# Unidad 9 - Localidad y Memoria Cache

## 1. Localidad

La localidad es una propiedad que se aprovecha para acelerar el acceso a la memoria desde el procesador. Se clasifica en:

- **Localidad Temporal**: si los datos han sido utilizados recientemente, es muy probable que se vuelvan a usar. Podemos aprovechar esto manteniendo los datos accedidos recientemente en memorias más rápidas.
- **Localidad Espacial**: si los datos han sido utilizados recientemente, es muy probable que se utilicen datos de direcciones próximas. Podemos aprovechar esto cargando, además del dato accedido, también los datos próximos en memorias más rápidas.

## 2. Memorias Locales

La localiadad de la que hablamos se aprovecha justamente con a través de memoria locales:

- Memorias Fuertemente Acopladas (TCM): región pequeña de memoria dedicada cercana al procesador. El procesador puede acceder a la TCM en cada ciclo.
- Memorias Scratchpad (SPM): memoria interna de alta velocidad utilizada para almacenamiento temporal de datos que no necesitan estar comprometidos en la memoria principal, relacionados a programas en ejecución.
- Memoria Cache: memoria extremadamente rápida que actúa como buffer entre la memoria principal y la CPU. Contiene los datos (e instrucciones) solicitados con más frecuencia para que estén disponibles cuando la CPU los necesite.

### 2.1. Memoria Fuertemente Acoplada

Esta pequeña memoria RAM embebida dentro del procesador es administrada por la aplicación. Al estar embebida en el procesador, utliza una arquitectura harvard, con una TCM de instrucción y una TCM de datos.
Desde el lado del software, no existe diferencia entre acceder a la TCM y a cualquier otra memoria, ya que se utilizan instrucciones de carga y almacenamiento regulares.
La TCM se utliza para:

- **Rutinas de Interrupciones**: cuando se necesitan tiempos deterministas y no se puede esperar errores de la cache.
- Acceso a **memorias RAM externas** configuradas para actualización automática.
- Operaciones que implique **apagar o reconfigurar la MMU**.

La TCM es gestionada a través de unos registros especiales, que sirven para leer el estado, la ubicación física y el tamaño de las TCM. El tamaño y ubicación de la TCM puede modificarse en tiempo de ejecución, e incluso esta puede dividirse en bancos separados, cada uno con sus propios registros de control, para por ejemplo dedicar alguno de los bancos para uso seguro.
Hay que tener en cuenta que estos registros no son como una tabla de la MMU, sino que mueven la ubicación física de la TCM, enmascarando incluso cualquier memoria subyacente al lugar donde la coloquemos, por lo que no es aconsejable superponer ninguna memoria física con la TCM.

### 2.2. Memoria Scratchpad

Como ya se dijo, la Scratchpad es una memoria de alta velocidad que almacena pequeños elementos para su rápida recuperación, como resultados de cálculos. Tiene los mismos problemas de localidad de referencia que las caches (no

tengo idea de a qué se refiere), pero a diferencia de esta la SPM es manipulada explícitamente por las aplicaciones y no almacena copias de/en la memoria principal.
Cabe destacar que no son memorias utlizadas en procesadores de escritorio, sino en sistemas integrados, donde el software a menudo se ajusta a una configuración de hardware.

## 2.3. Memoria Cache

La cache es el nivel más alto de la jerarquía de la memoria, que se utiliza para reducir el tiempo de acceso a la información de la memoria principal. Es una memoria pequeña y rápida, cercana al núcleo, que almacena copias de la información en la memoria principal que se utiliza frecuentemente.
Esta memoria mejora el desempeño general de la computadora, reduce la **brecha entre el procesador y la memoria principal**, funcionando como un **buffer** entre estos.

Algunas definiciones:

- Hit: acceso a la memoria cache en el que se encuentra el dato buscado.
- Miss: acceso a la memoria cache en el que NO se encuentra el dato buscado, forzando un acceso al siguiente nivel de la jerarquía.
- Penalidad de Miss: ciclos de reloj necearios para procesar un cache miss.

### 2.3.1. Organización de la Cache

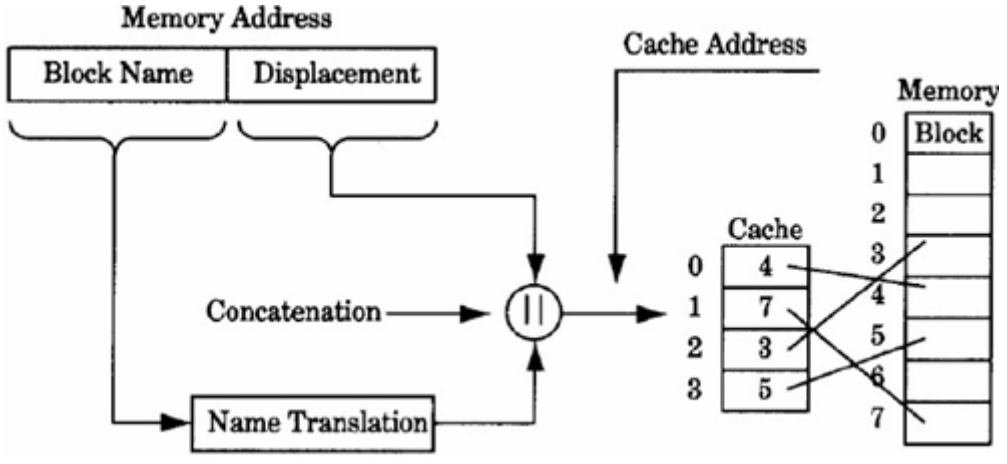
La cache se implementa como un bloque de memoria para almacenamiento temporal, con **B bloques** de memoria (determinado por la cantidad de bytes que se pueden transferir a cache en una operación), organizados en **S conjuntos** de bloques (determinados por la política de mapeo). El número de bloques por conjunto será igual a `N=B/S`, este número N se lo suele llamar **grado de asociatividad**.

### 2.3.2. Direccionamiento de Memoria

El proceso de **direccionamiento** a cache comienza obviamente desde el generador de direcciones del procesador, que genera la dirección virtual. En caché podemos etiquetar e indexar los bloques utilizando direcciones físicas o virtuales o una combinación de ambas, como veremos en Tipos de Indexado.

Las operaciones que realizaremos sobre la cache son:

- **Lectura**: en caso de que el dato esté en cache, se utliza desde la caché. En caso contrario, se lee desde la memoria principal y se guarda una copia en caché para futuros accesos.
- **Escritura**: en caso de encontrar el dato en cache, se debe escribir esta copia y también el dato "original" en la memoria principal. En caso de no encontrarse el dato en caché, se escribe en la memoria principal y se guarda la copia en cache para futuros accesos.



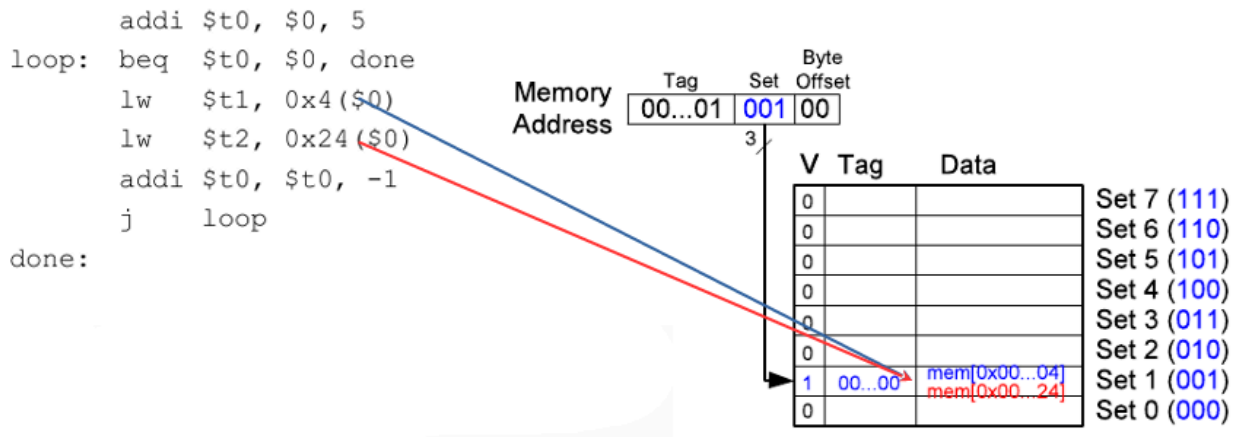
### 2.3.3. Mapeo de la Memoria

La función de mapeo determina cómo se asignan los bloques de la memoria principal a las entradas en caché. Existen tres tipo: **mapeo directo**, **mapeo asociativo**, **mapeo asociativo completo**

#### 2.3.3.1. Mapeo Directo

Cada conjunto de la caché contiene un unico bloque.
`S=B, N=1`

- **Ventajas**: acceso más rápido, ya que la posición del bloque está directamente determinada por su dirección en memoria.
- **Desventajas**: alta probabilidad de colisiones, ya que varios bloques de memoria pueden mapearse al mismo conjunto. Esto se debe principalmente a que la caché es limitada, por lo que habrá más bloques de memoria que entradas de caché.

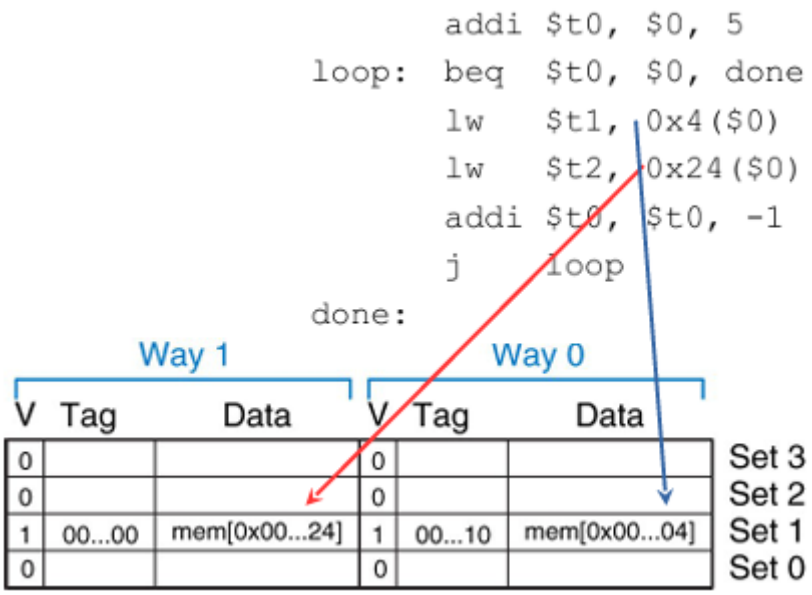



Mapeo Directo

#### 2.3.3.2. Mapeo Asociativo

Cada conjunto de la caché contiene múltiples entradas.
`S>1, N>1`

- **Ventajas**: reduce las colisiones con una velocidad de acceso razonable.
- **Desventajas**: dependerá del compromiso entre flexibilidad y simplicidad.




Mapeo Asociativo

#### 2.3.3.3. Mapeo Asociativo Completo

Existe un único conjunto que contiene todas las entradas en caché.
`S=1, N=B`

- **Ventajas**: minimización de colisiones, ya que cualquier bloque puede colocarse en cualquier posición de la caché.
- **Desventajas**: acceso más lento, ya que hay que comparar la etiqueta del bloque con todas las etiquetas en caché.

### 2.3.4. Tipos de Indexado

La caché se puede clasificar también según el tipo de direcciones utilizadas tanto para el índice (bits bajos) como para la etiqueta (bits altos). Para cada tipo de indexado hay que tener en cuenta:

- **Aliasing**: varias direcciones virtuales apuntando a la misma dirección física.
- **Homónimos**: una única dirección virtual asignada a varias direcciones físicas diferentes.

Los tipos de indexado/etiquetado son:

- Indexado Físico - Etiquetado Físico (PIPT): es un esquema simple pero lento, ya que se debe buscar primero la dirección física (dada por la MMU) antes de buscar la posición del bloque en caché.
- Indexado Virtual - Etiquetado Virtual (VIVT): da respuestas más rápidas ya que no requiere esperar a la MMU, pero es sensible a *aliasing* y *homónimos*.
- Indexado Virtual - Etiquetado Físico (VIPT): menor latencia que la PIPT, al poder buscar en paralelo en caché y en MMU, además de poder detectar *homónimos* dada la etiqueta como dirección física. Aún así, debe esperar a que la dirección física esté disponible.
- Indexado Física - Etiquetado Virtual (VIPT): puede producir falsos intentos de acceso a caché, al no tener ninguna etiqueta.

### 2.3.5. Fallos de Cache

Cuando nos referimos a un fallo de caché nos referimos a un intento fallido de lectura o escritura, resultando en una mayor latencia al acceder a la información. Se clasifican según su causa en:

- Compulsory: al acceder por primera vez a un bloque, se debe llevar a la caché desde la memoria principal. También se lo llama *fallo de arranque en frio* o *fallo de primera referencia*.

- Capacity: al ser el conjunto de trabajo del programa mucho mayor a la capacidad de la caché, esta no puede contener todos los bloques necesarios y se necesita descartar bloques.
- Conflict: cuando varios bloque se asignan al mismo *conjunto*, también llamado *fallo por colisión* o *fallo por interferencia*.

### 2.3.6. Estrategias de Reemplazo

Cuando no hay lugar libre disponible en la caché, se invoca a la estrategia de reemplazo. En el caso del *mapeo directo*, no es necesaria una estrategia, ya que solo se reemplaza la entrada mapeada, pero en el caso de *mapeo asociativo* se necesitan.
Cada estrategia de reemplazo tieen en cuenta el **compromiso entre la tasa de aciertos y la latencia**, que varian entre diferencies aplicaciones. Por ejemplo, en aplicaciones de transmisión de video y audio cada bit de dato se usa una sola vez, por lo que la tasa de aciertos es casi cero.
Las estrategias de reemplazo son:

- First in - First out (**FIFO**): la cache se comporta como una cola FIFO, donde desaloja el bloque al que se accedio primero.
- Last in - First out (**LIFO**): el opuesta a la FIFO, desaloja el bloque más reciente.
- **Least Recently Used** (LRU): descarta primero los que llevan más tiempo sin usarse. Esto requiere realizar un seguimiento del último uso de cada bloque con un *bit de edad*, lo cual es costoso.
- **Time Aware Least Recently Used** (TLRU): es una variante de la LRU, donde el bloque tiene una *timestamp* (marca de tiempo), que indica el tiempo de uso que se le va a dar al contenido.
- **Most Recently Used** (MRU): descarta el bloque usado más recientemente.
- **Pseuso-LRU** (PLRU): emplea un algorítmo que solo necesita un bit por elemento de caché para identificar *casi siempre* el bloque más "antiguo". Esta alternativa se debe a que, en cachés con asociatividad grande (>4), el costo de una LRU es prohibitivo. Esta solución tiene una tasa de error ligeramente mayor, pero una latencia mejor.
- **Segmented LRU** o 2Q LRU: la caché se divide en una cola *de prueba* y otra *protegida*. La primera vez, el elemento accedido se insertará a la cola de prueba. Al acceder a un dato de la cola de prueba, este será "promovido" a la cola protegida. Cuando alguna de las colas está llena, se descarta un elemento de la misma (como una FIFO), en el caso de la cola protegida, el elemento descenderá a la cola de prueba, mientras que en el caso de la cola de prueba el elemento es directamente descartado.
- **Least-Frequently Used** (LFU): descarta el bloque accedido menos frecuentemente. Cuenta la frecuencia con la que se necesita cada bloque, almacenando la cantidad de veces que se accedió.
- **Least Frequent Recently Used** (LFRU): combina LFU y LRU.

### 2.3.7. Niveles de Cache

Los procesadores modernos tienen múltiples niveles de cachés.

- Primer Nivel (**L1**): se ubica cerca del núcleo, y es el más pequeño. Se divide en un bloque de datos y otro de instrucciones.
- Segundo Nivel (**L2**): se ubica dentro del procesador y sirve como repositorio común para el L1. No se comparte entre los núcleos.
- Tercer Nivel (**L3**): también se ubica dentro del procesador, pero se comparte entre los núcleo. Es la más grande
- Cuarto Nivel (**L4**): se ubica fuera del procesador y en la memoria principal.

### 2.3.8. Cachés Especializadas

Existen otro tipos de caché, como la TLU (que forma parte de la MMU), que sirven para diferentes etapas de un procesador segmentada y evitan la planificación para compartir recursos:

- **Victim Cache**: retiene los bloques de datos desalojados de la cache principal (cuando se los reemplaza). Se ubica entre la cache principal y su ruta de llenado.
- **Trace Cache**: almacena las instrucciones decodificadas o retiradas, agregadas en grupos. Esto permite que la próxima vez que se necesite una instrucción, esta no tenga que ser decodificada.
- **Micro-operation Cache**: almacena las micro-operaciones de las instrucciones decodificadas tal como se reciben del decodificador de instrucciones. Esto permite reutilizar instrucciones ya decodificadas.
- **Branch Target Cache**: almacena los destinos de los primero saltos ejecutados. Se utiliza en procesadores de baja potencia que no necesitan un caché completo.