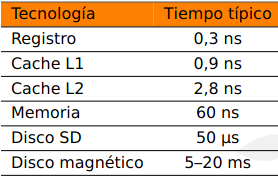
**Jerarquía de memoria**

**Gap CPU-memoria**

* Existe una disparidad de rendimiento entre memoria y CPU: las memorias son mucho más lentas y su rendimiento mejora más lentamente.
* Debido a esto, la memoria limita el rendimiento del sistema.
* Se soluciona organizando la memoria en niveles (jerarquía de memoria)

**Jerarquía de memoria**

1. **Registros de la CPU** (2 KiB[[1]](#footnote-0), 57 GB/s).
2. **Caches** (32Kib-4MiB, 25-45GB/s). Dividida en si misma en varios niveles.
   1. **Principio de inclusión**: Para pasar un dato de un nivel inferior a los registros pasando por cache, se va copiando el dato de un nivel a otro, dejando copias del dato en todos los niveles del cache.
3. **Memoria** **principal** (RAM) (4GiB, 5 GB/s)
4. **Disco duro** (>1000GiB, 150MB/s)
   1. También cintas de back-up, (varios TiB, 30MB/s) e incluye memorias externas (pen drive)

* En niveles superiores, es mayor la velocidad de almacenamiento y lectura, pero mayor coste/byte y menor capacidad.
* La CPU siempre accede al nivel más alto que sea posible para obtener información.
* Ejemplo: con un ciclo de reloj de 0.3ns:

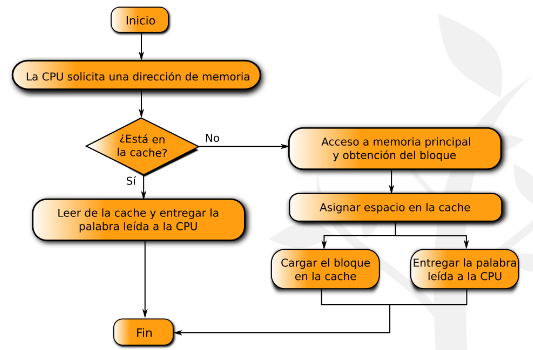
**Gestión de la jerarquía de memoria**

* El procesador gestiona registros y cache, los niveles superiores el sistema operativo.
* La jerarquía de memoria está adaptada para seguir los **principios de localidad:**
  + **Principio de localidad espacial:** Tras acceder a un dato, es común acceder a un dato cercano. Por ejemplo, al operar con un array.
    - En concreto, es común acceder al dato siguiente en memoria, denominado **principio de localidad secuencial**.
  + **Principio de localidad temporal:** Tras acceder a un dato, es común acceder a la misma dirección en breve. Por ejemplo, en un bucle.
* Debido a esto, cuando se accede a un dato en memoria, no sólo se transfiere este dato a caché, sino que también se mueven con el un conjunto de los datos próximos. Para el movimiento caché ←→ memoria este conjunto se denomina **línea cache**.
* Cuando se pide un dato a un nivel de la jerarquía, puede ocurrir:
  + **Acierto (hit):** El dato se encuentra en el nivel. Tasa de aciertos: nº de aciertos/nº de accesos.
  + **Fallo (miss):** El dato no se encuentra en el nivel. Tasa de fallos: nº de fallos/nº de accesos. En caso de fallo:
    - Se busca en un nivel inferior
    - Se le asigna un lugar en el nivel actual. Es posible que sea necesario desalojar otro dato.
* **Tiempo de acierto:** Tiempo de acceso + tiempo de comprobación
* **Penalización por fallo:** Tiempo de traer un dato desde un nivel inferior
* **Tiempo medio de acceso a nivel de memoria (AMAT):** Tiempo de acierto + (Penalización por fallo \* tasa de fallos)

**Ejemplo tiempo de acceso**

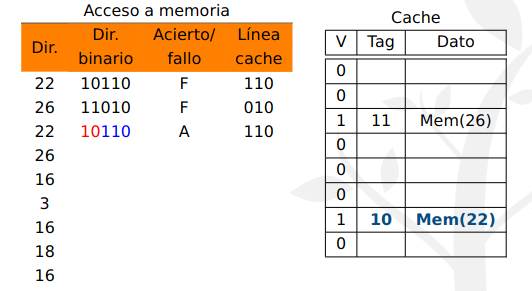
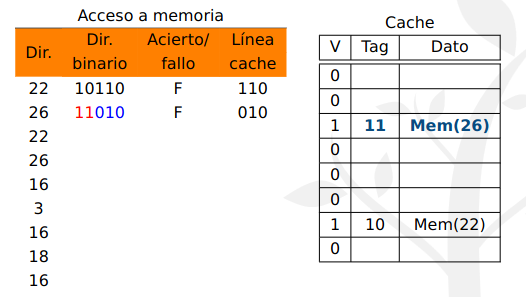
* Un programa lee 500GiB datos de la memoria principal, que lee a 5 GiB/s. Cada dato se utiliza 10 veces. Calcular tiempo sin cache y con cache de 45 GiB/s.
* Sin cache, se accede a 500x10 GiB de datos → 1000 s
* Con cache, se accede 1 vez a 500 GiB con memoria (100 s) y 9 veces con cache (4500GiB a 45 GiB/s) → 100 + 100 = 200 s.

**Memoria cache**

* Memoria pequeña y rápida entre CPU y memoria principal. Almacena la información en uso en un momento dado: datos utilizados en ALU e instrucciones, de uso inmediato.
* **Línea:** cada una de las posiciones de la caché. Puede contener 1 o más palabras (principio de inclusión). Unidad indivisible.
  + Líneas más grandes permiten aprovechar mejor la localidad espacial, pero reducen el nº de líneas que se pueden almacenar.
* **Tipos de cache:**
  + **Cache de asignación directa:** Cada bloque de memoria se coloca en una línea concreta de la cache
  + **Cache totalmente asociativa:** Cada bloque de memoria se puede colocar en cualquier línea
  + **Cache asociativa por conjuntos:** Las líneas se agrupan en conjuntos. Cada bloque de memoria se corresponde con un conjunto, pero dentro del conjunto el bloque se coloca en cualquier línea.

**Cache de asignación directa**

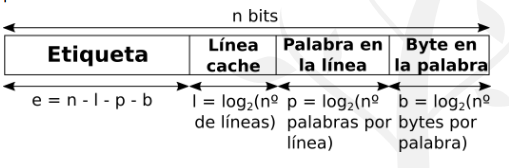
* Siendo **l** el nº de líneas en cache.
* Un dato de la dirección de memoria **d** se aloja en la línea cache número **d%l**.
  + Dada una dirección de memoria, los **log2l** bits menos significativos indican la línea cache. Los bits restantes se denominan etiqueta.
* Se añade a cada línea un bit de validez, que indica si la línea de la cache contiene información válida.
* Cada línea almacena también una **etiqueta**, que indica la dirección de memoria a la que corresponde.
  + El conjunto de todas las etiquetas se denomina **directorio cache**
* Ejemplo:



(si se indica una dirección que no está en caché) (si se indica una dirección que si está)

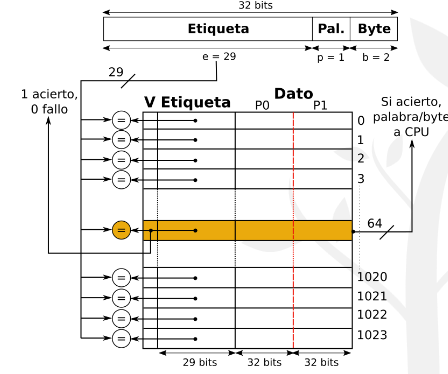
(se baja a memoria y se guarda el dato en la líne (se accede a la línea correspondiente, se

correspondiente, guardando además la etiqueta) comprueba si la etiqueta es la correcta y como lo es, se marca como acierto y devuelve el dato)

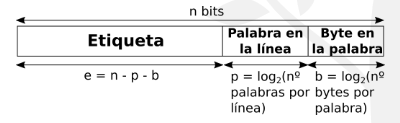
* Si se accede a una línea con una palabra, pero la etiqueta no coincide, se marca como fallo y se carga el dato correcto de memoria.
  + Esto lleva a que se reemplace el dato que había antes.
* Normalmente, las líneas cache contienen **múltiples palabras**. Los p=(log2 palabras/línea) bits menos significativos identifican a la palabra dentro de la línea.
* Además, si la ISA permite acceder a bytes individuales, los log2(bytes/palabra) bits menos significativos indican al byte en la palabra.
* Por ejemplo, en MIPS (cache de 32 líneas, líneas de 8 palabras), b=2, p=3, l=5, e=22
  + Dirección de memoria 0x0700010A (...000100001010): Se accede a la línea 8, a la palabra 4 y a su byte 2.

**Tipos de errores**

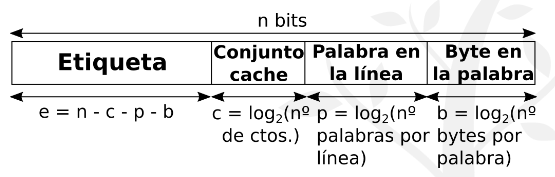
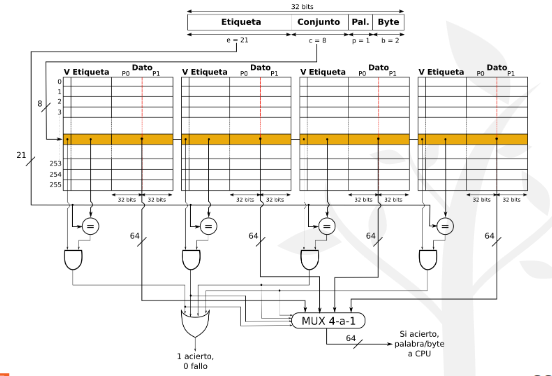
* **Obligatorios:** Se producen la primera vez que se referencia un dato
  + Se reducen aumentando tamaño de línea
  + Representan sólo el 0.006% de los fallos
* **De capacidad:** Cuando se referencia un dato que fue cargado pero después reemplazado porque la cache estaba llena
  + Se reducen aumentando tamaño de cache
* **De conflicto:** Cuando dos líneas compiten por la misma posición en cache
  + Se reducen con caches separadas de datos e instrucciones y aumentando asociatividad

**Cache totalmente asociativa**

* Un bloque de memoria se puede cargar en cualquier línea de la cache
  + Si está llena, se utiliza un algoritmo de reemplazo para escoger la línea a eliminar
* Para buscar un elemento, se compara su etiqueta con todas las entradas de la cache hasta encontrarlo



**Cache asociativa por conjuntos**

* La cache se divide en conjuntos de varias líneas
  + **Número de líneas:** nº de líneas por conjunto+
* Cada línea tiene asignado un conjunto, pero dentro del conjunto puede ir a cualquier sitio
* Más eficiente que directa y más sencilla/menor coste que totalmente asociativa. Ademais, non requiere leer todas as entradas, sólo as do conxunto correspondente
* Na imaxe, cada fila é un conxunto (en laranxa o conxunto que se está consultando actualmente, leense todas as entradas dese conxunto)

**Algoritmos de reemplazo**

* Emprégase cando a caché ou o conxunto a ser empregado está cheo
* Tipos:
  + Aleatorio: lol
  + FIFO: Elimínase o que entrase primeiro
    - Fácil de implementar, pouco eficiente
  + LRU: Elimínase o que leve máis tempo sin ser empregado
    - Máis utilizado, máis eficaz pero máis complexo. Eficacia depende moito do programa/arquitectura
  + LFU: Elimínase o menos empregado
    - Máis eficiente que FIFO

**Reducir penalización por fallo**

* **Cache multinivel:** Consiste en incluir niveles adicionales progresivamente mayores de cache (normalmente, 2 o 3)
  + Los niveles más cercanos a la CPU son más rápidos y pequeños, y suelen estar separadas en datos e instrucciones (otros niveles no)
* También es posible aumentar el ancho de banda (cantidad de datos que la RAM puede pasar en cada ciclo)

completar ancho de banda

1. KiB → 1024 bytes [↑](#footnote-ref-0)