

# Subsistema de Memoria

Organización de computadoras

Universidad Nacional de Quilmes

16 de junio de 2014

# Repaso

- a Notación científica
- b Punto Flotante
  - Interpretación
  - Rango
  - Resolución
  - Nacionalización y bit implícito
- c Estándar IEEE 754
  - Clases de números
    - ① Ceros
    - ② Denormalizados
    - ③ Normalizados
    - ④ Infinitos
    - ⑤ *Not A Number*
  - Interpretación

# Introducción

En las últimas décadas, cada 18 meses:

- 1 la velocidad de proceso (en cantidad de instrucciones ejecutadas por segundo) se duplica
- 2 el tamaño de la memoria se duplica
- 3 la velocidad de la memoria crece menos del 10 %

# Introducción

En las últimas décadas, cada 18 meses:

- 1 la **velocidad de proceso (en cantidad de instrucciones ejecutadas por segundo)** se duplica
- 2 el tamaño de la memoria se duplica
- 3 la velocidad de la memoria crece menos del 10 %

# Introducción

En las últimas décadas, cada 18 meses:

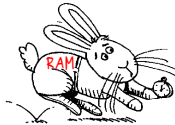
- 1 la velocidad de proceso (en cantidad de instrucciones ejecutadas por segundo) se duplica
- 2 el **tamaño de la memoria se duplica**
- 3 la velocidad de la memoria crece menos del 10 %

# Introducción

En las últimas décadas, cada 18 meses:

- 1 la velocidad de proceso (en cantidad de instrucciones ejecutadas por segundo) se duplica
- 2 el tamaño de la memoria se duplica
- 3 la **velocidad de la memoria crece menos del 10 %**

Crece la brecha entre la velocidad del procesador y  
la velocidad de la memoria



En la ejecución de cada  
instrucción se accede al menos  
una vez a memoria



En la ejecución de cada  
instrucción se accede al menos  
una vez a memoria



La velocidad de ejecución de  
instrucciones está limitada por el  
tiempo de acceso a memoria **que  
es mucho mayor al de la CPU**

## Jerarquía de memorias

¿Como mejorar la performance del procesador?

## Jerarquía de memorias

¿Como mejorar la performance del procesador?



Aumentar la cantidad de instrucciones ejecutadas  
por segundo

## Jerarquía de memorias

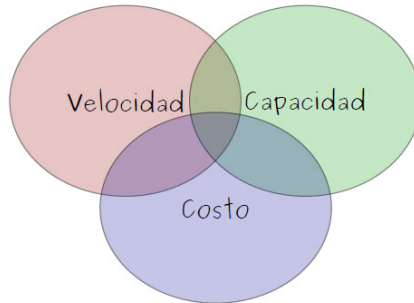
¿Como mejorar la performance del procesador?



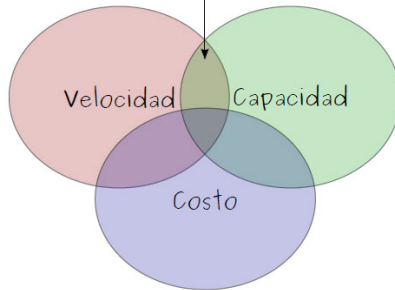
Aumentar la cantidad de instrucciones ejecutadas  
por segundo

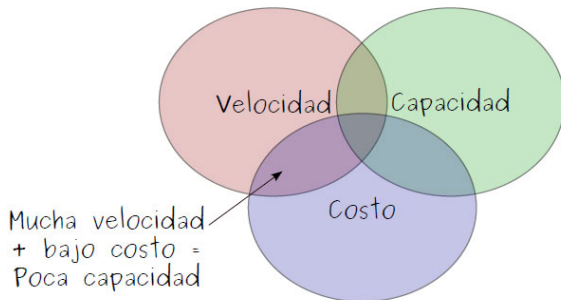
+ Disminuir el tiempo de acceso a memoria

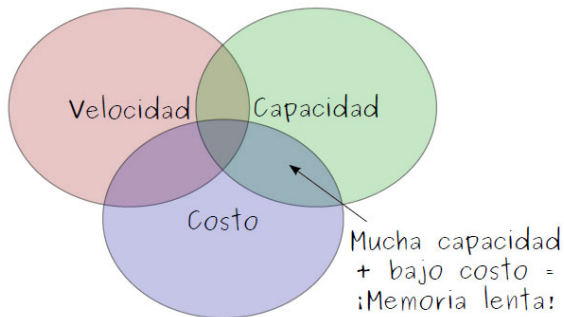
El dilema: ¿Cuánta? ¿Cuán rápida? ¿Cuán costosa?



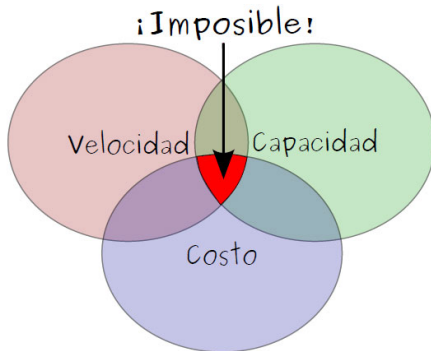
Mucha capacidad + mucha velocidad =  
¡Mucho costo!











Se necesita que la memoria tenga: gran capacidad y gran velocidad

## Jerarquía de memorias

Relación de compromiso entre el costo, la capacidad  
y el tiempo de acceso

## Jerarquía de memorias

Relación de compromiso entre el costo, la capacidad  
y el tiempo de acceso



- menor tiempo de acceso ➔ mayor costo por bit

## Jerarquía de memorias

Relación de compromiso entre el costo, la capacidad  
y el tiempo de acceso



- menor tiempo de acceso ➡ mayor costo por bit
- mayor capacidad ➡ menor costo por bit

## Jerarquía de memorias

Relación de compromiso entre el costo, la capacidad  
y el tiempo de acceso



- menor tiempo de acceso ➡ mayor costo por bit
- mayor capacidad ➡ menor costo por bit
- mayor capacidad ➡ mayor tiempo de acceso

## Camino hacia la solución:

- 1 Se incorpora memoria interna a la CPU (Registros)

## Camino hacia la solución:

- 1 Se incorpora memoria interna a la CPU (Registros)
- 2 Se incorpora memoria externa al sistema (operada a través de un módulo de E/S)

## Camino hacia la solución:

- 1 Se incorpora memoria interna a la CPU (Registros)
- 2 Se incorpora memoria externa al sistema (operada a través de un módulo de E/S)
- 3 Se utilizan muchos tipos de memoria combinados



## Camino hacia la solución:

- 1 Se incorpora memoria interna a la CPU (Registros)
- 2 Se incorpora memoria externa al sistema (operada a través de un módulo de E/S)
- 3 Se utilizan muchos tipos de memoria combinados

# Introducción

Subsistema de memoria: muchas memorias combinadas

## Introducción

Subsistema de memoria: muchas memorias  
combinadas



La interacción se optimiza para funcionar como una  
única memoria, grande y rápida

# Introducción

Subsistema de memoria: muchas memorias  
combinadas

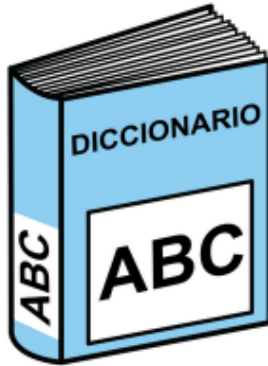


La interacción se optimiza para funcionar como una  
única memoria, grande y rápida



Se organizan en una jerarquía

# Antes, definiciones previas



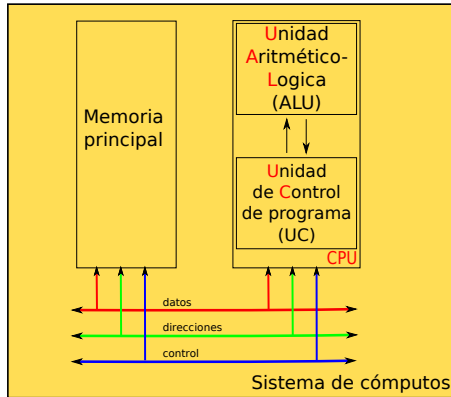
# Caracterización

Cada componente de almacenamiento puede caracterizarse según las dimensiones:

- Ubicación
- Unidad de transferencia
- Volatilidad
- Capacidad de escritura
- Método de acceso

# Caracterización

**Ubicación** Interna o Externa al sistema de cómputos.



## Caracterización

**Volatilidad** Una memoria es volátil cuando necesita alimentación eléctrica para mantener la información. Una memoria **no volátil** almacena información sin necesidad de tener alimentación y esa información no se deteriora.





## Caracterización

**Capacidad de escritura** Las memorias de **solo lectura** NO pueden ser alteradas luego de ser escritas. Necesariamente deben ser también **NO volátiles**.



## Read Only Memory (ROM)

Cuando un programa se carga en memoria permanece ahí hasta que

- a La memoria es sobreescrita
- b La computadora se apaga

## ROM: Memoria de sólo lectura

- 1 Para algunas aplicaciones, el programa no necesita ser cambiado y entonces puede ser 'hardwired' en una memoria de sólo lectura
- 2 Las ROMs se utilizan para almacenar programas en videojuegos, calculadoras, hornos de microondas, computadoras de vehículos, etc.


## ROM: Memoria de sólo lectura

- 1 Para algunas aplicaciones, el programa no necesita ser cambiado y entonces puede ser 'hardwired' en una memoria de sólo lectura
- 2 Las ROMs se utilizan para almacenar programas en videojuegos, calculadoras, hornos de microondas, computadoras de vehículos, etc.

¿Algún otro ejemplo?

# Métodos de acceso

## Caracterización: Método de acceso

**Secuencial** La dirección de cada dato está almacenada junto con él. Debe recorrerse secuencialmente para buscar el dato.  tiempo de acceso variable



Directo

Aleatorio

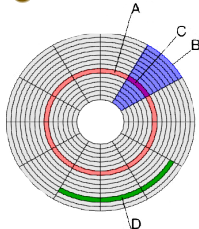
Asociativo

## Caracterización: Método de acceso

## Secuencial

**Directo** La dirección de cada dato se basa en su ubicación física. Etapas del acceso:

- 1 acceder directamente a la zona próxima del registro a buscar
- 2 buscar secuencialmente dentro de esa zona.




Aleatorio  
Asociativo

## Caracterización: Método de acceso

Secuencial

Directo

Aleatorio Cada dato tiene un mecanismo de acceso único y cableado físicamente  tiempo de acceso constante e independiente de los accesos anteriores. <sup>1</sup>

Asociativo

---

<sup>1</sup>Random:(Statistics) governed by or involving equal chances for each item.



## Caracterización: Método de acceso

Secuencial

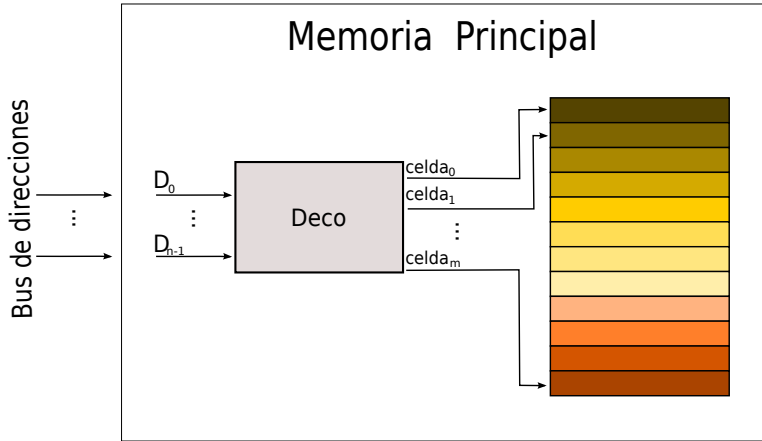
Directo

Aleatorio

**Asociativo** Cada celda es recuperada **en función de su contenido** en lugar de su dirección:

- Se busca un determinado patrón (clave) en **cierto conjunto de bits** de cada celda.
- Las comparaciones son simultáneas.
- **El tiempo de acceso es constante.**

## Método de acceso aleatorio

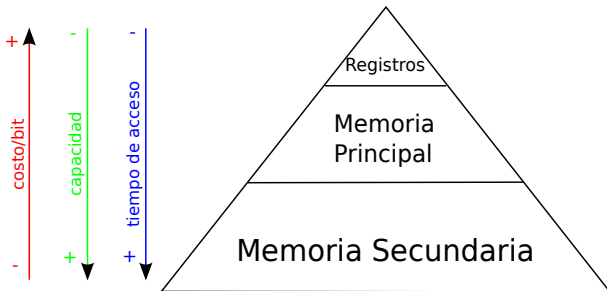


# Volvamos a la jerarquía de memorias

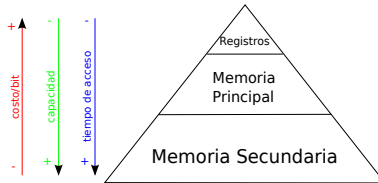
## Jerarquía de memorias

Solución: se emplea una jerarquía de memoria donde se complementan:

- costo por bit
- capacidad
- tiempo de acceso



## Jerarquía de memorias



- Registros: de velocidad igual a la cpu.
- Memoria principal: es volátil.
- Memoria secundaria: no volátil (ej: discos rígidos, tarjetas de memoria)

# Jerarquía de memorias

- ¿Porqué la memoria principal es volátil?
- ¿Porqué la memoria secundaria es no volátil?
- ¿Porqué las memorias principal y secundaria son de diferente tamaño?

## Jerarquía de memorias

¿Construir la memoria principal a base de registros?

## Jerarquía de memorias

¿Construir la memoria principal a base de registros?



¡NO!



# Principios de localidad

# Principios de localidad

## Localidad espacial

las posiciones de memoria mas cercanas a alguna referenciada recientemente son mas probables de ser referenciada que las mas distantes.

# Principios de localidad

## Localidad espacial

las posiciones de memoria mas cercanas a alguna referenciada recientemente son mas probables de ser referenciada que las mas distantes.



0000	ADD R0,R1
0001	SUB R1,R3
0002	AND R1,...
0003	... 0x0F0F

# Principios de localidad

## Localidad espacial

las posiciones de memoria mas cercanas a alguna referenciada recientemente son mas probables de ser referenciada que las mas distantes.



0000	ADD R0,R1
0001	SUB R1,R3
0002	AND R1,...
0003	... 0x0F0F

# Principios de localidad

## Localidad espacial

las posiciones de memoria mas cercanas a alguna referenciada recientemente son mas probables de ser referenciada que las mas distantes.



0000	ADD R0,R1
0001	SUB R1,R3
0002	AND R1,...
0003	... 0x0F0F

# Principios de localidad

## **Localidad temporal**

cuando un programa hace referencia de una posición de memoria, se espera que vuelva a hacerlo en poco tiempo

# Principios de localidad

## Localidad temporal

cuando un programa hace referencia de una posición de memoria, se espera que vuelva a hacerlo en poco tiempo



0000	ADD R0,R1
0001	SUB R1,R3
0002	JMP -3

# Principios de localidad

## Localidad temporal

cuando un programa hace referencia de una posición de memoria, se espera que vuelva a hacerlo en poco tiempo



0000	ADD R0,R1
0001	SUB R1,R3
0002	JMP -3



# Principios de localidad

## Localidad temporal

cuando un programa hace referencia de una posición de memoria, se espera que vuelva a hacerlo en poco tiempo



0000	ADD R0,R1
0001	SUB R1,R3
0002	JMP -3

# Principios de localidad

## Localidad temporal

cuando un programa hace referencia de una posición de memoria, se espera que vuelva a hacerlo en poco tiempo



0000	ADD R0,R1
0001	SUB R1,R3
0002	JMP -3

## Principios de localidad

Las celdas **mas usadas** se las quiere **mas a mano**

## Principios de localidad

Las celdas **mas usadas** se las quiere **mas a mano**



¿Cómo se relaciona esto con la jerarquía de memoria que propusimos?

## Principios de localidad

Las celdas **mas usadas** se las quiere **mas a mano**

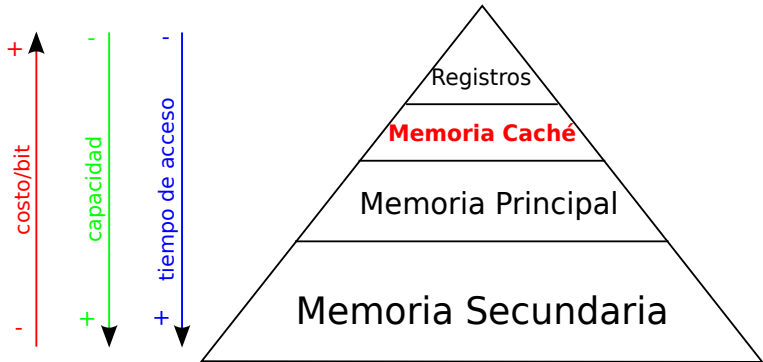


¿Cómo se relaciona esto con la jerarquía de memoria que propusimos?



Se puede agregar una memoria **mas pequeña** pero **mas rápida** que la principal, para almacenar estas celdas

## Jerarquía mejorada



## Objetivos

La memoria caché tiene como objetivo

## Objetivos

### La memoria caché tiene como objetivo

- proveer una velocidad de acceso cercana a la de los registros
- pero al mismo tiempo proveer un mayor tamaño de memoria

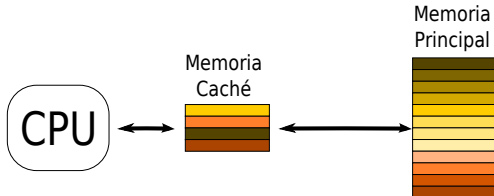


# Objetivos

## La memoria caché tiene como objetivo

- proveer una velocidad de acceso cercana a la de los registros
- pero al mismo tiempo proveer un mayor tamaño de memoria

Para esto la caché contiene una copia de porciones de la memoria principal



Pero... ¿Cómo es una lectura?

## Algoritmo de lectura

(1) La CPU pide una celda X

## Algoritmo de lectura

- (1) La CPU pide una celda X
- (2) La cache controla si *X está cacheada*

## Algoritmo de lectura

- (1) La CPU pide una celda X
- (2) La cache controla si X está *cacheada*
  - ✓ **Si está cacheada (hit o acierto):  
se otorga X a la CPU**

## Algoritmo de lectura

- (1) La CPU pide una celda X
- (2) La cache controla si X está *cacheada*
  - ✓ Si está cacheada (hit o acierto):  
se otorga X a la CPU
  - ✗ **Si NO está cacheada (fault o fallo):**
    - ① Se lee un bloque de memoria principal que contiene X
    - ② Se cachea el bloque
    - ③ se otorga X a la CPU

## Algoritmo de lectura

- (1) La CPU pide una celda X
- (2) La cache controla si X está *cacheada*
  - ✓ Si está cacheada (hit o acierto):  
se otorga X a la CPU
  - ✗ **Si NO está cacheada (fault o fallo):**
    - ① Se lee un bloque de memoria principal que contiene X
    - ② **Se cachea el bloque**
    - ③ se otorga X a la CPU

## Algoritmo de lectura

- (1) La CPU pide una celda X
- (2) La cache controla si X está *cacheada*
  - ✓ Si está cacheada (hit o acierto):  
se otorga X a la CPU
  - ✗ **Si NO está cacheada (fault o fallo):**
    - ① Se lee un bloque de memoria principal que contiene X
    - ② Se cachea el bloque
    - ③ **se otorga X a la CPU**



# Ajá... ¿Y la escritura?

## Algoritmo de escritura

(1) La CPU pide la escritura del valor **V** en la celda **X**

## Algoritmo de escritura

- (1) La CPU pide la escritura del valor V en la celda X
- (2) La cache controla si *X está cacheada*

## Algoritmo de escritura

- (1) La CPU pide la escritura del valor  $V$  en la celda  $X$
- (2) La cache controla si  $X$  está *cacheada*
  - ✓ Si está cacheada (hit o acierto):  
se escribe  $V$  en la copia de  $X$  en la cache

## Algoritmo de escritura

- (1) La CPU pide la escritura del valor  $V$  en la celda  $X$
- (2) La cache controla si  $X$  está *cacheada*
  - ✓ Si está cacheada (hit o acierto):  
se escribe  $V$  en la copia de  $X$  en la cache
  - ✗ Si **NO** está cacheada (fault o fallo):
    - ① Se lee un bloque de memoria principal que contiene  $X$
    - ② Se cachea el bloque
    - ③ se escribe  $V$  en la copia de  $X$  en la cache

## Algoritmo de escritura

- (1) La CPU pide la escritura del valor  $V$  en la celda  $X$
- (2) La cache controla si  $X$  está *cacheada*
  - ✓ Si está cacheada (hit o acierto):  
se escribe  $V$  en la copia de  $X$  en la cache
  - ✗ Si **NO** está cacheada (fault o fallo):
    - 1 Se lee un bloque de memoria principal que contiene  $X$
    - 2 **Se cachea el bloque**
    - 3 se escribe  $V$  en la copia de  $X$  en la cache

## Algoritmo de escritura

- (1) La CPU pide la escritura del valor V en la celda X
- (2) La cache controla si X está *cacheada*
  - ✓ Si está cacheada (hit o acierto):  
se escribe V en la copia de X en la cache
  - ✗ Si NO está cacheada (fault o fallo):
    - ① Se lee un bloque de memoria principal que contiene X
    - ② Se cachea el bloque
    - ③ se escribe V en la copia de X en la cache

¿Cómo comprueba la caché si  
X está cacheada?



## Memoria asociativa

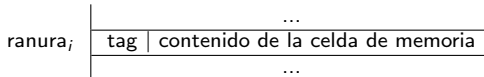
¿Cómo registra la caché qué celdas están  
cacheadas?

## Memoria asociativa

¿Cómo registra la caché qué celdas están cacheadas?



Almacena una etiqueta (**tag**) que identifica cada celda

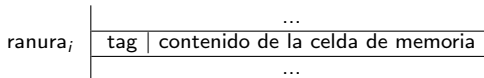


## Memoria asociativa

¿Cómo registra la caché qué celdas están cacheadas?



Almacena una etiqueta (**tag**) que identifica cada celda



Memoria Asociativa

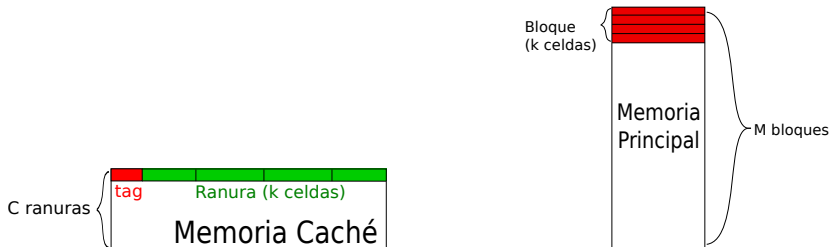
# Organización de la memoria caché

- 



## Organización de la memoria caché

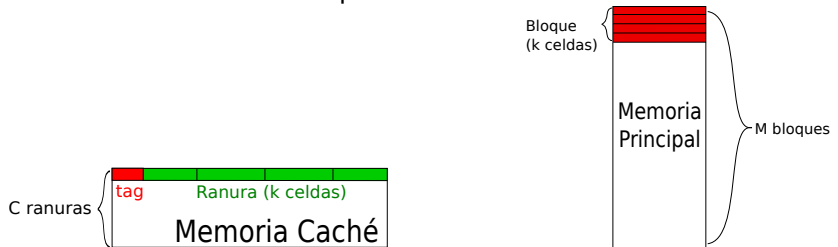
- La memoria principal se organiza en **M** bloques de **K** celdas cada uno.
- La memoria caché tiene **C** líneas (ranuras) de **K** celdas.



## Organización de la memoria caché

- La memoria principal se organiza en **M** bloques de **K** celdas cada uno.
- La memoria caché tiene **C** líneas (ranuras) de **K** celdas.

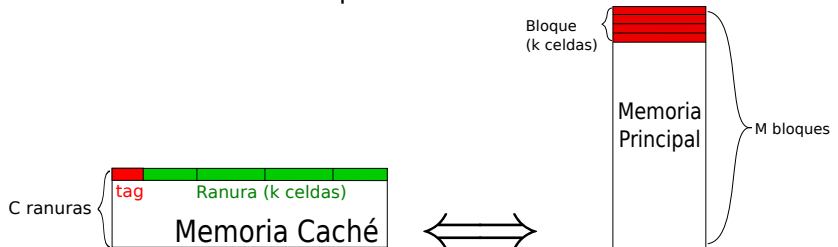
$C < M$  cada bloque no tienen ranura permanente se necesita identificar cada bloque en cada ranura



## Organización de la memoria caché

- La memoria principal se organiza en **M** bloques de **K** celdas cada uno.
- La memoria caché tiene **C** líneas (ranuras) de **K** celdas.

$C < M$  cada bloque no tienen ranura permanente se necesita identificar cada bloque en cada ranura



## Diseño de una caché: Tamaño

Debe ser:

- suficientemente pequeña para que el costo global se asemeje al de la memoria principal sola
- suficientemente grande para que el tiempo de acceso global se acerque al de la caché

Además, una caché mas grande tiende a ser un poco mas lenta.



# Función de correspondencia



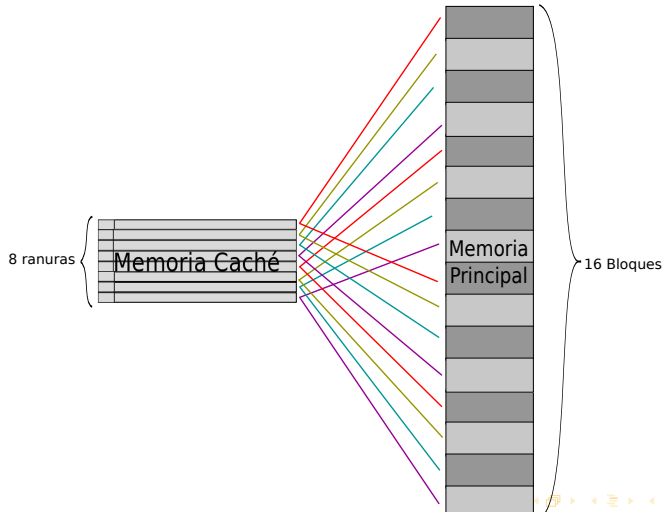
# Correspondencia directa

## Correspondencia directa

- Cada ranura se corresponde con un determinado conjunto de bloques de memoria, y cada bloque puede estar en solo una ranura.
- En cada ranura puede cargarse uno de  $16/8=2$  bloques de memoria.
- Para saber cual está cargado, se utiliza una etiqueta de 1 bit, y las direcciones de memoria se parten en los siguientes campos:

tag(1b)	ranura (3b)	palabra (2b)
---------	-------------	--------------

## Correspondencia directa



## Correspondencia directa

### Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda
- Función de correspondencia directa

¿Qué tamaño tiene el tag?

## Correspondencia directa

### Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda
- Función de correspondencia directa


¿Qué tamaño tiene el tag?



16 celdas = 16 bloques repartidos en 4 líneas

$16/4=4$  bloques por línea

Se debe distinguir cuál de esos 4 bloques es el que está almacenado

 tag=2bits

## Correspondencia directa

Para una dirección de memoria dada:

- 1 se identifica la ranura
- 2 se compara el primer bit con el tag de la ranura
- 3 si coinciden se recupera la palabra dentro del bloque según los últimos 2 bits

**Ventajas:** simple y económico para implementar. No requiere memoria de acceso aleatorio.

**Desventajas:** si un programa accede reiteradamente a palabras en diferentes bloques pero que se corresponden a la misma ranura ocurren muchos fallos.



## Correspondencia directa

### Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda
- Función de correspondencia directa

¿En que línea debo buscar la dirección 1010 ?

## Correspondencia directa

### Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda
- Función de correspondencia directa

¿En que línea debo buscar la dirección 1010 ?



(tag) 10	10 (línea)
----------	------------

# Correspondencia asociativa

## Correspondencia asociativa

- Cualquier bloque puede almacenarse en cualquier ranura.
- El tag corresponde al número de bloque

## Correspondencia asociativa

- Cualquier bloque puede almacenarse en cualquier ranura.
- El tag corresponde al número de bloque
- entonces la dirección se divide en:

tag(4b)	palabra (2b)
---------	--------------

## Correspondencia asociativa

- Cualquier bloque puede almacenarse en cualquier ranura.
- El tag corresponde al número de bloque
- entonces la dirección se divide en:

tag(4b)	palabra (2b)
---------	--------------

**Ventajas:** Flexibilidad

**Desventajas:** La política de reemplazo es compleja, requiriendo hardware y tiempo. Además, necesita una memoria de acceso aleatorio y un tag "grande".

## Correspondencia asociativa

### Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda
- Función de correspondencia asociativa

¿Qué tamaño tiene el tag?

## Correspondencia asociativa

### Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda
- Función de correspondencia asociativa

¿Qué tamaño tiene el tag?



No hay patrón de asignación... cualquier bloque en cualquier línea  
16 celdas = 16 bloques

Se debe distinguir cuál de esos 16 bloques es el que  
está almacenado en cada línea



## Correspondencia asociativa

### Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda
- Función de correspondencia asociativa

¿En que línea debo buscar la dirección 1010 ?

## Correspondencia asociativa

### Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda
- Función de correspondencia asociativa

¿En que línea debo buscar la dirección 1010 ?



¡En todas!

# Correspondencia asociativa por conjuntos

## Correspondencia asociativa por conjuntos

- Combina la flexibilidad del mapeo asociativo y la simplicidad del directo.
- Las ranuras se agrupan en conjuntos

## Correspondencia asociativa por conjuntos

- Combina la flexibilidad del mapeo asociativo y la simplicidad del directo.
- Las ranuras se agrupan en conjuntos

Suponer conjuntos de 2 ranuras  $\rightarrow$  la cache tiene 4 conjuntos.

- 1 Se usa el mapeo directo para encontrar el conjunto correspondiente a la dirección de memoria
- 2 Se usa mapeo asociativo dentro del conjunto para encontrar el bloque.

## Correspondencia asociativa por conjuntos

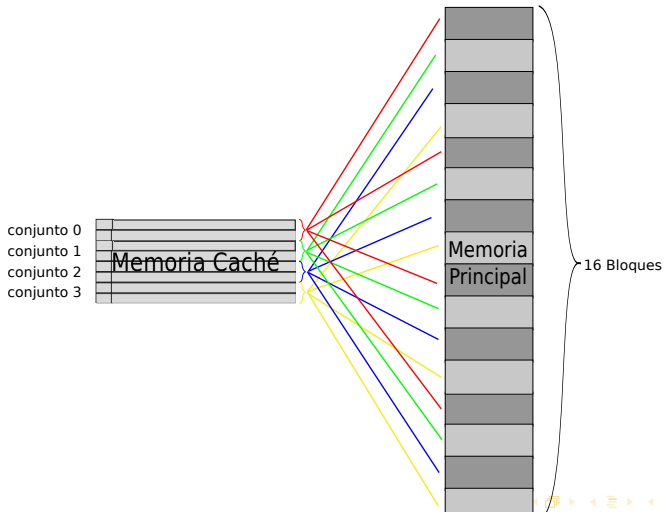
Para una dirección de memoria dada:

- 1 se identifica la ranura
- 2 se compara el primer bit con el tag de la ranura
- 3 si coinciden se recupera la palabra dentro del bloque según los últimos 2 bits

**Ventajas:** simple y económico para implementar. No requiere memoria de acceso aleatorio.

**Desventajas:** si un programa accede reiteradamente a palabras en diferentes bloques pero que se corresponden a la misma ranura ocurren muchos fallos.

## Correspondencia asociativa por conjuntos



## Correspondencia asociativa por conjuntos

La dirección se divide en:

tag(2b)	conjunto (2b)	palabra (2b)
---------	---------------	--------------

Para una dirección de memoria dada:

- 1 se identifica el conjunto
- 2 se comparan los primeros 2 bits con los tags de **las ranuras del conjunto**
- 3 si hay coincidencia se recupera la palabra dentro del bloque según los últimos 2 bits



## Correspondencia asociativa por conjuntos

### Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda y conjuntos de 2 líneas
- Función de correspondencia asociativa por conjuntos

¿Qué tamaño tiene el tag?


# Correspondencia asociativa por conjuntos

## Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda y conjuntos de 2 líneas
- Función de correspondencia asociativa por conjuntos


¿Qué tamaño tiene el tag?



16 celdas = 16 bloques. Conj de 2 líneas   $4/2 = 2$  conj

16 bloques/2 conjuntos = 8 bloques por conjunto

¿cuál de los 8 bloques está almacenado en cada línea del conjunto?

 tag=3bits

## Correspondencia asociativa por conjuntos

### Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda y conjuntos de 2 líneas
- Función de correspondencia asociativa por conjuntos

¿En que línea debo buscar la dirección 1010 ?

## Correspondencia asociativa por conjuntos

### Ejercicio

- Memoria principal de 16 celdas de 1 byte cada una
- Memoria caché de 4 líneas
- Bloques de 1 celda y conjuntos de 2 líneas
- Función de correspondencia asociativa por conjuntos

¿En que línea debo buscar la dirección 1010 ?



(tag) 101	0 (conjunto)
-----------	--------------

En todas las líneas del conjunto 0

# Algoritmos de reemplazo

# Algoritmos de reemplazo

Cuando se produce un fallo y un nuevo bloque debe ser cargado en la caché, debe elegirse una ranura (donde generalmente ya hay un bloque) para ocupar.

- Para el mapeo directo hay solo una posible ranura para un determinado bloque, así que no hay que elegir nada
- Para los mapeos **asociativo** y **asociativo por conjuntos** se necesita un algoritmo de reemplazo.

# Algoritmo LRU (Least Recently Used)

Menos recientemente usado

Se reemplaza el bloque que ha estado mas tiempo en la caché y sin referencias.

## Algoritmo LRU (Least Recently Used)

### Menos recientemente usado

Se reemplaza el bloque que ha estado mas tiempo en la caché y sin referencias.

Esto es facil de implementar en un caché asociativo por conjunto a 2 ranuras por conjunto:

- Cada ranura tiene un bit de USO.
- Cuando se hace referencia a un determinado bloque, su ranura se marca con  $USO=1$  y la otra ranura con  $USO=0$
- Cuando se trae un nuevo bloque se utiliza la ranura marcada  $USO=0$ .

Según el principio de localidad, este enfoque dará una buena tasa de aciertos.



## Algoritmo FIFO (First In-First Out)

El primero que llega es el primero que se va.

Se reemplaza el bloque que ha estado en la caché por mas tiempo.

## Algoritmo FIFO (First In-First Out)

El primero que llega es el primero que se va.

Se reemplaza el bloque que ha estado en la caché por mas tiempo.

Se implementa con una lista circular:

- Las referencias a los bloques no computan nada
- Cuando se trae un nuevo bloque:
  - 1 se pone en el lugar del primero de la lista
  - 2 se actualiza la lista para que el nuevo quede al final.

# Algoritmo LFU (Least Frequently Used)

Menos frecuentemente usado.

- Se reemplaza el bloque que ha sido usado menos veces.

## Algoritmo LFU (Least Frequently Used)

Menos frecuentemente usado.

- Se reemplaza el bloque que ha sido usado menos veces.
- Puede implementarse con un contador de accesos en cada ranura.

# Algoritmo Aleatorio

- Se elige una ranura al azar
- Algunos estudios de simulación muestran que esta implementación es apenas menos performante que los anteriores

## Políticas de escritura

Antes de que un bloque pueda ser reemplazado debe analizarse si fue modificado en caché pero no en memoria.

- ✗ Si NO fue modificado → se pisa con el bloque nuevo
- ✓ Si fue modificado → La memoria debe ser actualizada.

## Políticas de escritura

Las posibles políticas de escritura deben tener en cuenta lo siguiente:

- Más de un dispositivo puede tener acceso a la memoria principal (ej: modulo de entrada-salida) Si una palabra fue alterada en la caché es inválida en Memoria principal.
- Cuando varios procesadores comparten la memoria a través de distintas caché.

# Write-through

- Es la política mas simple.
- Todas las escrituras de caché se replican a memoria principal en el acto.

**Ventajas:** Puede haber otros dispositivos cach-cpu que monitoreen las lineas de escritura a memoria.

**Desventajas:** Puede generarse un cuello de botella en memoria



## Write-back

- Las actualizaciones a memoria se hacen sólo cuando un bloque es candidato a ser reemplazado.
- Se utiliza un bit *dirty* que se enciende cuando se hace una modificación en cache
- si *dirty*=1 cuando se reemplaza el bloque, se escribe en memoria

**Ventaja:** Las escrituras son mas eficientes

**Desventaja:** Algunas partes de la memoria son inválidas → los accesos de los módulos de entrada-salida deben pasar por caché (cuello de botella)

## Write-back vs. Write-through

- La experiencia muestra que solo el 15 % son accesos de escritura, por lo que una simple política Write-through es mas comunmente usada que Write-back

## Tamaño de bloque

**¿Cuál es el tamaño de bloque óptimo?**

## Tamaño de bloque

### ¿Cuál es el tamaño de bloque óptimo?

Cuando se hace referencia a una palabra, se carga en la caché un número de palabras adyacentes.

## Tamaño de bloque

### ¿Cuál es el tamaño de bloque óptimo?

Cuando se hace referencia a una palabra, se carga en la caché un número de palabras adyacentes.

- Si el bloque crece, la tasa de aciertos crece por el principio de localidad, hasta un cierto límite cuando se incluyen palabras demasiado distantes.
- Los bloques mas grandes reducen el número de bloques en caché → la información es reemplazada mas rápido

## Tamaño de bloque

### ¿Cuál es el tamaño de bloque óptimo?

Cuando se hace referencia a una palabra, se carga en la caché un número de palabras adyacentes.

- Si el bloque crece, la tasa de aciertos crece por el principio de localidad, hasta un cierto límite cuando se incluyen palabras demasiado distantes.
- Los bloques mas grandes reducen el número de bloques en caché → la información es reemplazada mas rápido

La relación entre tamaño de bloque y tasa de aciertos es compleja y no hay un valor óptimo para el tamaño de bloque. Un valor razonable es entre 4 y 8 palabras.

## Tasa de acierto y Tiempo efectivo de acceso

Dos indicadores de la performance de la memoria caché son:

- Si tenemos  $n$  accesos a memoria, y de esos  $k$  fueron hit, la tasa de aciertos se calcula como:

$$ta = \frac{\#k}{\#n}$$

- la tasa de fallos se calcula como:

$$tf = 1 - ta$$

## Tiempo de acceso Total

Si tenemos  $n$  accesos, una tasa de aciertos  $T_a$  y conocemos el tiempo de cache  $t_c$  y el tiempo de respuesta de la memoria  $t_m$ :

- $t_p = n \cdot T_a \cdot t_c + n \cdot (1 - T_a) \cdot (t_c + t_m)$



## Ejemplo

- Memoria principal con
  - a bloques de 16 palabras
  - b tiempo de acceso de 2500ns
- Memoria caché con
  - a mapeo (correspondencia) directo
  - b 4 ranuras de 16 palabras
  - c tiempo de acceso de 80ns
  - d inicialmente vacía
- Programa: Ensamblado en las celdas 48 a 95

## Ejemplo

Evento	Dirección	Bloque	Tiempo
1 fallo	48	3	$2500\text{ns} + 80\text{ns}$
15 aciertos	49-63	3	$80 \times 15\text{ns} = 1200\text{ns}$
1 fallo	64	4	$2500\text{ns} + 80\text{ns}$
15 aciertos	65-79	4	$80 \times 15\text{ns} = 1200\text{ns}$
1 fallo	80	5	$2500\text{ns} + 80\text{ns}$
15 aciertos	81-95	5	$80 \times 15\text{ns} = 1200\text{ns}$

## Ejemplo

#aciertos: 45

#fallos: 3

#refs 48

tasa de aciertos =  $45/48 = 0,9375$

tiempo de acceso efectivo =  $\frac{45 \cdot 80ns + 3 \cdot (2500ns + 80ns)}{48} = \frac{3600 + 7740}{48} =$   
 $\frac{11340ns}{48} = 236,25ns$

## Ejemplo

#aciertos: 45

#fallos: 3

#refs 48

tasa de aciertos =  $45/48 = 0,9375$

tiempo de acceso efectivo =  $\frac{45 \cdot 80ns + 3 \cdot (2500ns + 80ns)}{48} = \frac{3600 + 7740}{48} =$   
 $\frac{11340ns}{48} = 236,25ns$

!! el TAE es mucho mayor al TAC, pero se debe al alto costo de un acceso a memoria.



- 1 Jerarquía de memorias
- 2 Diseño de una caché
  - Tamaño
  - Función de correspondencia
  - Algoritmos de reemplazo
  - Políticas de escritura
    - Write-through
  - Tamaño de bloque
- 3 Tiempo de acceso
- 4 Performance de la caché

# Read Only Memory (ROM)

## ROM: Memoria de sólo lectura

Cuando un programa se carga en memoria permanece ahí hasta que

- a La memoria es sobrescrita
- b La computadora se apaga

## ROM: Memoria de sólo lectura

- 1 Para algunas aplicaciones, el programa no necesita ser cambiado y entonces puede ser 'hardwired' en una memoria de sólo lectura
- 2 Las ROMs se utilizan para almacenar programas en videojuegos, calculadoras, hornos de microondas, computadoras de vehículos, etc.



## ROM: Memoria de sólo lectura

- 1 Para algunas aplicaciones, el programa no necesita ser cambiado y entonces puede ser 'hardwired' en una memoria de sólo lectura
- 2 Las ROMs se utilizan para almacenar programas en videojuegos, calculadoras, hornos de microondas, computadoras de vehículos, etc.

¿Algún otro ejemplo?

# PROM: Memoria de sólo lectura programable

## PROM

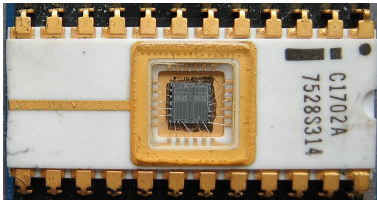
- 1 Para grandes producciones, la ROM se programa en fábrica, pero también existen las **ROM Programables (PROM)** que son escritas por el usuario usando un dispositivo para tal fin.
- 2 Las **PROM** pueden utilizarse para implementar Unidades de control o ALUs

# EPROM: Memoria de sólo lectura programable

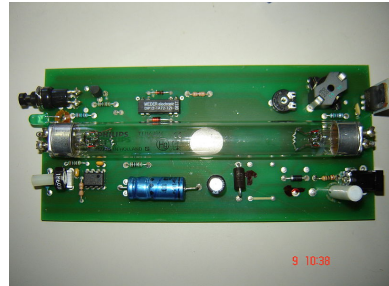
## EPROM y EEPROM

Si es necesario escribirla mas de una vez:

- 1 Las **EPROM** pueden ser borradas con luz ultravioleta,
- 2 Las **EEPROM** que pueden borrarse eléctricamente.



Memoria EPROM



Borrador de EPROM