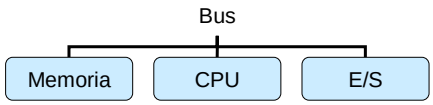
Tema 6: Memoria del Computador

- Conceptos Básicos
- Organización Caché
- Optimizaciones Caché
- Memoria Principal
- Ejemplos en procesadores comerciales

6.1 Conceptos básicos

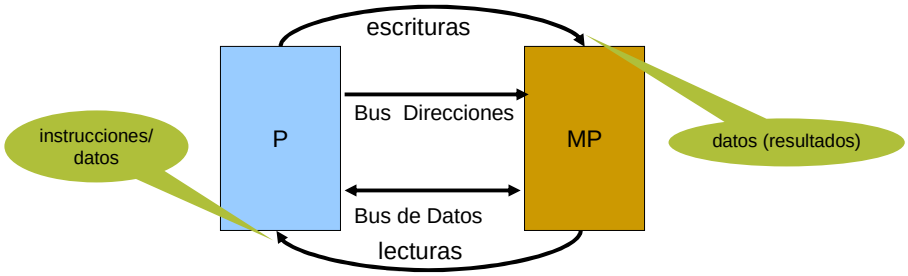


- Memoria: donde se almacenan datos y programa
- Memoria Principal (MP): aquella que 'directamente' interacciona con el Procesador (P).
  - ✓ Hay instrucciones del nivel ISA que acceden directamente a MP (loads y stores).

¿cómo interaccionan Memoria Principal y Procesador?

6.1 Conceptos básicos

- El procesador realiza 'referencias' a MP.
  - ✓ Para obtener un dato o instrucción (op. de lectura: load o búsquedas)
  - ✓ Para almacenar un dato/resultado (operación de escritura, store)



- Referencia: el procesador envía a MP la dirección del dato o instrucción a obtener y además, en el caso de escritura, el dato a almacenar.

6.1 Conceptos básicos

- Características
  - ✓ Capacidad : en bytes (B), bits (b) o palabras (w)
  - ✓ Unidad Direccionable: mínima ubicación que puede referenciarse (B, múltiplos).
  - ✓ Unidad de transferencia: mínimo numero de bytes de una trasferencia
  - ✓ Latencia: tiempo desde que el procesador emite la referencia hasta que se obtiene el dato o instrucción o se almacena el dato
  - ✓ Ancho de banda: número de B por seg que pueden extraerse de la memoria.

6.1.1 Impacto del Ancho de Banda

- Impacto del ancho de banda sobre el rendimiento
  - ✓ DLX: una operación aritmética supone ejecutar 4 instrucciones (dos loads, instrucción aritmética, store): 7 accesos a memoria

buscar 4 instrucciones	leer 2 operandos	escribir resultado

  - ✓ MFLOPS = millones operaciones en punto flotante / seg
  - ✓ De cada operación consideramos sólo tiempo acceso a MP ($t_{cálculo}$=0), datos 32bits.
    - 10 Mflops $\Rightarrow 10^7$ x7 accesos en 1 seg $\Rightarrow$ exigir ancho de banda de memoria de 280 Mbytes/seg
    - 100 Mflops $\Rightarrow$ exigir ancho de banda de memoria de 2,8 Gbytes/seg

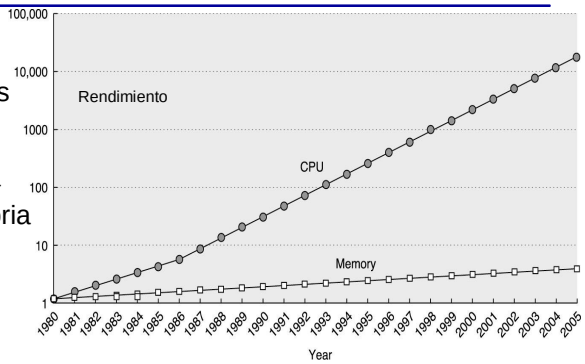
¡El acceso a memoria limita el rendimiento!

- Optimizar ancho de banda de memoria
  - Organización: jerarquía de memoria, organización interna de los chips de memoria (entrelazado)
- Disminuir latencia: organización y tecnología

6.1.2 'Gap' Tecnológico entre P y MP

- Tecnología
  - ✓ memorias más rápidas
- Organización
  - ✓ Jerarquía de memoria
  - ✓ Entrelazado de memoria

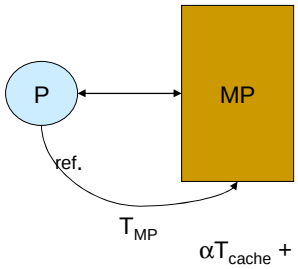
  Disminuyen latencia y aumentan el ancho de banda

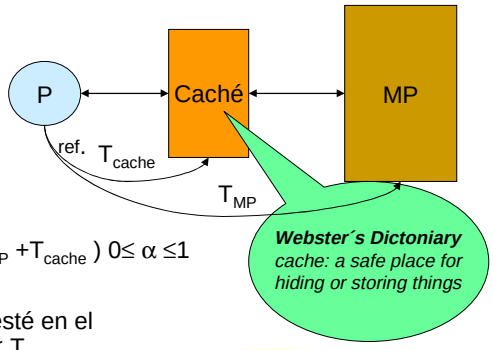


¡la tecnología no es suficiente!

6.1.3 Jerarquía de Memoria

- Idea básica: lo que se 'va' a utilizar ahora lo ubico en un dispositivo físico más rápido (aunque no todo).





$\alpha T_{cache} + (1-\alpha)(T_{MP} + T_{cache})$ $0 \leq \alpha \leq 1$

Latencia media: depende de α, la probabilidad de que la referencia esté en el dispositivo rápido (cache) $T_{cache} << T_{MP}$

¿podemos lograr el éxito? (α ~1)

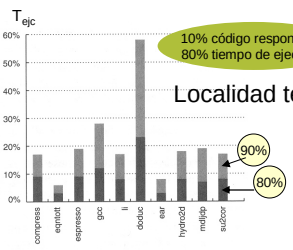
Conceptos | Org. Cache | Opt. Cache | Mem. Principal | Ejemplos

7

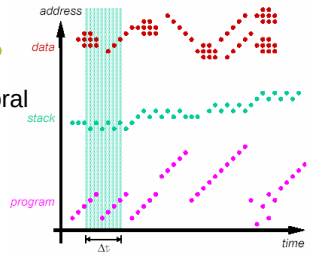
6.1.3 Jerarquía de Memoria (ii)

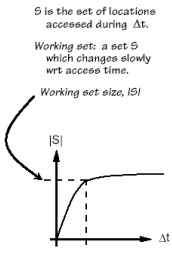
- Principio de localidad: para cualquier instante de tiempo, los programas acceden a una relativamente pequeña porción del espacio de direcciones
  - ✓ Localidad Temporal: es probable que lo recientemente referenciado por el programa se reutilice en un futuro cercano.
  - ✓ Localidad Espacial: es probable que porciones del espacio de direcciones cercanas a la actualmente referenciada sean utilizadas en el futuro cercano

una ref. a la posición X en instante t implica que la ref. X+ΔX en el instante t+Δt es tanto más probable conforme Δt y Δx se aproximen a cero



Localidad temporal



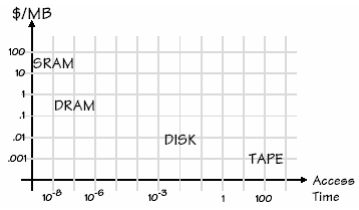


Conceptos | Org. Cache | Opt. Cache | Mem. Principal | Ejemplos

8

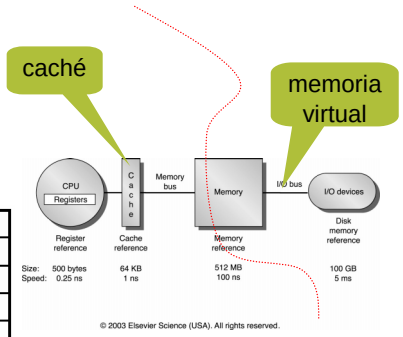
6.1.3 Jerarquía de Memoria (iii)

- Objetivo: que el sistema de memoria tenga la capacidad y precio por bit de un disco duro y la latencia de un SRAM (cache)



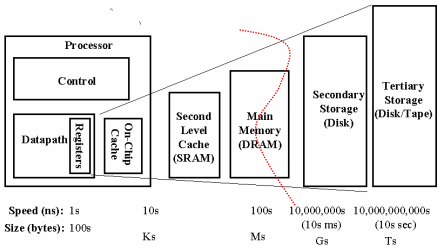
	Capacity	Latency	Cost
Register	100's of bits	20 ps	$$$$
SRAM	100's Kbytes	1 ns	$$$
DRAM	100's Mbytes	40 ns	$
Hard disk*	10's Gbytes	10 ms	¢
Want	100 Mbytes	1 ns	cheap

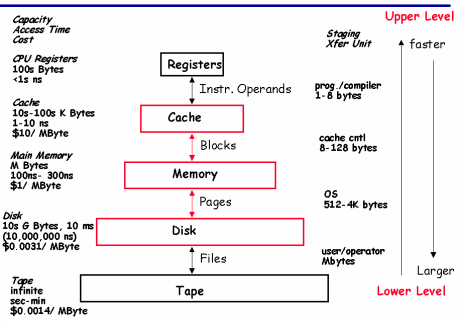
* non-volatile



6.1.3 Jerarquía de Memoria (iv)

- **Jerarquía de memoria: niveles**
  - ✓ **Registros**
  - ✓ **Caché L1**
  - ✓ **Caché L2**
  - ✓ **Memoria Principal**
  - ✓ **Caché de disco**
  - ✓ **Disco**
  - ✓ **Cinta, RAID**





Cada nivel debe almacenar una copia de una pequeña parte del nivel inmediatamente superior

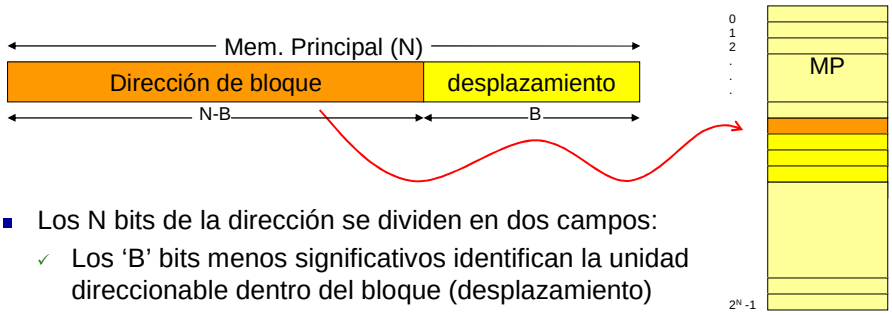
6.2 Memoria Caché

- Bloque (o línea) : unidad de transferencia entre MP y caché
  - ✓ Formado por varias unidades direccionables consecutivas en MP (potencia de 2)
  - ✓ Explota la localidad espacial
- Funcionamiento
  - ✓ El procesador busca primero en cache
    - Si está, el dato lo suministra la cache: acierto (hit), acceso rápido
    - Si no está, hay que acceder a MP: fallo (miss), acceso lento.
      - El bloque que contiene el dato se copia en cache (explota loc. temporal)
      - La cache suministra el dato al procesador
  - ✓ Ciclos de parada por referencias a memoria

Nº instrucciones x Nº medio de referencias a memoria por instrucción x porcentaje de fallos x penalización por fallo

6.2.1 Organización de la caché

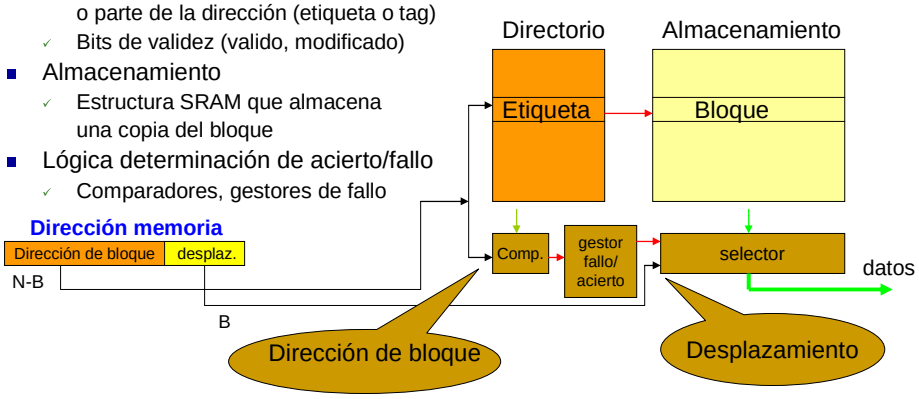
- Bloque: formado por 2^B unidades direccionables de MP (tamaño del bloque)
- Direcciones de N bits



- Los N bits de la dirección se dividen en dos campos:
  - ✓ Los 'B' bits menos significativos identifican la unidad direccionable dentro del bloque (desplazamiento)
  - ✓ Dirección de bloque: los N-B bits más significativos. Es el número de bloque en MP

6.2.1 Organización de la cache (ii)

- Directorio
  - Almacena la dirección de bloque o parte de la dirección (etiqueta o tag)
  - Bits de validez (valido, modificado)
- Almacenamiento
  - Estructura SRAM que almacena una copia del bloque
- Lógica determinación de acierto/fallo
  - Comparadores, gestores de fallo



6.2.1 Organización de la caché (iii)

Cuestiones básicas en la organización de la Jerarquía

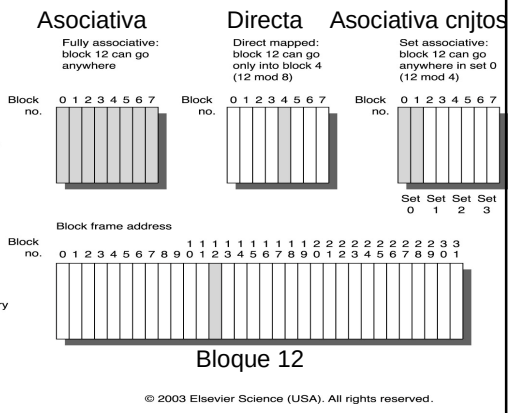
- A: Ubicación: ¿dónde colocar un bloque de MP en caché?
- B: Identificación: ¿cómo localizar si un bloque de MP está en caché?
- C: Reemplazo: ¿qué bloque reemplazar en caso de fallo en caché?
- D: Escrituras: ¿cuándo se actualizan las escrituras en MP?

organización y estructura

políticas de lectura, escritura y reemplazo

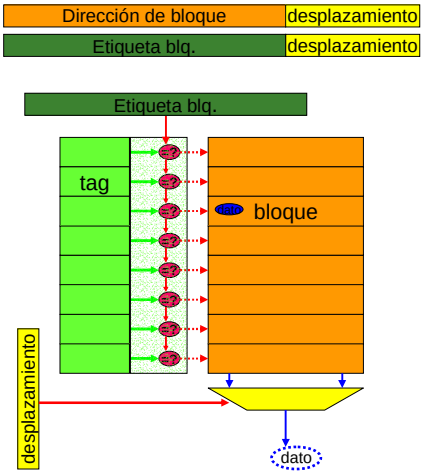
6.2.1 A: Ubicación
¿dónde colocar un bloque en cache?

- Asignación (o correspondencia)
  - ✓ Directa:
    - Dirección de bloque MOD nº bloques en cache
  - ✓ Asociativa:
    - En cualquier posición de la cache
  - ✓ Asociativa por conjuntos:
    - Dirección de bloque MOD nº de conjuntos en cache
    - Dentro del conjunto, en cualquier sitio
    - n-vías: n bloques / conjunto
    - Directa, n=1
    - Asociativa: n= nº bloques en cache= T_c



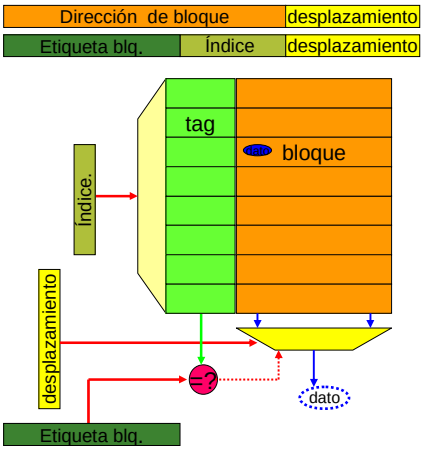
6.2.1 B: Identificación
¿cómo sabes si un bloque está en cache?

- Asignación asociativa
  - ✓ El bloque puede estar en cualquier entrada de la caché. Solo hay reemplazos por falta de capacidad
  - ✓ La etiqueta de bloque ('tag') es la dirección de bloque
  - ✓ Búsqueda asociativa: número de comparadores igual al de entradas en la caché
  - ✓ Los tags ocupan muchos bits, luego el tamaño del directorio es grande



6.2.1 B: Identificación (ii)
¿cómo sabes si un bloque está en cache?

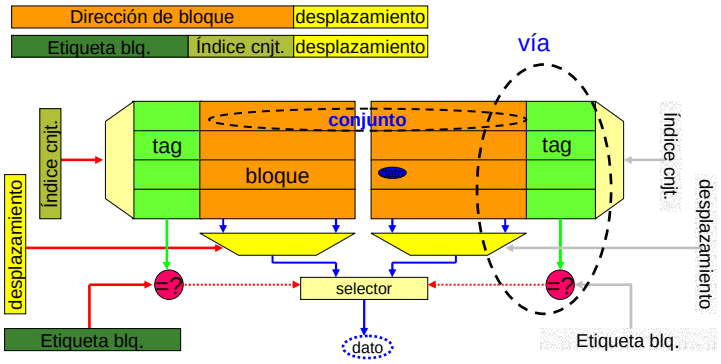
- Asignación directa
  - A cada bloque en MP le corresponde una única entrada en caché. Puede haber reemplazos por colisiones, aunque la caché no esté llena
  - Sólo un comparador
  - Si el número de bloques en la caché es $T_c = 2^m$, el campo Índice consta de m bits
  - El tamaño de la etiqueta ('tag') es el de la dirección de bloque menos m. Por tanto, el tamaño del directorio es mucho menor que en el caso de asignación asociativa
  - Permite la comparación de etiqueta en paralelo a la selección de dato



6.2.1 B: Identificación (iii)
¿cómo sabes si un bloque está en cache?

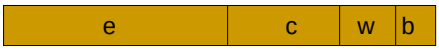
- Asignación asociativa por conjuntos (n-vías)
  - A cada bloque de MP corresponde un único conjunto, y puede alojarse en cualquiera de las n entradas del conjt. Reemplazos por colisiones si el conjt está lleno
  - Nº de comparadores igual al de 'vías'

- Entradas a la caché: $N_c \times n$
- **Indice cnjt** está formado por los últimos c bits de la dirección de bloque
- **Tamaño de 'tag'** es el de la dirección de bloque menos c
- Tamaño de directorio intermedio entre las dos asignaciones anteriores



6.2.1 B: Identificación Notación

- Tamaños
  - T_m : tamaño de MP en bloques
  - T_c : tamaño de cache en bloques
  - T_b : tamaño del bloque en Bytes
  - N_c : número de conjuntos en cache
  - n : asociatividad (nº bloques por conjunto)
- Número de bits
  - N : nº bits del bus de direcciones
  - m : para seleccionar un bloque de cache de forma directa
  - c : para especificar el conjunto
  - b : para especificar byte dentro de la palabra
  - w : para especificar la palabra dentro del bloque
  - e : tamaño en bits de la etiqueta
- Campos de la dirección



Relaciones

- $T_m = 2^N / 2^{w+b}$
- $T_c = 2^m$
- $T_b = 2^{w+b}$
- $N_c = 2^c$
- $n = T_c / N_c$

- $N = e+c+w+b$

Asignación directa
n=1, c=m

Asociativa
$n = T_c$, c=0

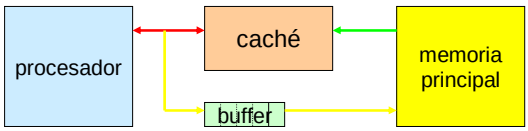
$1 \le n \le T_c$
$m \ge c \ge 0$

6.2.1 C: Reemplazo: ¿qué bloque reemplazar caso de fallo?

- ¿por qué reemplazar?
  - Asignación asociativa: la cache está llena
  - Otras asignaciones: por el tipo de correspondencia (colisiones)
- ¿qué bloque sustituir?
  - Asignación directa:
    - no hay ningún grado de libertad
  - Asignación asociativa o asociativa por conjuntos n-vías:
    - Libertad de elección dentro de cada conjunto (grado de libertad aumenta con n)
- En asignación con asociatividad: aproximar el futuro por pasado cercano
  - Random : dentro del conjunto, eliminar un bloque aleatoriamente
  - FIFO/LRR (first-in first-out / least-recently replaced) : dentro del conjunto, eliminar el que más tiempo lleva en la caché
  - LRU (Least-recently used) : dentro del conjunto, eliminar el bloque menos recientemente referenciado (mantiene en cache los bloques usados recientemente)
- LRU es la mejor alternativa (10% mejor para caches pequeñas)

6.2.1 D: Políticas de escritura ¿cuándo actualizar memoria principal?

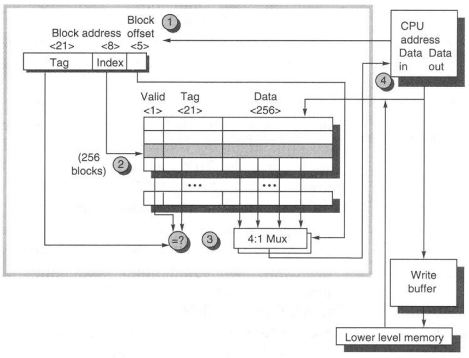
- Ante un acierto de escritura en caché. ¿se actualiza la información también en MP?
- **Escritura Directa** (*Write-Through*): escritura en caché y en MP
  - Esperar por actualización lenta de MP. Mejora: dejar en un buffer el dato a escribir en MP y el procesador puede seguir adelante
  - La MP tiene los datos actualizados (coherencia)



6.2.1 D: Políticas de escritura (ii) ¿cuándo actualizar memoria principal?

- Ante un acierto de escritura en caché...?
- **Post Escritura** (*Write-Back*): sólo se escribe en caché. La actualización en MP se efectúa cuando se haya de reemplazar el bloque
  - Se escribe a la velocidad de la cache
  - Varias escrituras en un mismo bloque se reflejarán con una sola escritura en MP (consume menos ancho de banda en bus cache-MP)
  - Reemplazo por fallo de lectura puede requerir actualizar MP si línea modificada
  - Se requiere un bit (dirty) por bloque para indicar si se ha escrito en el bloque
  - Incoherencia entre datos caché y MP
- Ante un fallo de escritura en la caché ¿traer el bloque desde MP?
  - Escritura con Ubicación (Write-Allocate): Se aloja el bloque en la caché y se procede como en acierto.
  - Escritura sin Ubicación (Non Write-Allocate): Se escribe directamente en MP
- ¿El procesador tiene que esperar que acabe una lectura o escritura?
  - Lecturas: SI, pero.. Escrituras: NO, procesador se deshace del dato

6.2.2 Ejemplos en procesadores comerciales



DEC Alpha 21064

- Cache de datos (8KB)
- Bloques de 32B
- 256 conjuntos, un bloque por cnjto
- Asignación Directa
- Escritura Directa
- 4 buffers de escritura tamaño bloque
- Sin alojamiento en fallo de escritura
- Direcciones de 34 bits
- Bus a memoria de 16 B
- 5 ck acceso a MP

→ 10 ck leer/escribir un bloque

Fallo lectura: 10 ck penalización (detención CPU)+ posible reemplazo
Fallo escritura: no penalización si alguno de los 4 buffers está libre

6.2.2 Ejemplos en procesadores comerciales (ii)

DEC Alpha 21064 versión asociativa de 2vías

- 128 conjuntos
- 2 bloques por cnjto

- Multiplexor extra

De cada vía se extrae un palabra. Este multiplexor selecciona la palabra que produzca acierto al comparar etiquetas

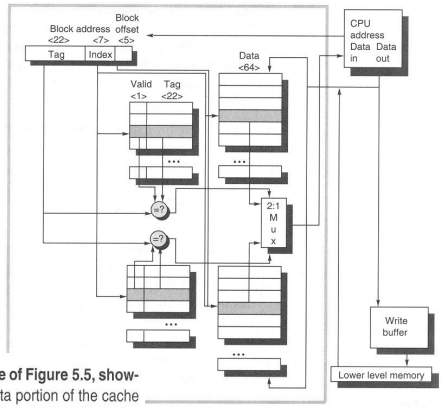


FIGURE 5.8 A two-way set-associative version of the 8-KB cache of Figure 5.5, showing the extra multiplexer in the path. Unlike the prior figure, the data portion of the cache is drawn more realistically, with the two leftmost bits of the block offset combined with the index to address the desired 64-bit word in memory, which is then sent to the CPU.

6.2.3 Optimizar las lecturas es más importante

- Entre el 80% y 90% de los accesos a memoria son lecturas
- Ejemplo en DLX: programa N instrucciones con 26% loads y 9% stores
  Tráfico de memoria : buscar N instrucciones, 0.26N loads, 0.09N stores
  Número de referencias a memoria: N+0.26N + 0.09N=1,35N
  - Los stores son 0.09 / 1,35 -> 6.7% del total de referencias a memoria
  - Los stores son 0.09 / 0.35 -> 25.7% del total de referencias a datos
- Las lecturas son más importantes-> optimizar la cache para lecturas.
  - **Posibilidades: ¿cuándo se inicia el acceso a MP? ¿cuándo se dispone del dato?**
    - **¿cuándo se inicia el acceso a MP?**
      - Secuencial: primero se detecta si fallo en caché, entonces inicia acceso a MP
      - Concurrente: acceso simultáneo a caché y MP
    - **Disponibilidad del dato:**
      - Simple: primer acceso, el bloque en caché; segundo acceso, dato a CPU
      - Adelantado: se solapa la escritura en caché con el suministro del dato al procesador

6.2.4 Tiempo medio de acceso a memoria

- Política lectura más utilizada: secuencial-adelantada
  - t_{ma} = tiempo medio de acceso a memoria
  - t_{hit} = tiempo de acceso a cache (tiempo de acierto, hit)
  - $Penalización_{miss}$ = penalización por fallo
    - Incluye acceso a MP, transportar y alojar bloque en caché y actualizar MP si hay reemplazo.
    - Depende de política de escritura, organización de MP, velocidad y ancho bus a MP
  - P_m = probabilidad de fallo en cache

$$t_{ma} = (1-P_m)\, t_{hit} + P_m\, (t_{hit} + Penalización_{miss}) = t_{hit} + P_m\, Penalización_{miss}$$

Penalización por fallo

Tiempo medio de acceso a memoria

Tiempo de acierto

Probabilidad de fallo en cache

6.2.5 Gestión de las instrucciones

Antecedentes

- Buffer de instrucciones
  - Almacena instrucciones previamente utilizadas (FIFO asociativa)
  - Explota localidad temporal (capturar lazos completos)
- Cola de instrucciones
  - Oculta la latencia de memoria mediante prebúsqueda
  - Posibilidad almacenar dos flujos de instrucciones (saltos)
  - Homogeniza latencia lectura instrucciones de longitud variable
  - Explota localidad espacial
  - En la actualidad sigue siendo un componente del procesador
- Caché de instrucciones
  - Ventajas del buffer y la cola (explota localidad espacial y temporal)
  - Solo lectura
- Caché de datos
  - Posibilidad de operaciones de lectura y de escritura
- Caché Unificada
  - ✓ Datos e instrucciones
    - No eficiente para L1 en diseños actuales (demasiados accesos por ciclo)

6.2.5 Gestión de las instrucciones

- Caché dividida: ventajas
  - ✓ Mejora el ancho de banda
    - Caminos independientes para datos e instrucciones (Arquitectura Harvard)
  - ✓ Gestión independiente de datos e instrucciones
    - Asignación, reemplazo y políticas de lectura y escritura
- Comparación cache unificada vs cache dividida (datos, instrucciones)
  - ✓ Hipótesis: Programa con 29% loads y 9% stores
  - ✓ -> % búsquedas instrucción 100/(100+29+9)=75%; % referencias a datos, 25%
  - ✓ Cache unificada de 32 KB, porcentaje de fallos 1.99%
  - ✓ Cache inst. 16 KB, porcentaje de fallos 0.64%
  - ✓ Cache datos 16 KB, porcentaje de fallos 6.47%
  - ✓ Tiempo acierto 1ck (en cache unificada, 2ck acierto en datos). Penalización por fallo, 50 ck
- **Comparación porcentaje de fallos global**
  Algo mejor la cache unificada
  C. dividida:0.75x0.64+0.25x6.47=2.1% frente 1.99 C. unificada
- **Comparación tiempo medio de acceso a memoria**
  C. unificada: 0.75 (1+1.99x50)+0.25(2+1.99x50)=2.24 ck
  C. dividida: 0.75 (1+0.64x50)+0.25(1+6.47x50)=2.05 ck

La cache dividida es mejor !!

6.3 Mejora del Rendimiento en Cachés

$t_{ma} = t_{hit} + P_m \text{ Penalización}_{miss}$

- A: Reducir la probabilidad de fallo (P_m)
- B: Reducir la penalización por fallo (Penalización$_{miss}$)
- C: Reducir el tiempo de acierto ($\approx t_{hit}$)

6.3 A: Reducir probabilidad de fallos

Caracterización de fallos ¿cómo y por qué?

- Obligatorios (*compulsory*): acceso por 1ª vez, el blq. no está en cache (fallos forzosos, en frio, de comienzo, de 1ª ref.). se producirían aunque la caché fuese de tamaño infinito
- Capacidad (*capacity*): causados por ser finita la caché, habrá blq's desechados que posteriormente son referenciados. Se producirían aunque fuese totalmente asociativa.
- Conflictos (*conflict*): causados por la estrategia de colocación de bloques, se remplazan blq's que posteriormente son referenciados.
- Coherencia (*coherence*): causados por política de coherencia de la caché

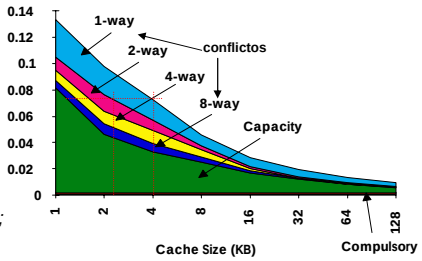
- Soluciones a groso modo:
  - ✓ Fallos obligatorios: aumentar tamaño de bloque y explotar localidad
  - ✓ Fallos de capacidad: cachés más grandes
  - ✓ Fallos de conflictos: aumentar asociatividad

6.3 A: Reducir probabilidad de fallos(ii)

T(KB)	w	miss	O	C	Cf
16	1	4.9	0.1	82	17
16	2	4.1	0.2	98	2
16	4	4.1	0.2	99	0
16	8	4.1	0.2	100	2
32	1	4.2	0.2	89	11
32	2	3.8	0.2	99	0
32	4	3.7	0.2	100	0
32	8	3.7	0.2	100	0
64	1	3.7	0.2	77	23
64	2	3.1	0.2	91	9
64	4	3.0	0.2	95	4
64	8	2.9	0.2	97	2

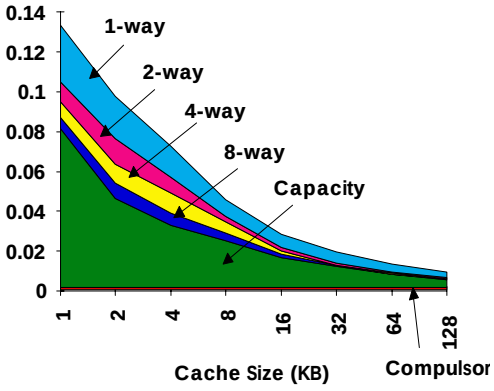
Reproducción parcial de la fig. 5.14 de H&P
T: tamaño cache ; w: nº vías; O: % fallos obligatorios;
C: % fallos de capacidad; Cf: % fallos de conflictos

- fallos obligatorios :independiente tamaño caché sólo depende del programa
- fallos capacidad independiente grado asociatividad, depediente del tamaño
- regla empírica:
  - ✓ porcentaje de fallos de caché directa tamaño K es aproximadamente igual a la de una asociativa por conjuntos de dos vías de tamaño K/2



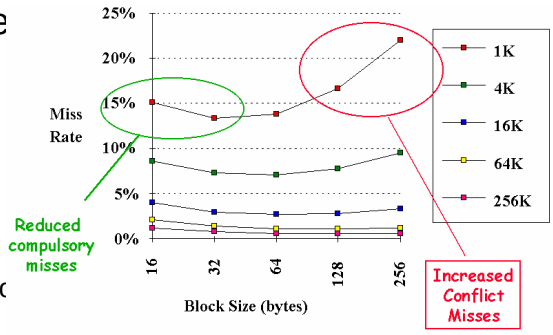
6.3 A: Reducir probabilidad de fallos(iii)

- Aumentar el tamaño de la caché
  - ✓ Reduce fallos de capacidad
    - El tamaño caché hoy es el de MP hace diez años)
  - ✓ Incrementa coste (área)
  - ✓ Afecta al tiempo de acierto



6.3 A: Reducir probabilidad de fallos(iv)

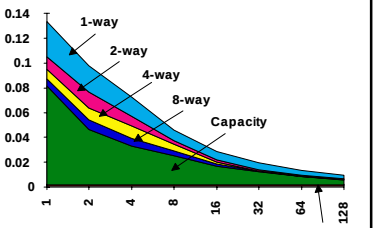
- Aumentar tamaño de bloque
  - ✓ Disminuyen fallos obligatorios (localidad espacial)
  - ✓ Puede incrementar fallos conflicto y capacidad en cachés pequeñas
  - ✓ Afecta a la penalización pc fallo (tiempo de transferencia)



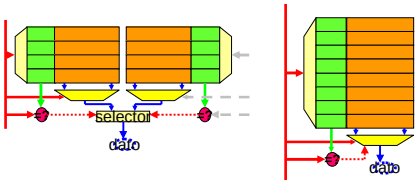
compromiso entre tamaño bloque, latencia y ancho de banda

6.3 A: Reducir probabilidad de fallos(v)

- Aumentar grado de asociatividad
  - ✓ Se reducen fallos de conflicto
    - La probabilidad de fallo se reduce en el mismo factor que aumentemos asociatividad.
    - Existen caches de 16 vías. Prácticamente, la probabilidad de fallos es igual a una totalmente asociativa
  - ✓ Mayor asociatividad incrementa tiempo de acierto($\Rightarrow t_{ck}$)



Δt_{ck} asoctv.(1.36,1.44,1.52)
Penalización fallo 25c



Cache size (KB)	Associativity			
	One-way	Two-way	Four-way	Eight-way
4	3.44	3.25	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.23	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.14	2.18	2.25
128	1.52	1.84	1.92	2.00
256	1.32	1.66	1.74	1.82
512	1.20	1.55	1.59	1.66

6.3 A: Reducir probabilidad de fallos y tiempo de acierto

- Selección de vía: aplicable a asociativas por conjuntos
  - ✓ Objetivo: mantener probabilidad de fallos de la asociativa y reducir tiempo de acierto como en la directa
  - ✓ Preseleccionar la vía de la asociativa con información de accesos previos (Bits extras en la caché predicen la vía o bloque del próximo acceso)
    - El tiempo de acierto predicho es el de una directa (early select)
    - El tiempo de acierto no-predicho es aprox. la de la asociativa
- I-Caché asociativa por conjuntos dos vías Alpha 21264
  - ✓ Acierto en caché: predictor acierto (1c) fallo (3c) – SPEC95 salva estados del pipeline en más del 85% de las búsquedas de instrucción.
- MIPS R4300 en aplicaciones empotradas 'embedded'
  - ✓ reduce consumo de potencia suministrándola sólo a la mitad predicha

6.3 A: Reducir probabilidad de fallos (Técnicas de compilación)

- Instrucciones: McFarling[1989] reducción 75% fallos en caché directa de 8KB bloques de 4B por software
  - ✓ Reordenación de procedimientos en memoria para reducir fallos por conflicto.
  - ✓ Elaboración de perfiles en busca de conflictos
- Datos
  - ✓ Mezcla de arrays (Merging Arrays): aprovechar localidad espacial al unificar arrays mediante elementos compuestos
  - ✓ Intercambio de bucles (Loop Interchange): ajustar anidamiento bucles al orden de los elementos en memoria (aumenta localidad)
  - ✓ Unión de bucles (Loop Fusion): combinar bucles independientes para solapar variables
  - ✓ Gestión de bloques (Blocking): mejorar localidad temporal accediendo repetidamente a bloques de datos

6.3 A: Reducir probabilidad de fallos (Técnicas de compilación)(ii)

- Merging Arrays – Mezcla de 'arrays'
  - ✓ Unificar elementos de varios 'arrays' diferentes para aumentar la localidad espacial de los mismos

```
/* Antes: 2 arrays consecutivos  memoria  */
int val[SIZE];
int key[SIZE];

/* Después: 1 array de estructuras */
struct merge {
        int val;
        int key;
};
struct merge merged_array[SIZE];
```

Reduce conflictos entre 'val' y 'key'
Mejora localidad espacial

6.3 A: Reducir probabilidad de fallos (Técnicas de compilación)(iii)

- Loop interchange – Intercambio de bucles
  - ✓ Intercambiar el orden de los bucles ajustando el orden de acceso de los elementos con el de almacenamiento en memoria (incrementa localidad espacial)

```
/* Antes */
for (k = 0; k < 100; k = k+1)
        for (j = 0; j < 100; j = j+1)
                for (i = 0; i < 5000; i = i+1)
                        x[i][j] = 2 * x[i][j];
/* Después*/
for (k = 0; k < 100; k = k+1)
        for (i = 0; i < 5000; i = i+1)
                for (j = 0; j < 100; j = j+1)
                        x[i][j] = 2 * x[i][j];
```

Acceso secuencial en lugar de ir a saltos en memoria de 100 palabras
Mejora de la localidad espacial

6.3 A: Reducir probabilidad de fallos (Técnicas de compilación)(iv)

- Loop fusion – Unión de bucles
  - ✓ Unir bucles independientes para compartir variables

```
/* Antes */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
            a[i][i] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
            d[i][j] = a[i][i] + c[i][j];
/* Después */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
    {       a[i][i] = 1/b[i][j] * c[i][j];
            d[i][j] = a[i][j] + c[i][j];}
```

- Pasa de dos fallos por acceso a los datos a[] y c[], a un fallo por acceso
- ✓ Mejora la localidad espacial
- ✓ También reduce el tráfico con caché

Conceptos Org. Cache Opt. Cache Mem. Principal Ejemplos

39

6.3 A: Reducir probabilidad de fallos (Técnicas de compilación)(v)

- Blocking – Gestión de bloques
  - ✓ Mejorar localidad temporal en accesos a diferentes matrices algunas recorridas por filas y otras por columnas
  - ✓ Procesar sub-matrices 'bloques' de modo que se maximice los accesos de datos en caché antes de reemplazarlos , función del tamaño de la caché (válido para asignación de registros)

```
/* Antes */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        {r = 0;
         for (k = 0; k < N; k = k+1){
              r = r + y[i][k]*z[k][j];};
         x[i][j] = r;
        };
```

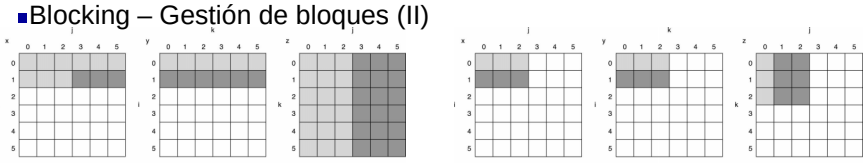
- **Dos bucles internos:**
  - ✓ Leer los NxN elementos de z[]
  - ✓ Leer repetidamente los N elements de 1 fila de y[]
  - ✓ Escribir N elementos de 1 fila de x[]
- **Fallos capacidad función de N y del tamaño de la caché:**
  - ✓ $2N^3 + N^2$ => (asumimos no fallos conflicto)
- **Idea: computar submatrices BxB de tamaño apropiado**

Conceptos Org. Cache Opt. Cache Mem. Principal Ejemplos

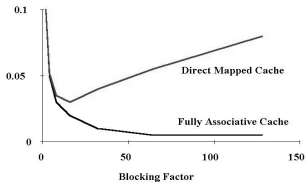
40

6.3 A: Reducir probabilidad de fallos (Técnicas de compilación)(v)

- Blocking – Gestión de bloques (II)



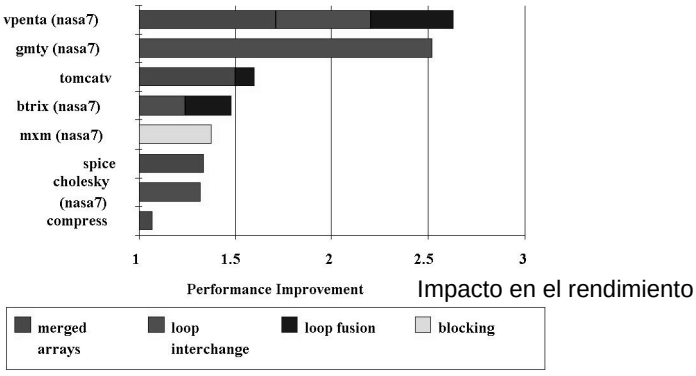
- B se denomina *Blocking Factor*
- Fallos Capacidad de $(2N^3 + N^2)$ a $(N^3/B+2N^2)$
- ¿Fallos por conflicto?




```
/* Después */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
    for (j = jj; j < min(jj+B-1,N); j = j+1)
        {r = 0;
         for (k = kk; k < min(kk+B-1,N); k = k+1) {
             r = r + y[i][k]*z[k][j];};
         x[i][j] = x[i][j] + r;
        };
```


6.3 A: Reducir probabilidad de fallos (Técnicas de compilación)(vi)

Resumen técnicas de compilación para reducción de índice de fallos

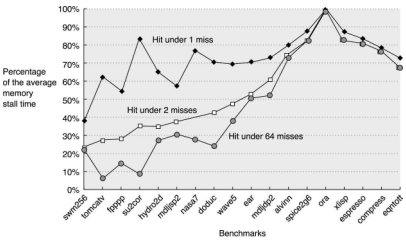


Impacto en el rendimiento

6.3 B: Reducir penalización por fallo Cache no bloqueante

- Ante un fallo, la caché es capaz de suministrar nuevos aciertos
  - ✓ Reduce la penalización por fallo al permitir a la CPU seguir trabajando (puede ocultar penalización por fallo L1)
  - ✓ Necesidad de múltiples bancos de memoria para ser soportado
  - ✓ Complejidad adicional del controlador de caché
    - Registros estado de peticiones, comparadores, pila de datos recibidos,....
  - ✓ Útil en procesadores segmentados con ejecución fuera de orden

- Acierto bajo fallo (Hit under miss):
- Acierto bajo múltiples fallos (Hit under multiple miss)



6.3 B: Reducir penalización por fallo: Prebúsqueda hardware

- Buscar instrucciones y datos antes de ser requeridos por el procesador

  Reduce probabilidad de fallo y mejora la penalización por fallos
  - ✓ Prebúsqueda de instrucciones (ej. Alpha 21064)
    - Cuando hay fallo se buscan dos bloques de MP, el bloque extra se coloca en un buffer 'stream buffer'
    - Bajo fallo, antes de ir a MP se chequea el buffer, si está se toma y se realiza la prebúsqueda del siguiente bloque
    - [Jouppi 90] caché directa 4KB bloq 16B, con 'stream buffer' de un bloque atrapa entre 15% y 25% los fallos. 'Stream buffer' de 4blq. mejora aciertos en un 50% y en un 72% con 16blq.
  - ✓ Similar en Datos
    - [Jouppi 90] caché directa 4KB, buffer de un dato atrapa un 25% los fallos.
    - [Jouppi 90] múltiples flujos en varios buffers, mejoras de hasta un 43% en aciertos para 4 'stream data'
    - [Palacharla, Kessler 94] programas científicos y 'stream buffer' para datos e instrucciones. Caché 64KB asociativa por conjuntos 4 vías, 8 'stream buffer' (1 instruc. / datos) captura entre un 50% y 70% de los fallos.
    - UltraSPARC III posee una caché de prebúsqueda
  - ✓ Hardware de prefetching con cola de instrucciones
    - Instrucciones: si hay vacantes en cola, hacer prebúsqueda
    - Datos: prebúsqueda mediante predecodificación de instrucciones 'loads' en la cola
  - ✓ Prebúsqueda demanda mayor ancho de banda a la MP, (deseable sin penalización)

6.3 B: Reducir penalización por fallo: Prebúsqueda software

- El compilador inserta instrucciones que consiguen los datos antes de ser necesarios
  - ✓ Prebúsqueda de registro: el valor se carga en un registro (HP PA-RISC)
  - ✓ Prebúsqueda de caché: el valor sólo se carga en caché (MIPS IV, PowerPC, SPARC v.9)
  - ✓ Prebúsqueda con fallo / sin fallo : ante fallo de memoria una prebúsqueda sin fallo se transforma en no_op. (instrucción Load clásica es una prebúsqueda de registro con fallo)
- Necesidad cachés no bloqueantes
- El procesador tiene que ejecutar más instrucciones (instrucciones prebúsqueda). La sobrecarga es menos significativa en superescalares.

```
for (i=0;i<3;i=i+1)
    for (j=0;j<100;j=j+1)
        a[i][j]=b[j][0]*b[j+1][0];
```

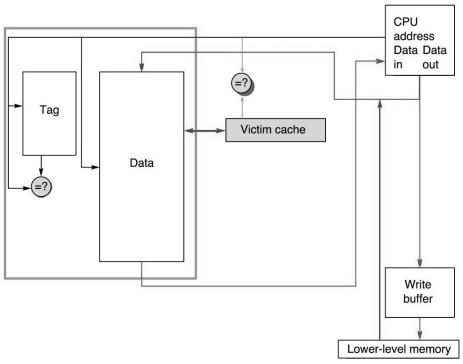
```
for (j=0;j<100;J0j+1){
    prefetch(b[j+7][0];
    prefetch(a[0][j+7];
    a[0][j]=b[j][0]*b[j+1][0];};
for (i=1;i<3;i=i+1){
    for (j=0;j<100;j=j+1)
        prefetch(a[i][j+7]);
        a[i][j]=b[j][0]*b[j+1][0];};
```

b(j,0),a(0,j) para 7 iteraciones después

a(i,j) para 7 iteraciones después

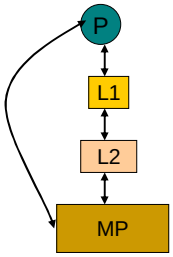
6.3 B: Reducir penalización por fallo: Cache víctima

- Pequeño buffer (caché asociativa) que almacena los últimos bloques reemplazados en la caché
  - ✓ Si hay falló en caché, se examina la caché víctima antes de acceder a MP
    - Reduce fallos por conflicto o se mejora la penalización por fallo
  - ✓ [Jouppi 1990] : una caché víctima de 4 entradas reduce entre el 20% y el 95% de los fallos por conflicto de una caché directa de 4KB
  - ✓ Alpha y procesadores HP



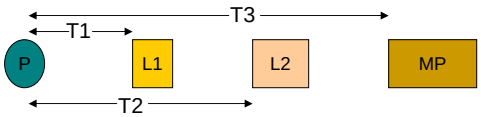
6.3 B: Reducir penalización por fallo: Caches multinivel

- Incrementar el número de niveles de la jerarquía de memoria
  - mejora el tiempo de acierto y la penalización por fallo
- Añadir un nivel(es) adicional en la jerarquía de memoria
  - **L1 : primer nivel de caché**, pequeña y rápida para ajustarse al ciclo de reloj del procesador
  - **L2: segundo nivel de caché**, grande para capturar la mayoría de los acceso a memoria (sólo fallos obligatorios y de conflicto) y reducir la penalización por fallo
- Alternativas de diseño distintas en L1 y L2 (L2 no dialoga directamente con el procesador).
- Problema para mantener propiedad de inclusión
  - Inclusivas: la caché interna mantiene copias de la externa
  - Exclusivas: bajo fallo, puede que el bloque se aloje en L1 y no en L2



6.3 B: Reducir penalización por fallo: Caches multinivel (ii)

- El concepto de un segundo nivel de caché es sencillo pero el análisis de prestaciones se complica

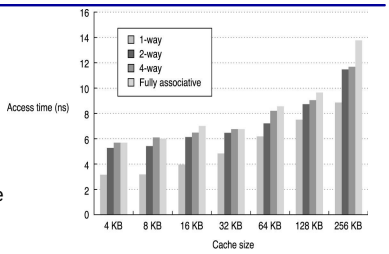


$$t_{ma} = t_{hit1} + P_{m1}\,(t_{hit2} - t_{hit1}) + P_{m1}\,P_{m2}\,\text{Penalización}_{miss}$$

- Penalización$_{miss}$ es el coste de transferir bloque desde MP a L2, desde L2 a L1y desde L1 a la CPU P
- Razón de fallos local: razón de fallos en la caché respecto a referencias a dicha caché. Ejemplo: P_{mL1}, P_{mL2}
- Razón de fallos global: razón de fallos en la caché respecto al total de referencias a memoria. Ejemplo: P_{mL1}, $(P_{mL1} \times P_{mL2})$

6.3 C: Reducir tiempo de acierto

- Cachés pequeñas y sencillas
  - ✓ Simplificar mecanismo traducción
- Solapar traducción direcciones virtuales e indexación de la caché.
  - ✓ Uso de direcciones virtuales: evita tiempo de traducción
    - Anulación en conmutación de contexto
    - Duplicidad direcciones virtuales para diferentes procesos



- Caches segmentadas: Pentium (1ck) Pentium Pro – Pentium III (2k), Pentium4 (4ck)
  - ✓ Realmente incrementa el ancho de banda de las búsquedas de instrucciones
- Caché de trazas: Los bloques de caché contienen trazas dinámicas de instrucciones ejecutadas por el procesador (en lugar de secuencias estáticas determinadas por la posición en memoria) Microarquitectura NetBurst de Intel (Pentium 4).
  - ✓ Mediante seguimiento de trazas integra predicción de saltos
  - ✓ Mecanismo de traducción de direcciones mucho más complejo
  - ✓ Las mismas instrucciones pueden estar duplicadas en diferentes trazas

6.3 Cuadro resumen de técnicas de mejora del rendimiento de la caché

Technique	Miss penalty	Miss rate	Hit time	Hardware complexity	Comment
Multilevel caches	+			2	Costly hardware; harder if block size L1 ≠ L2; widely used
Critical word first and early restart	+			2	Widely used
Giving priority to read misses over writes	+			1	Trivial for uniprocessor, and widely used
Merging write buffer	+			1	Used with write through; in 21164, UltraSPARC III; widely used
Victim caches	+	+		2	AMD Athlon has eight entries
Larger block size	–	+		0	Trivial; Pentium 4 L2 uses 128 bytes
Larger cache size		+	–	1	Widely used, especially for L2 caches
Higher associativity		+	–	1	Widely used
Way-predicting caches		+		2	Used in I-cache of UltraSPARC III; D-cache of MIPS R4300 series
Pseudoassociative		+		2	Used in L2 of MIPS R10000
Compiler techniques to reduce cache misses		+		0	Software is a challenge; some computers have compiler option
Nonblocking caches	+			3	Used with all out-of-order CPUs
Hardware prefetching of instructions and data	+	+		2 instr., 3 data	Many prefetch instructions; UltraSPARC III prefetches data
Compiler-controlled prefetching	+	+		3	Needs nonblocking cache too; several processors support it
Small and simple caches		–	+	0	Trivial; widely used
Avoiding address translation during indexing of the cache			+	2	Trivial if small cache; used in Alpha 21164, UltraSPARC III
Pipelined cache access			+	1	Widely used
Trace cache			+	3	Used in Pentium 4

6.3.1 Cachés en procesadores comerciales

Consejo: consultar datos de fabricantes

- Microarquitectura P5
  - Pentium
    - Bloques 32 Bytes
    - L1I 8KB + L1D 8KB asoc-cnjt 2vías
- Microarquitectura P6
    - Bloques 32 Bytes
  - Pentium Pro
    - L1I 8KB (4vias) + L1D 8KB (2vías) asoc-cnjt
    - L2 256 KB (4vias) asoc-cnjt.
  - Pentium II
    - L1I 16KB + L1D 16KB asoc-cnjt 2vías
    - L2 512 KB
  - Pentium III
    - L1I 16KB + L1D 16KB asoc-cnjt 2vías
    - L2 256 KB (on-chip)
    - Prebúsqueda configurable, tamaños L2 vbl´s según modelos

6.3.1 Cachés en procesadores comerciales (ii)

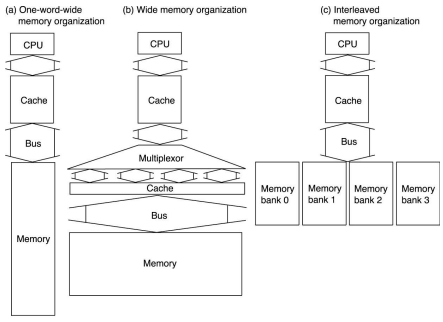
- Microarquitectura NetBurts
  - Pentium 4
    - Bloques 64 Bytes
    - L1I (caché de trazas) 12000 µop + L1D 8KB (4vías) asoc-cnjt, Bloques = 64B
    - L2 integrada, 256KB (8vias) asoc-cnjt. Bloques=64B
    - L3 externa, 1-2 MB, (8 vías)
- Microarquitectura IA-64
    - Bloques 32 Bytes
  - Itanium
    - L1I 16KB(4vias) + L1D 16KB (4vias) asoc-cnjt - Blq.32B
    - L2 96KB(6vias) asoc-cnjt 2vías - Blq.64B
    - L3 dependiente encapsulado (4MB)
  - Itanium 2
    - L1I 16KB(4vias) + L1D 16KB (4vias) asoc-cnjt - Blq.64B
    - L2 256KB(4vias) asoc-cnjt - Blq.128B
    - L3 3-6MB(12vias) asoc-cnjt - Blq.128B (on-chip)

6.3.1 Cachés en procesadores comerciales (iv)

- AMD Athlon
  - ✓ Primer nivel caché de datos (L1-D)
    - 64 Kbytes – Asoc. Cnjt. 2 vías – líneas 32 Bytes
    - Dos puertos , 8 bancos
  - ✓ Primer nivel caché instrucciones (L1-I)
    - 64 Kbytes – Asoc. Cnjt. 2 vías – Líneas 32 Bytes
    - Dos puertos
  - ✓ Segundo nivel caché unificada (L2)
    - 256/512 Kbytes – Asoc. Cnjt. 16 vías
  - ✓ Tercer nivel caché unificada (L3)
  - ✓ TLB
    - Dividida, multinivel de 512 entradas

6.4 Memoria Principal

- Papel de la Memoria Principal
  - Satisfacer peticiones de caché
  - Servir de interfaz con I/O
- Aumento de prestaciones
  - ✓ Latencia: afecta principalmente a caché
  - ✓ Ancho de banda: afecta a I/O y sistemas multiprocesadores
    - Es más fácil mejorar ancho de banda que latencia
    - Grandes bloques (L2) requieren mayor ancho de banda
- Técnicas de mejora del ancho de banda
  - ✓ Memoria principal 'ancha'
  - ✓ Memoria entrelazada
  - ✓ Bancos de memoria independientes



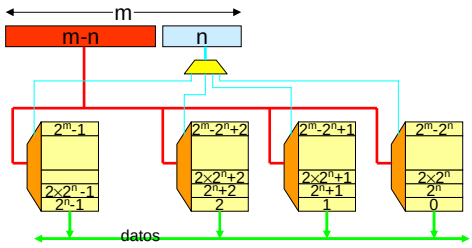
6.4.1 Memoria Entrelazada

- Entrelazado de memoria: distribución del espacio de direcciones de memoria en módulos disjuntos
  - ✓ Posibilita el acceso simultaneo a posiciones ubicadas en módulos distintos.
  - ✓ El mapa de direcciones de divide en clases de congruencia $N=2^n$, donde 'N' es el número vías o módulos disjuntos.
  - ✓ Si las direcciones constan de 'm' bits, 'm-n' bits direccionan el desplazamiento en el módulo y 'n' seleccionan el módulo.
- Dependiendo de la posición de los 'n' bits se habla de:
  - ✓ Entrelazado de orden inferior: se selecciona el módulo con los 'n' bits menos significativos de la dirección.
  - ✓ Entrelazado de orden superior: se selecciona el módulo con los 'n' bits más significativos de la dirección.
  - ✓ Entrelazado de orden mixto: los 'n' bits de selección del módulo se divide a su vez en campos.

6.4.1 Memoria Entrelazada (ii)

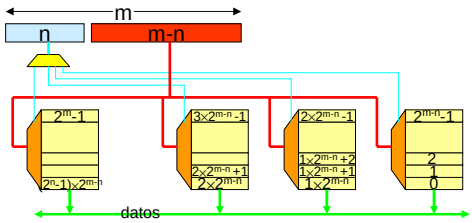
- Esquema entrelazado orden inferior

  se selecciona el módulo con los 'n' bits menos significativos de la dirección



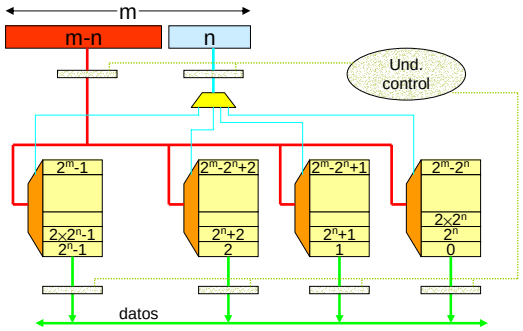
- Esquema entrelazado orden superior

  se selecciona el módulo con los 'n' bits más significativos de la dirección



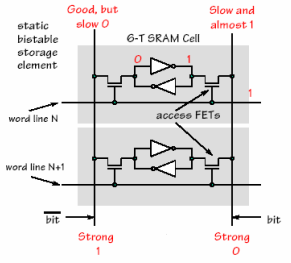
6.4.1 Memoria Entrelazada (iii)

- Latches y control
  - ✓ Mediante control y desacoplo de direcciones y/o salidas, puede accederse simultáneamente o iniciarse accesos solapados a módulos diferentes

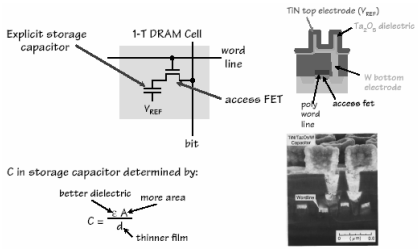


6.4.2 Tecnología Mem. Principal

- Celda SRAM
  - ✓ Almacenamiento del bit en FF
  - ✓ No necesita refresco
  - ✓ 6T (menos simple, más área)
  - ✓ Alto coste por bit
  - ✓ Rápida



- Celda DRAM
  - ✓ Almacenamiento del bit como carga capacitativa
  - ✓ Necesidad de refresco
  - ✓ 1T (simple y poca área)
  - ✓ Bajo coste por bit
  - ✓ Lectura destructiva
  - ✓ Lenta



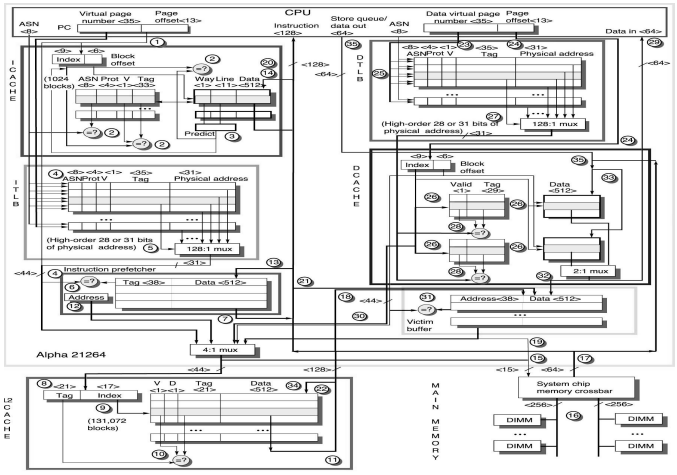
6.4.2 Tecnología Mem. Principal (ii)

- Organización interna de una DRAM: conjunto de matrices bidimensionales de celdas de memoria (1-4Mbits) que permite optimizar el ancho de banda
  - ✓ 'Fast Page Mode': en cada matriz bidimensional, un buffer almacena toda una fila de celdas, lo que permite accesos rápidos a celdas consecutivas.
  - ✓ ' Synchronous DRAM' (SDRAM): se añade una señal de reloj que sincroniza las transferencias entre DRAM y el controlador de memoria.
  - ✓ 'Double Data Rate SDRAM' (DDR SDRAM): se transfieren datos tanto en el flanco de subida como en el flanco de bajada de la señal de reloj (x2).
- Nuevo interfaz de DRAM: RAMBUS (4.8 GBits/sec.)
  - ✓ Cada chip de memoria consiste en varios módulos entrelazados
    - Cada chip contiene entre 4 y 16 bancos de memoria (cada banco tiene asociado un buffer)
    - Buses independientes para direccionar las filas y columnas permiten atender simultáneamente tres peticiones.

6.4.2 Tecnología Mem. Principal (iii)

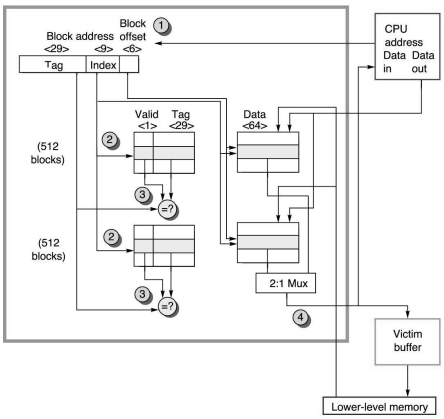
- Los procesadores embebidos requieren pequeñas memorias de almacenamiento no volátil.
  - ✓ ROM
    - ❑ Se programa en tiempo de fabricación
    - ❑ 1 TRT por bit de almacenamiento
    - ❑ Es indestructible, por lo que proporciona un alto nivel de protección del código
  - ✓ FLASH
    - ❑ Permite la alteración del contenido de la memoria después de la fabricación (EEPROM)
    - ❑ Las lecturas requieren tiempos similares los de la tecnología DRAM, pero las escrituras son 10 a 100 veces + lentas

6.5 Jerarquía de memoria en procesadores comerciales: Alpha 21264



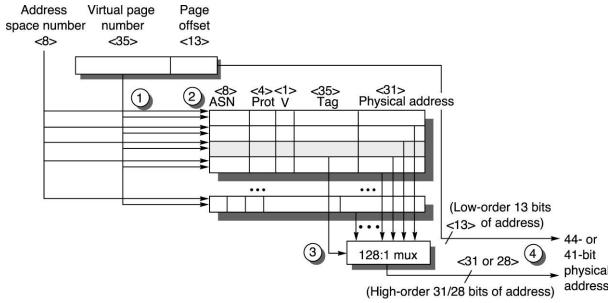
6.5 Jerarquía Memoria en procesadores comerciales: Alpha 21264 (ii)

- Caché de datos del Alpha 21264
  - ✓ Compaq Alpha-Server ES40
  - ✓ D-caché 64KB, bloques=64B, asociativa por conjuntos de dos vías, post_escritura con ubicación bajo fallo



6.5 Jerarquía Memoria en procesadores comerciales: Alpha 21264 (iii)

- Traducción direcciones en TLB de datos del Alpha 21264
  - ✓ 128 entradas totalmente asociativas



Bibliografía

- J.L. Hennesy, D. Patterson. “Computer Architecture: a Quantitative Approach”. 3ª Ed., Edit. Morgan Kaufman, 2003
  - ✓ Tema 5
- H.G. Cragon. “Memory Systems and Pipelines Processors”, Edit. Jones and Barlett, 1996.
  - ✓ Tema 1