UT 3. Subsistema de memoria

Tema 3.1 Prestaciones del subsistema de memoria

J. Flich, P. López, V. Lorente,
A. Pérez, S. Petit, J.C. Ruiz, S. Sáez, J. Sahuquillo

Departamento de Informática de Sistemas y Computadores
Universitat Politècnica de València

UNIVERSIDAD POLITECNICA DE VALENCIA

DISCA

Índice

Bibliografía

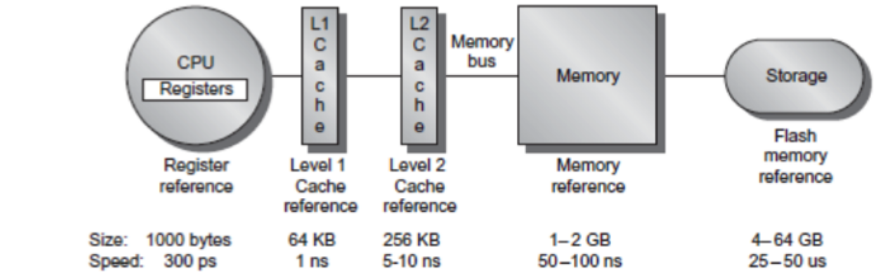
John L. Hennessy and David A. Patterson.
Computer Architecture, Fifth Edition: A Quantitative Approach.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5 edition, 2012.

Índice

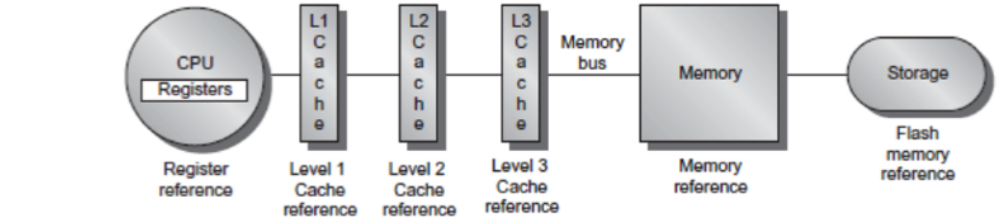
Jerarquía de memoria

"...los programadores pretenden acceder a cantidades ilimitadas de memoria rápida..."
$\rightarrow$ No existe ninguna tecnología capaz de proporcionarlo
Solución: organización jerárquica utilizando distintas tecnologías.
Jerarquía de memoria organización de la memoria en diferentes niveles, donde cada nivel es más pequeño, más rápido y más caro que el nivel inferior.

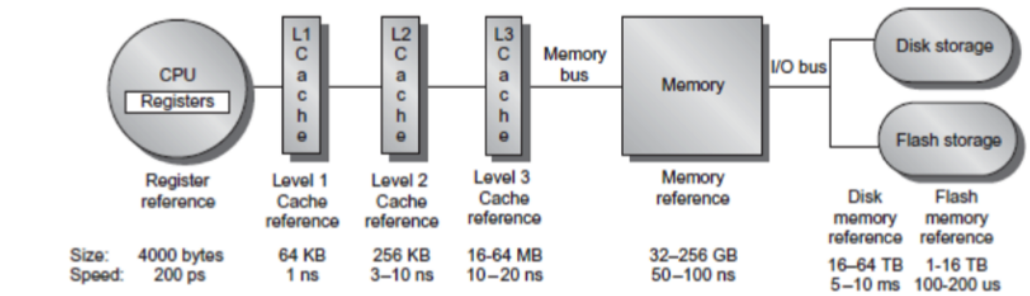
Jerarquía de memoria



(A) Memory hierarchy for a personal mobile device



(B) Memory hierarchy for a laptop or a desktop



(C) Memory hierarchy for server

Jerarquía de memoria

¿Por qué la jerarquía proporciona buenas prestaciones?

- **Objetivo:** velocidad cercana al más rápido.
- Principio de localidad. Los programas tienden a reutilizar el código y los datos utilizados recientemente.

  Localidad temporal: los elementos ya accedidos serán accedidos nuevamente en el futuro próximo.
  - Alta en el código: "El 10 % del código se ejecuta durante el 90 % del tiempo de ejecución de un programa".

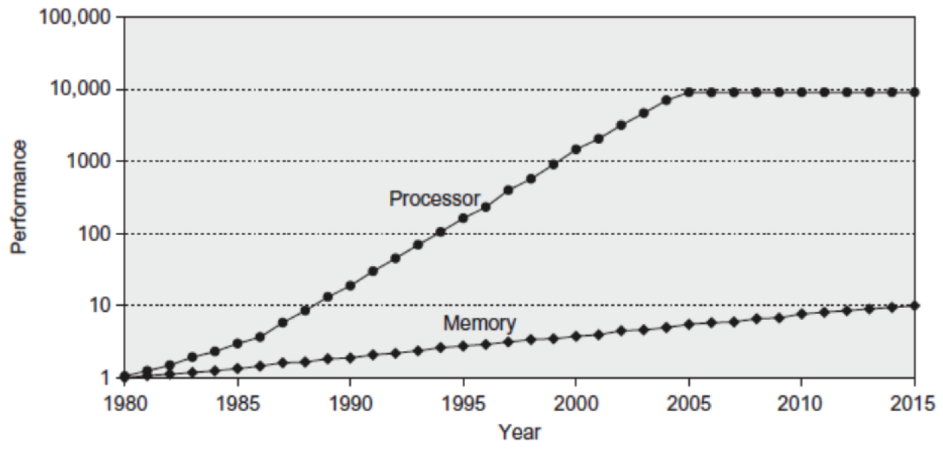
  Localidad espacial: los elementos con direcciones cercanas tienden a referenciarse cercanamente en el tiempo. $\rightarrow$ organización en bloques.

¿Por qué es una solución eficiente?

- Combina tecnologías con distintas características.
  - Rápidas pero caras para prestaciones.
  - Densas pero baratas (coste por bit) para almacenamiento.

Importancia de la jerarquía de memoria

La diferencia de velocidad entre el procesador y la memoria DRAM ha crecido exponencialmente en la última década $\rightarrow$ necesidad de jerarquía de cache.



- En 1980, los procesadores no llevaban caches.
- En 2001, dos niveles de cache en el propio chip.
- Actualmente muchos sistemas incorporan tres niveles de cache.

Requisitos diferentes según tipo de computador

Computador de sobremesa Un usuario, pocas aplicaciones, memoria virtual.

- Objetivo: reducir latencia.

Servidores Múltiples usuarios, múltiples aplicaciones, memoria virtual

- Objetivos: ancho de banda, reducir latencia, protección.

Computadores empotrados Una aplicacion, a veces sin sistema operativo. Memoria principal pequeña. SIN memoria virtual. Objetivos:

- Tiempo-real (importante conocer las prestaciones del peor-caso).
- Bajo consumo.

Memorias *cache*

*"**Cache:** un sitio seguro para esconder o almacenar cosas"*

Cache: primer nivel de la jerarquía de memoria.

El término cache se emplea actualmente cuando se almacena información que se reutilizará: caches de ficheros, cache de disco, cache de nombres, etc.

Acierto: el procesador encuentra el dato en la cache.

Fallo: en caso contrario. El *bloque* que contiene la palabra accedida se trae desde los niveles inferiores de la jerarquía (siguiente nivel de cache o memoria principal).

La gestión de la cache: aciertos, fallos y reemplazos, se realiza mediante *hardware*.

Memoria Virtual

Si el sistema soporta memoria virtual, los objetos referenciados por un programa deben estar en memoria principal o en disco.

El espacio de direccionamiento se divide en *páginas* que deben estar en memoria para ejecutarse.

Si se produce un *fallo de página*, la página entera se transfiere desde el disco a la memoria principal.

Los fallos de página se gestionan por *software* y no detienen el procesador. El procesador cambia de contexto, ejecutando otra tarea mientras se realiza el acceso al disco.

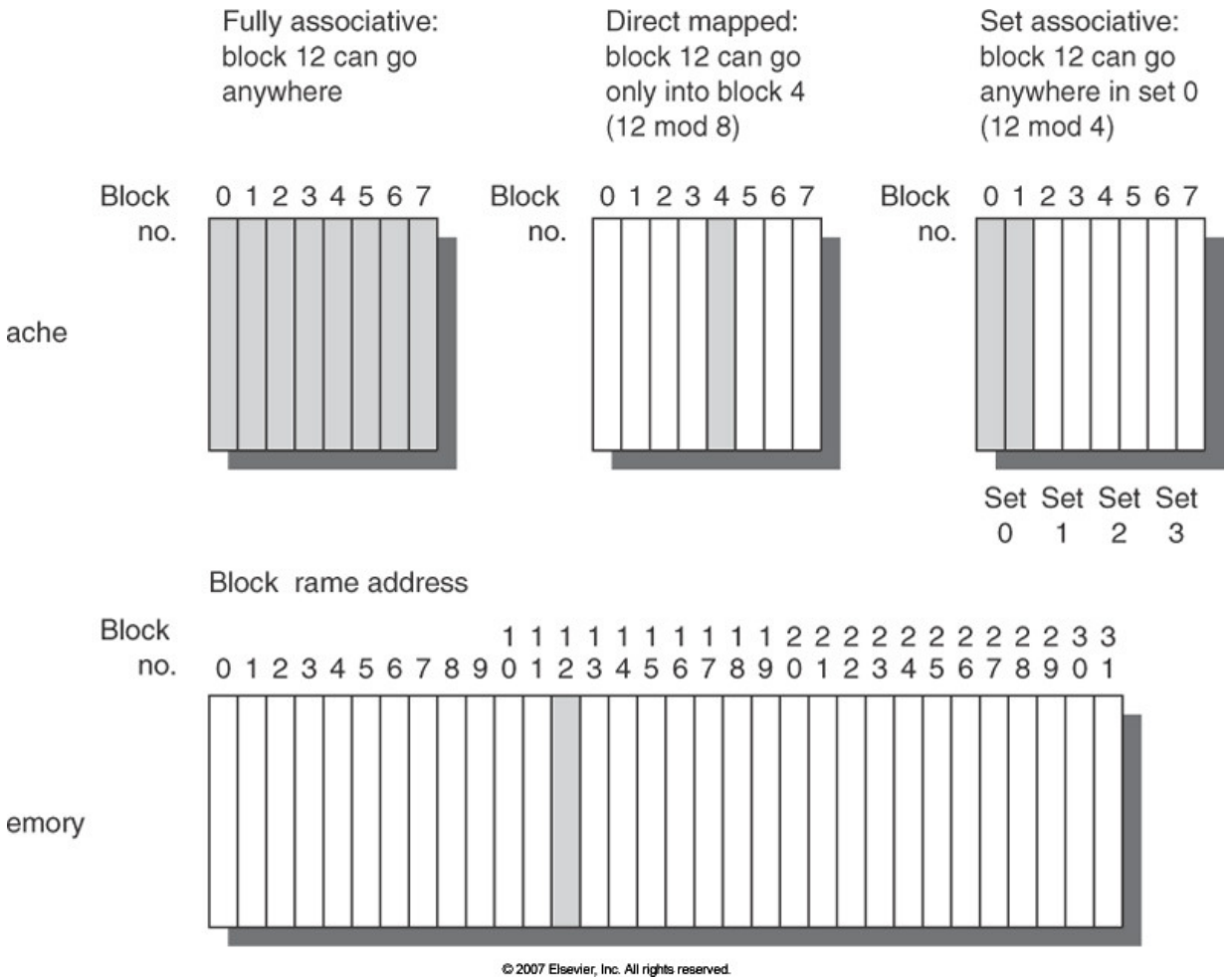
Índice

Características de las memorias cache

Cualquier nivel de la jerarquía de memoria puede caracterizarse respondiendo a las preguntas siguientes:

- Ubicación de un bloque. ¿Dónde puede ubicarse un bloque?
- Identificación de un bloque. ¿Cómo se encuentra un bloque?
- Reemplazamiento. ¿Qué bloque se elimina ante un fallo?
- Política ante escrituras. ¿Qué se hace ante una escritura?

Ubicación de un bloque



Ubicación de un bloque (cont.)

Correspondencia *directa* Un bloque sólo puede estar almacenado en una línea de la cache.

```
#línea = #bloque_referenciado mod #líneas_de_cache
```

Correspondencia *totalmente asociativa* Un bloque puede almacenarse en cualquier línea de la cache.

Correspondencia *asociativa por conjuntos de n vías* Un bloque sólo puede almacenarse en un conjunto que contiene n líneas de la cache.

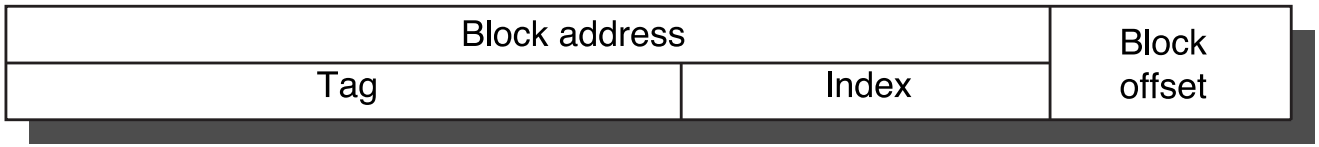
```
#conjunto = #bloque_referenciado mod #conjuntos_de_cache
```

Identificación de un bloque

Cada bloque almacenado en la cache tiene asociada una etiqueta.

Para saber si un bloque se encuentra en la cache, se compara el campo etiqueta de la dirección del bloque con las etiquetas del conjunto destino.

Partes de una dirección emitida por el procesador:



Un bit ‘válido” (V) indica si una línea tiene o no información válida. Solo se comparan aquellas etiquetas que tienen el bit válido activo.

Identificación de un bloque (cont.)

¿Cómo comparar?

- En paralelo con todas las etiquetas. Con correspondencia directa, sólo una comparación.
- No hace falta incluir la palabra dentro del bloque (*offset*), ya que, el bloque completo está presente o ausente.

Para un mismo tamaño de cache, al aumentar la asociatividad aumenta el número de bloques por conjunto y disminuye el número de conjuntos, por lo que se reduce el tamaño del índice y aumenta el de la etiqueta.

Reemplazamiento

Cuando hay un fallo de cache, el bloque referenciado se trae desde los niveles inferiores de la jerarquía. Si el conjunto está lleno, ¿cuál se elimina?

Con correspondencia directa es trivial (solo 1 candidato).

Con correspondencia asociativa pueden emplearse varias estrategias:

LRU Menos recientemente usado. Explota la localidad temporal.

SeudoLRU Menos costoso que el LRU, prestaciones similares con muchas vías.

Aleatoria Se elije un candidato al azar. Fácil de implementar. Útil para estructuras con poca localidad.

Políticas de escritura

Solo se modifica un dato (byte, palabra, . . .) del bloque.
El bloque debe actualizarse también en la memoria principal (MP).

Estrategias en caso de acierto:

Write-through Se escribe tanto en la cache como en MP.

- Más fácil de implementar.
- La memoria principal siempre está actualizada.

Write-back Solo se escribe en la cache.

- Los bloques ”sucios” (bit dirty activo) se actualizan en MP cuando se reemplazan.
- Emplea menos ancho de banda de memoria que *Write-through*.

Políticas de escritura (cont.)

Estrategias en caso de fallo:

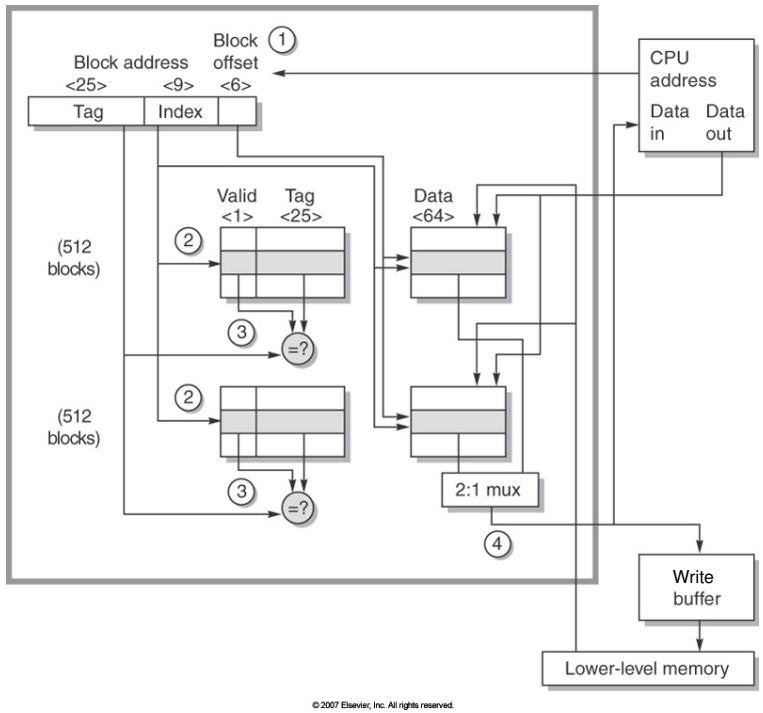
Write allocate El bloque se trae a la cache.
Después, se llevan a cabo las acciones de escritura con acierto.
Habitual con *write-back*

No-write allocate El bloque no se trae a la cache. Sólo se modifica en el nivel inferior. Habitual con *write-through*

Políticas de escritura (cont.)

Ejemplo: Cache de datos del Opteron

- 64KB, 2 vías, 64 bytes/bloque. Reemplazo LRU
- Implementa direcciones físicas de 40 bits (de los 64 bits posibles)
- Write-back, write-allocate



Políticas de escritura (cont.)

1. El procesador envía la dirección (40 bits)
   = Dirección de bloque: 34 bits + Desplazamiento: 6 bits
2. Selección del conjunto (Índice):

$$2^{\text{Indice}} = \frac{\text{Capacidad cache}}{\text{Tam.bloque} \cdot \text{Num.vias}} = \frac{65536}{64 \cdot 2} = 512 = 2^9$$

   Etiqueta: 34–9=25 bits
3. Lectura de las dos etiquetas del conjunto y comparación con la emitida por la CPU. En paralelo, lectura de los dos datos del conjunto.
4. En caso de acierto, selección de la entrada del multiplexor y envío del dato a la CPU.

Índice

Tiempo de acceso medio

$T_{\text{acceso}} = TA + TF \times PF$

donde TA: tiempo de acierto en cache
TF: tasa de fallos
PF: penalización de fallo

Modificación de la ecuación del tiempo de ejecución para incluir el tiempo de la cache:

$$T_{\text{ej}} = T_{\text{ej cpu}} + T_{\text{extra memoria}}$$

donde $T_{\text{ej cpu}}$ incluye el tiempo de acierto de cache y $T_{\text{extra memoria}}$ el tiempo para gestionar los fallos.

Procesador con ejecución en orden escalar

Asunciones:

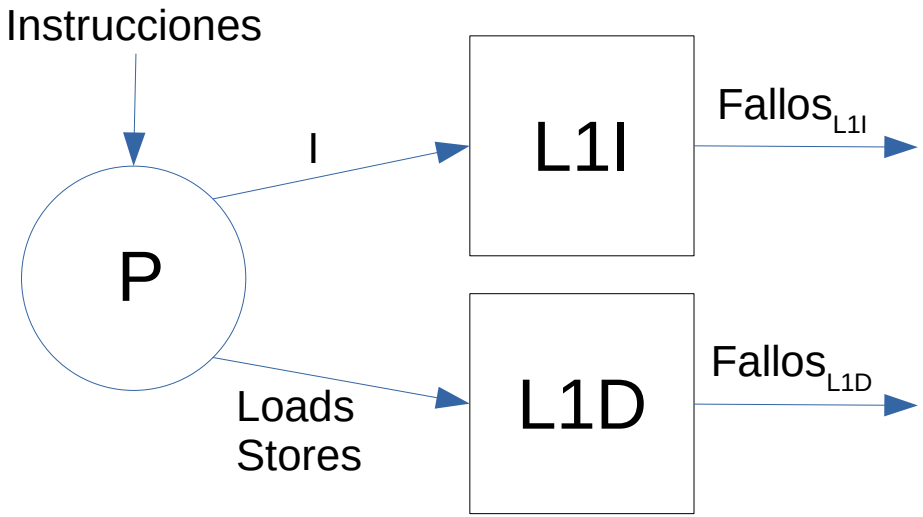
- No puede lanzar más de una instrucción por ciclo.
- Los fallos detienen inmediatamente al procesador.
- Los fallos se sirven de uno en uno (no hay paralelismo al acceder a niveles inferiores de la jerarquía).

Procesador con ejecución en orden escalar (cont.)

- $T_{\text{ej cpu}} = \text{I} \times \text{CPI} \times \text{T}$
- $T_{\text{extra memoria}} = \text{Ciclos parada mem.} \times \text{T}$
- Ciclos parada mem. = Nº de fallos $\times$ Penaliz. por fallo = NF $\times$ PF
- Nº de fallos = Nº de accesos $\times \frac{\text{Fallos}}{\text{Accesos}} = \text{NA} \times \text{TF}$
- Nº de accesos = Instrucciones $\times \frac{\text{Accesos}}{\text{Instrucciones}} = \text{I} \times \text{API}$

$$T_{\text{extra memoria}} = \text{I} \times \text{API} \times \text{TF} \times \text{PF (en ciclos)} \times \text{T}$$

Cache separadas de instrucciones y datos



Si consideramos caches separadas para instrucciones (L1I) y datos (L1D):

$$\text{API} = \frac{\text{Accesos}}{\text{Instrucciones}} = \frac{(\text{Accesos}_{\text{L1I}})+(\text{Accesos}_{\text{L1D}})}{\text{Instrucciones}} = \frac{(\text{Instrucciones})+(\text{Loads}+\text{Stores})}{\text{Instrucciones}}$$

Número medio de accesos por instrucción a L1I : $\text{API}_{\text{L1I}} = \frac{\text{Instrucciones}}{\text{Instrucciones}} = 1$

Número medio de accesos por instrucción a L1D : $\text{API}_{\text{L1D}} = \frac{\text{Loads}+\text{Stores}}{\text{Instrucciones}}$

Desglose del API:

$$\text{API} = \text{API}_{\text{L1I}} + \text{API}_{\text{L1D}}$$

Cache separadas de instrucciones y datos (cont.)

$$T_{\text{extra mem.}} = (\text{Fallos}_{\text{L1I}} + \text{Fallos}_{\text{L1D}}) \times \text{PF} \times \text{T}$$

Puesto que:

- $\text{TF}_{\text{L1I}} = \frac{\text{Fallos}_{\text{L1I}}}{\text{Accesos}_{\text{L1I}}} = \frac{\text{Fallos}_{\text{L1I}}}{\text{I}} \rightarrow \text{Fallos}_{\text{L1I}} = \text{I} \times \text{TF}_{\text{L1I}}$
- $\text{TF}_{\text{L1D}} = \frac{\text{Fallos}_{\text{L1D}}}{\text{Accesos}_{\text{L1D}}} = \frac{\text{Fallos}_{\text{L1D}}}{\text{Loads+Stores}} \rightarrow \text{Fallos}_{\text{L1D}} = (\text{Loads+Stores}) \times \text{TF}_{\text{L1D}}$

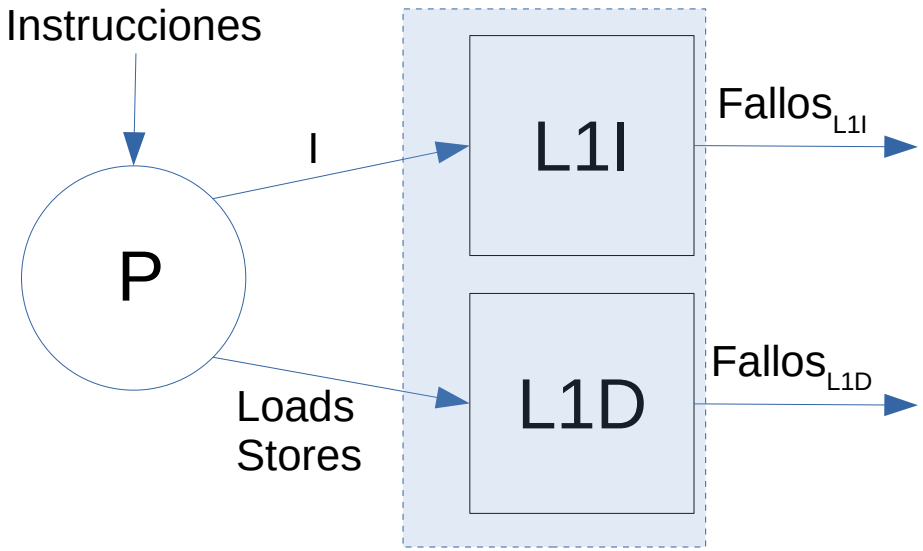
Entonces:

$$\begin{aligned} T_{\text{extra mem.}} &= (\text{I} \times \text{TF}_{\text{L1I}} + (\text{Loads+Stores}) \times \text{TF}_{\text{L1D}}) \times \text{PF} \times \text{T} \\ &= \text{I} \times \left(\frac{\text{I}}{\text{I}} \times \text{TF}_{\text{L1I}} + \frac{(\text{Loads+Stores})}{\text{I}} \times \text{TF}_{\text{L1D}}\right) \times \text{PF} \times \text{T} \end{aligned}$$

Sustituyendo API_{L1I} y API_{L1D}, queda:

$$T_{\text{extra mem.}} = \text{I} \times (\text{API}_{\text{L1I}} \times \text{TF}_{\text{L1I}} + \text{API}_{\text{L1D}} \times \text{TF}_{\text{L1D}}) \times \text{PF} \times \text{T}$$

Tasa de fallos unificada



$$TF' = \frac{Fallos_{L1I}+Fallos_{L1D}}{I+Loads+Stores} = \frac{TF_{L1I}\times I+TF_{L1D}\times(Loads+Stores)}{I+Loads+Stores} = \frac{TF_{L1I}\times\frac{I}{I}+TF_{L1D}\times\frac{Loads+Stores}{I}}{\frac{I+Loads+Stores}{I}}$$

$$TF' = TF_{L1I}\times\frac{API_{L1I}}{API}+TF_{L1D}\times\frac{API_{L1D}}{API}$$

Si se usa la tasa de fallos unificada no es necesario desglosar el API:

$$T_{\text{extra mem.}} = I\times API\times TF'\times PF\times T$$

Diferentes penalizaciones de fallo

Si las penalizaciones de fallo son diferentes (por ejemplo, entre L1I o L1D, o entre lecturas y escrituras en L1D), no es posible utilizar la tasa de fallos unificada y es necesario desglosar el API:

$$
\begin{aligned}
T_{\text{extra mem.}} = {} & I \times \frac{I}{I} \times TF_{L1I} \times PF_{L1I} \times T + \\
& I \times \frac{\text{Loads}}{I} \times TFL_{L1D} \times PFL_{L1D} \times T + \\
& I \times \frac{\text{Stores}}{I} \times TFE_{L1D} \times PFE_{L1D} \times T
\end{aligned}
$$

Donde:

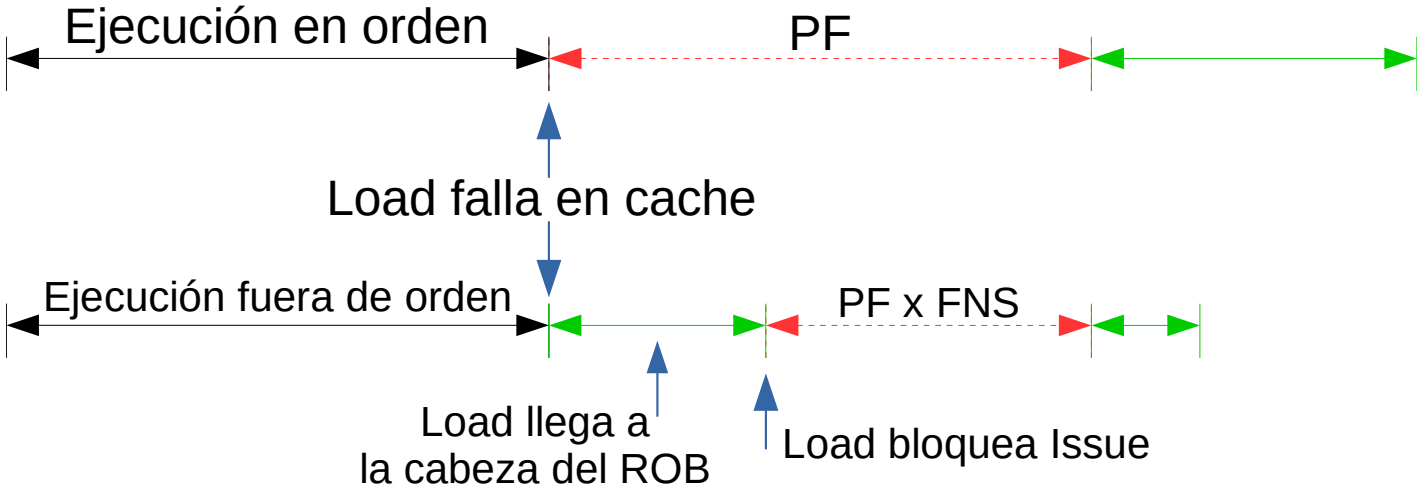
- $TF_{L1I} = \frac{\text{Fallos}_{L1I}}{I}$, $TFL_{L1D} = \frac{\text{Fallos de Lectura}_{L1D}}{\text{Loads}}$, y $TFE_{L1D} = \frac{\text{Fallos de Escritura}_{L1D}}{\text{Stores}}$
- PF_{L1I}, PFL_{L1D}, y PFE_{L1D} son las penalizaciones correspondientes a cada tipo de fallo.

Procesador con ejecución fuera de orden

Busqueda de instrucción Los fallos en la cache de instrucciones detienen la búsqueda de instrucciones, decodificación, e Issue (*front-end*).

Lecturas Ante un fallo de lectura, el front-end no se detiene hasta que no haya una entrada disponible en ROB o estaciones de reserva/buffers.
$\rightarrow$ PFL_{L1D} no solapada = $\text{PFL}_{\text{L1D}} \times$ FNS (Fracción No Solapada)

Ejemplo:



Escrituras Las escrituras se realizan en buffers de escritura y sus fallos prácticamente no detienen el front-end. $\rightarrow$ $\text{PFE}_{\text{L1D}} \approx 0$

Procesador con ejecución fuera de orden (cont.)

Por lo tanto:

$$T_{\text{extra mem.}} = \mathrm{I} \times \mathrm{TF}_{\mathrm{L1I}} \times \mathrm{PF}_{\mathrm{L1I}} \times \mathrm{T} + \mathrm{I} \times \frac{\text{Loads}}{\mathrm{I}} \times \mathrm{TFL}_{\mathrm{L1D}} \times \mathrm{PFL}_{\mathrm{L1D}} \times \mathrm{FNS} \times \mathrm{T}$$

Nota: Si el ROB o las estaciones de reserva/buffers se llenan,
se generan ciclos de parada que pueden solaparse con las penalizaciones del front-end (por fallos en L1I),
pero se asume que ese solape es despreciable.