

Subsistema de Memoria

Organización de computadoras

Universidad Nacional de Quilmes

<http://orga.blog.unq.edu.ar>

a Compuertas lógicas:

- 1 Básicas: AND, OR, NOT
- 2 Derivadas: NOR, NAND, XOR

b Circuitos secuenciales:

- 1 Suma de productos (SOP)
- 2 Cómo construir un circuito:

Enunciado  Tabla de verdad  SOP  Circuito

- 3 Circuitos mas usados:

- Mux/demux
- Codec/deco

- 4 Circuitos aritméticos:

- Half Adder para 1 bit
- Full Adder para 1 bit
- Full Adder para N bits
- Restador de 1 bit

Algunos Datos

Modelo	Año	Máxima Velocidad
8086, 8088	1978-1979	4 Mhz
80286	1982	48 Mhz
80386, 386DX	1985	33 Mhz
486DX, 4863DX, 486DX4	1989	100 Mhz
Pentium, PentiumMMX	1992	300 Mhz
Pentium Pro, Pentium II, Celeron, Pentium II Xeon	1995	550 Mhz
Pentium III	1998	1000 Mhz
Pentium 4	2000	1400 Mhz

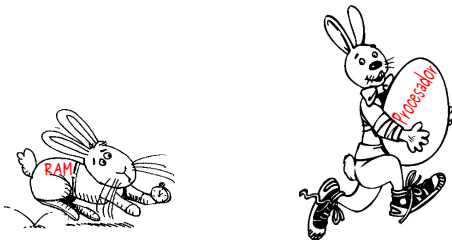
En las últimas décadas:

- 1 Cada 18 meses: la velocidad de la CPU (en cantidad de instrucciones ejecutadas por segundo) se duplica manteniendo su costo.
- 2 Cada 18 meses: el tamaño de la memoria, se duplica

La velocidad de la memoria solo creció a una taza menor al 10 % anual



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



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

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

Subsistema de memoria: muchas memorias
combinadas

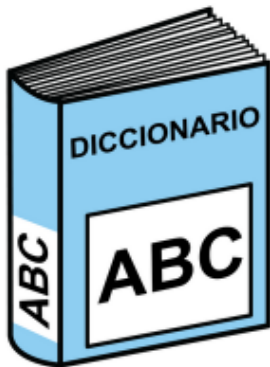


La interacción se optimiza para funcionar como una
única memoria, grande y rápida



Se organizan en una jerarquía

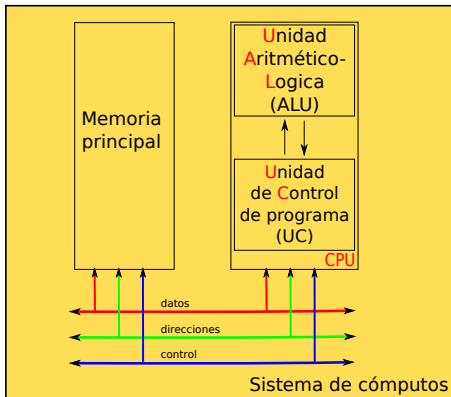
Antes, definiciones previas



Cada componente de almacenamiento puede caracterizarse según las dimensiones:

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

Ubicación Interna o Externa al sistema de cómputos.



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.

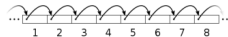


Capacidad de escritura Las memorias de **solo lectura** NO pueden ser alteradas luego de ser escritas. Necesariamente deben ser también **NO volátiles**.



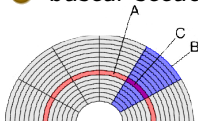
Caracterización: Método de acceso

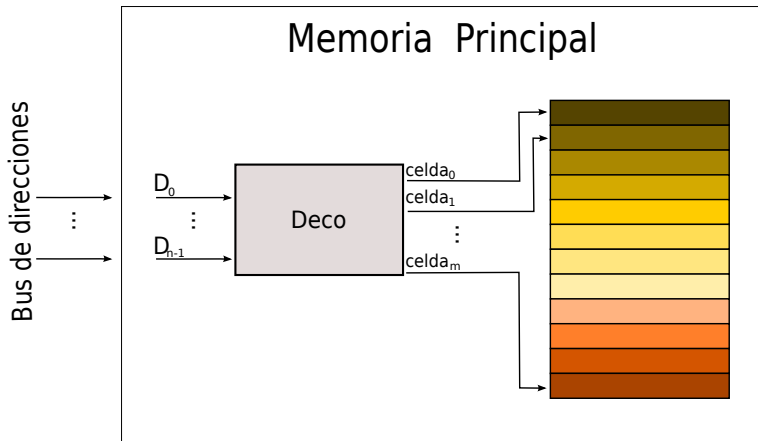
Secuencial Los datos se organizan en celdas cuya dirección está almacenada junto con el dato. Debe recorrerse secuencialmente para acceder a un registro determinado.  tiempo de acceso variable



Directo Los datos tienen una dirección basada en su ubicación física. El acceso tiene dos etapas:

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





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

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

¿Algún otro ejemplo?

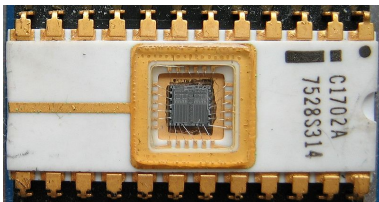
PROM

- 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

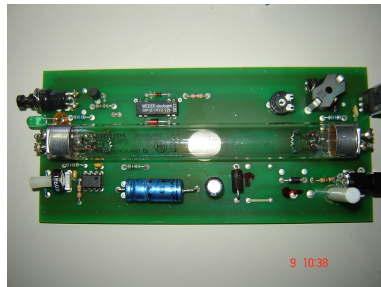
EPROM y EEPROM

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

Jerarquía de memorias

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



Se necesita: gran capacidad y mejor performance
(menor tiempo de acceso)

Existe una relación de compromiso entre el costo, la capacidad y el tiempo de acceso

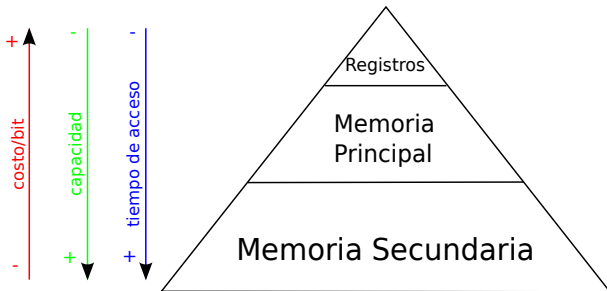


- menor tiempo de acceso mayor costo por bit
- mayor capacidad menor costo por bit
- mayor capacidad mayor tiempo de acceso

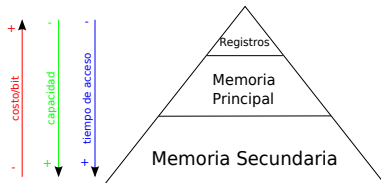
Jerarquía de memorias

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)

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

¿Como mejorar la performance del procesador?



Aumentar la cantidad de instrucciones ejecutadas por segundo
+ Disminuir el tiempo de acceso a memoria



¿Construir la memoria principal a base de registros?



¡NO!

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

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

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

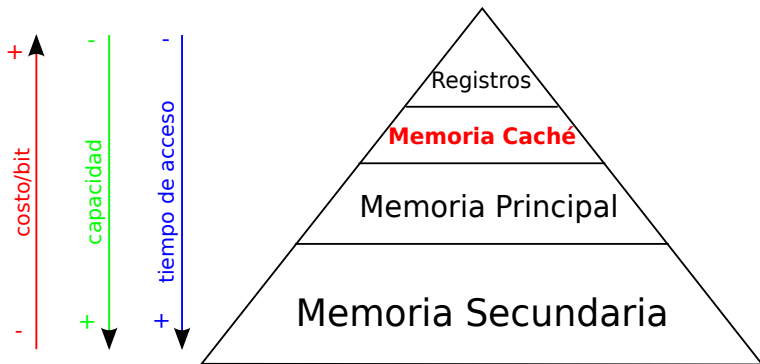


¿Cómo se relaciona esto con la jerarquía de memoria que propusimos?



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

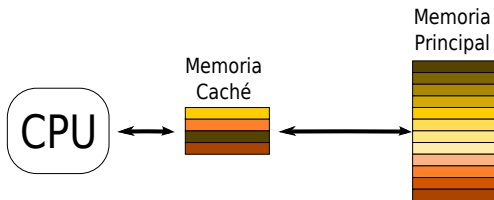
Jerarquía mejorada



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?

- (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

(1) La CPU pide una celda X

(2) 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

Ajá... ¿Y la escritura?

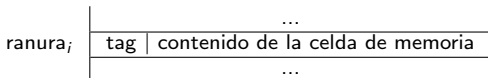
- (1) La CPU pide la escritura del valor **V** en la celda **X**
- (2) La cache controla si **X** *está cacheada*
 - ✓ **Si está cacheada (hit o acierto):**
se escribe **V** en la copia de **X** *en la cache*
 - ✗ **Si NO está cacheada (fault o fallo):**
 - ① Se lee un bloque de memoria principal que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

¿Cómo registra la caché qué celdas están cacheadas?



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

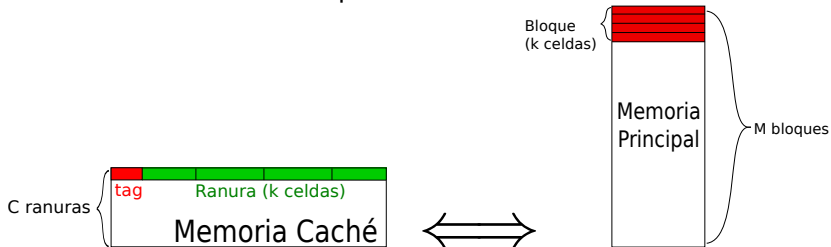


Memoria Asociativa

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



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.

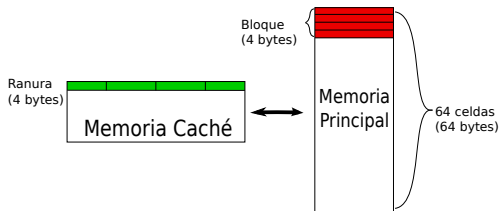
Función de correspondencia

Diseño de una caché: Función de correspondencia

La función de correspondencia (mapeo) asocia los bloques de memoria con ranuras de la caché.

Suponer:

- una memoria de $2^6 = 64$ celdas
- un tamaño de bloque de 4 celdas $\rightarrow$ 16 bloques
- una caché de 8 ranuras de 4 celdas cada una

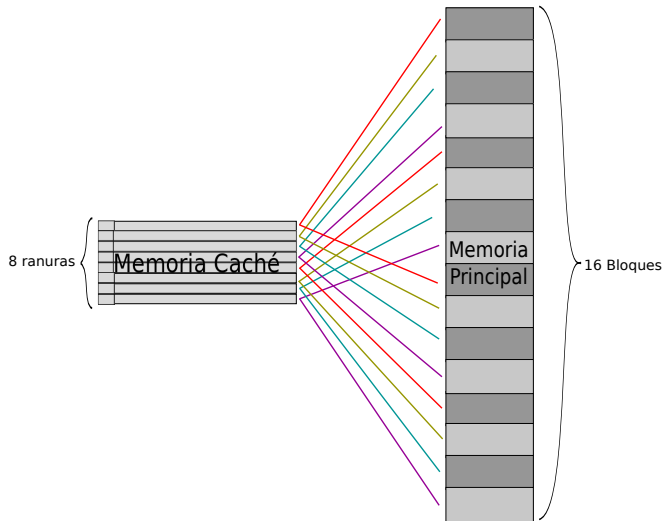


Correspondencia directa

- 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



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

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.

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

- 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".

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

 tag=4bits

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

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

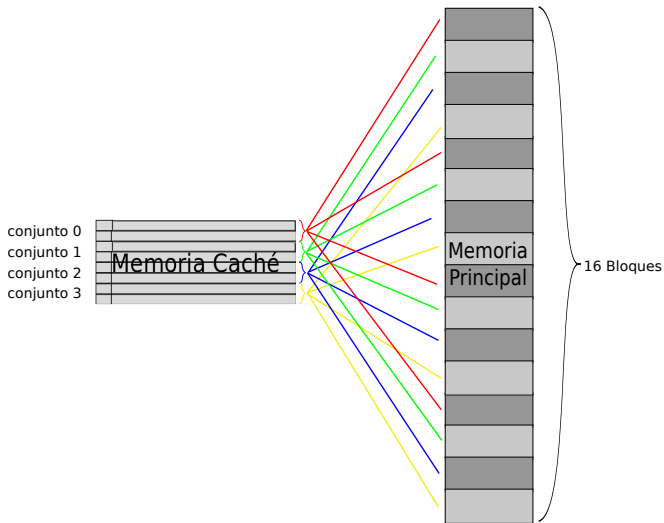
Para una dirección de memoria dada:

- 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

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

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

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.

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.

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.
- Puede implementarse con un contador de accesos en cada ranura.

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

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.

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.

- 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 líneas de escritura a memoria.

Desventajas: Puede generarse un cuello de botella en memoria

- 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

¿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.

Dos indicadores de la performance de la memoria caché son:

- La tasa de aciertos:

$$\frac{\#aciertos}{\#refs}$$

donde:

#aciertos es la cantidad de accesos con éxito

#refs es la cantidad total de accesos

Tasa de acierto y Tiempo efectivo de acceso

Dos indicadores de la performance de la memoria caché son:

- El tiempo efectivo de acceso:

$$\begin{aligned}\frac{T_{total}}{\#refs} &= \frac{\#aciertos * t_a + \#fallos * t_f}{\#refs} = \\ &= \frac{\#aciertos * TAC + \#fallos * (TAM + TAC)}{\#refs}\end{aligned}$$

donde

T_{total} es el tiempo total de acceso

t_a es el tiempo de un acierto

t_f es el tiempo de un fallo

TAC es el tiempo de acceso a caché

TAM es el tiempo de acceso a memoria principal

- 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

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

Ejemplo

#aciertos: 45

#fallos: 3

#refs 48

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

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

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

Redondeando...

¿Preguntas?