Fundamentos de Computadores

Segundo cuatrimestre

2015-2016

fc2

Módulo 10: El Sistema de Memoria

- ✓ Organización de memorias de semiconductores: RAM estática
- ✓ Jerarquía de Memoria
- ✓ Introducción a la memoria Cache

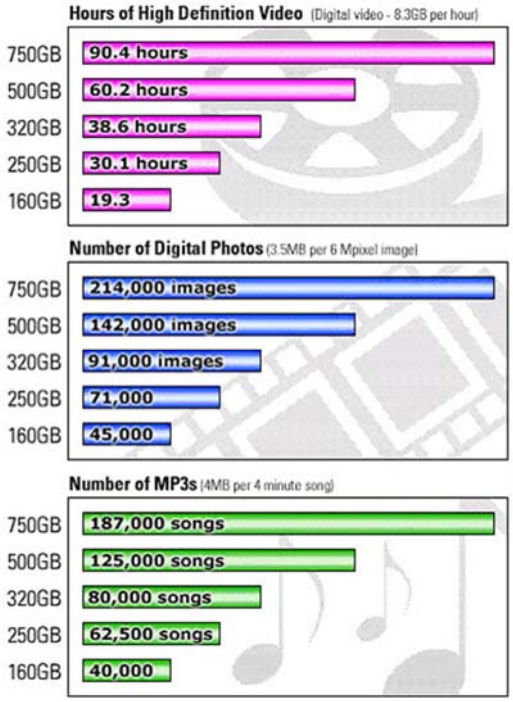
fc2

Sistema de memoria de un computador

- ¿Qué es una memoria? ¿Cómo se implementa?
- ¿Es la memoria lo suficientemente rápida como para satisfacer las transferencias demandadas por el procesador?
- ¿Por qué TODOS los computadores tienen memoria cache?
- ¿Cómo funciona?

Introducción

- El manejo de datos tiene un coste:
  - Ocupan bastante **espacio** en memoria.
  - Acceder a ellos requiere un cierto **tiempo**.

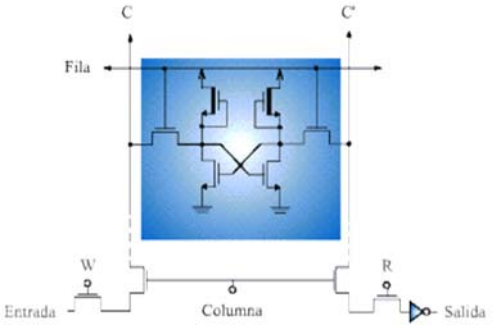


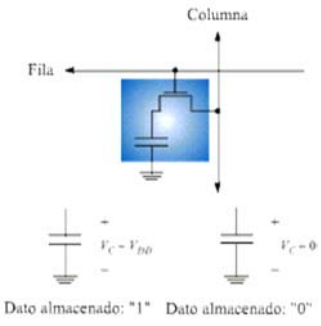
Tipos de memoria semiconductora

- Memoria no volátil ROM
  - PROM, EPROM,
  - EEPROM, FLASH,
  - Disco magnético, Disco de estado sólido.
- Memoria volátil RAM
  - SRAM
  - DRAM

Tipos de memoria semiconductora

- Celda básica de memoria RAM





Celda SRAM

Ventajas
- Tiempo de acceso y de ciclo reducido

Desventajas
- Disipan mucha energía
- Baja densidad de integración
- Coste elevado

Celda DRAM

Ventajas
- Bajo consumo de energía
- Alta densidad de integración
- Coste reducido

Desventajas
- Tiempo de ciclo elevado
- Necesidad de refresco

Introducción

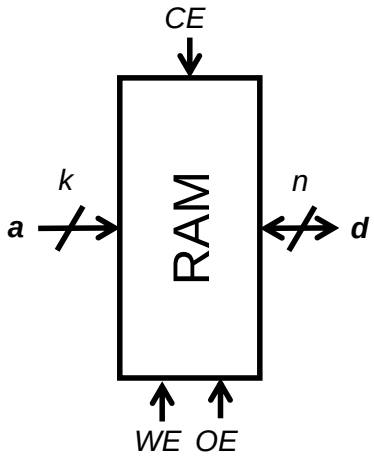
- Valores típicos de tiempos de acceso y precios por Gbyte (año 2015)

Tipo de memoria	Tiempo de acceso (ns)	$ / Gbyte	Ancho de Banda (Gbytes/s)
SRAM	0.5	5,000	25+
DRAM	10-50	7	10
Disco de Estado Sólido (SSD)	20,000	0.40	0.5
Disco Magnético	5,000,000	0.05	0.75

fc²

RAM (Random-Access Memory)

[Revisión del módulo 7]



RAM $2^k \times n$
(2^k n-bit words)

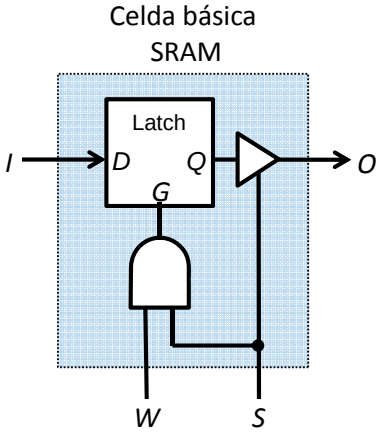
Interfaz externo del dispositivo	
a	k líneas de dirección (entrada)
d	n líneas de datos (entrada/salida)
CE	1 entrada de habilitación del dispositivo
OE	1 entrada de habilitación de lectura
WE	1 entrada de habilitación de escritura

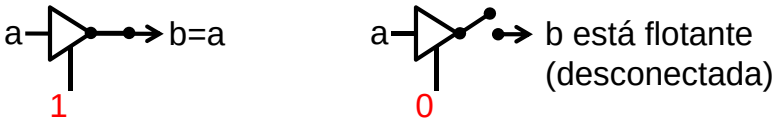
Memoria volátil capaz de almacenar 2^k palabras de n bits

fc²

RAM (Random-Access Memory)
[Revisión del módulo 7]

- Celda básica de una SRAM (Static RAM)
  - Cada bit se almacena en un latch
  - El contenido de la celda se mantiene mientras esté conectada a la alimentación
  - Si se activa (1) la señal S (Select) el contenido de la celda aparece en la salida O
  - Cuando las señales W (Write) y S están las dos a 1 el valor de la entrada I se escribe en el latch
  - Comportamiento de un driver tri-estado:



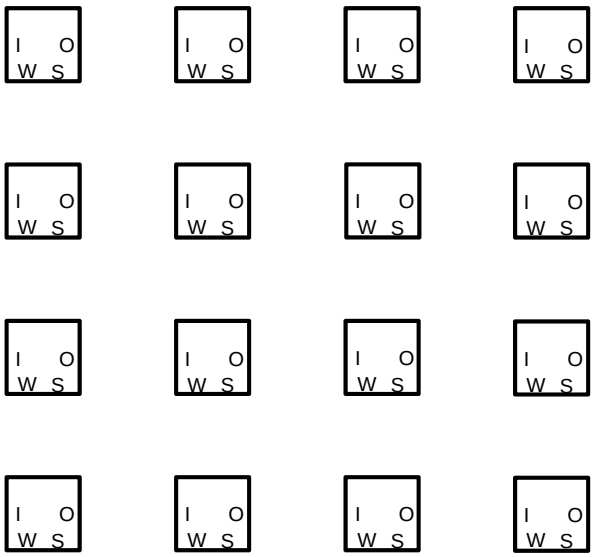


fc²

RAM (Random-Access Memory)
[Revisión módulo 7]

Ejemplo: Implementación de una memoria SRAM 4x4

Paso 1: Replicar la celda básica para formar un array 4×4 (4 palabras de 4 bits cada una)

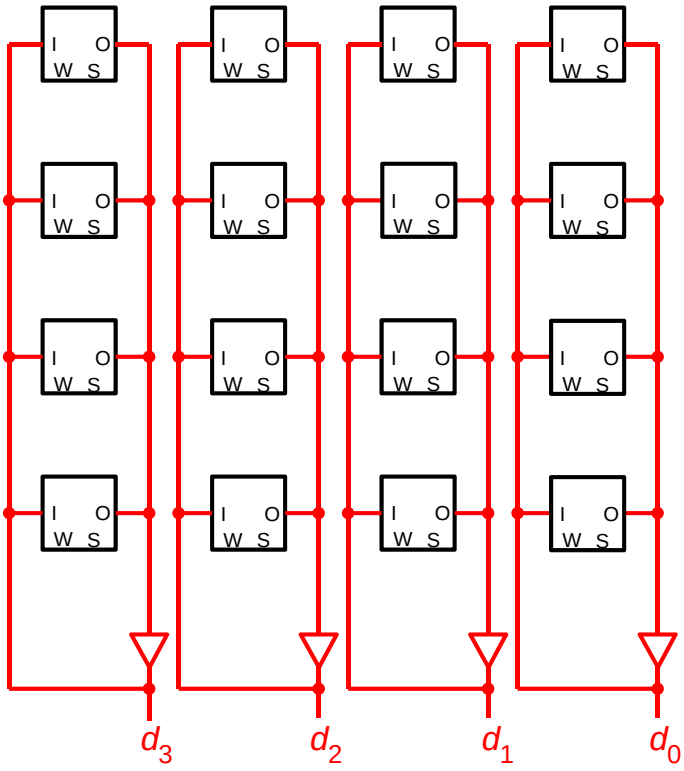


fc²

RAM (Random-Access Memory)

[Revisión del módulo 7]

Paso 2: Implementar las líneas de datos bidireccionales

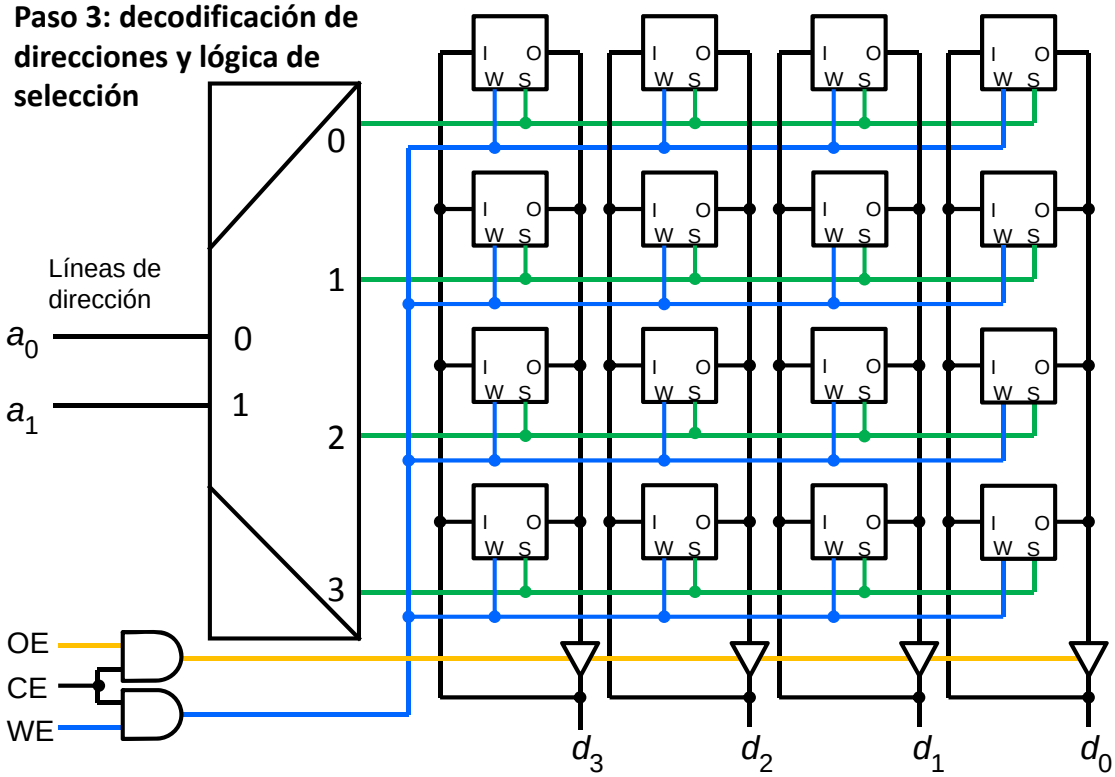


fc²

RAM (Random-Access Memory)

[Revisión del módulo 7]

Paso 3: decodificación de direcciones y lógica de selección

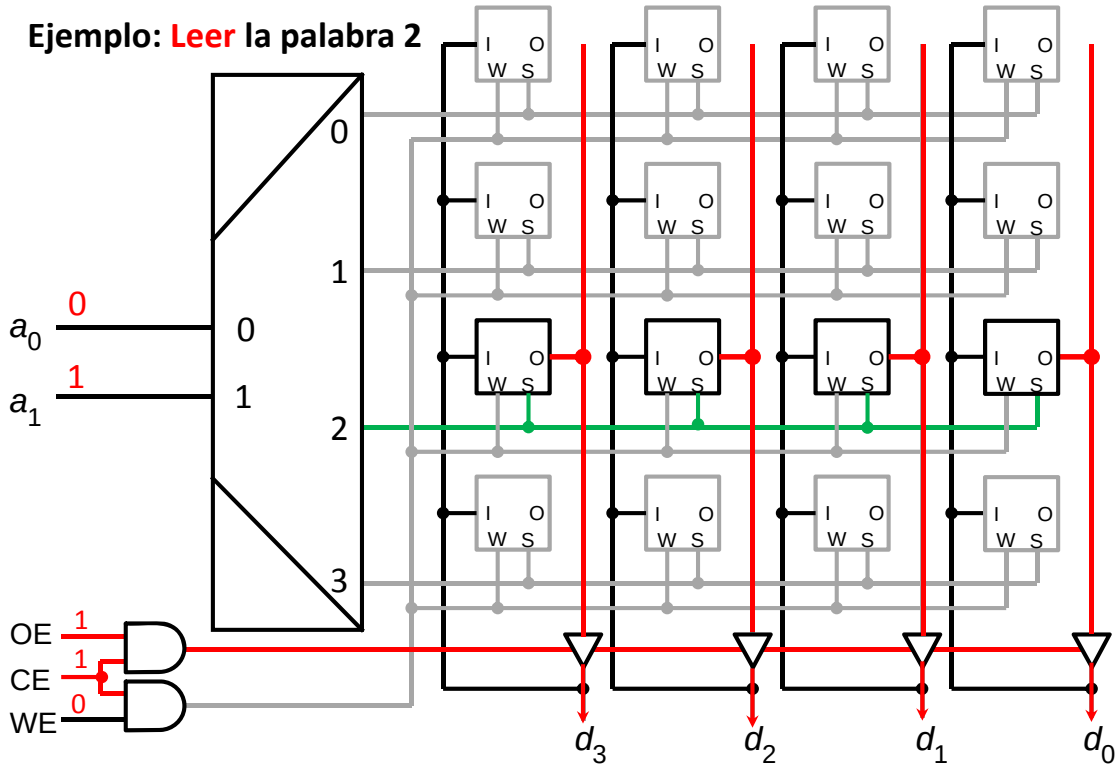


fc²

RAM (Random-Access Memory)

[Revisión del módulo 7]

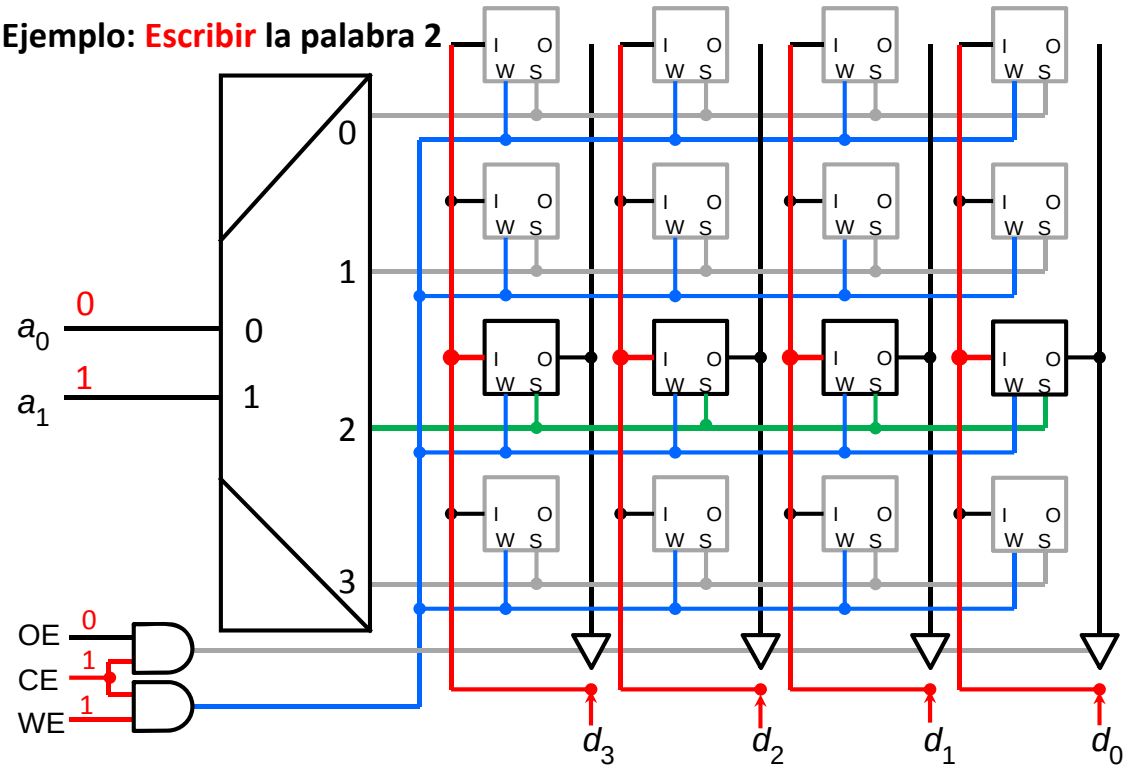
Ejemplo: Leer la palabra 2



RAM (Random-Access Memory)

[Revisión del módulo 7]

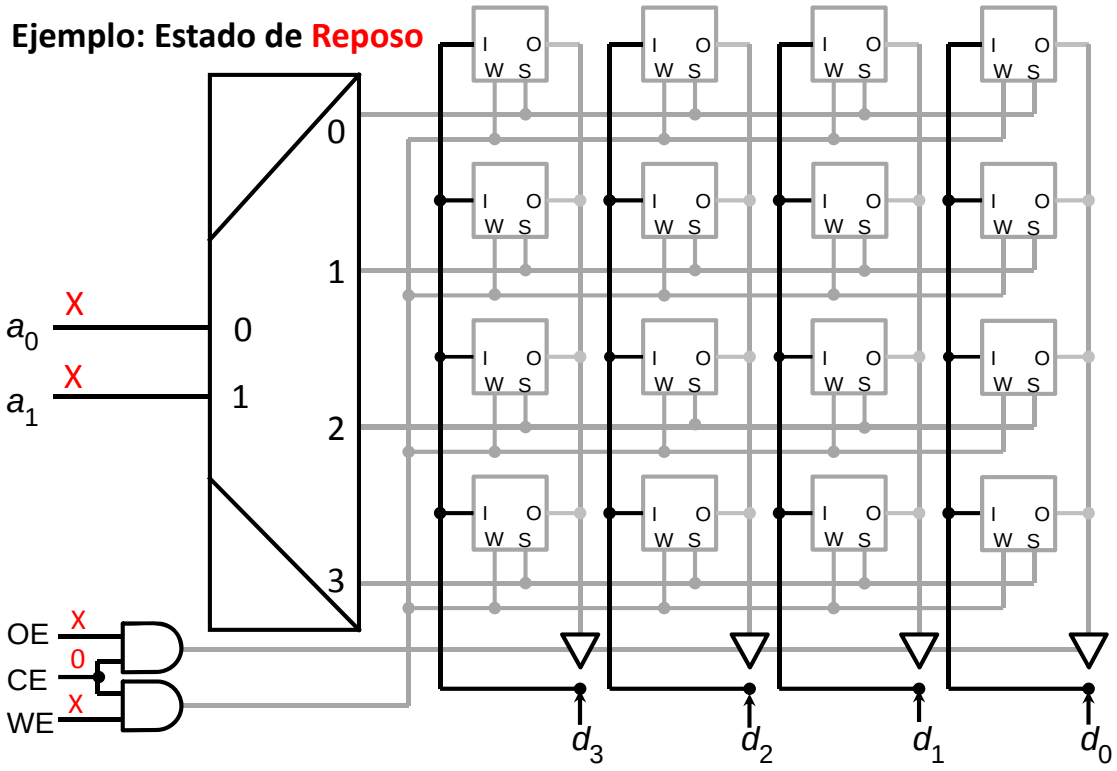
Ejemplo: Escribir la palabra 2



RAM (Random-Access Memory)

[Revisión del módulo 7]

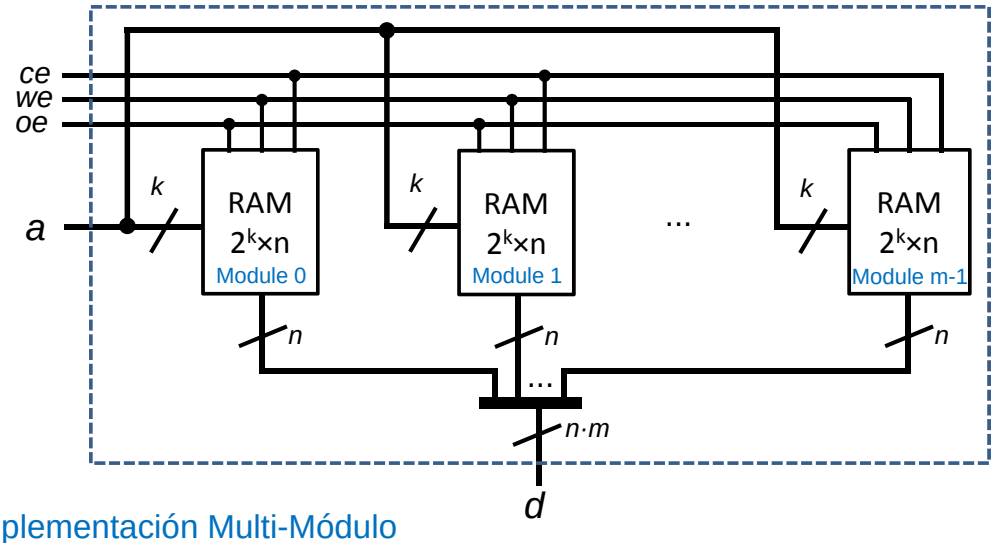
Ejemplo: Estado de Reposo



RAM (Random-Access Memory)

[Revisión del módulo 7]

- Podemos conectar varias RAMs para que se comporten como una RAM con mayor tamaño de palabra

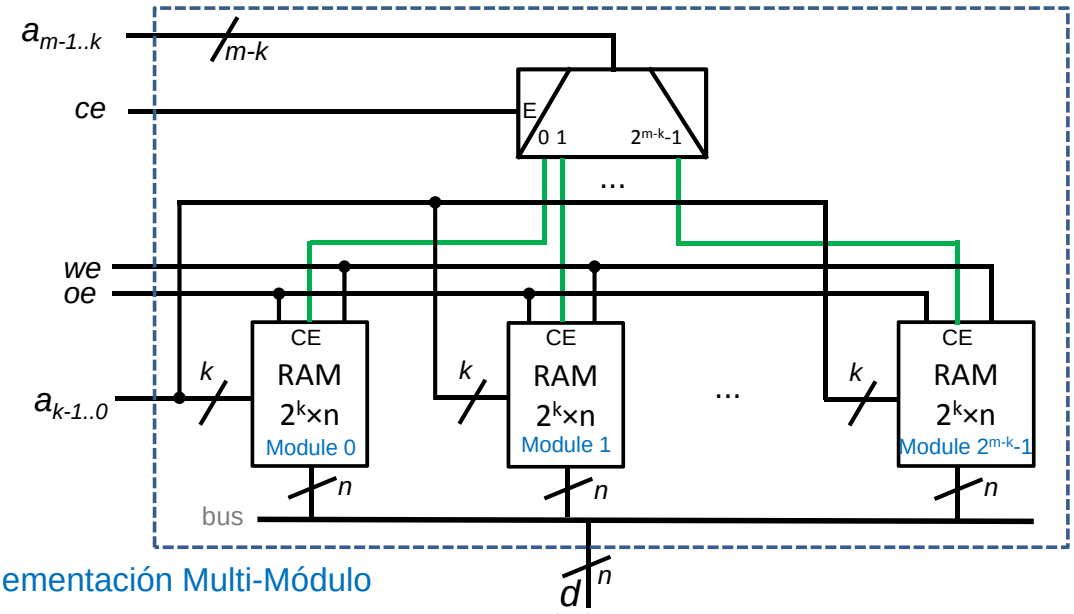


Implementación Multi-Módulo

RAM de tamaño $2^k \times (n \cdot m)$ utilizando m RAMs de tamaño $2^k \times n$

RAM (Random-Access Memory)
[Revisión del módulo 7]

- Podemos combinar varias RAMs para que se comporten como una RAM de mayor profundidad (más direcciones)



Implementación Multi-Módulo

RAM de tamaño $2^m \times n$ utilizando RAMs de tamaño 2^{m-k} $2^k \times n$

Chips SRAM

- Ejemplo: AS6C62256 (Alliance Memory, Inc. 2007)
  - 262.144-bit (256 Kbit) low power CMOS
  - Organizada como una RAM de 32.768 (32 K) palabras de 8 bits
    - 32 kpalabras (= 2^{15}) implica 15 líneas de dirección (A14-A0)

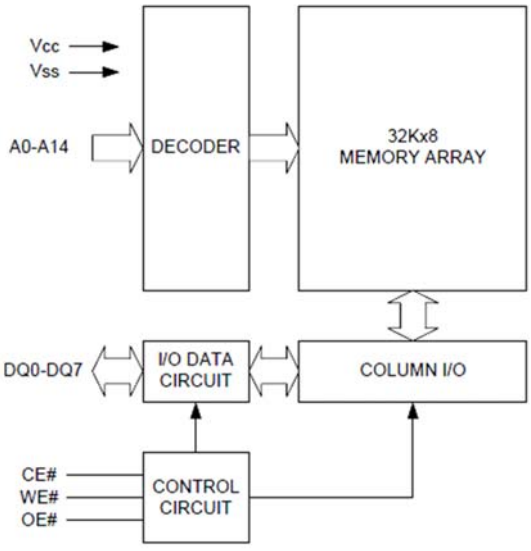


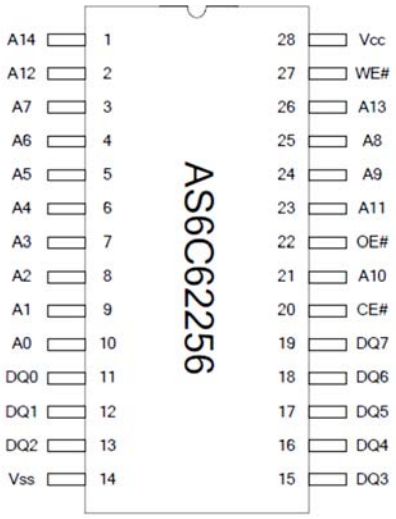
Diagrama de bloques

- El array de memoria es similar (pero mucho más grande) al del ejemplo anterior de una SRAM 4x4
- El símbolo # en la líneas CE, WE y OE indica que estas líneas son activas a baja (i.e. activas a 0)
- Hay un decodificador de 15 a 2^{15}

Chips SRAM

- Ejemplo: AS6C62256 (cont)

PIN CONFIGURATION



PIN DESCRIPTION

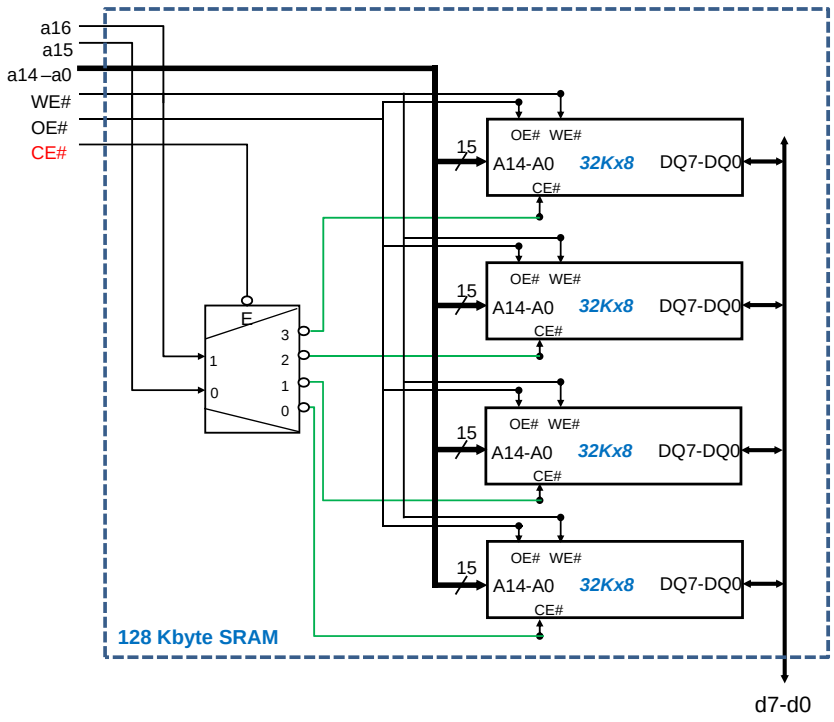
SYMBOL	DESCRIPTION
A0 - A14	Address Inputs
DQ0 – DQ7	Data Inputs/Outputs
CE#	Chip Enable Input
WE#	Write Enable Input
OE#	Output Enable Input
Vcc	Power Supply
Vss	Ground

Tabla de verdad

Mode	CE#	OE#	WE#	Data Lines
Standby	H	X	X	Hi-Z
Read	L	L	H	Dout
Write	L	H	L	Din

Diseño de memorias con chips SRAM

Ejemplo 1: módulo de memoria de 128 Kbyte implementado con chips AS6C62256



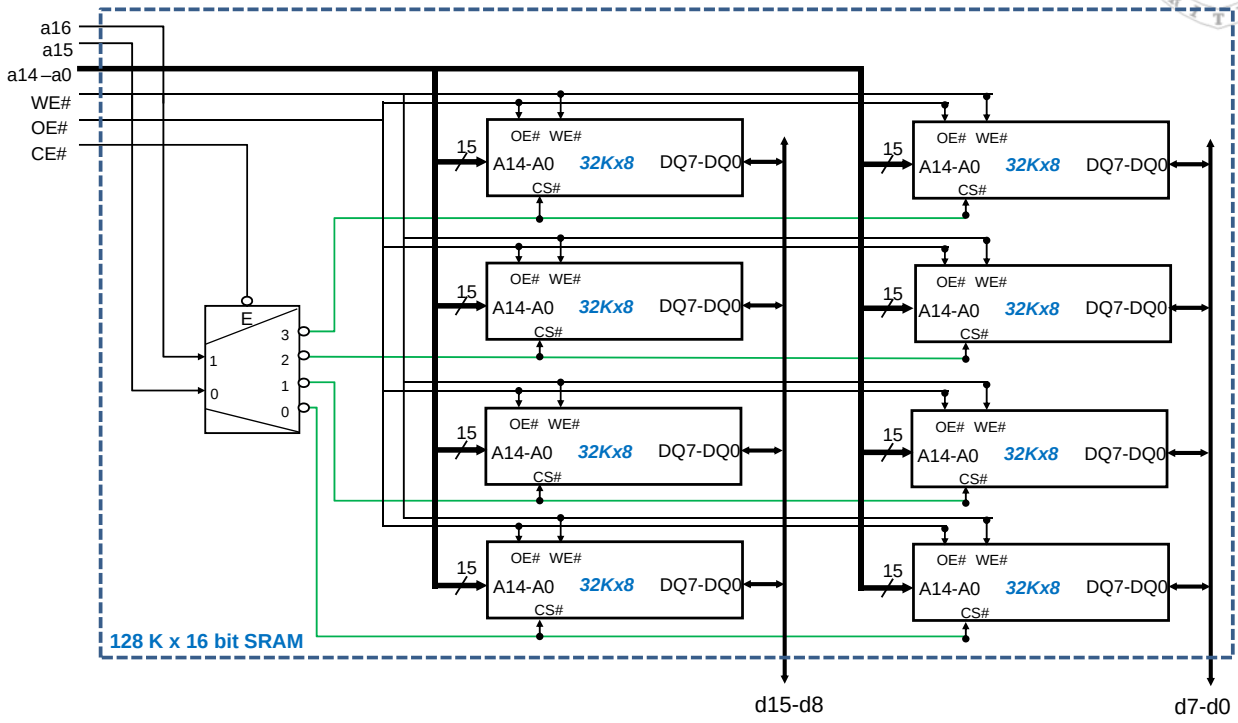
El decodificador selecciona (como mucho) uno de los chips, en función del valor de las líneas *a16-a15*.

Los otros están en estado de reposo (standby), debido a que tienen a 1 sus entradas CE#

Si la entrada CE# global del módulo está a 1, se deshabilita el decodificador y todos los chips: toda la memoria se encuntra en estado de reposo

Diseño de memorias con chips SRAM

Ejemplo 2: un módulo de memoria de 128 Kpalabras de 16 bits implementada con chips AS6C62256

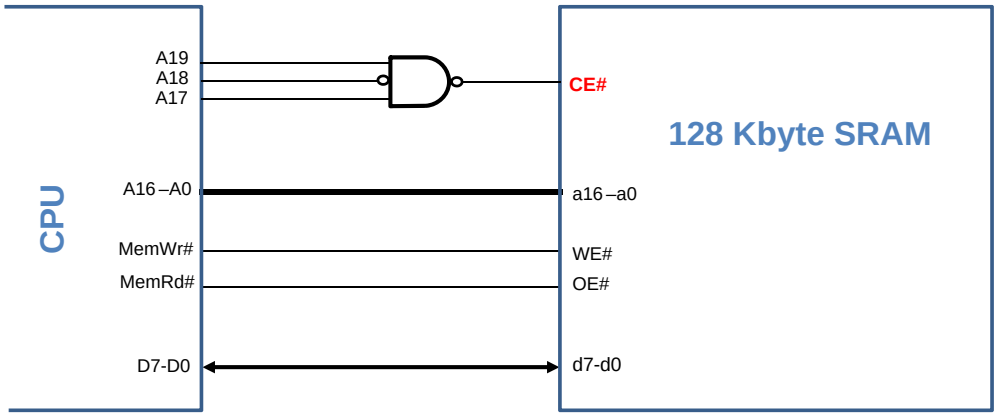


Emplazamiento de módulos de memoria en el mapa de memoria

- Antes de utilizar un módulo de memoria en un computador debemos decidir qué rango de direcciones le vamos a asignar
- Ejemplo 1:
  - Supongamos un computador con 20 líneas de dirección (A19-A0).
  - Supongamos que la memoria es direccionable por bytes
  - Emplazar el módulo de memoria SRAM de 128 Kbyte diseñado previamente en el rango de direcciones 0xA0000 a 0xBFFFF
    - Observar que todas las direcciones de este rango tienen los tres bits más significativos a 101

Emplazamiento de módulos de memoria en el mapa de memoria

- Solución:

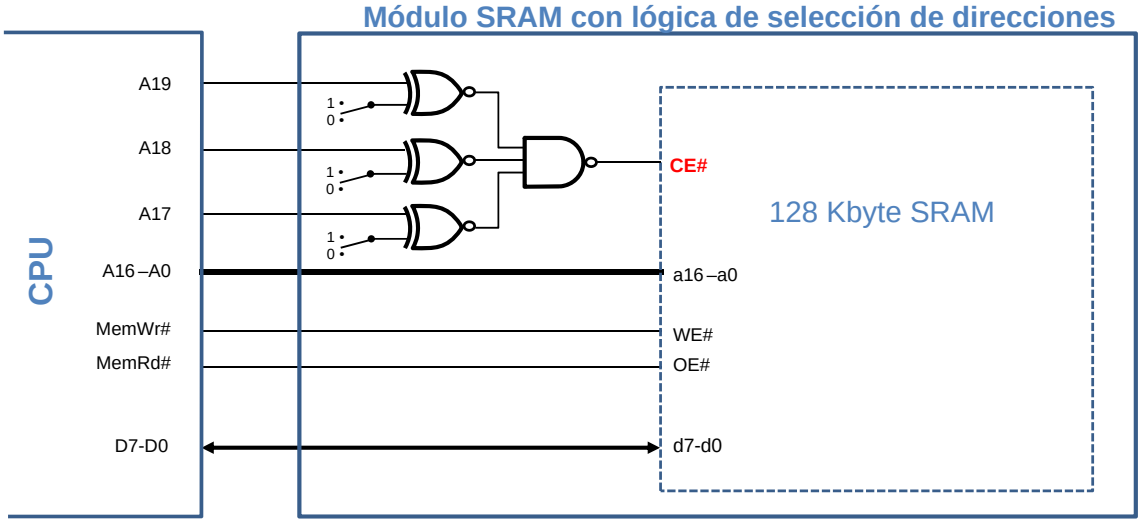


  - Cualquier dirección de la forma 101x xx…xx (i.e. todas las que empiezan por A o B) llevarán la entrada CE# a 0, y por tanto habilitarán el módulo de memoria para hacer una operación de lectura o escritura según el computador active las señales MemWr# y MemRd#
  - Para cualquier dirección fuera del rango 0xA0000-0xBFFFF, la entrada CE# tomará el valor 1, dejando al módulo de memoria en estado de reposo
    - Esto permite conectar varios módulos de memoria al sistema, utilizando las mismas líneas de dirección, datos y Control (MemWr# y MemRd#), emplazando cada módulo en un rango de direcciones diferente. Sólo se necesitaría una puerta nand adicional por cada nuevo módulo.

fc²

Emplazamiento de módulos de memoria en el mapa de memoria

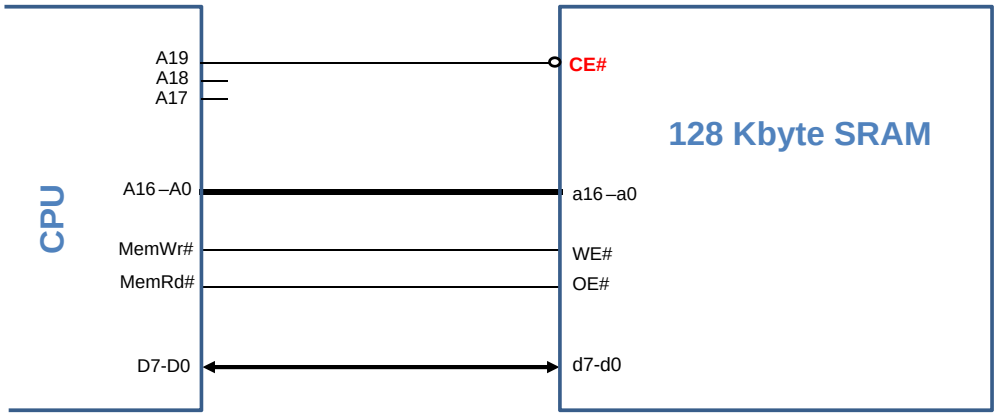
- Ejemplo 2: Diseñar un módulo SRAM de 128 Kbyte que pueda ser emplazado en cualquier rango de direcciones de la forma 0xY0000 – 0x(Y+1)0000, dónde Y es un dígito hexadecimal par



fc²

Emplazamiento de módulos de memoria en el mapa de memoria

- Ejemplo 3 (decodificación parcial de direcciones): Emplazar el módulo SRAM de 128 Kbyte de forma que ocupe la mitad superior del mapa de memoria (con aliasing).



  - El módulo se activará con cualquier dirección que empiece por 1
  - Cada byte del módulo de memoria tendrá 4 direcciones válidas (aliasing)
    - Por ejemplo, el primer byte del módulo será accedido con cualquiera de las siguientes direcciones: 0x80000, 0xA0000, 0xC0000 y 0xE0000

Jerarquía de Memoria

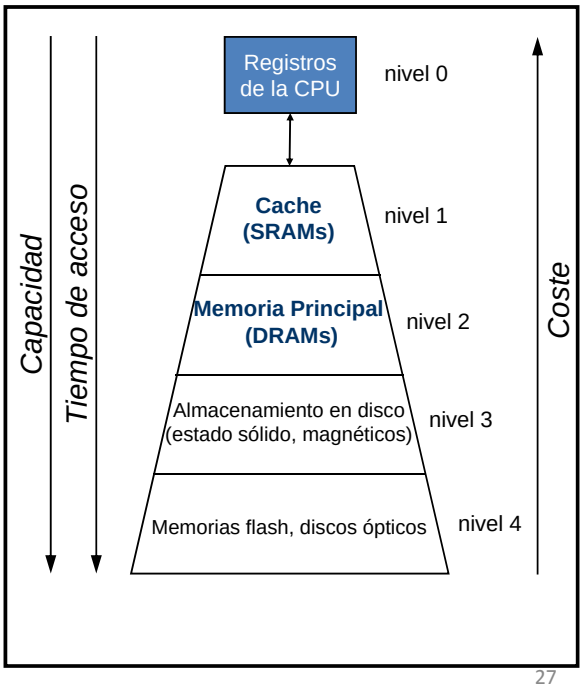
- Objetivo:
  - Conseguir una memoria de gran tamaño, rápida y al menor coste posible.
  - De forma transparente al usuario
- ¿Cómo organizar la memoria?
- Base: Principio de localidad

  "Cualquier programa accede a una porción relativamente pequeña de su espacio de direcciones en cualquier instante de tiempo"

Jerarquía de memoria

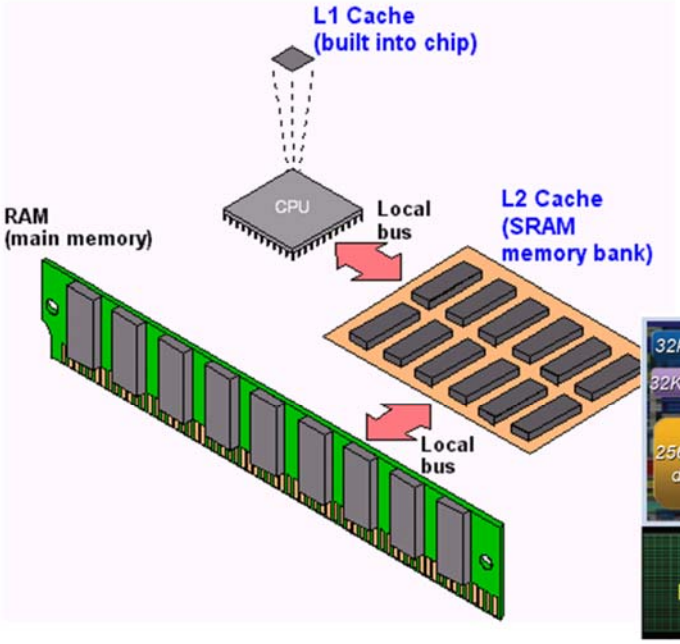
Niveles de la Jerarquía de memoria

- Un computador típico está formado por diversos niveles de memoria, *organizados de forma jerárquica*:
  - Registros de la CPU
  - Memoria Cache
  - Memoria Principal
  - Memoria Secundaria (discos)
  - Memorias flash y CD-ROMs
- El coste de todo el sistema de memoria excede al coste de la CPU
  - Es muy importante optimizar su uso

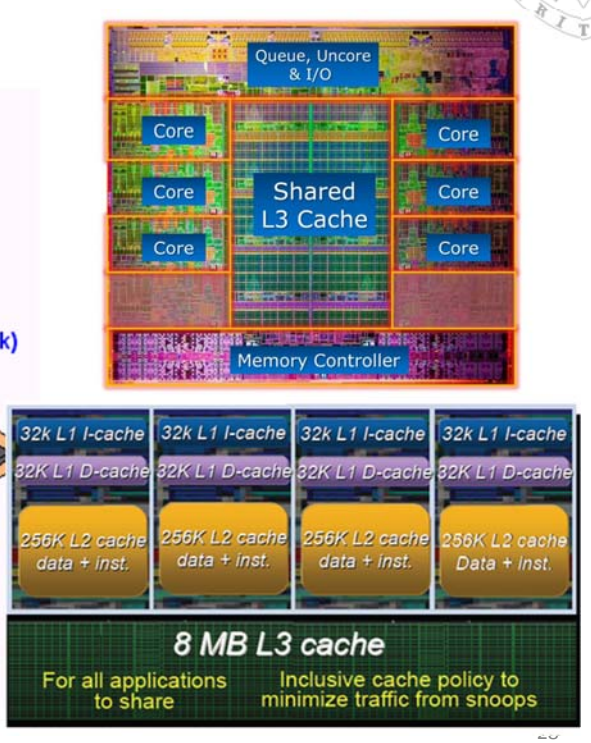


Jerarquía de memoria

- CPU Caché

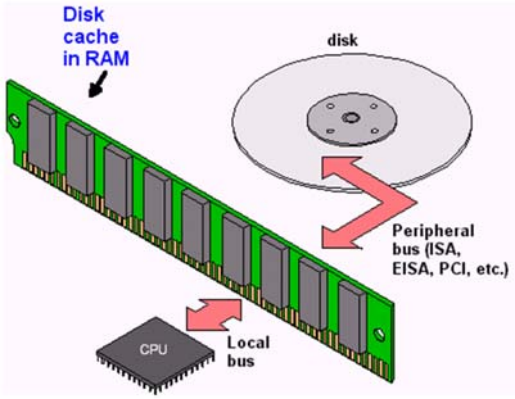


- Intel Core I7

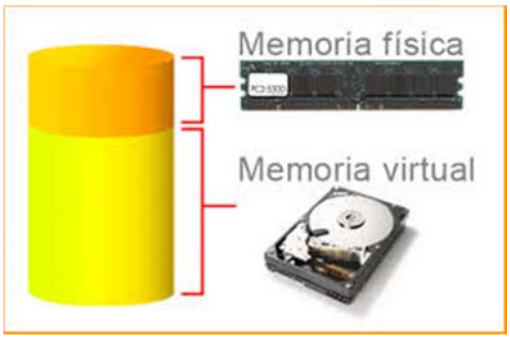


Jerarquía de memoria

- Caché de Disco



- Memoria virtual/swap:
  - Hacer creer a la CPU que tiene más memoria principal
  - Lo gestiona el Sist. Oper.



Principio de localidad

- Localidad temporal: Un dato usado en un determinado instante tiende a ser pronto reutilizado
- Localidad espacial: Si un dato es utilizado en un determinado instante, es muy probable que los datos cercanos a él sean también pronto utilizados

Localidad espacial

- Es habitual que un programa acceda datos que están almacenados en posiciones consecutivas
- Los arrays se almacenan en posiciones consecutivas y se suelen acceder secuencialmente

```
sum= 0;
for (i = 0; i < MAX; i++)
        sum= sum + a[i];
```

fc²

Localidad temporal

- Es habitual que un programa acceda la misma variable varias veces
- Lo óptimo es mantenerla en registro, pero no siempre es posible

```
sum= 0;
for (i = 0; i < MAX; i++)
        sum= sum + a[i];
```

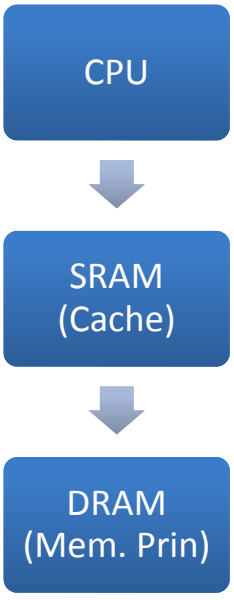
fc²

Jerarquía de memoria

- Jerarquía de memoria: múltiples niveles de memoria de diferentes velocidades y tamaños
- Se basa en el principio de localidad
- Mantiene los datos usados recientemente cerca del procesador (localidad temporal)
- Mueve junto con el dato pedido, otros datos cercanos a él, a niveles de memoria más cercanos al procesador (localidad espacial)

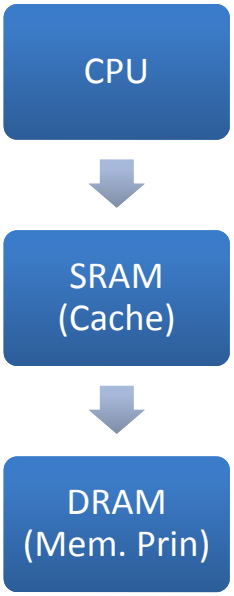
¿Cómo explotar la localidad temporal?

- Cada vez que se accede a memoria principal, llevamos una copia del dato a una memoria más pequeña y rápida (SRAM)
  - Introduce una penalización en el primer acceso
  - Será amortizada si se producen más accesos, pues se usará la copia y se evitará un acceso a memoria principal.



¿Cómo explotar la localidad espacial?

- Cada vez que se accede a la dirección i, llevamos una copia del dato a la cache
- Pero además, se copian varios datos más
  - Por ejemplo, si se accede al elemento a[0], se traen a la cache a[0],a[1],a[2] y a[3]
  - De nuevo, hay una penalización inicial que suele amortizarse.



fc²

Jerarquía de memoria

- La memoria más rápida (cache) se coloca más cerca del procesador ➡ Accesos más rápidos
- Y el nivel de memoria más lento será el más alejado
- La búsqueda de los datos comienza siempre en la cache, y de no encontrarse en ella, continúa por los demás niveles de memoria

Objetivo: Crear una Ilusión a la CPU.
- Memoria rápida (velocidad de la Caché)
- Memoria amplia (tamaño de la RAM)

fc²

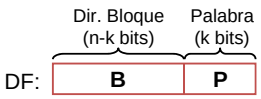
Gestión de la jerarquía de memoria

- **Vamos a trabajar con una memoria en 2 niveles**
  - Cache y memoria principal
- **Bloque**: unidad mínima de transferencia entre los dos niveles
- **Acierto (hit):** el dato solicitado está en la cache
- **Fallo (miss):** el dato solicitado no está en la cache y es necesario buscarlo en la memoria principal
- **Tasa de Aciertos (hit rate):** fracción de los accesos a memoria que producen un acierto en cache
  - Número de accesos que son acierto / Número total de accesos
- **Tasa de fallos (miss rate):** (1 – HitRate)

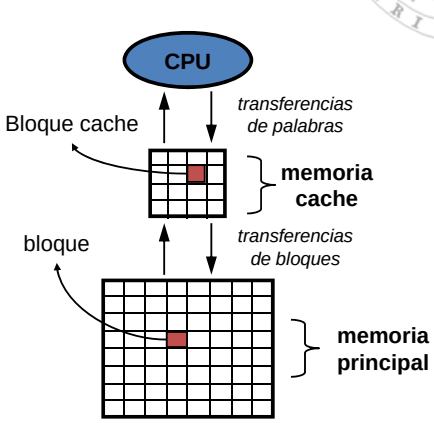
fc²

Memoria Cache

- **Memoria pequeña y rápida** situada entre el procesador y la memoria principal
  - Almacena una copia de la porción de información actualmente en uso de la memoria
- **Objetivo**: disminuir el tiempo de acceso a memoria
- **Estructura** del sistema memoria cache/principal:
  - *MP (memoria principal):*
    - formada por 2^n palabras direccionables
    - "dividida" en nB bloques de tamaño fijo de 2^K palabras por bloque
    - Campos de una dirección física:



  - *MC (memoria cache):*
    - formada por nM bloques (o líneas) de 2^K palabras cada uno (nM<<nB)
  - *Directorio (en memoria cache):*
    - Para cada bloque de MC, indica cuál es el bloque de MP que está alojado en él.



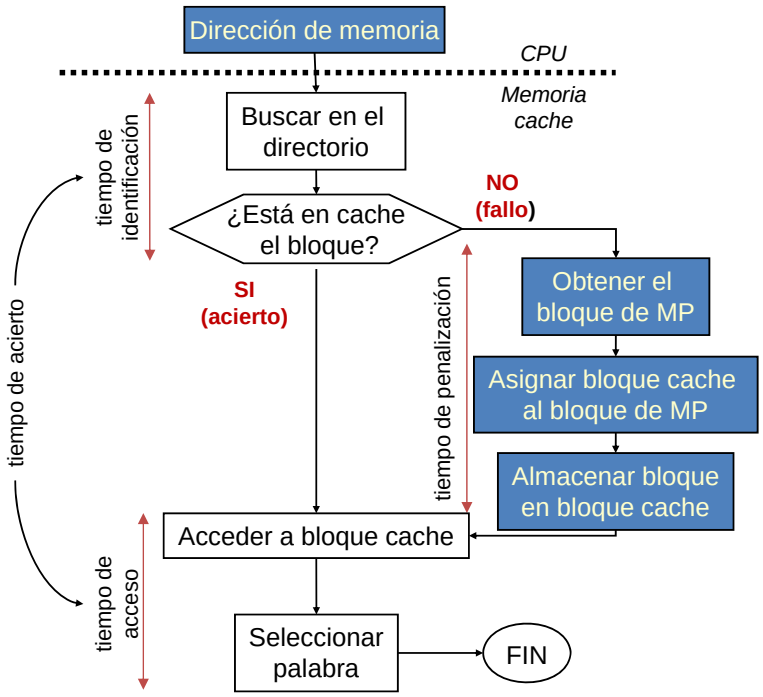
nB: número de bloques
nM: número de bloques de cache
B: dirección del bloque
M: dirección del bloque de cache
P: palabra dentro del bloque

fc²

Gestión de la jerarquía de memoria

- Cuando la CPU genera una referencia, busca en la cache
  - Si la referencia no se encuentra en la cache: **FALLO**
  - Cuando se produce un fallo, no solo se transfiere una palabra, sino que se lleva un **BLOQUE** completo de información de la MP a la MC
  - *Por la propiedad de localidad temporal*
    - Es probable que en una próxima referencia se direccione la misma posición de memoria
    - Esta segunda referencia no producirá fallo: producirá un **ACIERTO**
  - *Por las propiedades de localidad espacial*
    - Es probable que próximas referencias sean direcciones que pertenecen al mismo bloque
    - Estas referencias no producen fallo

Memoria cache



Principales objetivos:

- Maximizar la tasa de aciertos
- Minimizar el tiempo de acceso
- Minimizar el tiempo de penalización
- Reducir el coste hardware

Tiempo medio de acceso a M (T_{MAM})

$$T_{MAM} = T_{acierto} + (1-H)T_{penalizacion}$$

(frecuencia de hits)

Memoria Cache

- ¿Cómo sabemos que un dato está en la cache?
- Y si está, ¿cómo lo encontramos?

POLÍTICAS DE EMPLAZAMIENTO

Políticas de emplazamiento

- Política de emplazamiento:
  - Necesaria ya que existen menos bloques en MC que bloques en MP
  - Determina en qué bloque, o bloques, de MC, puede cargarse cada bloque de MP
- Existen diferentes políticas:
  - **Emplazamiento directo**
  - Emplazamiento asociativo
  - Emplazamiento asociativo por conjuntos

Políticas de emplazamiento

Emplazamiento directo:

Cada inquilino tiene asignado un **único** piso donde alojarse

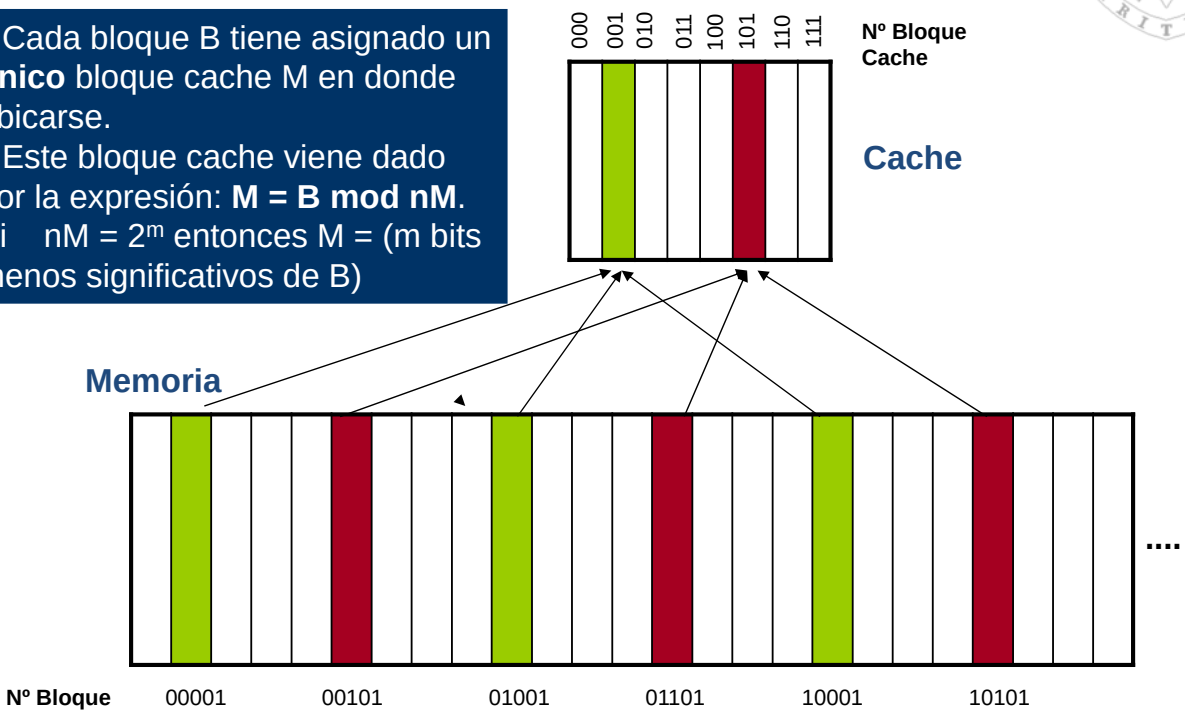
Cada bloque puede ir a un único lugar de la cache → **Búsqueda rápida**

↓

La localización en la cache se basa en la dirección del bloque en memoria

Emplazamiento directo

- Cada bloque B tiene asignado un **único** bloque cache M en donde ubicarse.
- Este bloque cache viene dado por la expresión: **M = B mod nM**. Si nM = 2^m entonces M = (m bits menos significativos de B)

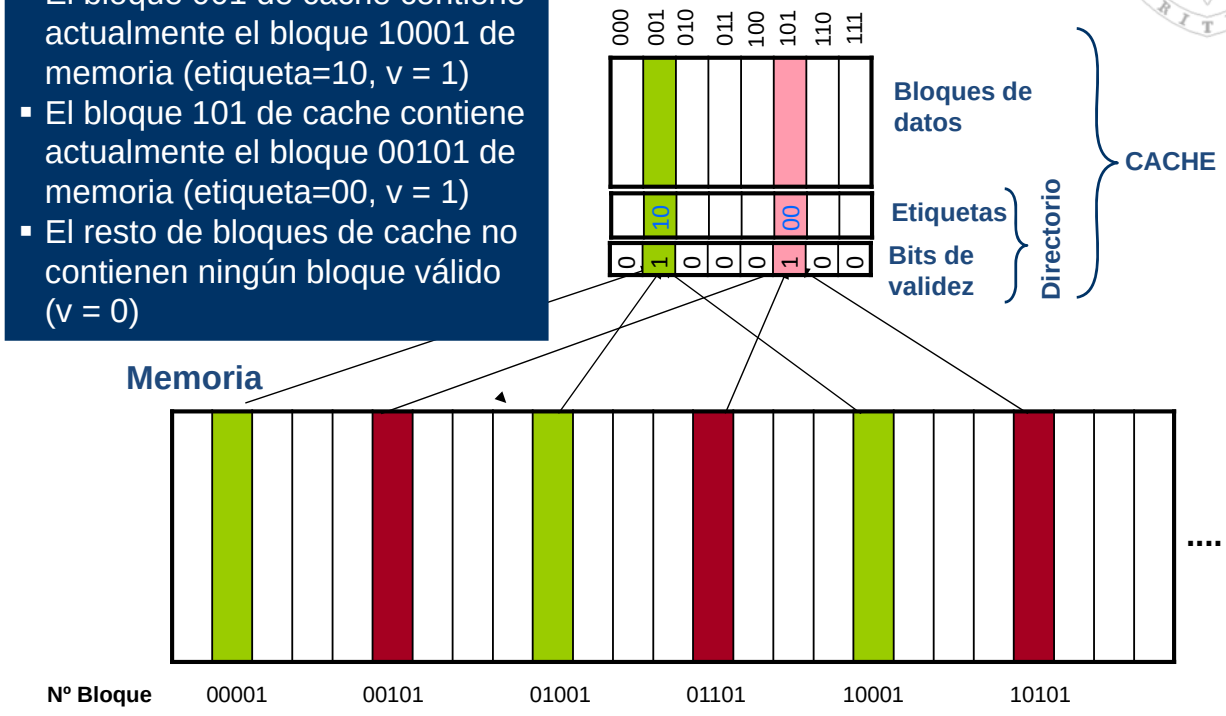


Emplazamiento directo

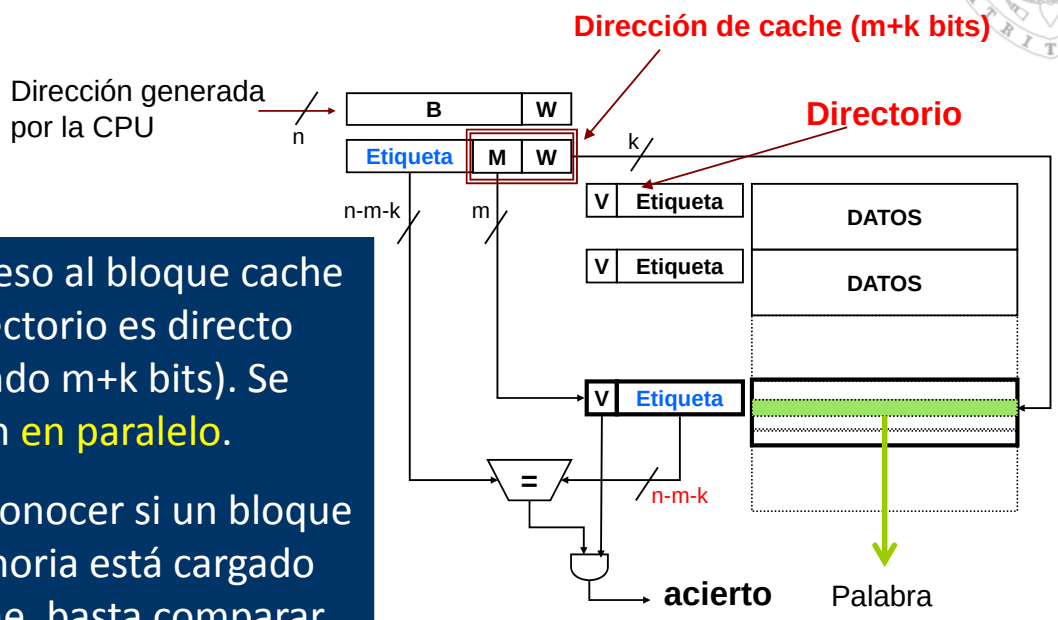
- Cada bloque de la cache puede contener diferentes bloques de memoria
  - ¿Cómo saber si el dato de la cache es el dato buscado?
- Solución:
  - El directorio almacena para cada bloque cache una etiqueta con los n-k-m bits que completan la dirección del bloque almacenado

Emplazamiento directo

- El bloque 001 de cache contiene actualmente el bloque 10001 de memoria (etiqueta=10, v = 1)
- El bloque 101 de cache contiene actualmente el bloque 00101 de memoria (etiqueta=00, v = 1)
- El resto de bloques de cache no contienen ningún bloque válido (v = 0)



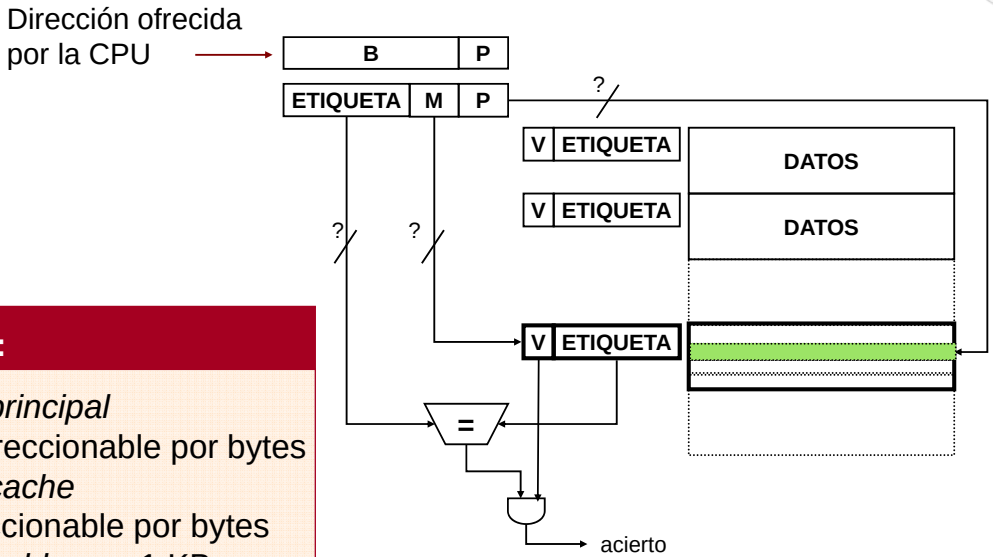
Emplazamiento directo



- El acceso al bloque cache y al directorio es directo (utilizando m+k bits). Se acceden en paralelo.
- Para conocer si un bloque de memoria está cargado en Cache, basta comparar las etiquetas y comprobar el bit de validez

fc²

Emplazamiento directo

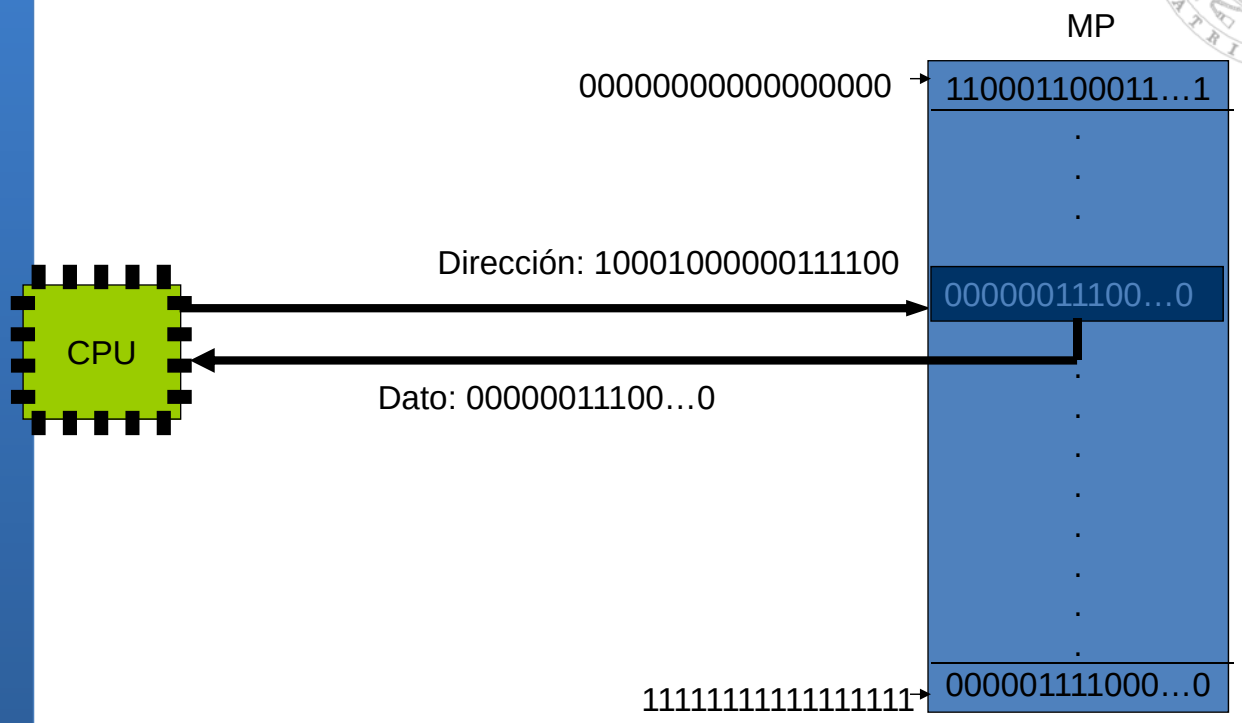


Ejemplo:

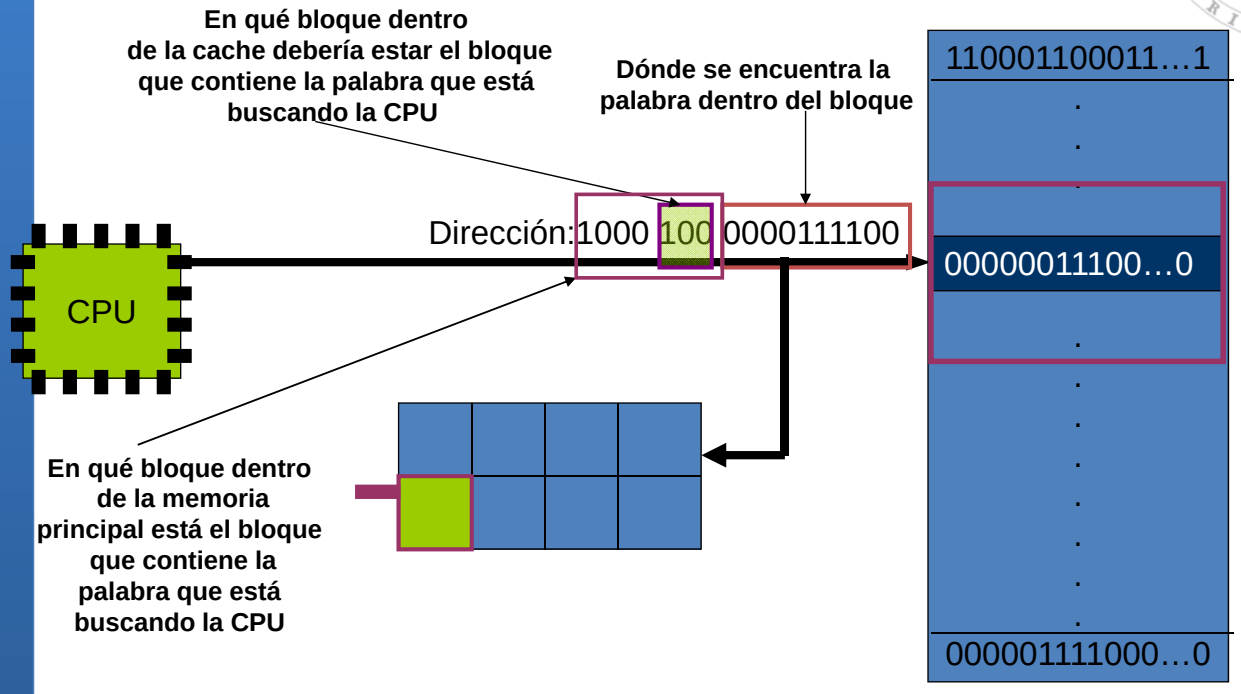
Memoria principal
128 KB direccionable por bytes
Memoria cache
8 KB direccionable por bytes
Tamaño de bloque: 1 KB
¿Número de bloques en MP?
¿Número de bloques en MC?

fc²

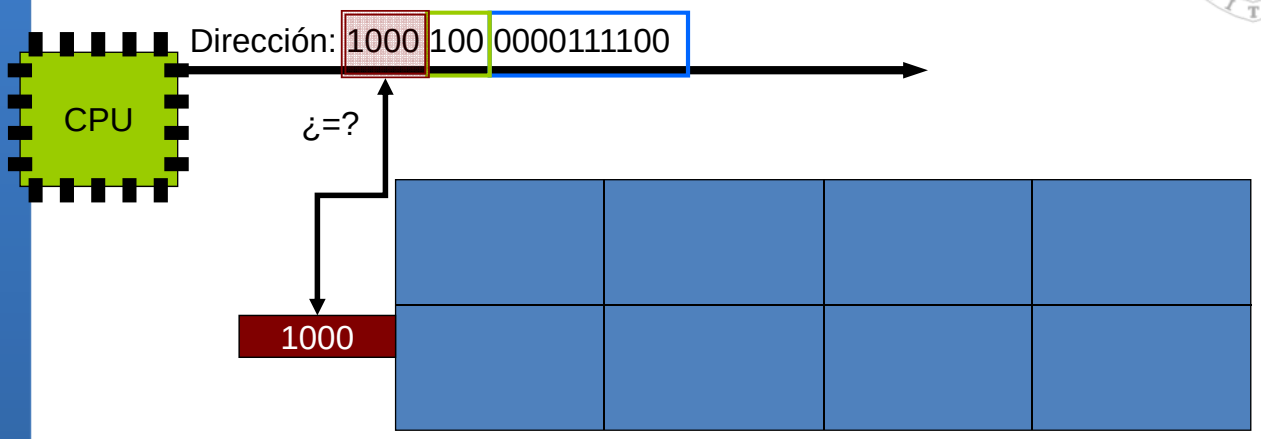
Emplazamiento directo



Emplazamiento directo

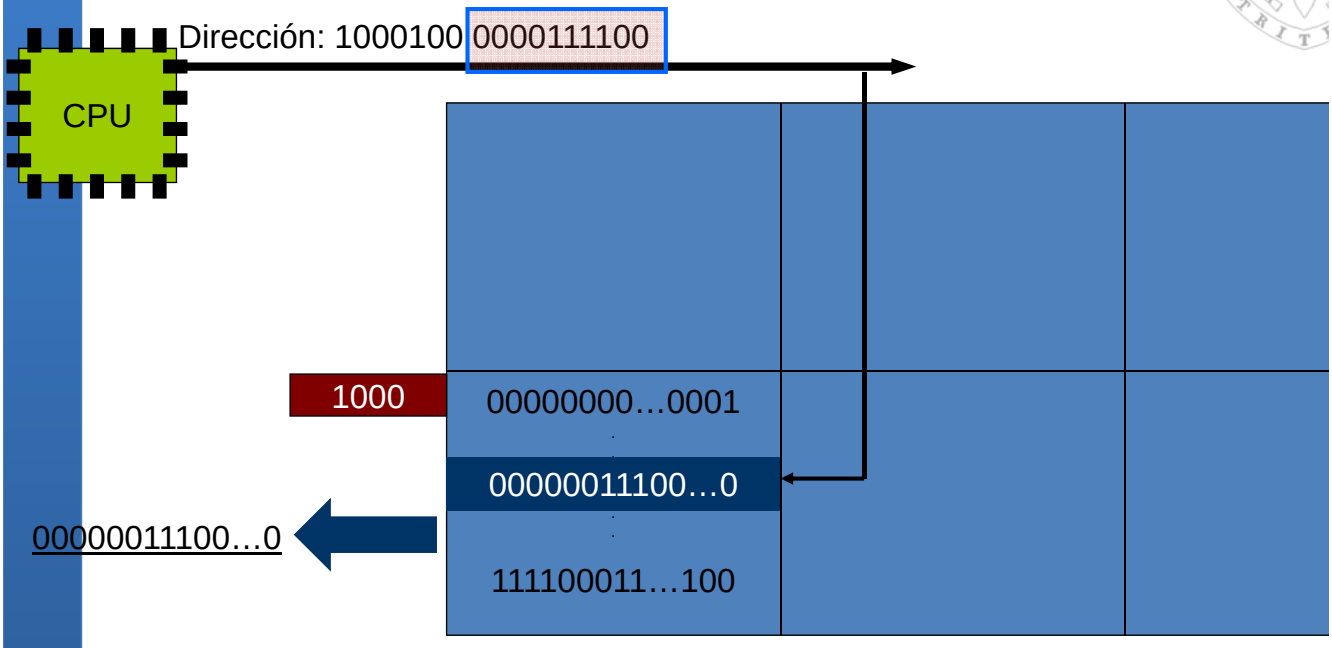


Emplazamiento directo



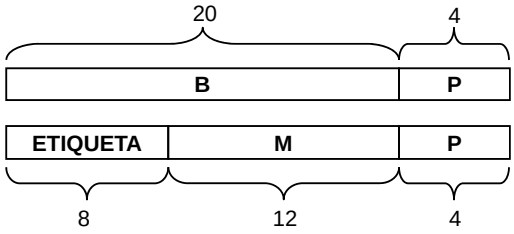
Si las etiquetas son iguales, entonces el bloque guardado en el bloque de cache es el bloque que estamos buscando

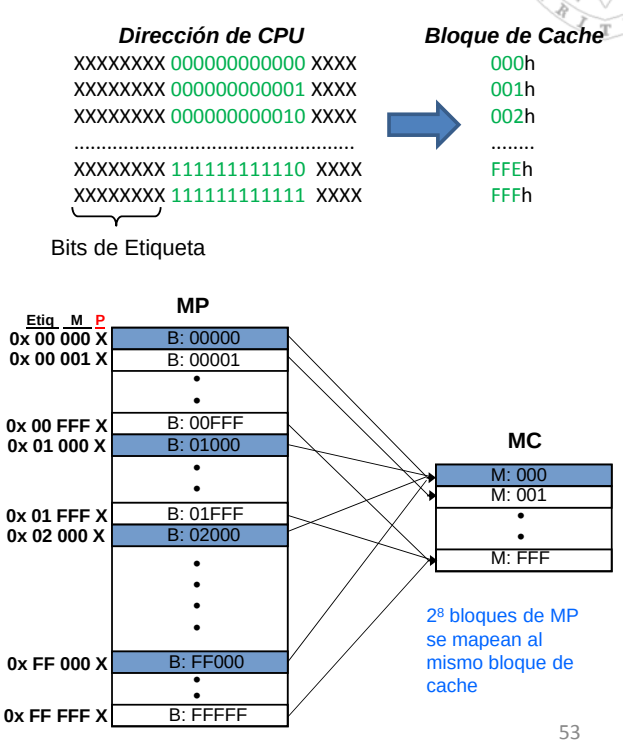
Emplazamiento directo



Emplazamiento directo

- *Memoria principal*
  - 16 Mb direccionable por bytes (dirección de 24 bits)
- *Memoria cache*
  - 64 Kb
- *Tamaño de bloque*: 16 bytes
  - Número de bloques en MP
    - nB = 1M (dirección de 20 bits)
  - Número de bloques en MC
    - nM = 4K (dirección de 12 bits)





Emplazamiento directo

Ejemplo:

- Se tiene una memoria con las siguientes características
  - *Memoria principal*
  - 1 MB direccionable por bytes
  - *Memoria cache*
  - 4 KB direccionable por bytes
  - *Tamaño de bloque*: 1 KB
- Se producen los siguientes accesos
  - 00005
  - 00006
  - 14605
  - 03805
  - 10005
  - 000F5

Emplazamiento directo

Ejemplo:

- 00005
  [0000 0000] [00] 00 0000 0101
  etiqueta m FALLO!
- 00006
  [0000 0000] [00] 00 0000 0110
  etiqueta m ACIERTO!
- 14605
  [0001 0100] [01] 10 0000 0101
  etiqueta m FALLO!
- 03805
  [0000 0011] [10] 00 0000 0101
  etiqueta m FALLO!
- 10005
  [0001 0000] [00] 00 0000 0101
  etiqueta m FALLO!
- 000F5
  [0000 0000] [00] 00 1111 0101
  etiqueta m FALLO!

Emplazamiento directo

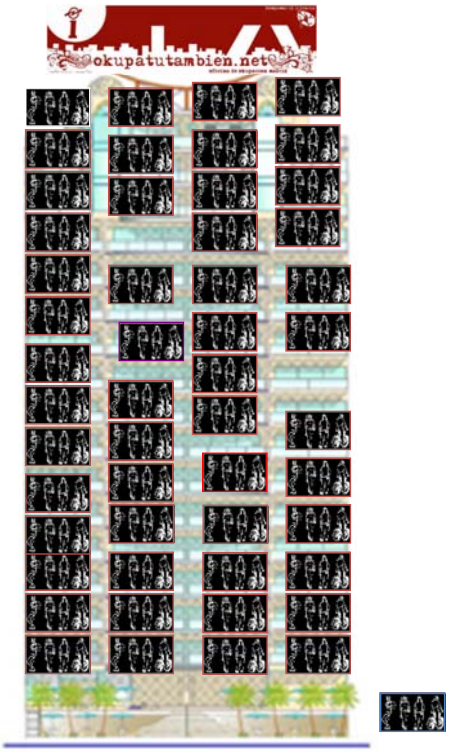
Ventajas:

- baja complejidad hardware
- rapidez en la identificación
- directorio pequeño

Principal problema:

Alta tasa de fallos cuando varios bloques compiten por el mismo bloque de MC

Políticas de emplazamiento



Emplazamiento asociativo:

Cada inquilino puede alojarse en **cualquier** piso

Cada bloque de memoria principal puede ubicarse en cualquier bloque cache

fc²

Políticas de emplazamiento

Emplazamiento asociativo por conjuntos:

Cada inquilino tiene asignada una **planta** puede ubicarse en **cualquiera piso** de dicha planta

La MC está dividida en nC conjuntos de nM/nC bloques cache cada uno

Cada bloque B tiene asignado un **conjunto fijo** C y puede ubicarse en **cualquiera de los bloques cache** que componen dicho conjunto

fc²

Políticas de actualización

¿Qué sucede cuando la CPU escribe una palabra?

Si se escribe sobre uno de los bloques cargados en la cache, el contenido de la MC y de la MP no coinciden ⇒ es necesario actualizar el bloque correspondiente de MP

Política de actualización

- **Escritura inmediata (*write-through*):** cada vez que se hace una escritura en la MC se actualiza inmediatamente la MP.
  - *Ventajas:* bajo coste hardware y consistencia en todo momento
  - *Desventajas:* aumenta el tráfico entre MC-MP

Políticas de actualización

- **Post-escritura (*copy-back*):** la MP se actualiza sólo cuando se reemplaza el bloque
  - Se utiliza un **bit de actualización** por bloque que indica si debe actualizarse en MP.
  - *Ventajas:* disminuye el tráfico entre MC y MP y disminuye el tiempo de acceso para escritura
  - *Desventajas:* inconsistencia
  - Para evitar retrasos en los reemplazamientos se utiliza un buffer intermedio (1 bloque)