie de celdas que contienen los datos.

Hay diversos tipos de memorias de semiconductores, y también varios criterios de clasificación. Uno de estos criterios puede ser el siguiente: ¿Cómo se referencia cada una de las celdas de datos de la memoria?

Según este criterio, hay dos tipos de memorias:

- Seleccionables por el contenido, esto es, las asociativas.
- Seleccionables por la dirección de la celda (las llamaremos “convencionales”).

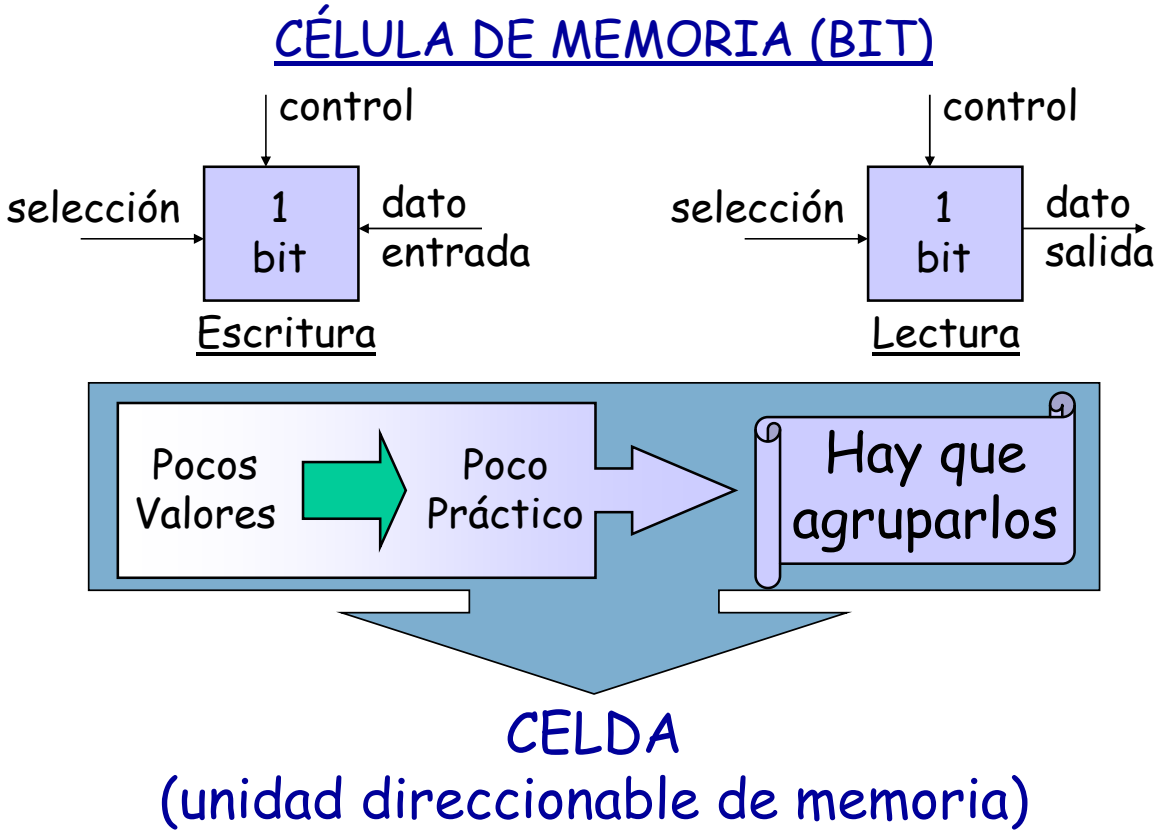
Las memorias asociativas son las que se utilizan para las memorias caché. Por su parte, las memorias convencionales tienen distintos usos dependiendo de su tecnología, y podemos encontrarnos memorias RAM, ROM, PROM, EPROM, Flash y EEPROM. Entre todas estas, la que suele ocupar la mayor parte del mapa de memoria principal es la memoria RAM, es decir, memoria volátil de acceso directo de lectura/escritura.

La memoria RAM a su vez admite distintas tecnologías, como las memorias estáticas (más rápidas y que se suelen emplear para las memorias caché) y las dinámicas (más lentas, y utilizadas para la memoria principal), pero aquí no vamos a bajar a este nivel, y vamos a ocuparnos de las características y conexiones de los módulos de memoria RAM y ROM en general. Los módulos de memoria ROM los consideraremos con las mismas características que la memoria RAM, excepto la posibilidad de escribir en ella.

Aunque un mapa de memoria principal puede estar formado por distintos tipos de memoria, lo más frecuente es encontrar simplemente memoria RAM y ROM (o alguna variedad), donde la RAM ocupa la inmensa mayoría del espacio de direcciones.

En la memoria ROM de los ordenadores es donde se encuentra el programa inicial de arranque (IPL) y un conjunto de rutinas básicas de entrada/salida. Con ayuda de este programa se arranca el mecanismo de carga del sistema operativo de cada ordenador.

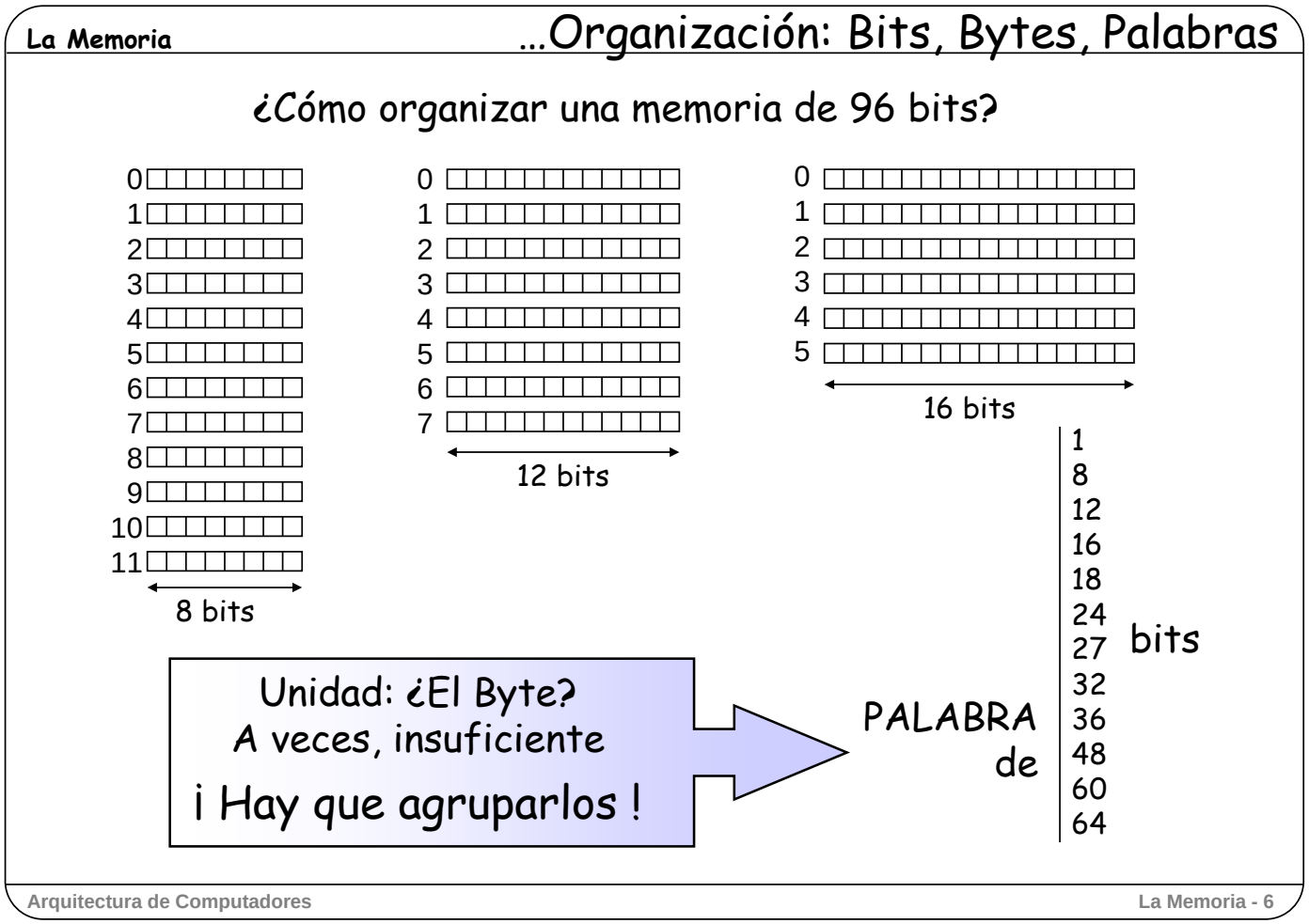
Organización: Bits, Bytes, Palabras



Sabemos que la memoria está compuesta por bits, pero un bit resulta insuficiente como unidad lógica de almacenamiento, por lo que se requiere que las celdas de memoria estén compuestas de varios bits. Es decir, a partir de la serie de bits de que está compuesta la memoria, necesitamos organizarlos en grupos de bits, a los que llamaremos celdas de memoria, tal que cada una de estas celdas sí sea capaz de almacenar un dato significativo (un carácter, un número, ...).

Cada una de estas celdas de memoria tiene asignada una dirección, lo que significa que **es la unidad direccionable de memoria**.

La unidad direccionable de memoria es la mínima porción de memoria a la que la CPU puede hacer referencia para leer o escribir en ella. Es decir, la CPU no puede acceder directamente a un bit concreto de la memoria. Para leer o escribir un bit concreto, tiene que leer o escribir la celda completa de memoria en la que está dicho bit. Una vez que la celda está en algún registro de la CPU, ésta ya puede acceder a cada bit según las instrucciones disponibles.



En la figura vemos que una memoria de 96 bits puede organizarse de varias maneras. Algunas de ellas son: 6 celdas de 16 bits, 8 celdas de 12 bits o 12 celdas de 8 bits.

Aunque el número de bits por celda ha variado mucho a lo largo de la corta historia de los ordenadores digitales, hoy día casi todos los fabricantes de ordenadores de propósito general han estandarizado la celda de 8 bits, llamada **byte**.

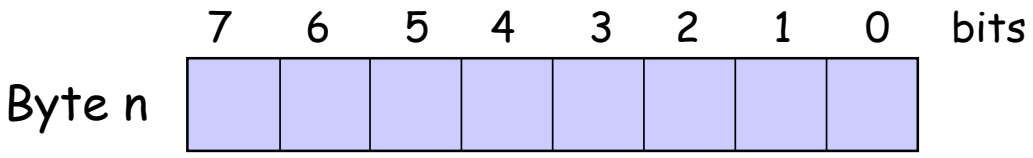
Un byte es capaz de almacenar datos tales como caracteres y valores numéricos pequeños, pero resulta insuficiente para manejar números de cierta magnitud. Por esto, los bytes se agrupan a su vez en **palabras**. El tamaño de una palabra viene determinado por el ancho de los registros generales de la CPU, por lo que un procesador con registros de 32 bits tiene una palabra de 32 bits. Esto significa que se pueden hacer operaciones (aritméticas, de movimiento, etc.) con datos de hasta 32 bits (en este caso).

El tamaño de las palabras ha pasado por los 8 y los 16 bits, creando los tipos de datos byte y *word* (palabra); años después, cuando se pasó a registros de 32 bits, para mantener los mismos tipos de datos, para referirse a los datos de 32 bits (4 bytes) se creó la doble palabra. En resumen, desde un punto de vista estricto, la palabra de un ordenador indica el ancho de los registros generales, pero en lenguaje ensamblador, para referirse a los tipos de datos que maneja, suele hablar de bytes (8 bits), palabras (2 bytes), dobles palabras (4 bytes), etc., aunque cada fabricante suele darles nombres distintos a estos tipos de datos.

Aunque parece que lo natural son palabras de 8, 16, 32, ... bits, ha habido palabras desde un modelo de Burroughs con una palabra de 1 bit, pasando por 8, 12, 16, 18, 24, 27, 32, 36, 48, los 60 bits del superordenador CDC de Cyber, hasta llegar a los 64 bits de los microprocesadores actuales.

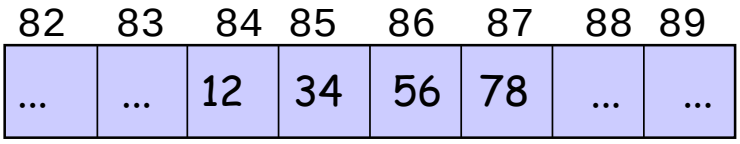
Obsérvese que como cada byte tiene su propia dirección de memoria, una palabra de varios bytes ocupa varias direcciones consecutivas de memoria. La dirección de una palabra viene determinada por la dirección del primero de sus bytes, esto es, el de dirección más baja.

Orden: Bits, Bytes, Palabras



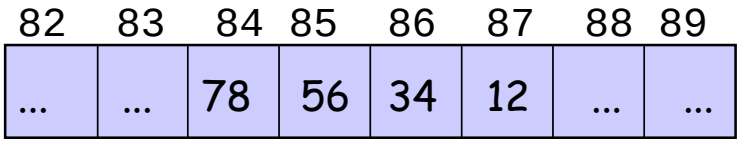
¿Cómo se ordenan los bytes de una palabra?

Dato: 12345678 H en la Dirección: 84



Big-Endian

El byte más significativo en la dirección más baja



Little-Endian

El byte más significativo en la dirección más alta

Ya hemos visto que la memoria es una colección de muchos bits agrupados, formando bytes, palabras, palabras largas o dobles palabras, etc. Veamos ahora cómo organizar los bits dentro de cada byte y cómo organizar los bytes dentro de cada palabra.

Los bits. Si vemos los bits de un byte como una ristra de ceros y unos de izquierda a derecha, para esa combinación de bits, que debe expresar un número, se debe tomar el acuerdo de cómo numerar los bits dentro del byte, y de establecer cuál es el peso de cada bit.

Aunque se pueden encontrar procesadores con cualquier orden, lo que se suele hacer es numerar los bits de derecha a izquierda, considerando que el bit de menor peso es el de orden cero, es decir, que el bit del extremo derecho, el de orden cero, es el de menor peso, y el bit del extremo izquierdo, el de orden siete, el de mayor peso.

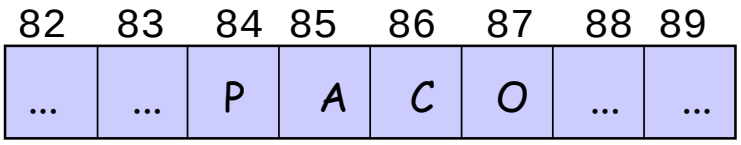
Los bytes. Supongamos que tenemos el número hexadecimal 12345678H almacenado en una palabra de 32 bits de un ordenador cuya unidad de direccionamiento es el byte, y que lo ponemos en la dirección 84. El valor consta de 4 bytes, donde el menos significativo contiene el valor 78, y el más significativo el valor 12. Pues bien, como se puede ver en la figura, hay dos formas de almacenar el valor 12345678:

- La primera opción, conocida como ***big-endian***, almacena el byte más significativo del número en la dirección más baja de memoria. Esto es equivalente al orden normal de escritura de los lenguajes occidentales.
- En la representación inferior, denominada ***little-endian***, es el byte menos significativo del número el que se almacena en el byte de la dirección más baja de memoria.

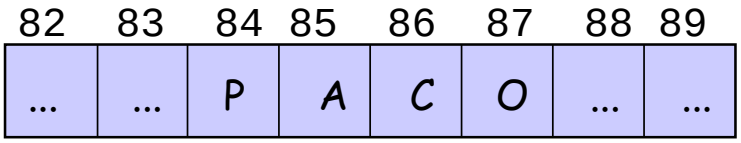
Como podemos ver, para un valor escalar compuesto por múltiples bytes, una opción es simplemente la ordenación contraria de la otra.

…Orden: Bits, Bytes, Palabras

Dato: "PACO" en la Dirección: 84



Big-Endian



Little-Endian

¿ Algún Problema ?

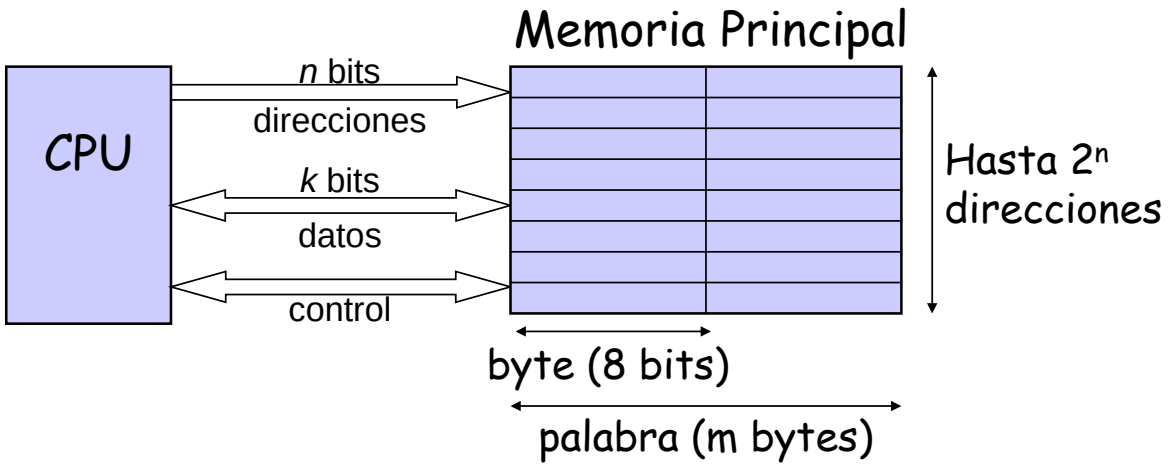
¡SI!
Al transmitir datos multi-byte entre máquinas con distintos modelos

Para los valores que ocupan un solo byte, como los caracteres, tanto *big-endian* como *little-endian* resultan en la misma representación.

El modelo *little-endian* es el utilizado por los procesadores de Intel, mientras que el *big-endian* tiene como seguidores a Motorola, a los *mainframes* de IBM, y a la mayoría de las máquinas RISC. El PowerPC es ***bi-endian***, puesto que es configurable y puede trabajar en ambos modos.

Aunque los partidarios de cada modelo esgrimen razones para preferirlo, no hay diferencias prácticas entre ellos. **El problema aparece cuando se transmiten datos entre dos máquinas con diferentes modelos de ordenación**, pues de no tenerse en cuenta este aspecto, los datos escalares de múltiples bytes se trastocarían completamente al pasar de una máquina a otra.

Características de la Memoria



Al hablar de una memoria se debe tener en cuenta

- ☞ Capacidad ≠ Espacio de direccionamiento
- ☞ Unidad Direccionable
- ☞ Palabra
- ☞ Unidad de Transferencia
- ☞ Tiempo de Acceso

La memoria principal suele estar formada por varios tipos de memoria; normalmente se encuentran, al menos, memorias de tipo RAM y ROM. Para todas ellas hay una serie de características o términos comunes que debemos aclarar:

Una característica obvia de la memoria es su **capacidad**. Ésta puede venir expresada por su número de celdas de memoria, por lo que en la mayoría de los sistemas basados en microprocesadores, viene expresada en bytes (1 byte = 8 bits). Otra forma de expresarla es indicando el número de celdas y el número de bits de datos de cada celda. Así, por ejemplo, 16 K x 8, quiere decir que tiene 16 K celdas direccionables, y cada una se compone de 8 bits, o lo que es lo mismo, 16 Kbytes.

Dirección o ubicación. Cada celda de memoria está asociada unívocamente a una dirección, de tal manera que mediante esa dirección se puede acceder a ella para leer o escribir un dato.

No se debe confundir el espacio de direccionamiento de un procesador con la cantidad de memoria disponible. El espacio de direccionamiento viene impuesto por el número de hilos, *n*, del bus de direcciones, de tal manera que el espacio de direccionamiento de un procesador es 2^n unidades direccionables, con lo que su rango de direcciones va de 0 a 2^n-1. La cantidad de memoria de un ordenador se corresponde solamente con la memoria realmente instalada. El máximo de memoria que se puede instalar es 2^n.

Unidad direccionable o resolución de acceso. Es el número de bits accesibles en cada dirección de memoria. La mínima unidad de almacenamiento es el bit, pero con solo dos valores posibles no resulta muy útil como unidad para el usuario, por eso se elige un tamaño mayor como mínima unidad de direccionamiento. La unidad de direccionamiento, es la mínima unidad de memoria a la que se puede hacer referencia mediante las señales de direcciones y selección de la CPU. Si bien el tamaño de la palabra es el apropiado para los números, hay otros tipos de datos para los que esta unidad resulta demasiado grande, por ejemplo los caracteres, que ocupan solamente un byte. Por este motivo, es normal que la unidad de direccionamiento sea el byte y no la palabra. Normalmente el espacio de direccionamiento, o capacidad, viene expresado en unidades direccionables.

Palabra. Su tamaño suele ser igual al máximo número de bits utilizado para representar los números en los registros internos de la CPU. Los primeros microprocesadores tenían palabras de 8 bits (como el 8080 y 8085 de Intel), el 8086 tenía una palabra de 16 bits, y la palabra del 68000 era de 32 bits. Los microprocesadores actuales tienen 64 bits, como el Pentium y los últimos modelos de PowerPC. Como vemos, la palabra de memoria viene establecida por la CPU, aunque, como veremos más adelante, influye directamente en el diseño de la organización de la memoria.

Unidad de transferencia. Es el máximo número de bits que pueden transferirse por el bus en cada operación de lectura o escritura en memoria. Este número viene impuesto por el número de hilos del bus de datos y, aunque suele ser igual a la palabra, también puede ser un submúltiplo de ésta.

A la hora de medir las prestaciones de una memoria debemos considerar el **tiempo de acceso**; es el tiempo necesario para realizar una operación de lectura/escritura, es decir, el tiempo que transcurre desde el instante en que se pone la dirección en el bus de direcciones hasta que el dato ha sido almacenado o puesto a disposición de la CPU. Actualmente las memorias estáticas tienen un tiempo de acceso del orden de 3 ns, mientras que las dinámicas están alrededor de 20 ns.

Algunos Ejemplos

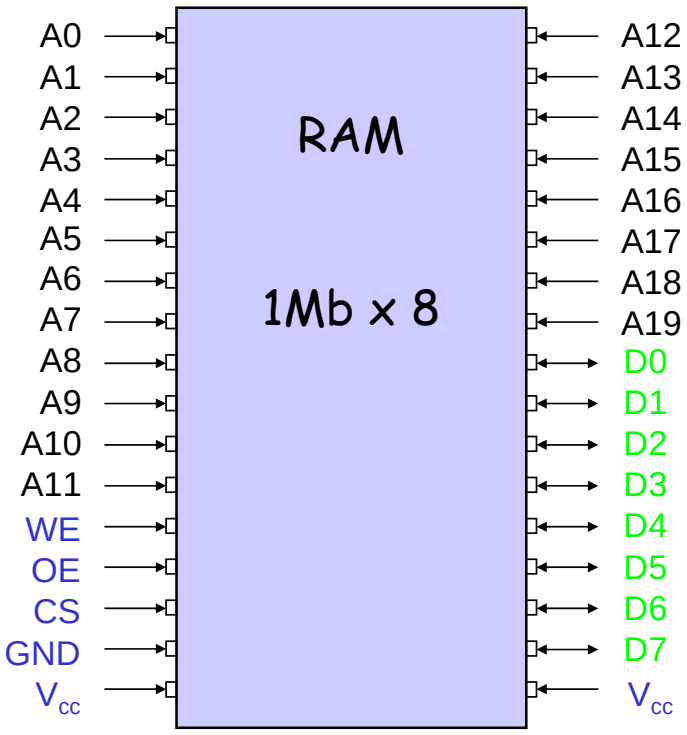
	Bus datos	Longitud registros	Unidad direccionable
8086	16	16	byte
8088	8	16	byte
486	32	32	byte
Pentium	64	64	byte
68000	16	32	byte
68020	32	32	byte
PowerPC 750	64	64	byte

Arquitectura de Computadores La Memoria - 10

Interfaz de una Memoria

Tiene patas de:

- ✓ Direcciones
- ✓ Datos
- ✓ Control



Aquí tenemos una pastilla de memoria RAM de 1 Mbit x 8 (1 Mbyte). Veamos la interfaz que nos ofrece para su conexión con el resto de los componentes del ordenador.

Patas de direcciones: A_0-A_{19}. Puesto que tiene un espacio de direccionamiento de 1 Mega (1.048.576 direcciones) necesita 20 patas de direcciones para poder seleccionar la celda deseada, pues 2^{20} = 1.048.576. Esta claro que estas señales son de entrada.

Patas de datos: D_0-D_7. Cada celda direccionada es un byte, luego necesita 8 hilos. Ya que esta pastilla es de memoria RAM, se puede leer o escribir en ella, por lo que estas patas son de entrada/salida, dependiendo de si la operación es escritura o lectura.

Señales de lectura/escritura: OE y WE. Cuando se desea realizar una operación de lectura, además de indicar la dirección correspondiente en las patas de dirección, debe activarse la señal OE (*Output Enable*). Si lo que se desea es una operación de escritura, debe activarse la señal WE (*Write Enable*). Obviamente, estas señales son mutuamente excluyentes.

Selección de pastilla: CS. Más adelante veremos que el mapa de memoria de un ordenador normalmente está compuesto por varias pastillas de memoria a las que llegan las señales de direcciones que salen de la CPU. Sin embargo, en cada operación concreta de lectura/escritura no debe leerse o escribirse en todas estas pastillas, sino solamente en la que corresponda según la dirección. Ya veremos que un módulo de decodificación se encarga de seleccionar la pastilla o pastillas que deben activarse al recibir una señal OE o WE, y para ello activará la señal CS (*Chip Select* o CE, *Chip Enable*) de la pastilla o pastillas que deban responder a la operación. Es decir, que esta señal le indica a la pastilla si debe responder o no a la operación de lectura/escritura que se arranca.

Tensión de alimentación: V_{cc}. Por esta patilla se recibe la tensión de alimentación de la pastilla.

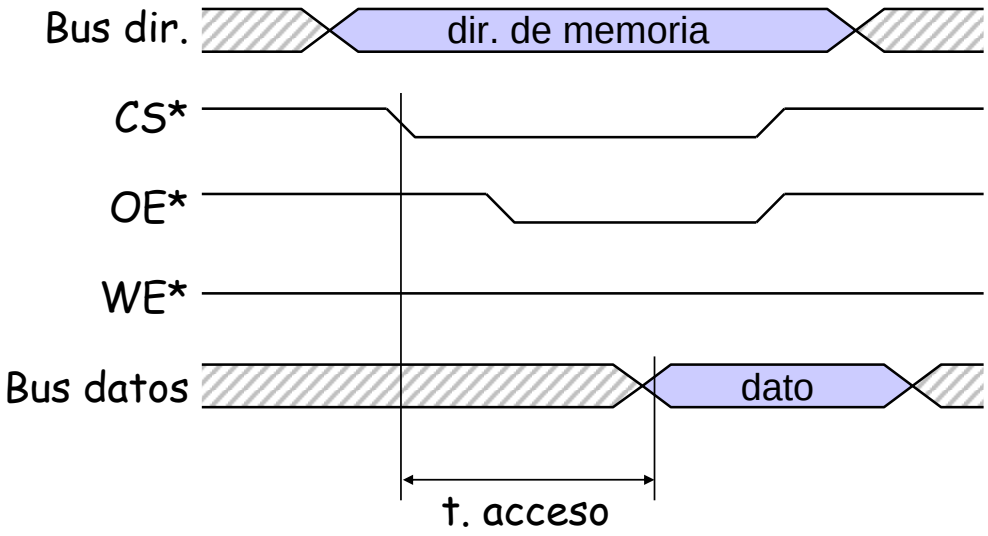
Tierra: GND. Esta pata se une a la toma de tierra (*Ground*) del ordenador.

Las patillas de alimentación o de toma de tierra suelen estar duplicadas en los encapsulados comerciales.

Las diferencias con otros tipos de pastilla suelen ser mínimas. En las pastillas ROM, por ejemplo, no está presente la señal *Write Enable*; y en las pastillas de tipo EEPROM (memoria no volátil de lectura/escritura) se dispone de una señal de tensión especial para la escritura.

Se debe tener en cuenta que al decir que una señal está activa no quiere decir que esa señal tiene 5 voltios, por ejemplo, pues depende de si la señal tiene lógica positiva o lógica negada. Esto quiere decir que una señal concreta puede estar activa al tener 5V o al estar a 0V o -5V, dependiendo de la lógica utilizada.

Cronograma de Lectura



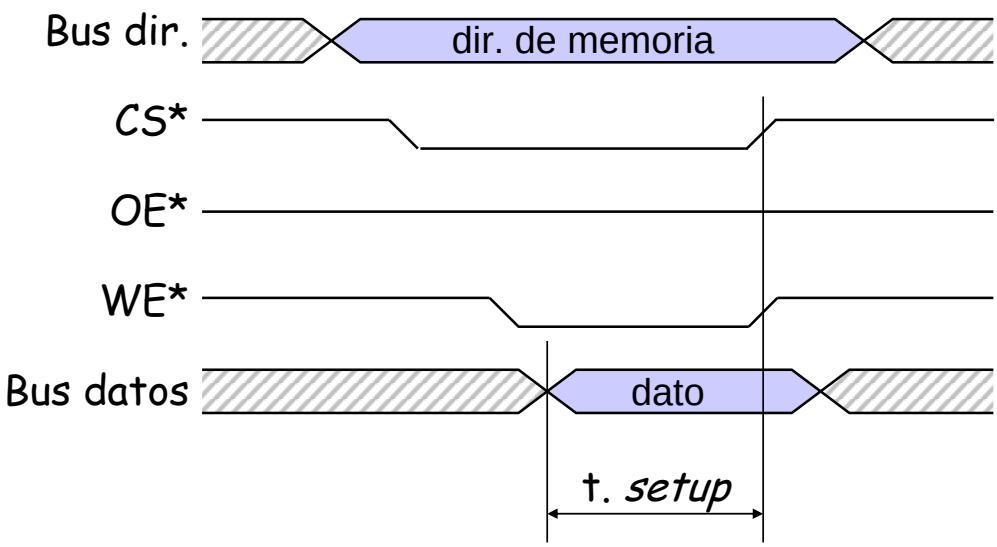
Vamos a ver a continuación los cronogramas simplificados de las operaciones de lectura y escritura en una pastilla genérica de memoria.

En un **ciclo de lectura** síncrona, todas las señales las activa el dispositivo que quiere realizar la lectura de memoria (generalmente, la CPU), excepto las señales de datos, ya que, al tratarse de una lectura, será la propia pastilla de memoria la que los active convenientemente.

En primer lugar se debe poner la dirección del dato a leer en las señales de direcciones; a continuación se activará la señal de selección de pastilla CS (o CE, *Chip Enabled*) y, por último, OE, la señal que indica que se trata de una lectura. Obviamente, la señal de escritura WE se mantiene desactivada durante toda la operación.

Pasado un cierto tiempo, la pastilla de memoria pone el contenido de la dirección de memoria indicada en las señales que componen el bus de datos. El tiempo máximo que transcurre desde que se activa la señal CS hasta que entrega el dato se denomina “tiempo de acceso”, y su duración depende de la tecnología utilizada. Como ya hemos comentado anteriormente, las memorias dinámicas (las normalmente utilizadas en RAM) tienen un tiempo de acceso entre 10 y 20 ns, mientras que las estáticas (las utilizadas para memorias caché) son de alrededor de 3 ns.

Cronograma de Escritura



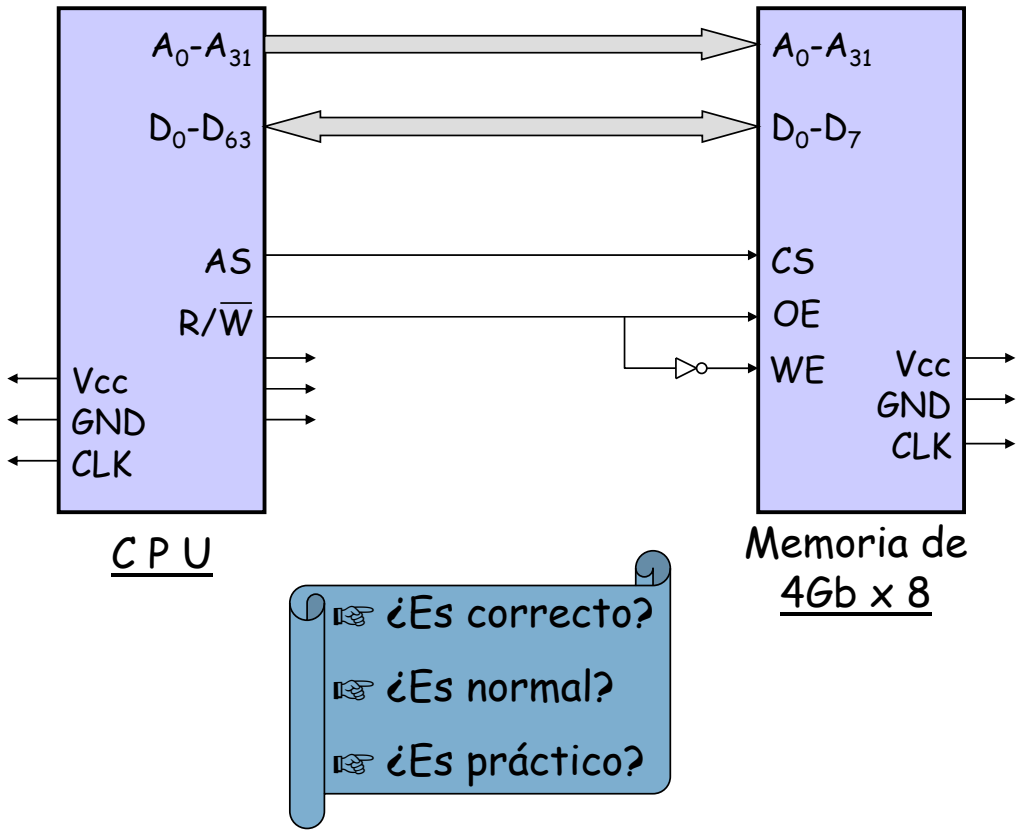
En el caso de un **ciclo de escritura** síncrona, todas las señales las activa el dispositivo que va a escribir en la memoria, incluidas, claro está, las señales de datos.

En la figura se puede ver el orden en el que se tienen que ir activando las distintas señales. Ahora es la señal OE la que se mantiene desactivada durante toda la operación.

Las pastillas de memoria exigen, generalmente, que el dato que se desea escribir esté presente en el bus de datos un tiempo mínimo (tiempo de “setup”) antes de desactivar las señales CS y WE, de tal forma que le de tiempo a la pastilla a capturar el dato correctamente.

Tanto al ciclo de lectura como al de escritura se los conoce de forma genérica como **Ciclo de Memoria** (o de Bus). Debe quedar claro que aunque tengan un nombre genérico común, un ciclo de lectura no tiene porqué durar lo mismo que uno de escritura; es decir, que los ciclos de memoria pueden tener distintas duraciones.

Interconexión CPU-Memoria



Una vez vista la interfaz que ofrecen la CPU y las pastillas de memoria, vamos a describir la interconexión entre ambos.

Supongamos un procesador más o menos actual, con 64 hilos en el bus de datos, y 32 en el de direcciones, lo que quiere decir que cuenta con un espacio de direccionamiento de 4 Gbytes. Por otra parte, supondremos una memoria con 2^{32} bytes, es decir, los 4 Gbytes.

Ahora debemos conectar los 32 hilos de direcciones de la CPU con los 32 de la memoria y los de datos de la CPU con los de datos de la memoria. También hay que conectar la salida AS (*Address Strobe*) de la CPU con la entrada CS (*Chip Select*) de la memoria, de tal forma que cuando la CPU dé por estables los valores del bus de direcciones, se seleccione la pastilla de memoria. Por último, nos queda conectar la salida R/W de la CPU a las entrada OE (*Output Enable*) y WE (*Write Enable*) de la memoria, para indicar cuándo se desea leer o escribir en ella.

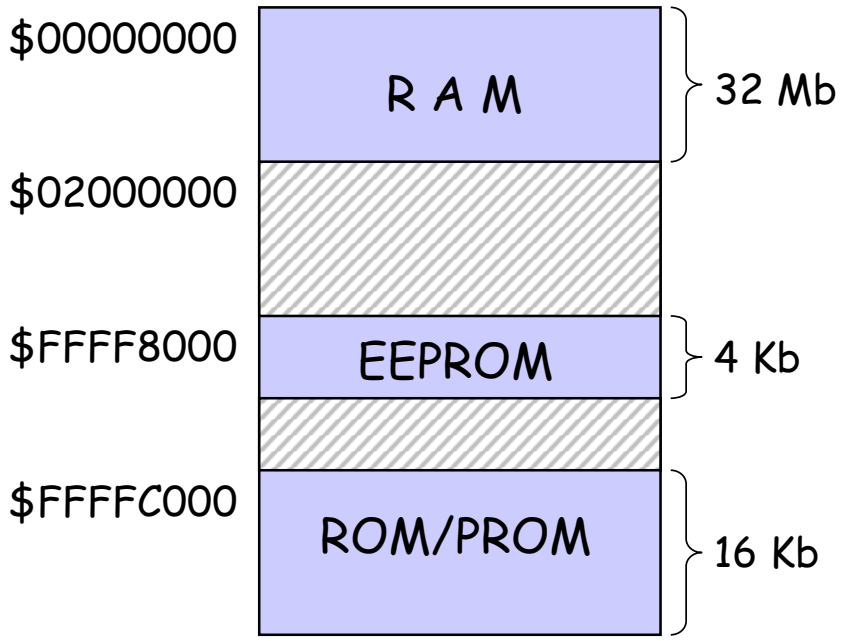
Esto parece fácil y razonable, pero hay algo que no hemos dejado claro. El procesador tiene 64 hilos de datos, y la memoria por su parte ofrece 2^{32} unidades de direccionamiento de 8 bits cada una, luego entonces debe tener 8 hilos de datos. Entonces ¿cómo se conectan los 64 hilos de datos del procesador a los 8 de la memoria?

Hay más cuestiones todavía.

- ¿Es normal que se desee rellenar todo el espacio de direccionamiento (los 4 Gbytes) con memoria?
- ¿Es práctico que todo el espacio de memoria disponible sea del mismo tipo (RAM, ROM, ...)?

Como vemos, antes de realizar la conexión de la CPU con la memoria debemos solucionar algunas cuestiones prácticas que todavía no hemos tratado. Vamos a comentarlas a continuación.

El Mapa de Memoria



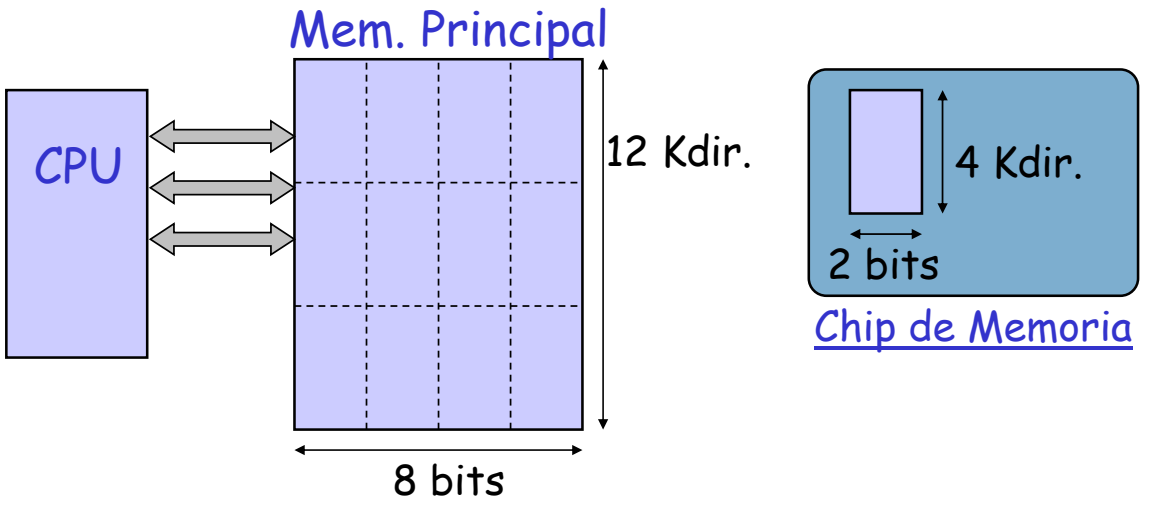
Los ordenadores personales convencionales suelen contar, actualmente, con un vasto espacio de direccionamiento (2^{32} bytes, 2^{40} y 2^{52}); pero no suele ser necesario llenar todo este espacio de direccionamiento con los chips de memoria correspondientes, sino que solamente se cubre en parte; por ejemplo, con 512 Mbytes, 1 ó 2 GBytes. (Aunque aquí no nos ocuparemos de ellos, los *mainframes* y los supercomputadores disponen de una memoria principal de varios gigabytes).

Por otra parte, también resulta normal contar con varios tipos de memoria: la mayor parte de RAM, que suele comenzar en las direcciones bajas; y un poco de ROM o alguna variante (con el programa de arranque) que suele estar en las direcciones más altas de memoria. En algunos sistemas también se puede contar con una pequeña cantidad de memoria EEPROM que puede estar situada en cualquier parte del hueco existente entre la RAM y la ROM.

Así, nos encontramos con que el espacio de direccionamiento está formado por varias zonas o rangos de direcciones en las que hay distintos tipos de memoria y huecos en los que no hay instalada ningún tipo de memoria.

El **Mapa de Memoria** representa la distribución del espacio de direccionamiento de una máquina entre los distintos tipos de memoria instalados.

Organización de los Módulos



Hay que agrupar chips para conseguir

- El espacio de direccionamiento
- La longitud de la celda de memoria

La pastilla de memoria que hemos visto recientemente (1 Mbit x 8) se adapta muy bien a las necesidades normales de un ordenador, sin embargo, no suelen ofrecerse comercialmente. En su lugar se ofrecen pastillas con otras combinaciones, como 2 Mbit x 4, o simplemente de 1 Mbit x 1.

Las pastillas de memoria se comercializan con un número de celdas de memoria (direcciones) y un tamaño de celda (bits por celda) que no tienen por qué coincidir con el tamaño de la palabra de la CPU ni con las necesidades de memoria que hay que instalar. Por esto, es normal que haya que agrupar varias pastillas de memoria para ofrecerle a la CPU la imagen de una única memoria con tantas direcciones como se desea y con un tamaño de celda igual al de la palabra de la CPU.

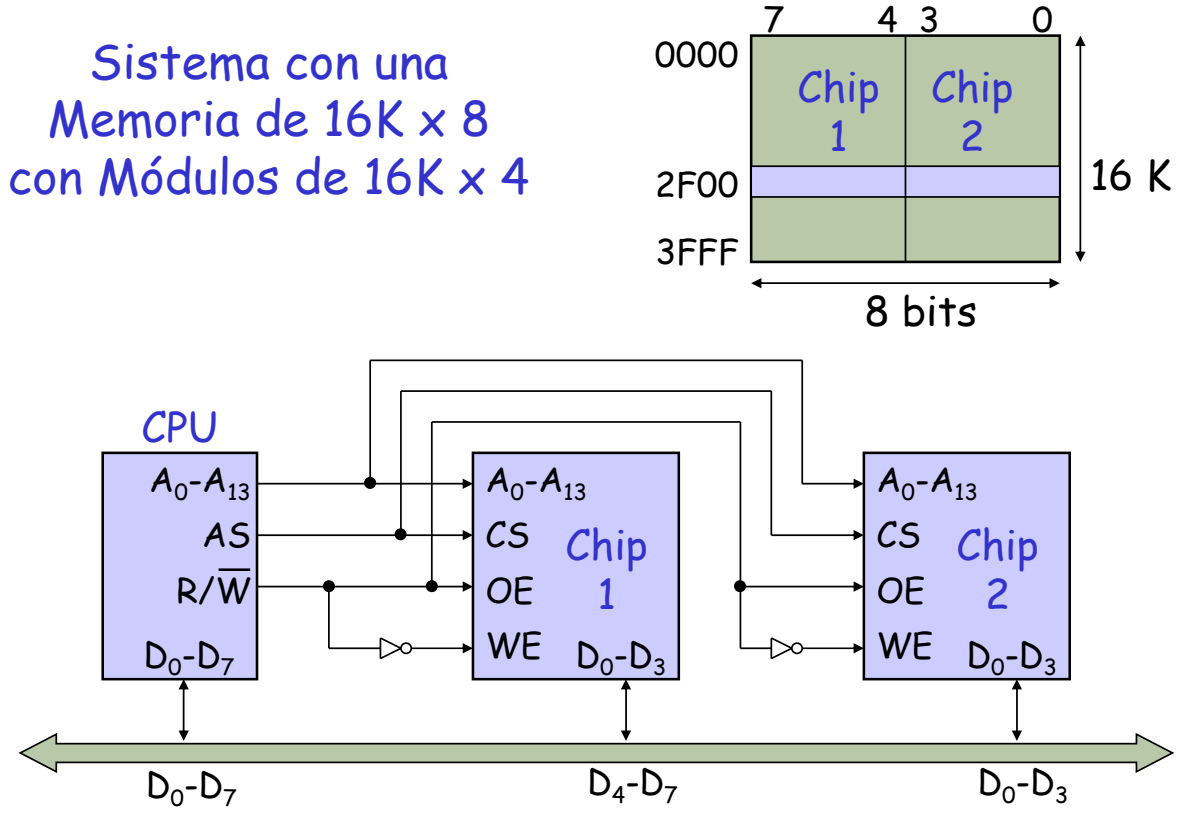
Aquí mostramos el ejemplo de un ordenador al que se le quiere dotar de un espacio de direccionamiento de 12 Kbytes. Pero resulta que en el mercado no está disponible una pastilla de memoria de 12 K x 8; supondremos que lo más parecido disponible son pastillas de 4K x 2 bits.

Así pues, las pastillas de memoria vamos a tener que agruparlas por dos motivos:

1. Para conseguir el espacio de memoria (direcciones) deseado.
2. Para conseguir el tamaño de celda o unidad de direccionamiento que ve la CPU.

Vamos a ver algunos ejemplos en los que se muestra cómo deben agruparse las pastillas o módulos de memoria para conseguir la memoria deseada.

Agrupación por Longitud de Celda



EJEMPLO 1: Agrupación por Longitud de Celda

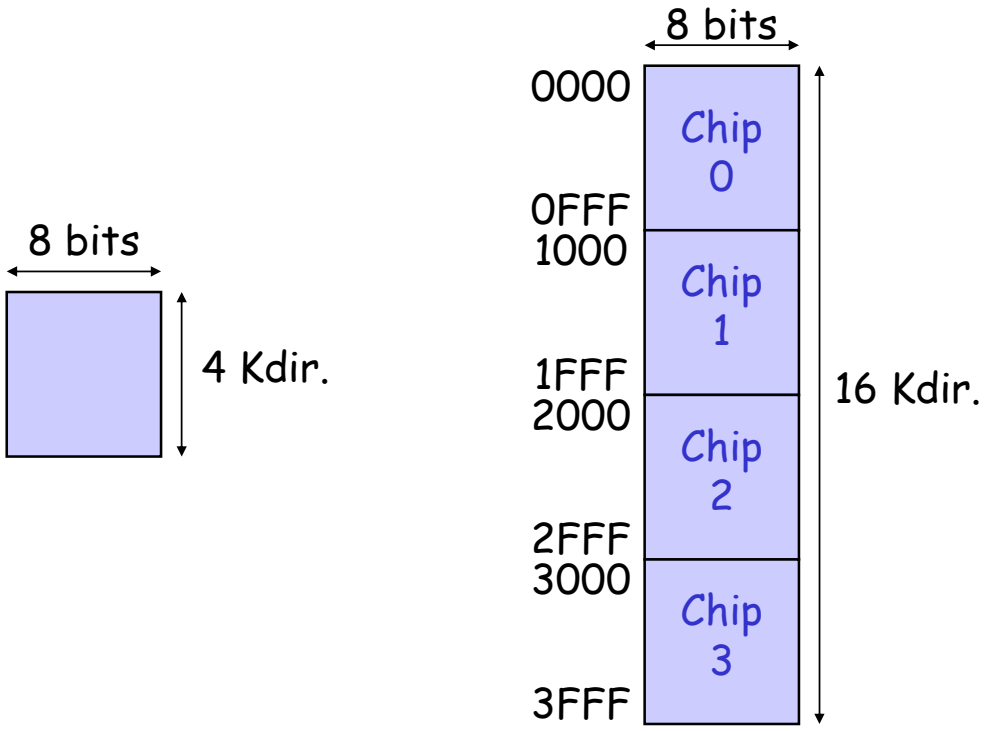
Supongamos que se desea conseguir un banco de memoria de 16 Kpalabras de 8 bits (es decir una memoria de 16K x 8); y para ello disponemos de módulos de 16K x 4 (4 bits de datos cada celda).

Vemos que cada módulo de memoria cubre el espacio de direccionamiento requerido (16K direcciones), pero solamente con la mitad de la longitud de una celda en cada dirección. Así pues, lo que tendremos que hacer es agrupar convenientemente dos módulos de 16K x 4 para conseguir una visión de 16K x 8.

Por una parte, lo que tendremos que hacer es unir directamente las patas de direcciones de las dos pastillas, puesto que ambas tienen 16K direcciones. Por otra parte tendremos que formar palabras de 8 bits a partir de dos bloques de 4. Para ello simplemente tomaremos los 4 bits de datos de la pastilla 1 y los consideraremos como los 4 bits de mayor peso de la palabra, y los 4 bits de datos de la pastilla 2 serán los 4 bits de menor peso de la palabra.

Así, cuando la CPU realice una lectura de la dirección 2F00, ambas pastillas se seleccionarán (CS activo); la primera pastilla aportará los 4 bits de su dirección 2F00 a los 4 bits mayor peso, y la segunda pastilla aportará los 4 bits de su dirección 2F00 a los 4 bits de menor peso de la palabra. De esta manera, la CPU habrá leído una palabra de 8 bits de la dirección 2F00 de su espacio de direcciones.

Agrupación por Direcciones



Supongamos que para implementar el mismo bloque de memoria de 16K x 8, disponemos ahora de módulos de 4K x 8.

Como vemos, cada módulo ahora ofrece palabras completas (de 8 bits), pero no disponemos de un módulo que ofrezca 16 Kpalabras, por lo que tendremos que agrupar 4 pastillas de tal forma que cada una de ellas nos ofrezca 1/4 del espacio de direccionamiento requerido, consiguiendo los 16K en total (0 - 3FFF).

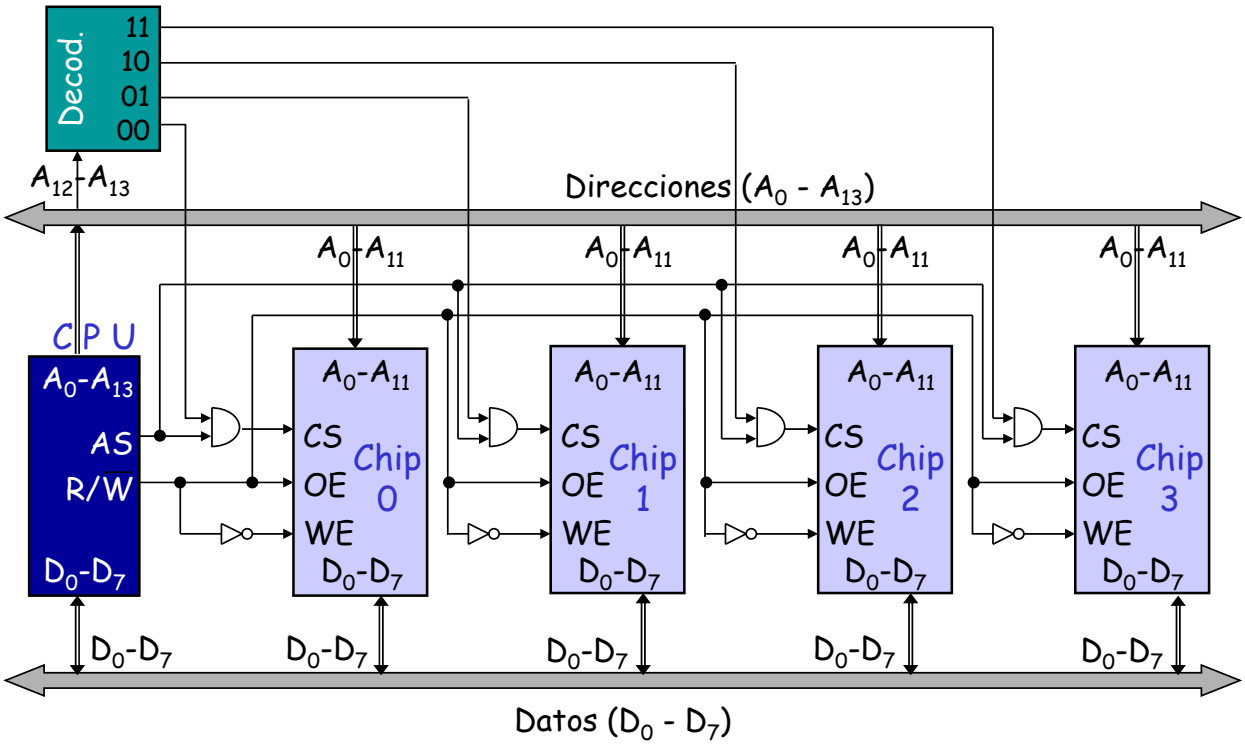
Lo que parece razonable es hacer que un módulo de memoria contenga el primer cuarto de direcciones, otro módulo el segundo cuarto, y otros dos módulos para los dos últimos cuartos del espacio de direccionamiento. Así tendríamos lo siguiente:

Direcciones	0000 - 0FFF	→ en Módulo 0
"	1000 - 1FFF	→ en Módulo 1
"	2000 - 2FFF	→ en Módulo 2
"	3000 - 3FFF	→ en Módulo 3

En la siguiente página podemos observar la interconexión de estos módulos de memoria con la CPU.

...Agrupación por Direcciones

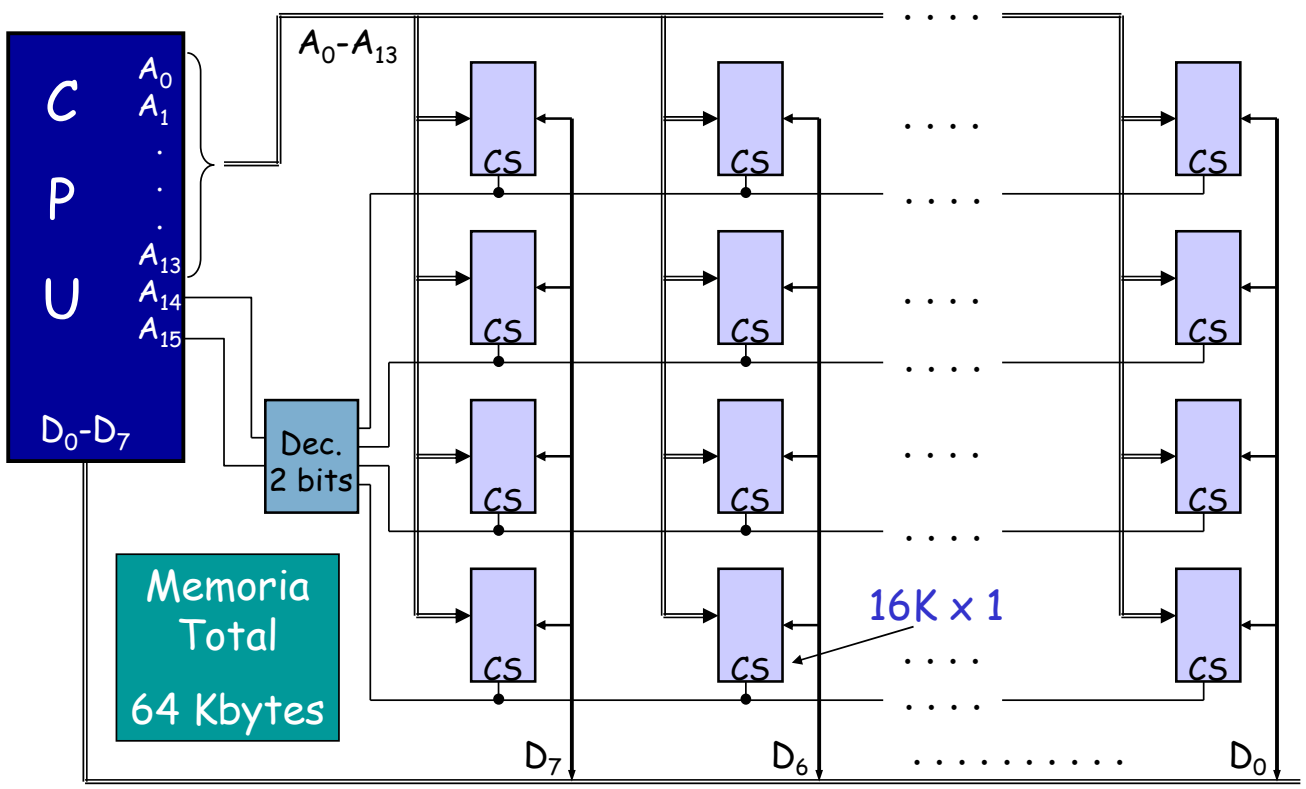
Sistema con una Memoria de 16K x 8 con Módulos de 4K x 8



En este caso, cada pastilla entrega los 8 bits de cada palabra, pero no se tienen que seleccionar todas simultáneamente, sino que solamente tendrá que seleccionarse la que contenga la dirección referenciada por la CPU.

En estas situaciones en las que cada uno de los espacios de direccionamiento está repartido entre múltiples pastillas de memoria, se requiere un mecanismo que sepa averiguar el módulo que contiene la dirección referenciada por la CPU y activar únicamente la señal *Chip Select* de ese módulo. A este mecanismo se le conoce como **decodificación de direcciones**.

...Organización de los Módulos



En este ejemplo tenemos un sistema con una memoria de 64 Kbytes formada a base de módulos de 16 K x 1, esto quiere decir que cada chip dispone de 14 patas de direcciones y una de datos.

Vamos a tener 4 bancos (filas) de 16 Kdirecciones cada uno, y cada banco, a su vez, debe estar formado por 8 pastillas, ya que cada una de ellas entrega un bit. Así pues necesitaremos 4x8 pastillas, o sea, 32 en total.

En cuanto a los hilos de datos, puede verse que cada columna de pastillas entrega un bit de datos del mismo peso, (el D_7, el D_6, ...), pero como en un momento dado solamente estará activada una pastilla por columna, el resto de las pastillas de la columna se quedarán en alta impedancia, y no aportarán nada al bus de datos.

Ya que cada pastilla tiene 16 Kdirecciones, dispone de 14 hilos de direcciones, mientras que de la CPU salen los 16 hilos correspondientes a un direccionamiento de 64 K. Lo primero que debemos hacer es conectar los 14 hilos de direcciones de menor peso (A_0-A_{13}) a las 14 patas de cada módulo de memoria.

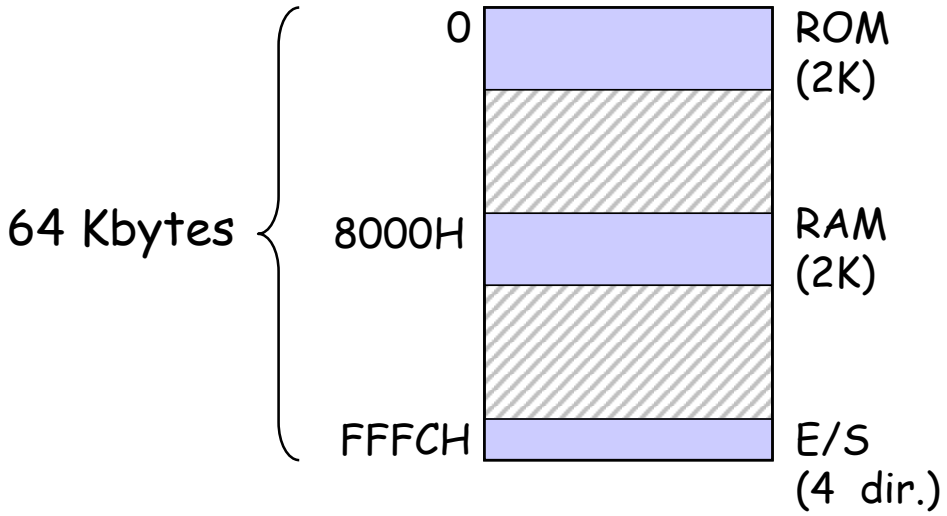
Pero claro, con esta conexión resulta que para cualquier dirección que genere la CPU se seleccionarán todas las pastillas de memoria, mientras que nosotros queremos que para una dirección en el rango 0-3FFF se seleccionen solamente las 8 pastillas que forman los bytes de las 16 Kdirecciones primeras; si la dirección está en el rango 4000 - 7FFF, se deben seleccionar las 8 pastillas del segundo banco de memorias; para las direcciones de 8000 - BFFF, las 8 pastillas del tercer banco; y, por último, si la dirección está en el rango C000 - FFFF, se deben seleccionar solamente las 8 pastillas del último banco.

Pues bien, si nos fijamos en los 16 hilos de direcciones que salen de la CPU, las señales que indican el banco de direcciones de 16 K que se está referenciando, son las dos señales de mayor peso, la A_{14} y la A_{15}, con las siguientes combinaciones:

A_{15}	A_{14}	**Banco seleccionado**
0	0	1º (0000-3FFF)
0	1	2º (4000-7FFF)
1	0	3º (8000-BFFF)
1	1	4º (C000-FFFF)

Lo único que queda por hacer es meter estas dos señales en un decodificador de 2 bits que se encargue de activar una de sus 4 salidas, de tal manera que cada una de ellas se conectará a la pata CS (*Chip Select*) de todas las pastillas que conforman el banco correspondiente de 16 K. Obsérvese que para activar una pastilla no basta con conectar la salida del decodificador a la patilla CS, sino que para activar una pastilla debe estar activa la señal correspondiente del decodificador más la señal AS (*Address Strobe*) de la CPU.

Decodificación de Direcciones

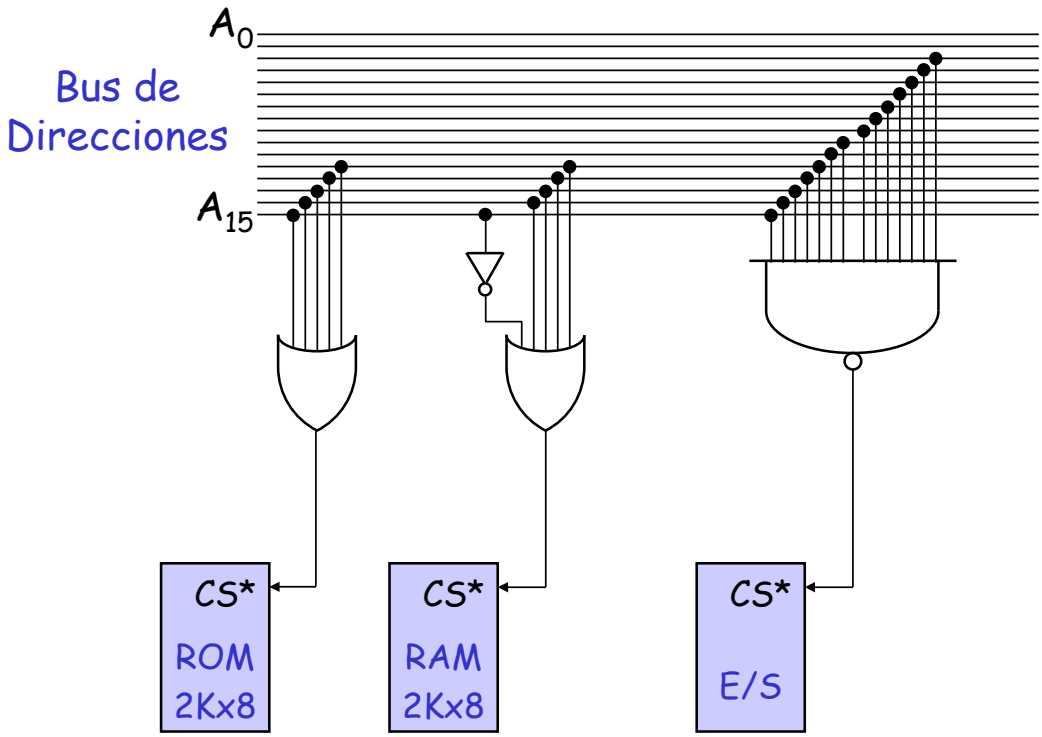


En la página anterior hemos descubierto la necesidad de decodificar direcciones para saber qué pastillas hay que activar en cada referencia a memoria.

Hemos visto que la decodificación de direcciones es necesaria incluso en espacios de direccionamiento ocupados por múltiples pastillas de memoria del mismo tipo (RAM, ROM, ...). En la práctica lo que sucede es que el espacio de direccionamiento suele estar ocupado solo parcialmente y, además, por diferentes tipos de memoria o puertos de controladores de periféricos *mapeados* en memoria (ocupando direcciones del espacio de direccionamiento normal de la CPU), con lo cual se añade un motivo más para la decodificación de direcciones.

Veamos como ejemplo sencillo un sistema con un espacio de direccionamiento de 64 K, ocupado por 2 Kbytes de memoria ROM a partir de la dirección 0, y otros 2 Kbytes de RAM a partir de la dirección 8000H. Además, las últimas 4 direcciones están utilizadas para hacer referencia a los 4 registros o puertos del controlador de un dispositivo periférico.

Decodificación Total



Según el mapa de memoria que hemos establecido en la transparencia anterior, la memoria ROM deberá seleccionarse por cualquier dirección binaria del tipo 00000xxxxxxxxxx. Es decir, siempre que los 5 bits de mayor peso estén a cero indicará que la dirección está dentro de los dos primeros Kbytes de memoria. Dado que normalmente la señal CS se activa por lógica inversa (a cero), la selección de la ROM puede conseguirse entonces mediante una puerta OR de cinco entradas, de tal manera que solamente cuando las cinco estén a 0, el resultado será un 0.

El mismo principio puede aplicarse a la memoria RAM, cuyas direcciones tendrán el formato 10000xxxxxxxxxx. En este caso, la selección de esta pastilla se puede conseguir con la misma lógica que la de la ROM, más un inversor para la señal de mayor peso a la entrada de la OR.

La decodificación de las direcciones de los puertos es igual de sencilla, pues al tener las direcciones del tipo 11111111111111xx, su activación se consigue con una puerta NAND a la que entran las 14 señales de direcciones de mayor peso.

La única pega práctica que presentan estas soluciones es que no se comercializan las pastillas OR que hemos comentado. Lo que sí está disponible son pastillas NOR, NAND e inversores, con las cuales puede obtenerse cualquier lógica *booleana*, teniendo en cuenta las leyes de De Morgan:

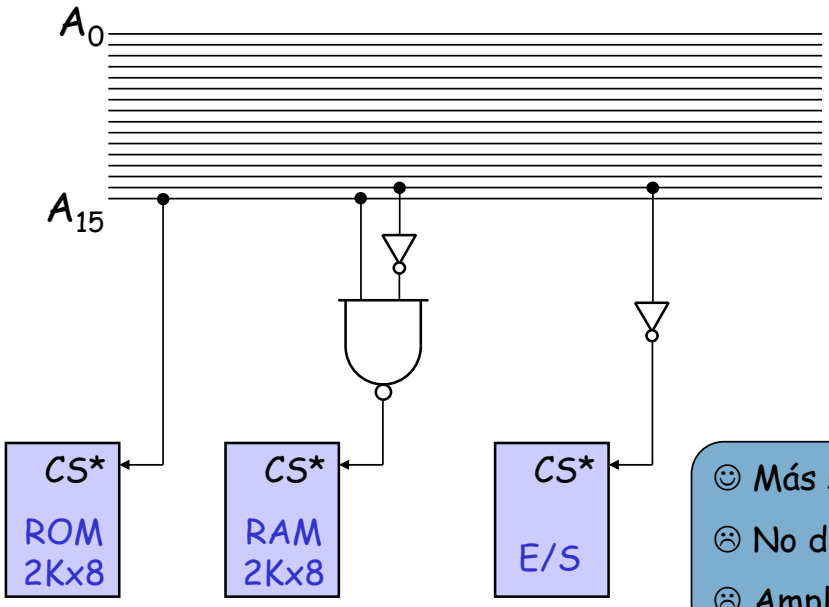
$$a + b = \overline{\overline{a} \cdot \overline{b}}$$

$$a \cdot b = \overline{\overline{a} + \overline{b}}$$

Esto que acabamos de ver es una **decodificación total de direcciones**, ya que para la selección de cada pastilla intervienen todas las señales de direcciones que identifican directa y unívocamente tal pastilla.

Decodificación Parcial

Direcciones ROM: **0**000 0xxx xxxx xxxx
Direcciones RAM: **10**00 0xxx xxxx xxxx
Puertos de E/S: 1**1**11 1111 1111 11xx



☺ Más simple
☹ No detecta dirs. inexistentes
☹ Ampliaciones más difíciles

En el caso del ejemplo anterior, hemos necesitado las 5 señales de dirección de mayor peso para identificar la ROM y la RAM, y las 14 de mayor peso para identificar las direcciones de los puertos de E/S.

Sin embargo, podríamos utilizar una artimaña para tratar de simplificar el número de señales implicadas en la decodificación. Se trata de observar las diferencias entre los distintos grupos de direcciones, y utilizar lógica para decodificar direcciones basándose solamente es esas diferencias.

Veamos nuestro ejemplo. Tenemos que las direcciones ROM, y solamente las direcciones ROM, tienen de particular que su señal de direcciones más alta (A_{15}) siempre es 0. Así, podríamos conectar directamente A_{15} a la señal CS.

En cuanto a la memoria RAM, tenemos que sus direcciones son las únicas cuyas señales de direcciones A_{15} y A_{14} son, respectivamente, 1 y 0.

Por último, nos encontramos con que las direcciones de los puertos de E/S son las únicas cuya señal A_{14} es siempre 1.

Como podemos ver, tenemos que para distinguir los tres grupos de direcciones y poder seleccionar la pastilla adecuada, nos basta con utilizar solamente las dos señales de dirección más altas, lo cual se consigue, como se puede ver en la figura, con solo una puerta NAND y dos inversores.

Este método de decodificación de direcciones se denomina **decodificación parcial de direcciones**, puesto que no intervienen todas las señales que definen cada grupo de direcciones.

Esto tiene una pega: Una lectura de cualquiera de estas direcciones

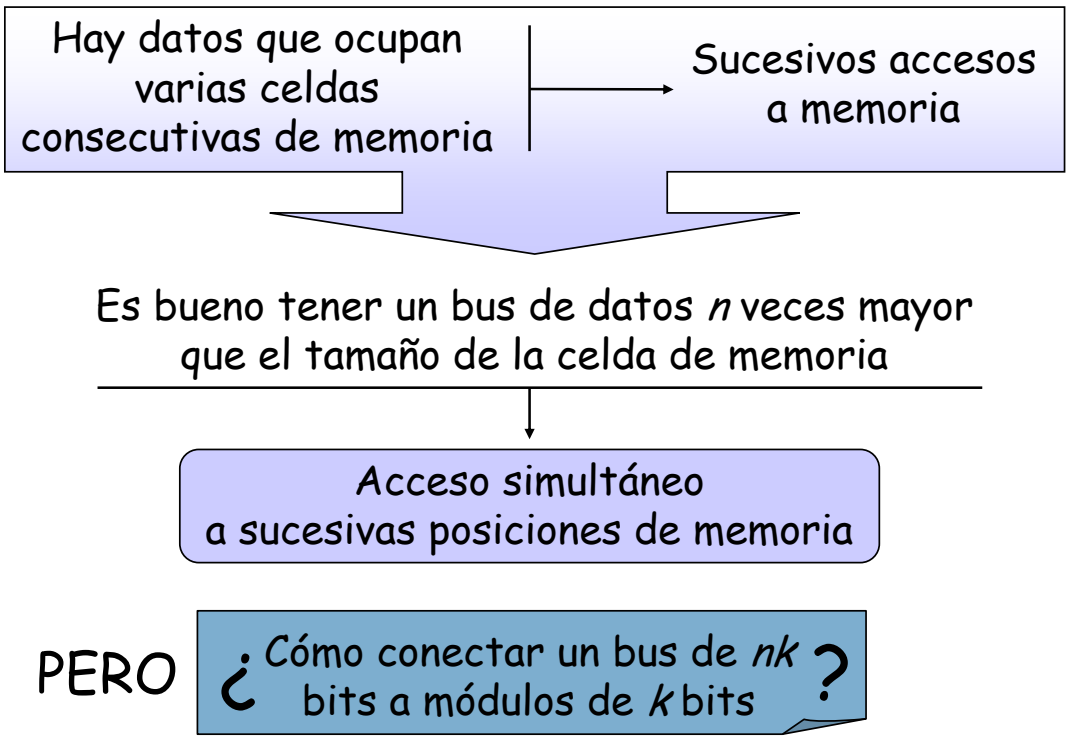
0001 0000 0000 0000
0001 1000 0000 0000
0010 0000 0000 0000

producirá el mismo resultado. De hecho, cualquier lectura de la mitad inferior de las direcciones (sin variar los 11 bits de menor peso) producirá el mismo efecto, pues cualquiera de ellas seleccionará la pastilla ROM, y con los mismos 11 bits de direcciones. Por esto, hay que pensar que quizás sería mejor detectar las referencias a direcciones de memoria no existentes. Si se piensa que el ordenador puede ampliarse en el futuro con otras zonas de memoria, se nos puede quedar inservible la lógica de decodificación; así, al añadir más módulos de memoria, con decodificación total solamente hay que añadir la lógica correspondiente a los módulos de memoria añadidos, mientras que con decodificación parcial, es muy posible que haya que rehacer toda la lógica general de decodificación de direcciones.

La decodificación total de direcciones, aunque es más compleja y costosa, **tiene dos ventajas**:

- Detecta referencias a direcciones inexistentes de memoria.
- Permite ampliaciones del mapa de memoria con otras zonas de memoria.

...Organización de los Módulos



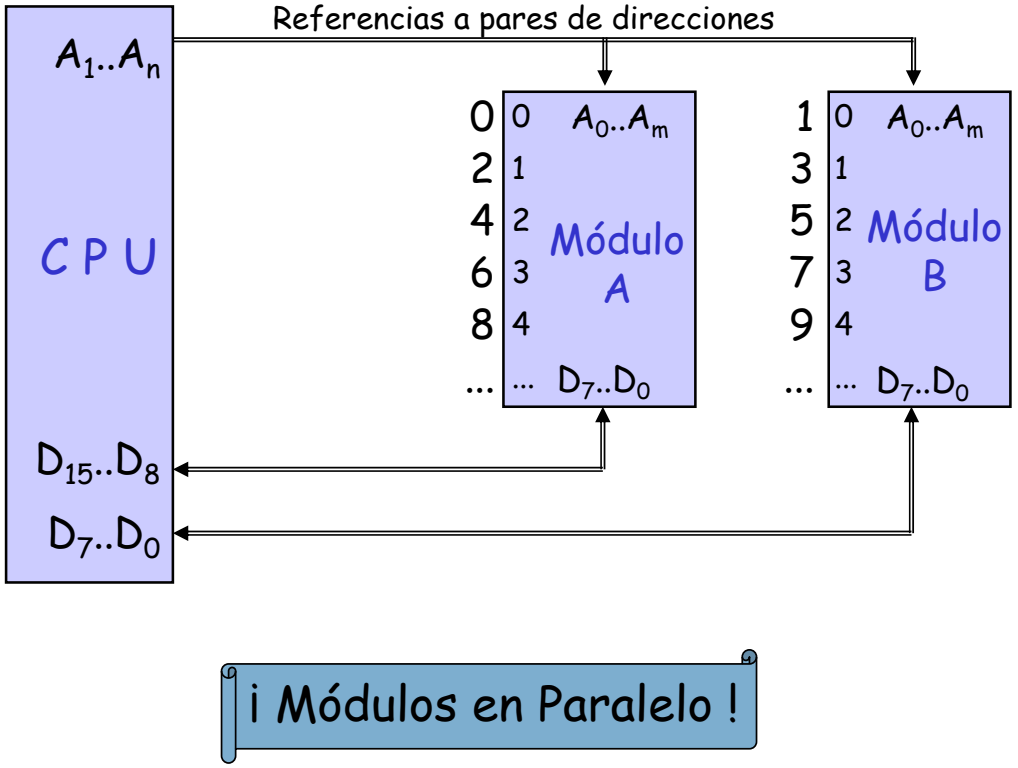
Ya hemos visto que un módulo básico de memoria está formado por una o varias pastillas de memoria, tal que, en conjunto, ofrece una serie de celdas de direcciones consecutivas de *k* bits cada una y que, por lo tanto, el módulo básico tiene *k* hilos de datos para su conexión al bus de datos del procesador.

Por otra parte tenemos que los procesadores suelen contar con registros generales que son de mayor tamaño que las celdas de memoria, para poder trabajar con datos que ocupan varias celdas (palabras, dobles palabras, etc.). Así, por ejemplo, cuando se va a leer un dato que ocupa varias celdas de memoria, se deben realizar sendas lecturas de celdas consecutivas en memoria. Por esto último, es normal que los procesadores tengan un bus de datos que es *n* veces más ancho que el tamaño de la celda de memoria, para así poder realizar un acceso simultáneo a *n* celdas con direcciones consecutivas en memoria.

Supongamos el siguiente escenario: Tenemos un procesador cuya resolución de acceso a memoria es a celdas de 8 bits, por lo que nos hacemos con módulos de memoria de celdas de 8 bits y, por lo tanto, con 8 hilos de datos. Pero ahora nos encontramos con que el bus de datos del procesador es de 16 hilos, para poder realizar accesos simultáneos a datos que ocupen 2 direcciones consecutivas de memoria. **¿Cómo conectamos el bus de 16 hilos del procesador a los módulos de memoria de 8 hilos de datos?**

Veamos en la página siguiente cómo se resuelve esto.

...Organización de los Módulos



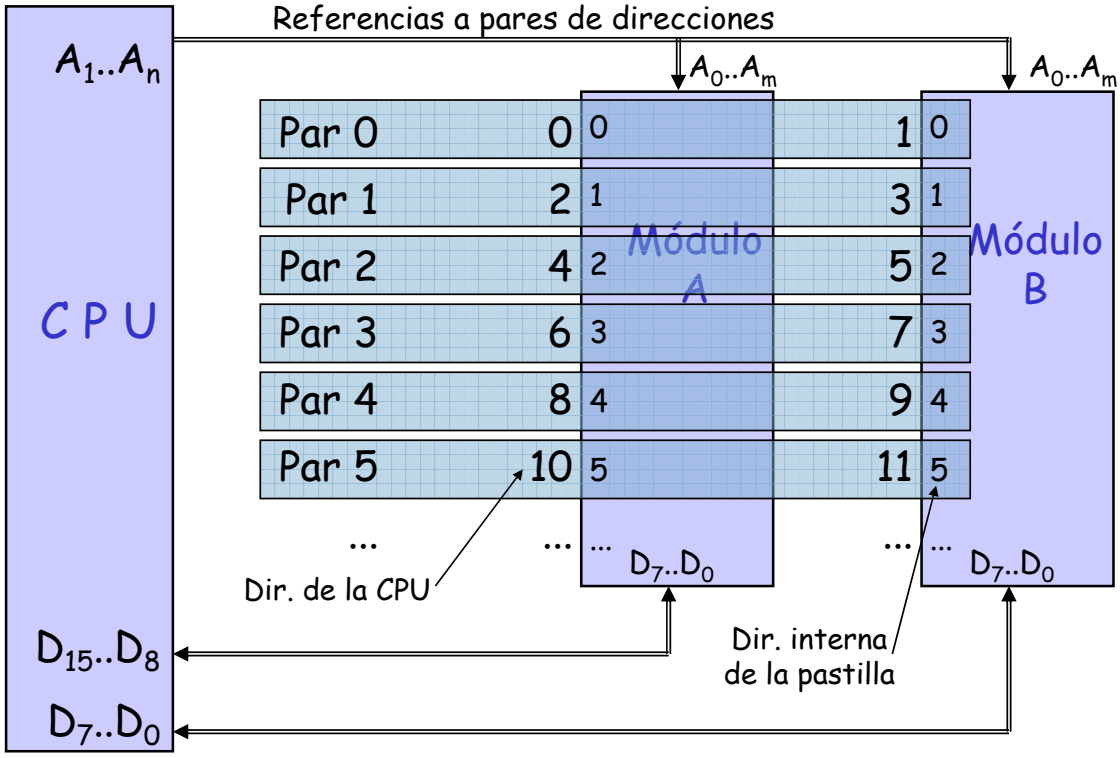
¡ Módulos en Paralelo !

A primera vista, ya parece obvio que no se pueden conectar directamente 16 hilos del procesador a los 8 del módulo de memoria, por lo que parece que se necesitan más módulos de memoria. Entonces la respuesta es fácil: "conectando los ocho hilos de menor peso del bus de datos del procesador a un módulo de memoria, y los otros ocho hilos de mayor peso a otro módulo". No obstante hay que tener en cuenta una cosa: **los 16 hilos del bus de datos esperan acceder a 2 celdas de memoria de direcciones consecutivas**, pero como hemos tenido que conectar los 16 hilos de datos a dos módulos distintos, tenemos que **las direcciones de un módulo de memoria no pueden representar direcciones consecutivas del espacio de direccionamiento del procesador**, pues, por ejemplo, la dirección 0 estará en el módulo A y la dirección 1 en el módulo B, la 2 en el A y la 3 en el B, ... Es decir, que un módulo tendrá las direcciones pares y otro las impares.

Debemos sacar en conclusión que si el ancho del bus de datos del procesador es *n* veces mayor que la celda de memoria, los módulos de memoria deben organizarse montándose en series de *n* módulos en paralelo, de tal manera que, desde el punto de vista de la CPU, las direcciones del primer módulo serán: *0, n, 2n, 3n*, ...; las direcciones del segundo módulo serán: *1, n+1, 2n+1, 3n+1*; las direcciones del tercero: *2, n+2, 2n+2, 3n+2, ...* etc.

El ancho del bus de datos de los procesadores suele ser una potencia exacta de 2 y tiende a ser igual a la longitud de la palabra, por lo que para celdas de 8 bits, el bus de datos puede ser 8, 16, 32, 64, ...

...Organización de los Módulos



Según lo visto en la página anterior, en realidad, para arquitecturas con un bus de datos de 16 bits, desde la CPU se hace referencia a “pares de direcciones”, donde el par 0 se refiere a las direcciones 0 y 1; el par 1, a la 2 y la 3; el par 2, a la 4 y la 5, etc. etc.

Ya que, externamente, la CPU no hace referencias a direcciones de celdas individuales, sino a pares de direcciones, lo que debe salir por sus patas de direcciones es el “número de par” de direcciones a cuyo contenido se quiere acceder.

Si la CPU, internamente, quiere acceder a una dirección concreta ¿que valor tiene que sacar por sus patas de direcciones? Es fácil, se divide la dirección por 2 y se toma el cociente entero. Por ejemplo, la dirección 7 está en el par 3, luego por las patas de direcciones se debe sacar el valor 3, que hace referencia a las direcciones del par 3, es decir, a la dirección 6 y la 7.

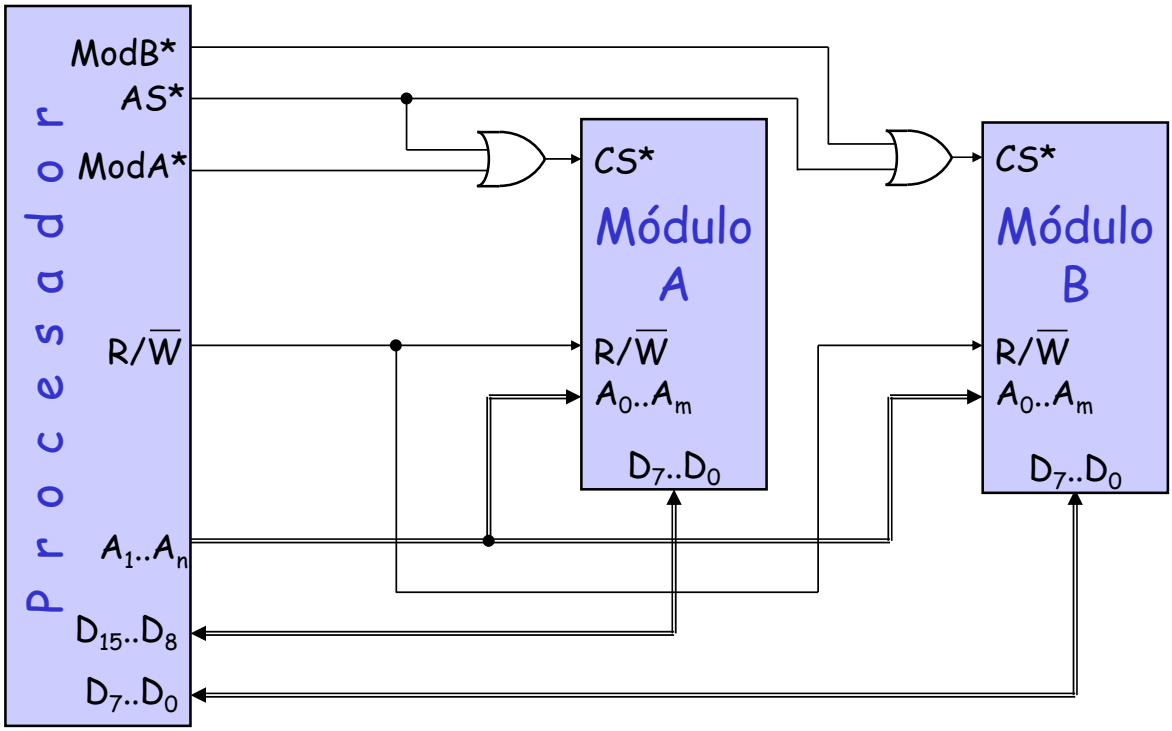
Ya hemos visto que para calcular el número de par la CPU simplemente tiene que dividir por 2. Pero esto se puede simplificar más. Con una numeración en base 2 (la binaria), para dividir un número entre un valor n que sea potencia de 2, simplemente hay que realizar un desplazamiento a la derecha de $\log_2$ n bits (es decir, para dividir un número entre 2, se desplaza un bit; para dividirlo por cuatro, dos bits; para dividirlo por 8, 3 bits, …). Esto es lo mismo que eliminar directamente los $\log_2$ n bits de menor peso del número (o sea, eliminar el bit de menor peso, los dos bits de menor peso, los 3 bits de menor peso, …).

En nuestro caso, para dividir la dirección por 2, hay que realizar un desplazamiento a la derecha, o lo que es lo mismo, eliminar el bit de menor peso de la dirección. Es decir, que si de las patas de dirección A_0-A_{23} eliminamos la pata A_0 y nos quedamos con los hilos A_1-A_{23}, obtenemos directamente el número de par en el que se encuentra la dirección a la que se quiere acceder. Es por esto por lo que en el Motorola 68000, las patas de direcciones son A_1-A_{23}.

Obsérvese que el “número de par” coincide con las direcciones internas de cada una de las pastillas de memoria. Es por esto por lo que los hilos de direcciones del procesador se pueden conectar directamente a las patas de direcciones de las pastillas de memoria. En realidad se utilizan solamente los hilos del bus de direcciones que sean necesarios para hacer referencia a la cantidad de memoria instalada. Así, si a un 68000 se le instalan 2 pastillas de 8 MBytes de memoria, se conectaran los hilos A_1-A_{23} directamente a las patas A_0-A_{22} de las dos pastillas. Si se instalaran 2 pastillas de 4 MBytes, solo se conectarían los hilos A_1-A_{22}, también a las dos pastillas.

Para un procesador con un bus de datos de 32 bits, en cada referencia a memoria se puede acceder al contenido de 4 direcciones en paralelo. En este caso, las direcciones que salen de la CPU harán referencia a “cuartetos de direcciones”, por lo que para saber a qué cuarteto corresponde la dirección de una celda, hay que dividir por 4, o, lo que es más fácil, desplazar a la derecha 2 bits, es decir, eliminar los dos bits de menor peso. Entonces, en el bus de direcciones se habrán eliminado los hilos A_0 y A_1. Para un bus de datos de 64 bits, los accesos a memoria se hacen de 8 en 8; hay que dividir por 8 (desplazamiento a la derecha de 3 bits), por lo que no estarán los hilos A_0, A_1 y A_2.

...Organización de los Módulos



Ya hemos visto la necesidad de colocar en paralelo los módulos de memoria en sistemas con un procesador cuyo bus de datos es *n* veces mayor que el tamaño de la celda de memoria; de esta manera se consigue el acceso en paralelo a *n* posiciones sucesivas de memoria. Por ejemplo, si la celda es de 8 bits y el bus de datos es de 16, en un único acceso a memoria se lee el contenido de dos celdas consecutivas. Así, cuando las patas de direcciones tienen el valor 0, en realidad se accede al contenido de la dirección 0 y al de la 1; cuando tienen la dirección 1 se accede a la dirección 2 y a la 3, etc. etc.

Una vez en este punto **¿porqué no encerrar a todas las pastillas de memoria en un único módulo de 16 hilos de datos?**

Si así lo hiciéramos, nos encontraríamos con un nuevo problema: **¿Qué pasaría, entonces, si se quisiese acceder a un dato que solamente ocupa una celda de memoria?** El problema está en el hecho de que solamente queremos 8 bits de datos y el sistema nos da 16; queremos el contenido de una celda y nos dan el de dos.

Una solución sencilla a este problema podría ser la siguiente: El procesador es listo, y como sabe que sólo quiere el contenido de 8 bits (los más altos o los más bajos del bus de datos), ignora los otros bits y forma un dato de solo 8 bits.

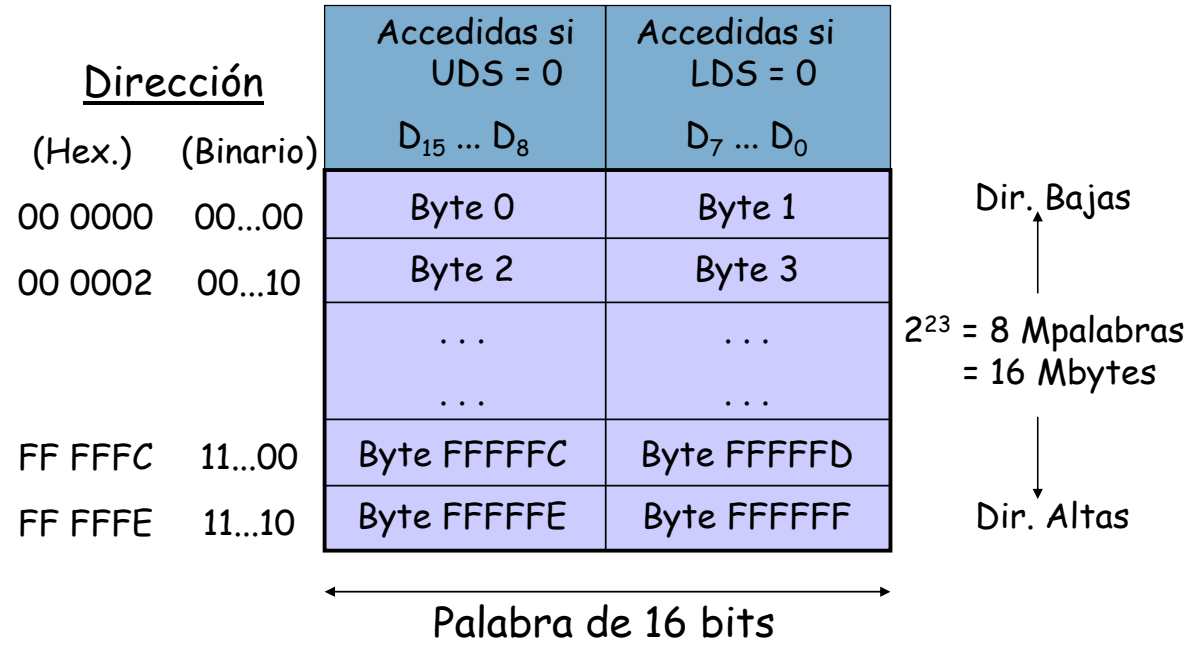
Esta solución podría funcionar, pero no es así. Ciertamente esta solución resuelve el problema de las lecturas, pero ¿qué pasa con las escrituras? Es decir, cuando se da una orden de escritura ¿qué se escribe: el contenido de los 16 bits del bus de datos sobre dos celdas consecutivas de memoria, o solamente se escriben 8 bits? Por lo que sabemos hasta ahora, parece claro que se escriben los 16 bits, pues están conectados a dos módulos de memoria, y ambos reciben una orden de escritura. Es decir, que la unidad de escritura en memoria correspondería a un par de celdas de 8 bits, mientras que nosotros queríamos que la unidad fuese solamente una celda.

Por lo tanto, se necesita algo más que nos ayude a saber en el exterior del procesador si se quiere acceder a una palabra de dos bytes o a un único byte y, en este último caso, hay que saber si el acceso va dirigido al byte de la dirección menor o al de la mayor (de las dos que proporciona el bus de datos).

Esta información sobre el tipo de acceso que se está realizando se consigue mediante unas señales de salida del procesador en las que éste indica si la operación va dirigida al byte de la dirección más alta, al de dirección más baja o a los dos bytes. Obsérvese que, por ejemplo, en el caso de un procesador con un ancho de bus cuatro veces mayor que la celda de memoria, se requieren cuatro señales de salida del procesador para indicar las direcciones o módulos de memoria que deben estar implicados en la operación.

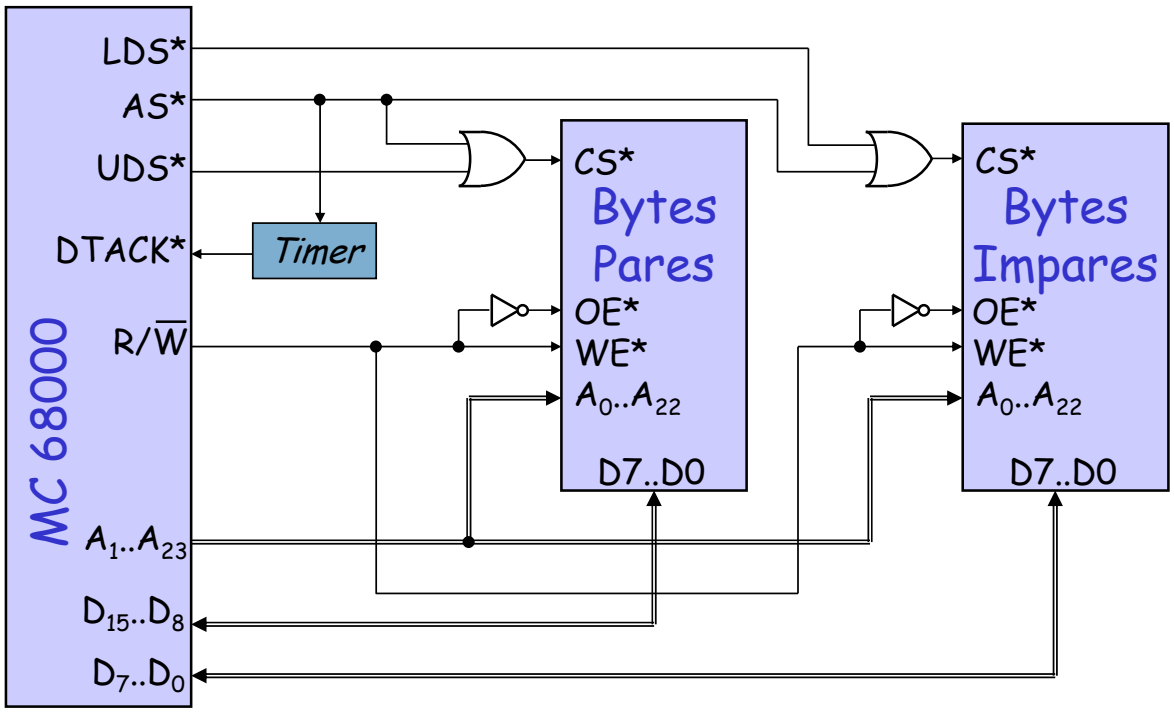
Estas señales se utilizan junto con la lógica de descodificación de direcciones para establecer si debe activarse o no la señal de selección (CS) de cada módulo de memoria.

Espacio de Direccionamiento



Este gráfico solamente trata de recordar el aspecto del espacio de direcciones visto desde el 68000, y cómo se hace referencia a los bytes de cada palabra con ayuda de las señales UDS y LDS.

Sistema Basado en el MC68000



Ahora presentaremos, mediante un ejemplo simple, la interconexión entre el procesador 68000 de Motorola y 16 Mbytes de memoria formada mediante dos módulos o chips de 8 Mbytes cada uno.

Ya que el bus de datos es de 16 bits, cada operación de lectura/escritura es capaz de realizarse sobre 16 bits simultáneamente.

Como disponemos de dos pastillas de memoria de 8 Mbytes cada una, podríamos utilizar una para los primeros 8 Mbytes del espacio de direccionamiento de la CPU, y la otra para los 8 últimos; pero esto presenta el problema de que cuando se quiere acceder a una palabra (formada por el byte de cualquier dirección par más el de la siguiente dirección) este esquema no es capaz de proporcionar 16 bits en paralelo, solamente 8. Por esto, vamos a utilizar una pastilla de memoria para contener los bytes de direcciones pares (bits 8..15 de cada palabra) y otra para las direcciones impares (bits 0..7), de tal manera que ante cualquier dirección, cada pastilla entrega 8 bits, en total, los 16 de la palabra.

Así, los 8 hilos de datos de la pastilla de los bytes pares se conectarán a los hilos D_8-D_{15} del bus de datos, mientras que los 8 hilos de la otra pastilla se conectarán a los hilos D_0-D_7 del bus de datos.

Aunque el bus de datos transporta palabras de dos bytes, para ser capaz de referenciar los bytes individuales de cada palabra, se utilizan las señales UDS (*Upper Data Strobe*) y LDS (*Lower Data Strobe*). Así vemos cómo estas señales van a parar, respectivamente, a la pastilla de direcciones pares y a la de las direcciones impares. Ambas, junto con la señal AS (*Address Strobe*) activan la señal CS (*Chip Select*) de sus respectivos módulos de memoria.

Puesto que ambas pastillas de memoria tienen 8 Mdirecciones, sus 23 hilos de direcciones se conectarán directamente a las señales A_1-A_{23} de la CPU.

La señal R/W de la CPU se conecta directamente a la pata WE (*Write Enable*) de ambas memorias; y la misma señal, negada, se conecta a OE (*Output enable*).

Por último, nos queda por conectar la pata DTACK (*Data Acknowledge*) de la CPU, (la que le indica a la CPU que el dato a leer de la memoria ya está listo, o que el dato a escribir ya ha sido recogido). Ya que se suele trabajar con memorias síncronas, es decir que la memoria responde siempre en un tiempo predefinido, la señal DTACK la suele generar un dispositivo temporizador, de tal manera que a partir de su activación mediante la señal AS, pasado el tiempo indicado en las especificaciones de la memoria, activa la señal DTACK.

Obsérvese que la lógica de estas conexiones está realizada teniendo en cuenta que las señales de control son de lógica negativa (*), esto es, activas a nivel bajo.

Ejemplos de Acceso a Memoria

Acceso de lectura al byte almacenado en la dir. $7A0003

Bus de Direcciones	UDS*	LDS*	AS*	R/$\overline{W}$
0111 1010 0000 0000 0000 001(1)	1	0	0	1

Acceso de escritura al byte almacenado en la dir. $7A0006

Bus de Direcciones	UDS*	LDS*	AS*	R/$\overline{W}$
0111 1010 0000 0000 0000 011(0)	0	1	0	0

Aquí tenemos algunos ejemplos de referencias a memoria para lectura y escritura de datos de tamaño *byte*, tanto a direcciones pares como impares.

Para cada ejemplo podemos ver el valor del bus de direcciones y de las señales de control correspondientes al acceso que se desea realizar.

Téngase en cuenta que el bus de direcciones contiene únicamente los 23 bits de mayor peso de la dirección real. El bit de menor peso se muestra entre paréntesis para saber cuál es la dirección completa formada internamente en la CPU.

...Ejemplos de Acceso a Memoria

Acceso de lectura a la palabra almacenado en la dir. $7A0006

Bus de Direcciones	UDS*	LDS*	AS*	R/$\overline{W}$
0111 1010 0000 0000 0000 011(0)	0	0	0	1

Acceso de escritura a la palabra larga almacenado en la dirección $7A0006

Bus de Direcciones	UDS*	LDS*	AS*	R/$\overline{W}$
0111 1010 0000 0000 0000 011(0)	0	0	0	0
0111 1010 0000 0000 0000 100(0)	0	0	0	0

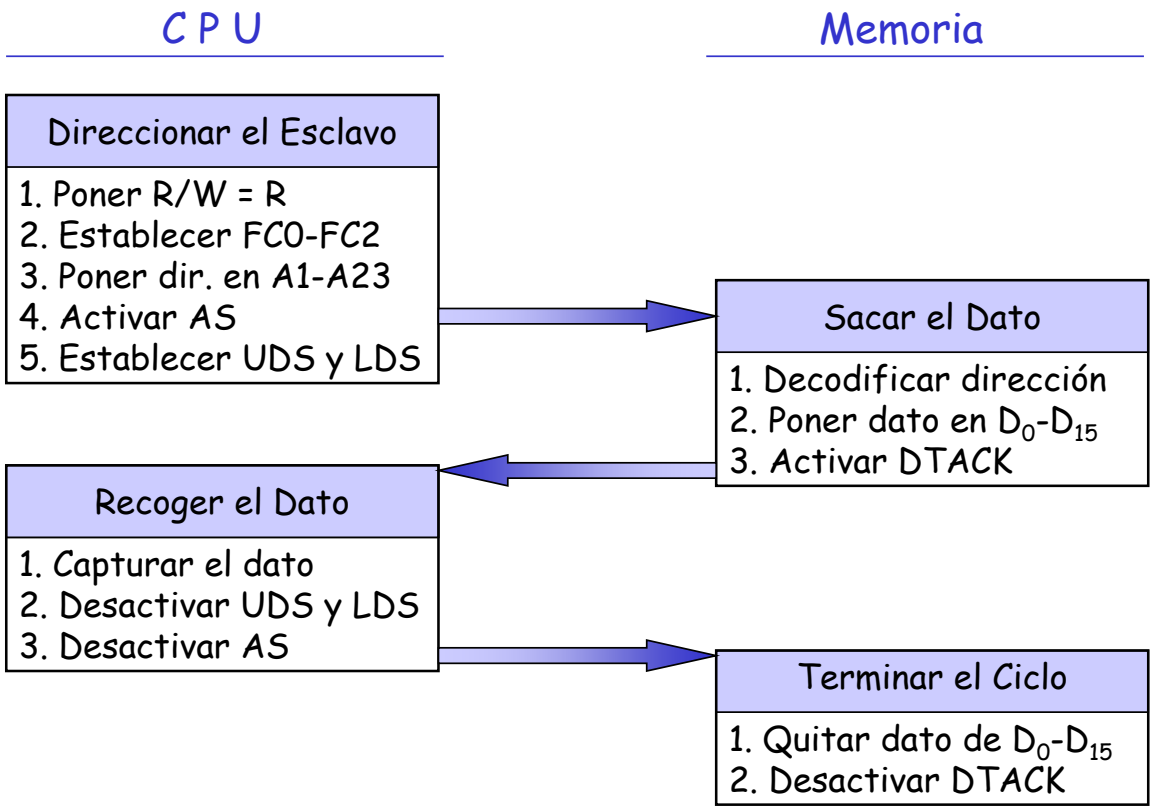
Ahora se muestran algunos ejemplos de referencias a memoria para lectura y escritura de datos de tamaño *word* y *long word*.

Para cada ejemplo podemos ver el valor del bus de direcciones y de las señales de control correspondientes al acceso que se desea realizar.

Téngase en cuenta que el bus de direcciones contiene únicamente los 23 bits de mayor peso de la dirección real. El bit de menor peso se muestra entre paréntesis para saber cuál es la dirección completa formada internamente en la CPU.

Obsérvese cómo en el último ejemplo se requieren dos ciclos de memoria para escribir la doble palabra de la dirección 7A0006.

Ciclo de Lectura



En este diagrama se muestran los pasos que se siguen para la lectura de un dato desde el MC68000. Como vemos, consta de cuatro etapas:

1. La CPU direcciona el esclavo (memoria o dispositivo de E/S)

2. El esclavo proporciona el dato solicitado por la CPU

3. La CPU adquiere el dato

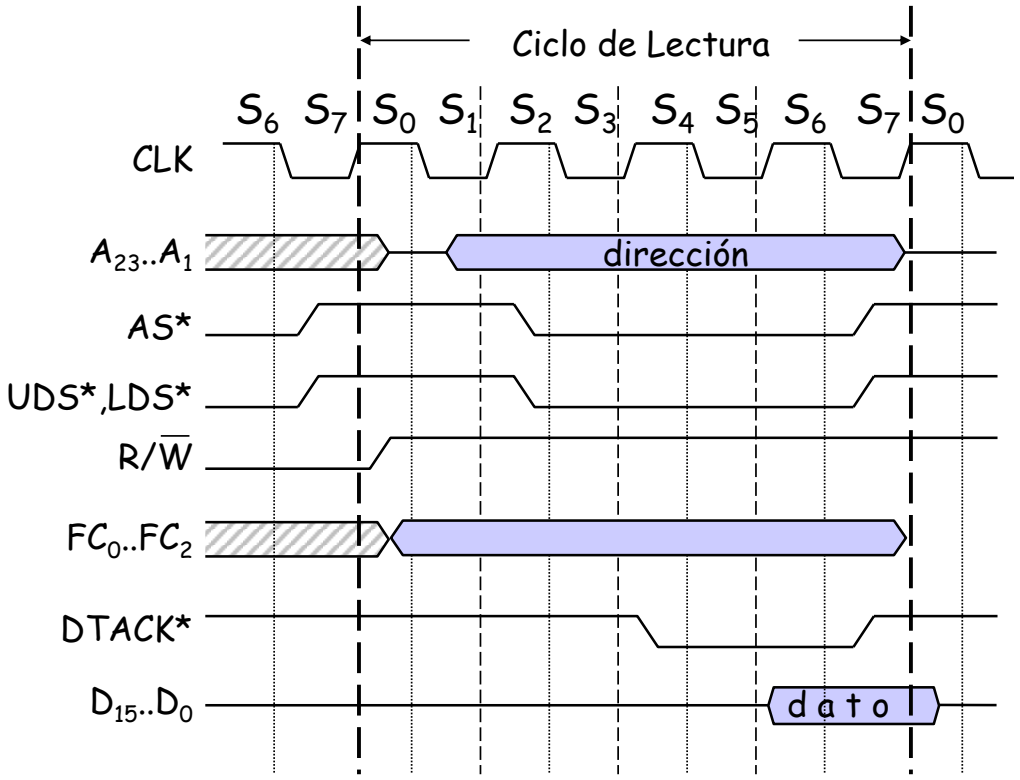
4. El esclavo da por terminada la lectura

Los pasos de cada una de las etapas se muestran en la figura de arriba.

En el paso 3 de la fase “Sacar el Dato”, se indica que la memoria activa la señal DTACK de la CPU, lo que indica que en este caso se trata de una memoria asíncrona. Si la memoria fuese síncrona, esta señal la activaría un *timer*, como ya hemos visto en algún ejemplo anterior.

Este diagrama indica el orden en el que se activan las señales, pero no da información del instante en el que se activa cada señal y cuánto dura su activación. Para ver esta lectura con detalle temporal pasemos a ala página siguiente para ver el cronograma correspondiente.

...Ciclo de Lectura



Este cronograma contiene la evolución temporal de todas las señales implicadas en la lectura de un dato desde la CPU.

En primer lugar tenemos la señal de reloj. La duración de cada ciclo de reloj es demasiado larga para los tiempos que se manejan en el procesador, por eso, cada ciclo se divide en dos subciclos. Así, diremos que los sucesos tienen lugar en uno u otro subciclo. Veamos entonces lo que sucede en los 8 subciclos de reloj que requiere una operación de lectura.

Comencemos viendo lo que sucede a partir del subciclo S_0 de nuestra operación. Las señales AS, UDS y LDS permanecen desactivadas (son de lógica negativa), así como DTACK. La pata R/W también está a nivel bajo. El contenido del bus de direcciones (zona sombreada) no es significativo. Lo mismo sucede con los valores de las señales de estado FC_0-FC_2. En este momento se supone que todos los dispositivos conectados al bus de datos están en alta impedancia, por lo que éste no contiene ninguna señal.

Lo primero que sucede (en S_0) es que se activa la señal de R/W, indicando que se trata de una lectura. Los valores de FC_0-FC_2 se establecen de acuerdo a la operación. También se pone la dirección de lectura en el bus de direcciones, que acaba estando estable en S_1.

En S_2, con el bus de direcciones estable, se activa AS y las señales UDS o LDS que correspondan al tipo y dirección del dato solicitado.

Con las señales activadas, la memoria ya puede comenzar a recuperar el dato solicitado. Dispone para ello desde el subciclo S_3 hasta el S_6.

En S_4 se activa la señal DTACK, indicándole al procesador que el dato estará presente en el bus de datos en S_7.

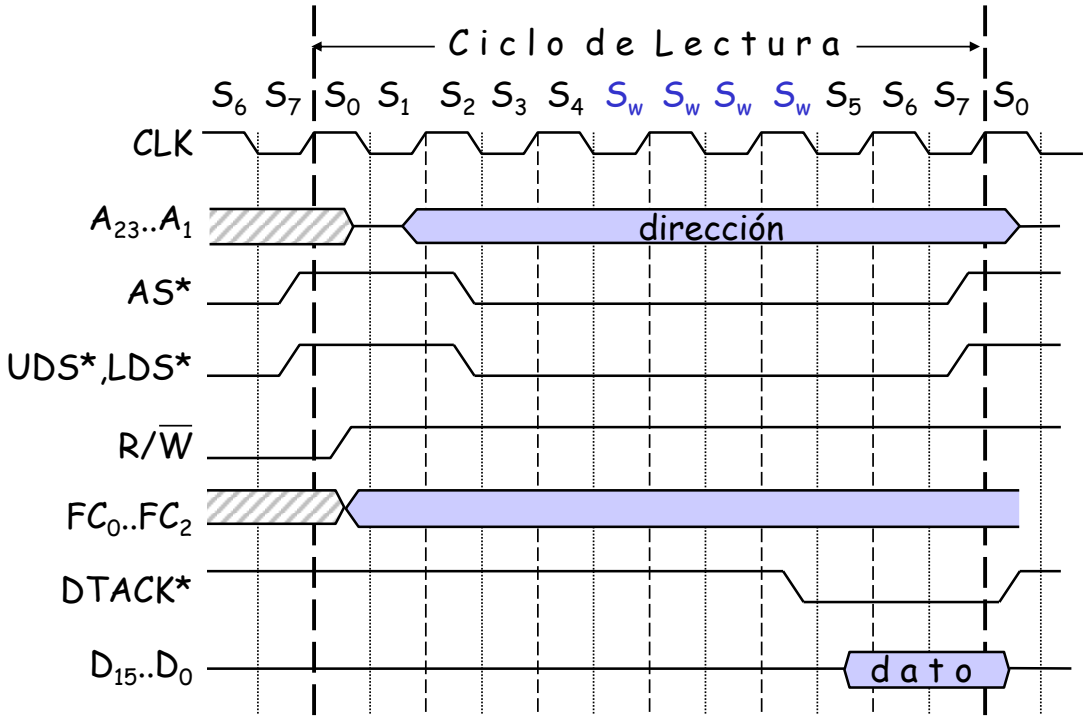
Durante el subciclo 5 no se altera ninguna señal de bus.

Así, en el subciclo S_6 la memoria pone el dato en el bus de datos. Al comienzo del subciclo S_7 la CPU lee el dato del bus de datos. A continuación desactiva las señales AS y UDS y/o LDS, dando por acabada la lectura en lo que al procesador le concierne.

Cuando la memoria detecta la desactivación de la señal CS, pone en alta impedancia sus patas D_0-D_{15} y se desactiva la señal DTACK.

La lectura ha finalizado. Todas las lecturas comienzan en S_0 con el flanco de subida del reloj, y finalizan con el flanco de bajada en S_7.

Ciclo de Lectura con Ciclos de Espera



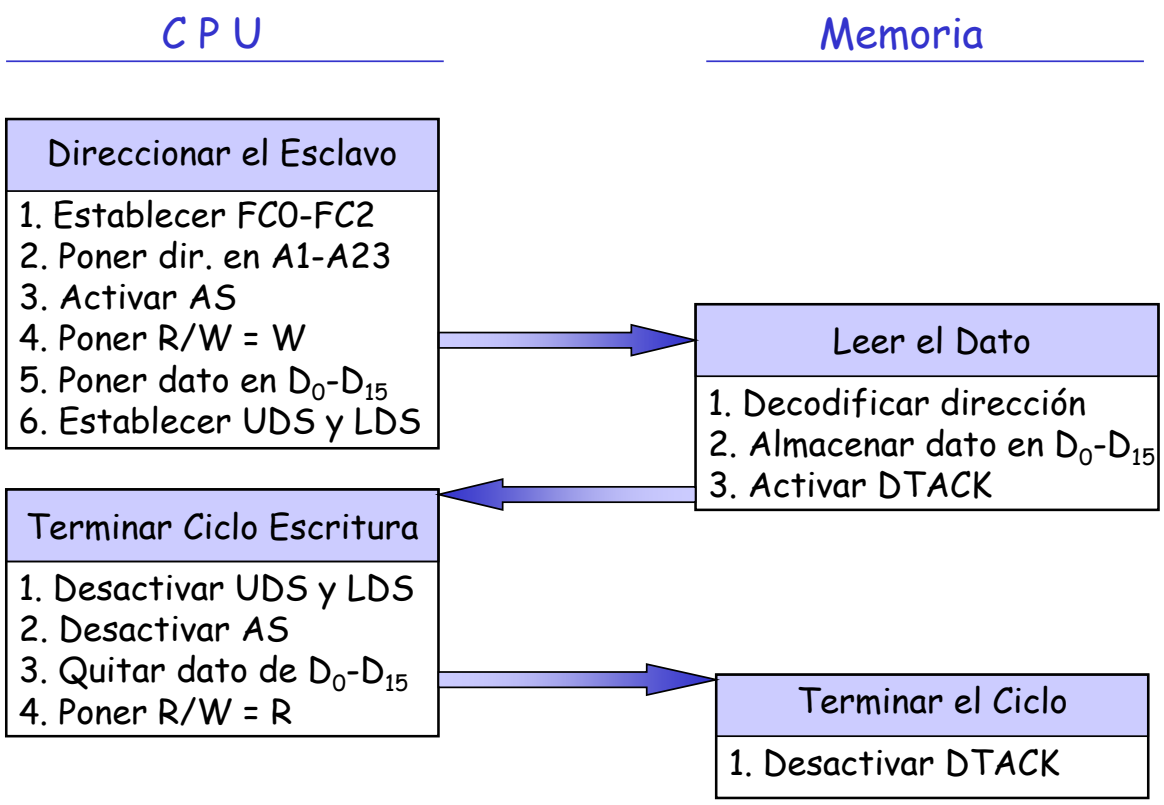
En el anterior cronograma de lectura, hemos supuesto que la memoria es capaz de contestar en el tiempo impuesto por la CPU. No obstante algunas memorias pueden resultar lentas para la velocidad del procesador, por lo que no pueden contestar en el momento previsto por la CPU.

En tal caso, la señal DTACK no debe activarse en el subciclo S_4, sino que debe posponerse hasta que la memoria esté dispuesta a entregar el dato solicitado. Cuando esto sucede, a continuación de S_4 se insertan subciclos de espera conocidos como **estados de espera**.

Obsérvese en este cronograma cómo ahora el dato es entregado justo a continuación de que la señal DTACK es activada.

Como se puede apreciar, la lógica general de este cronograma es idéntica a cuando no hay estados de espera, con la excepción de que se alarga el tiempo entre la activación de AS y la de DTACK con los subciclos de espera necesarios.

Ciclo de Escritura



En este diagrama se muestran los pasos que se siguen para la escritura de un dato desde el MC68000. Como vemos, consta de cuatro etapas:

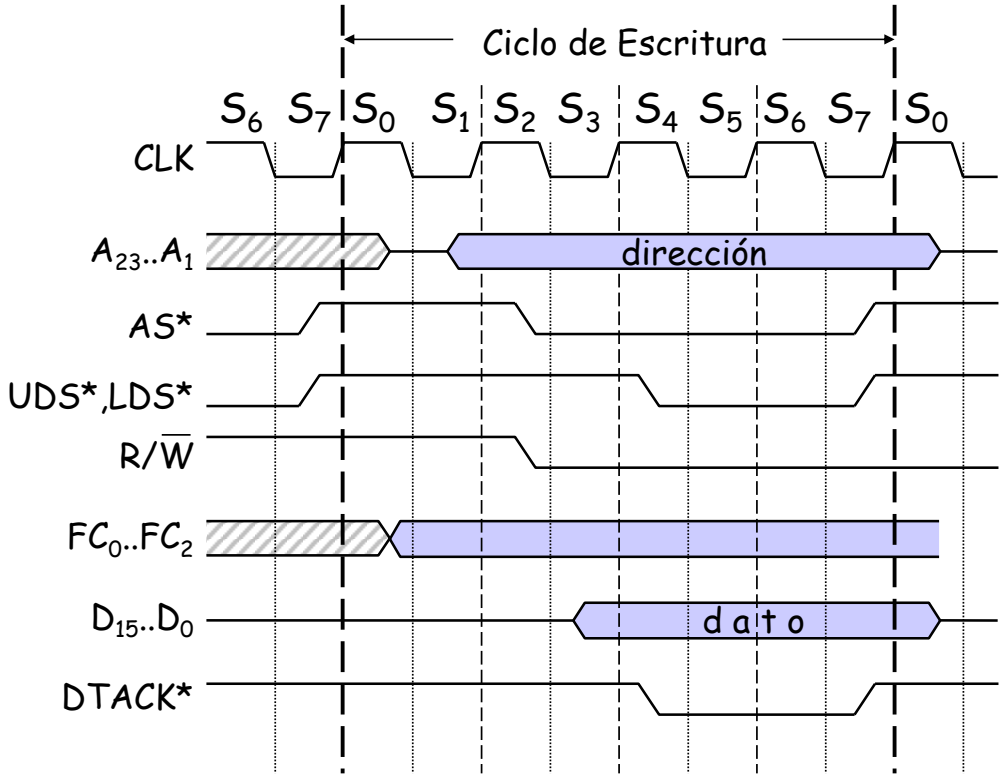
1. La CPU direcciona el esclavo (memoria o dispositivo de E/S) y pone el dato a escribir en el bus de datos.
2. El esclavo captura el dato del bus de datos.
3. La CPU da por terminada la escritura.
4. El esclavo termina el ciclo de escritura.

Los pasos de cada una de las etapas se muestran en la figura de arriba.

Obsérvese que aquí se activa DTACK para indicar que ya se ha leído el contenido del bus de datos. Así, a continuación la CPU desactiva las señales UDS/LDS y AS.

Para ver esta escritura con detalle temporal pasemos a ver el cronograma correspondiente en la página siguiente.

...Ciclo de Escritura



La escritura comienza, igualmente, poniendo en el subciclo S_0 la dirección de memoria en la que se va a escribir. Los valores de FC_0-FC_2 se establecen también en S_0. Al final de S_1 queda estable el contenido del bus de direcciones.

En el siguiente paso, en S_2, se debe poner a nivel bajo la señal R/W (indica escritura), y activar AS. A continuación, en S_3, se pone el dato en el bus de datos, y cuando éste se hace estable, se activan las señales UDS y/o LDS (en S_4).

Al activarse la señal CS de la memoria, el contenido del bus de datos se almacena rápidamente en un registro *latch* entre el bus de datos y la memoria, desde el cual se envía después a la memoria RAM. Tan pronto como el dato está en el *latch*, se activa la señal DTACK. Es por esto que en el cronograma no se observa un tiempo apreciable entre la activación de las señales UDS/LDS y DTACK. Al activarse la señal DTACK la CPU desactiva AS y UDS/LDS, pero la memoria no desactiva DTACK hasta que el dato que se había capturado en el *latch* se ha escrito realmente en la memoria.