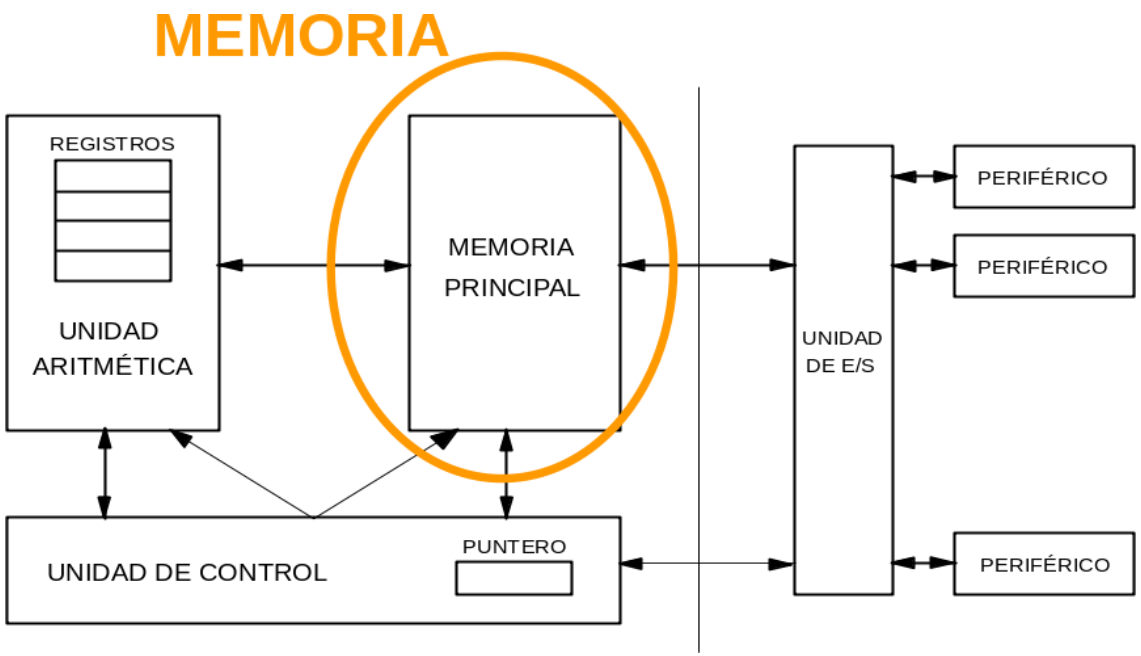
La memoria

Índice

La memoria

Índice

- **Memoria**
  - ➔ **Bloque fundamental de la arquitectura de von Neumann**
  - ➔ **Su función es contener la información que está tratando el sistema durante el procesamiento**
    - ➔ Contiene instrucciones de máquina y datos del programa (o programas) actualmente en ejecución



La memoria

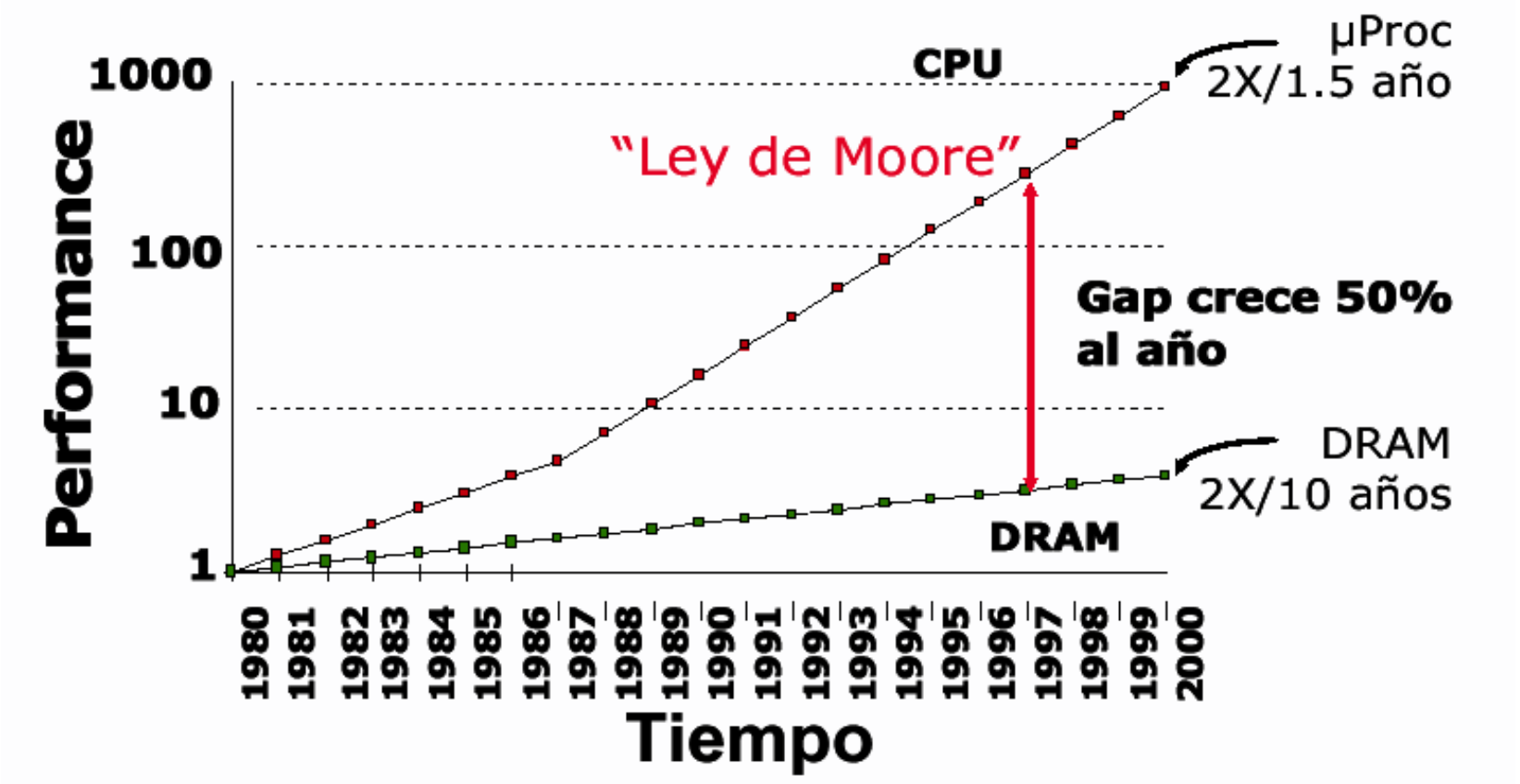
1. Nociones fundamentales

- **La memoria “alimenta” al procesador durante la ejecución de manera que han de trabajar bien SINTONIZADOS si se desea un alto rendimiento**

  - **Procesador → muchos transistores activos en cada momento**

  - **Memoria → muy pocos transistores activos en el mismo instante de tiempo**

- **Existe una gran diferencia de rendimiento entre el procesador y la memoria**



La memoria

1. Nociones fundamentales

- **El sistema de memoria almacena los programas y los datos que requiere el procesador**

- **El programador ve:**

  - **Memoria principal → se reserva espacio en memoria al programar tanto para el código como para datos**

  - **Memoria secundaria → pertenece al sistema de E/S y se accede a través del S.O. de manera transparente**

La memoria

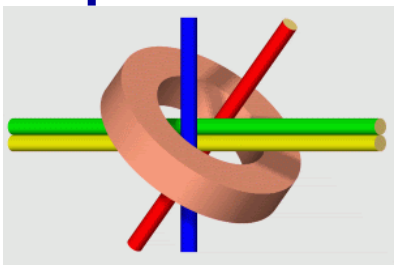
1. Nociones fundamentales

- **Conviene distinguir la MEMORIA de los DISPOSITIVOS DE ALMACENAMIENTO (discos duros, cintas, etc.) que pertenecen al bloque de entrada/salida**
  - ➔ **Los dispositivos de almacenamiento son el archivador mientras que la memoria sería el escritorio o lugar de trabajo**

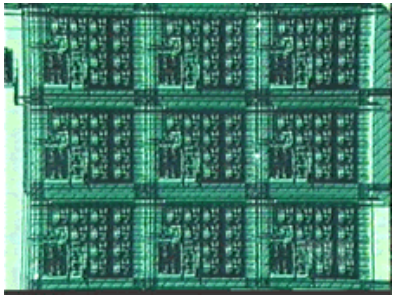
La memoria

1. Nociones fundamentales

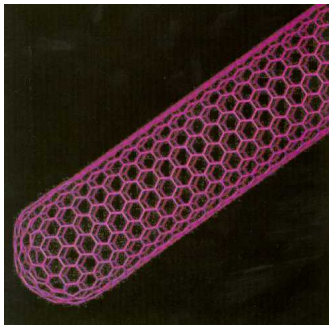
- **La tecnología con la que se realiza la memoria puede cambiar pero siempre requiere elementos capaces de retener uno entre varios estados**
  - ➔ **Inicialmente se construyeron con núcleos de ferrita, relés, etc.**



  - ➔ **A partir de 1975 son de semiconductor, también llamadas de estado sólido**



  - ➔ **En el futuro podrían realizarse gracias a la nanotecnología (nanotubos de carbono), la codificación de moléculas orgánicas, fenómenos cuánticos u otros mecanismos**



La memoria

1. Nociones fundamentales

- **El almacén básico es la celda que, cuando codificamos en binario, puede retener uno entre dos estados (bit)**

- **Sin embargo, la posibilidad de disponer de tecnologías que sean capaces de conmutar entre más de dos estados daría lugar a dispositivos de menor tamaño**

La memoria

1. Nociones fundamentales

- **Normalmente las memorias son referenciadas por un código que identifica de manera univoca un conjunto de celdas**

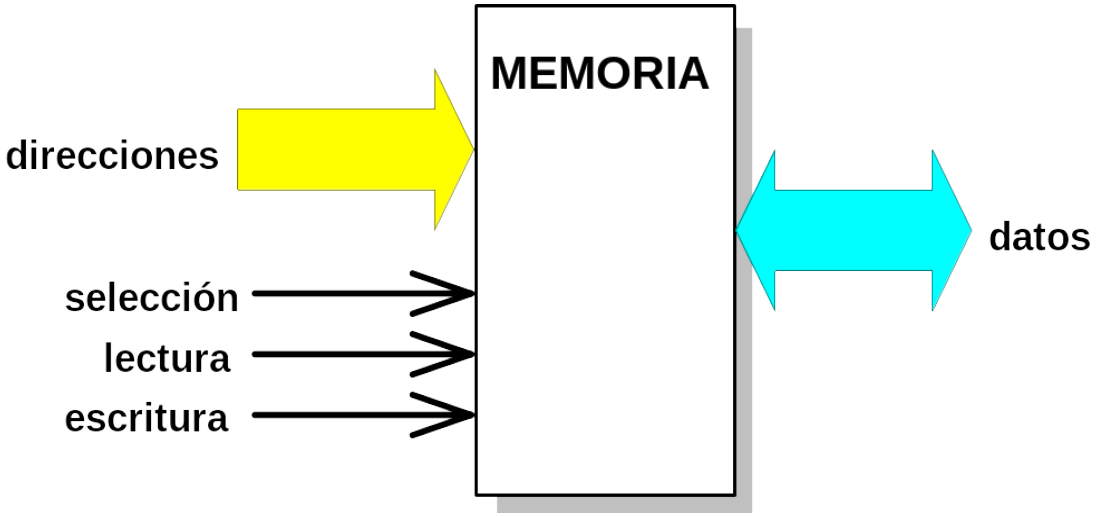
  - **Código de “continente” → dirección**

  - **Contenido → valor**

La memoria

1. Nociones fundamentales

- **Las celdas se suelen reunir formando una unidad lógica que se conoce como *"palabra"***
- **Cada palabra tiene asociada una *"dirección"* o *"posición" de memoria* que la identifica**
- **El procesador debe generar esta dirección para leer o escribir en ese elemento lógico de almacenamiento**



- **Propiedades:**
  - ➔ **Velocidad ➔ máxima frecuencia de reloj a la que pueden trabajar**
  - ➔ **Capacidad de transferencia ➔ cantidad de información transferida por unidad de tiempo (tb. *"ancho de banda"*)**
  - ➔ **Volatilidad ➔ capacidad para retener los datos después de retirar la alimentación**
  - ➔ **Necesidad de recuperación ➔ pérdida de los datos aún con suministro de alimentación**

- **Parámetros (I):**
  - **<u>Capacidad</u> → producto del número de palabras por la longitud de la palabra**
  - **<u>Organización</u> → asociación de celdas para dar la longitud de la palabra; por ejemplo, 1Mbit**
    - 1M x 1 — 1M de direcciones x palabra de 1bit
    - 128K x 8 — 128K de direcciones x palabra de 1byte
    - 64K x 16 — 64K de direcciones x palabra de 16bits
  - **<u>Capacidad de direccionamiento</u> → anchura del bus de direcciones (*n* líneas permiten direccionar 2^n posiciones)**
    - Distinguir entre posición (continente) y dato (contenido)

- **Parámetros (II):**
  - ➔ **Tiempo de escritura (t_e): el que transcurre entre el momento en que se presenta la información a almacenar en la memoria y el momento en que queda registrada**
  - ➔ **Tiempo de lectura (t_l): el que transcurre entre la orden de lectura y el que la información esté disponible en la salida**
  - ➔ **Tiempo de acceso (t_a): es la media de los dos anteriores**

- **Parámetros (III):**
  - ➔ **Tiempo de ciclo (t_c): después de una operación de lectura o escritura, es posible que la memoria necesite un tiempo de reinscripción (memorias de núcleos de ferrita, por ejemplo), o de recuperación (DRAM). El tiempo de ciclo es entonces la suma de este tiempo y del tiempo de acceso**
    - ➔ También denominado ciclo de memoria, es el tiempo transcurrido desde que se solicita un dato a la memoria hasta que ésta se halla en disposición de efectuar una nueva operación de lectura o escritura
    - ➔ También se habla en este caso de ***"latencia"***

- **Parámetros (IV):**
  - **Velocidad de transferencia (V_t): cantidad de información transferida por segundo; depende de la organización de la memoria, es decir, de la longitud de palabra que se transfiere en cada acceso**
  - **En general:**

$$V_t = \text{tamaño de palabra accedida} / t_c$$

  - **Se suele expresar en** Mbytes/seg **independientemente del ancho de palabra**
    - A veces se habla de "*ancho de banda*" para referirse a este término… aunque no es muy propio

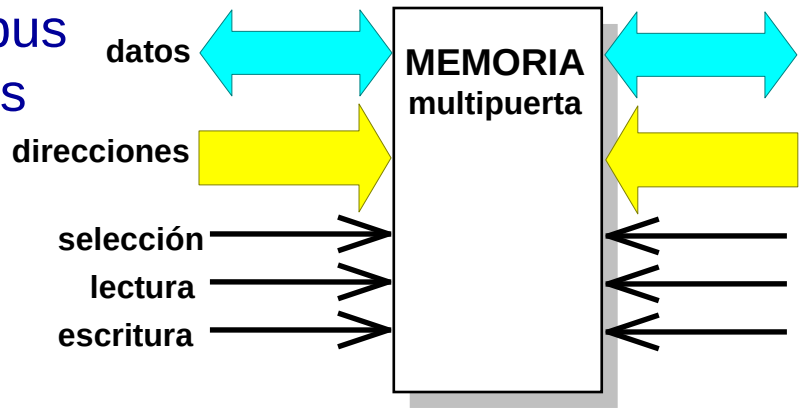
- **Clasificación:**
  - ➔ **Por la forma de acceso**
  - ➔ **Por el modo de almacenamiento**
  - ➔ **Por su función**
  - ➔ **Por el modo de interconexión**

- **Por la forma de acceso:**
  - **Secuenciales**
    - El tiempo de acceso depende de la ubicación
    - Las primeras máquinas tenían memorias de tipo cinta o disco
  - **Acceso aleatorio**
    - Cualquier celda puede leerse o escribirse con el mismo tiempo de acceso
    - Responden a las siglas RAM (*Ramdom Access Memory*)
  - **Por contenido**
    - Se accede por contenido, no por dirección
    - La salida puede ser “encontrado/no encontrado” o un código

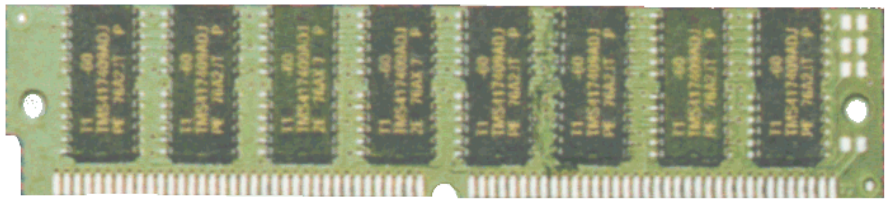
- **Por el modo de almacenamiento:**
  - ➔ **Volátil**
    - ➔ Pierde la información almacenada al cortar la alimentación
  - ➔ **No volátil**
    - ➔ Retiene la información de modo permanente aunque se elimine la alimentación

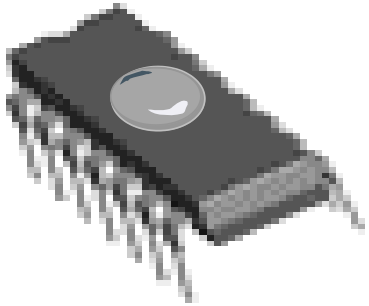
- **Por su función:**
  - ➔ **Memorias tampón (*buffer*): La información se almacena durante un corto periodo de tiempo (*borrador*, *de paso* o *adaptadoras*)**
    - ➔ Son de baja capacidad y acceso rápido
    - ➔ Actúan como memorias auxiliares en las transferencias de E/S
      - FIFO (First In Firts Out; 1º en entrar 1º en salir): es una cola
      - LIFO (Last In First Out; último en entrar 1º en salir): es una pila
  - ➔ **Memoria central: Es la que almacena los programas y los datos implicados en la ejecución de las instrucciones**
  - ➔ **Memoria caché: Es una porción de memoria rápida interpuesta entre el procesador y la memoria central**
    - ➔ "alimenta" al procesador de una manera más eficiente que la memoria central

- **Por el modo de interconexión:**
  - **Un único puerto de conexión**
    - Los accesos se realizan a través de un único sistema de direccionamiento
    - Los datos se transfieren sobre un solo bus
  - **Multipuerta**
    - La memoria permite el acceso simultáneo desde varios puertos
    - Cada puerto cuenta con su bus de direcciones y bus de datos
      - Permiten multiplicar el "ancho de banda"



- **Memorias de estado sólido[1]:**
  - RAM
  - SRAM
  - DRAM
  - SSRAM
  - ROM
  - EPROM
  - EEPROM
  - Flash





[1] "Estado sólido" se refiere a componentes electrónicos, dispositivos o sistemas basados enteramente en semiconductor

- **Estructura general de una memoria de semiconductor**

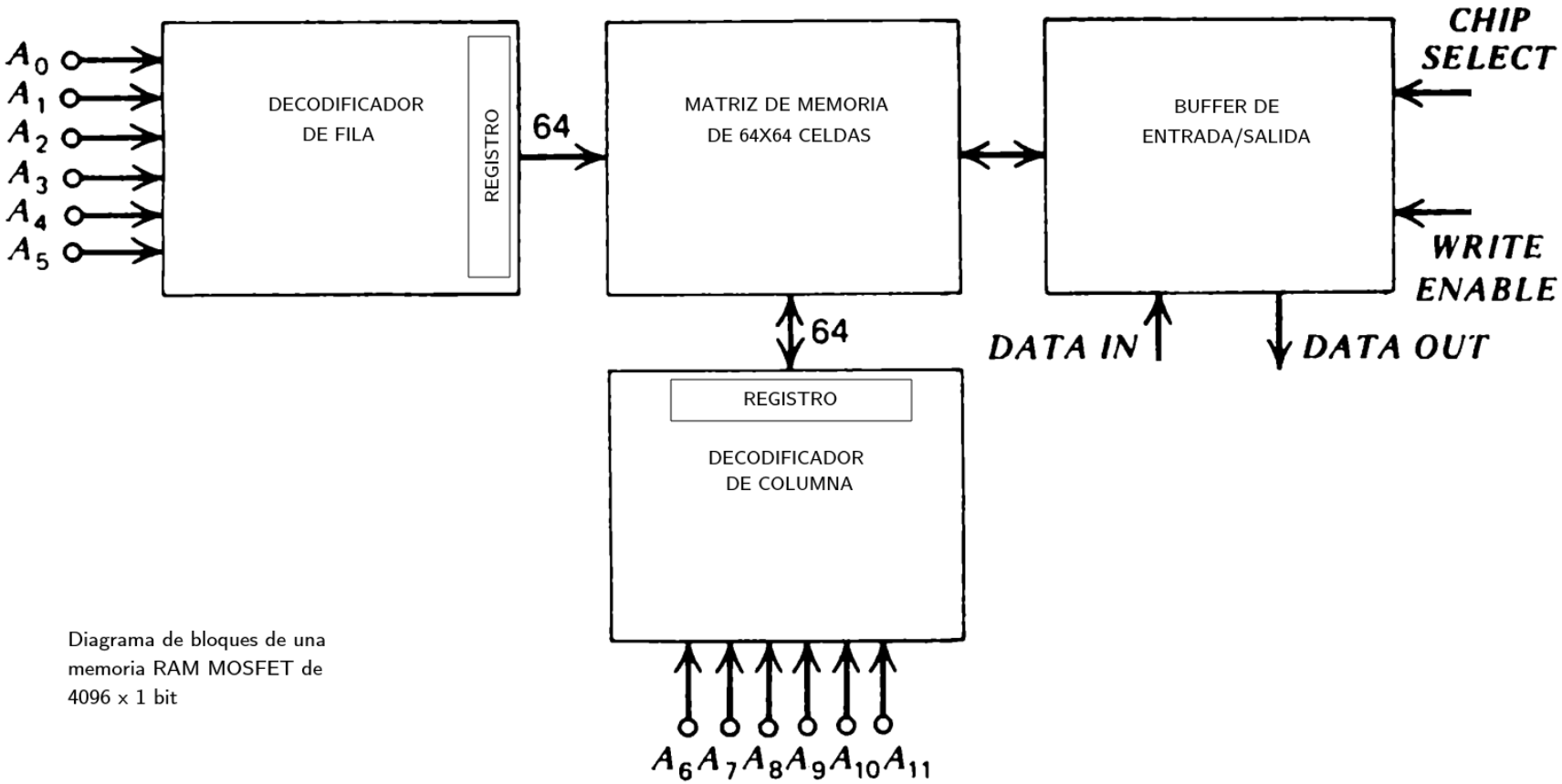
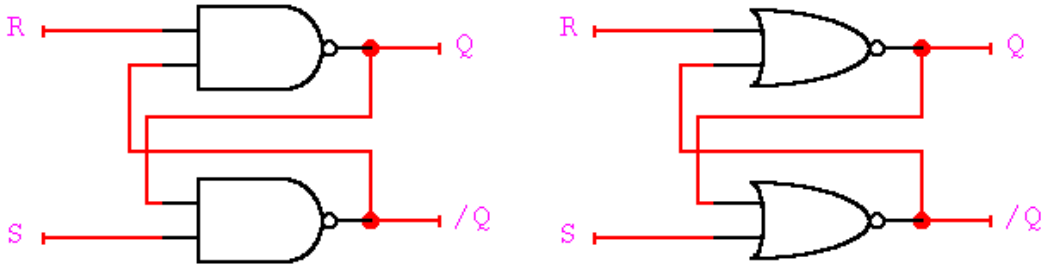


Diagrama de bloques de una memoria RAM MOSFET de 4096 x 1 bit

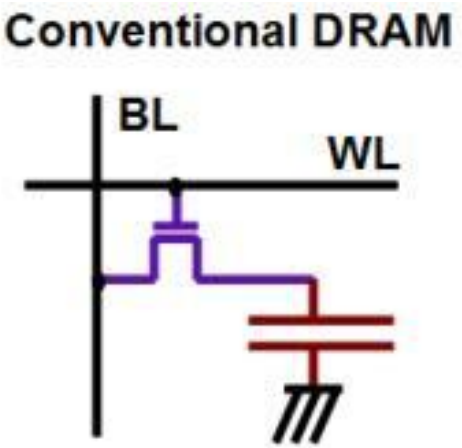
- **Señales de control típicas:**
  - **CS o CE: *C*hip *S*elect o *C*hip *E*nable**
    - Activa la pastilla y, a veces, también la salida
  - **OE: *O*utput *E*nable**
    - Activa la salida pasando de alta impedancia a transferencia
  - **WE: *W*rite *E*nable**
    - Indica que se desea escribir en memoria
  - **RAS y CAS: *R*ow *A*ddress *S*trobe y *C*olumn *A*ddress *S*trobe**
    - En memorias con bus de direcciones multiplexado indican qué parte de la dirección se ha enviado

- **RAM (*Ramdom Access Memory*)**
  - **➔ Son memorias en las que podemos leer y escribir**
  - **➔ Son de acceso aleatorio**
  - **➔ Son volátiles**
  - **➔ Clases:**
    - ➔ Estática ➔ SRAM
    - ➔ Dinámica ➔ DRAM

- **SRAM**
  - **Memoria RAM constituida por biestables**
    - Cada bit es un biestable completo
  - **Rápida pero de elevado consumo energético**
  - **Ocupa una gran área de silicio**
  - **Sencilla de manejar**
  - **Tiempo de acceso de ≈10ns**

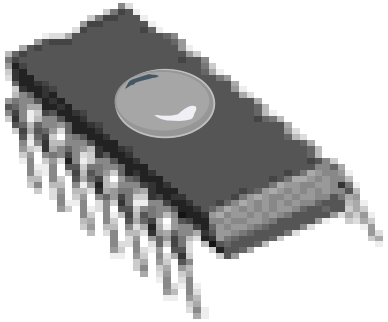


- **DRAM (I)**
  - ➔ **Memoria RAM donde cada bit es un transistor MOSFET asociado a un condensador**



- **DRAM (II)**
  - ➔ **Requiere continuo refresco para no perder la carga**
  - ➔ **Lenta pero consume poco**
  - ➔ **Ocupa poca superficie de silicio**
  - ➔ **Tiempo de acceso de ≈60ns**
  - ➔ **Su gran capacidad ha forzado la multiplexación de direcciones para reducir el patillaje (RAS y CAS)**
  - ➔ **Multiplexación y refresco la hace más complicada de manejar que la estática**

- **ROM (*Read Only Memory*)**
  - ➔ **Son memorias de sólo lectura**
  - ➔ **Son de acceso aleatorio como las RAM**
  - ➔ **Son no-volátiles o permanentes**
  - ➔ **Existen variantes que permiten realizar una escritura inicial o varios ciclos borrado-escritura:**
    - ➔ PROM ➔ ROM programable
    - ➔ EPROM ➔ ROM programable y borrable (luz ultravioleta)
    - ➔ EEPROM ➔ ROM programable y borrable eléctricamente (completa, no bloques o palabras)



- ***Flash***
  - ➔ **Se accede por bloques**
    - ➔ No es, por eso, apta para ser usada como RAM
    - ➔ Se adapta muy bien a trabajar con los sistemas de ficheros debido a la analogía entre sectores y bloques
    - ➔ Número limitado de ciclos de lectura-escritura
      - Se incluyen más celdas de las nominales para sustituir a las degradadas por muchos ciclos de uso

- **Aumento de prestaciones**

  - **Solapar etapas de acceso**
    - SSRAM y SDRAM
    - FPM
    - EDO

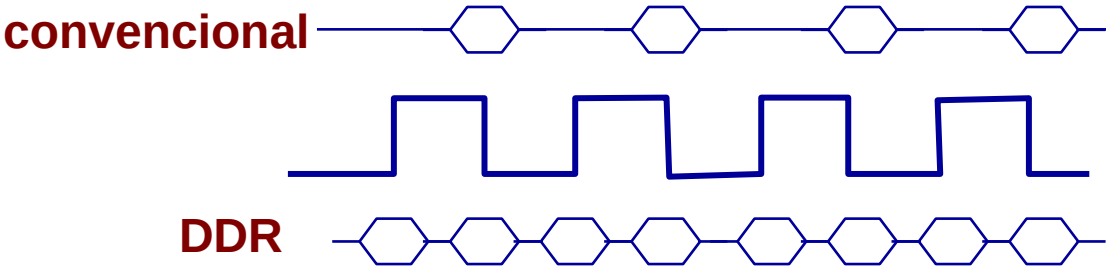
  - **Doblar el reloj**
    - DDR (***D****ouble* ***D****ata* ***R****ate*) combinando flancos de subida y bajada

- **SSRAM ➔ memorias SRAM síncronas**
  - **➔Tienen registros de direcciones y datos de entrada y salida para poder bajar el tiempo de acceso**
  - **➔Los accesos se pueden segmentar en tres etapas:**
    - ➔direccionar (*address setup*);
    - ➔acceso a RAM; y
    - ➔salida de dato
  - **➔El tiempo de acceso que se consigue es de 7 a 12ns**
  - **➔El funcionamiento segmentado está gobernado por un reloj sincronizado con el procesador**
    - ➔Cada acceso solapa sus etapas con las de otros accesos

- **FPM RAM (*Fast Page Mode RAM*)**
  - ➔ **DRAM que permite un acceso más rápido a la misma fila o página**
  - ➔ **Se elimina la necesidad de direccionar filas si los datos se localizan en una fila previamente referenciada**

- **EDO (*Extended Data Out DRAM*)**
  - ➔ **Tipo de DRAM 10-15% rápida que la convencional**
  - ➔ **Mientras que la convencional sólo puede acceder a un bloque en cada acceso, la EDO DRAM puede comenzar la búsqueda del siguiente bloque de memoria a la vez que envía el bloque previo a la CPU**

- **DDR RAM**
  - ➔ ***Double Data Rate***
  - ➔ **Transfiere datos en los dos flancos del reloj**



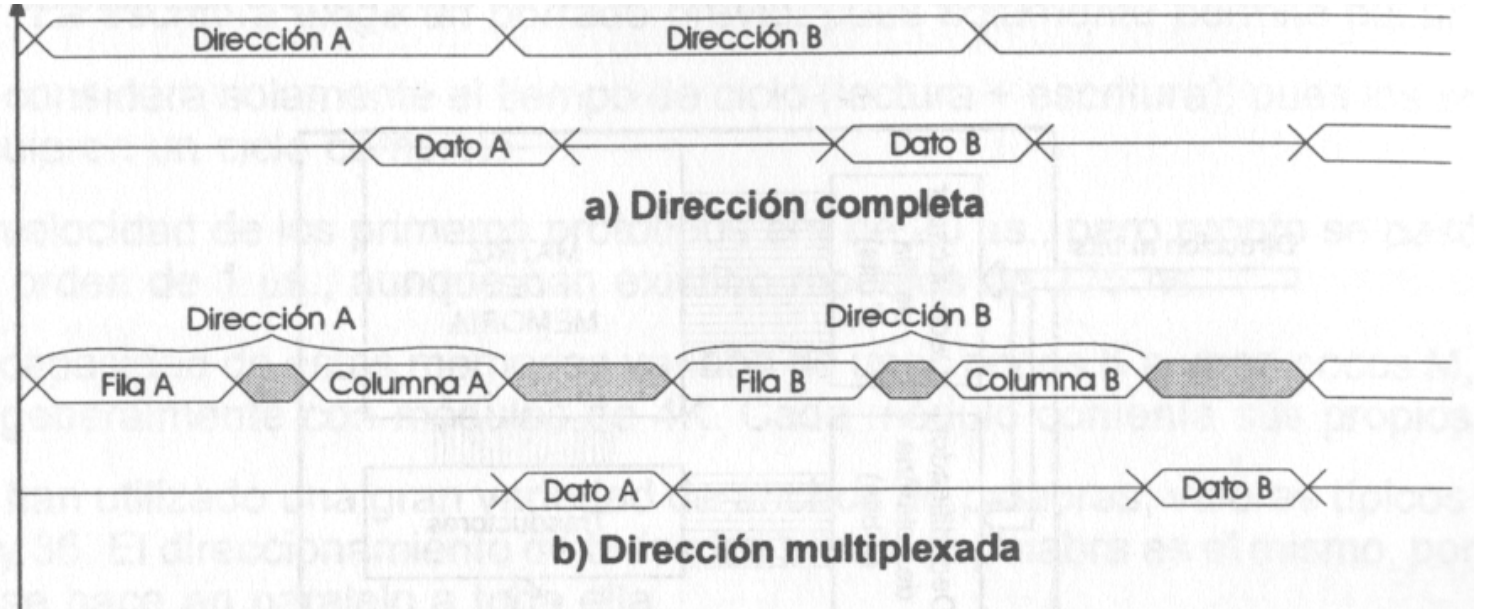
  - ➔ **Trabajando a 266MHz alcanzan un ancho de banda de 2.1GB**

- **Mejora de prestaciones (cronogramas)**

Figura tomada de Pedro de Miguel. *Fundamentos de los computadores*. Paraninfo, 1999

a) SRAM: se emite la dirección y se obtiene la salida en el tiempo de latencia

b) DRAM: la dirección se ha de emitir en dos pasos, primero fila y luego columna

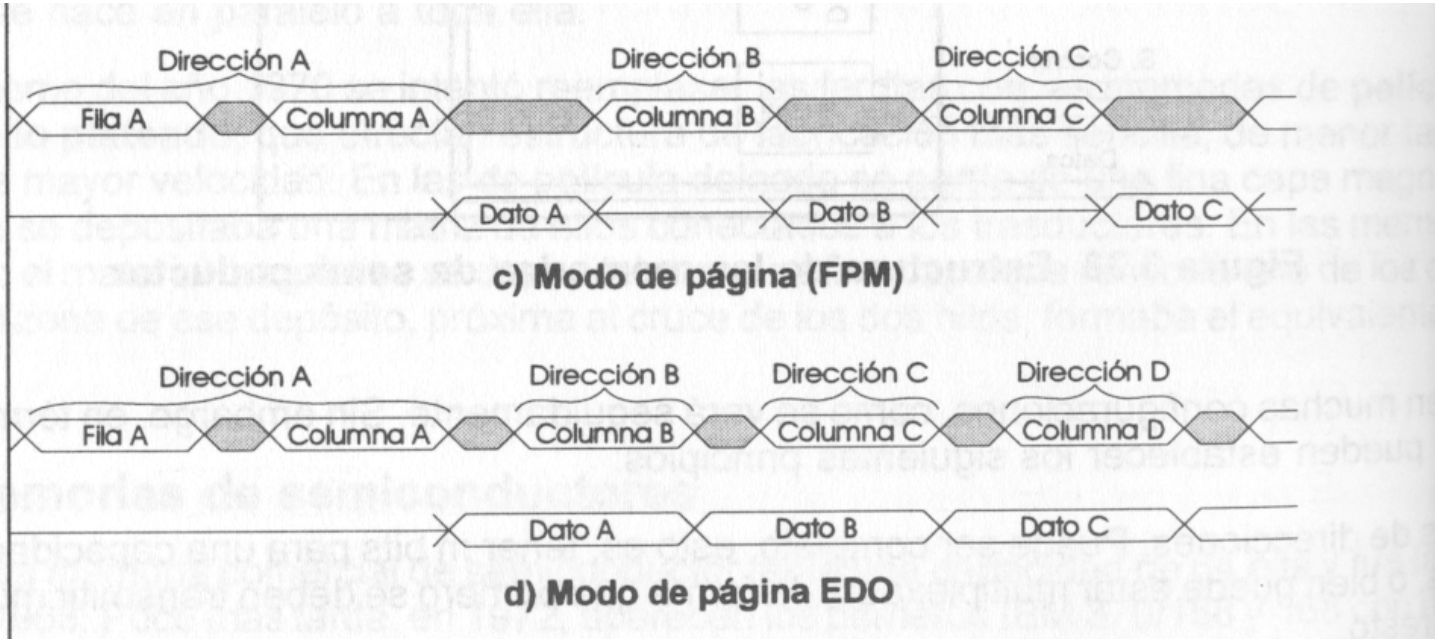


- **Mejora de prestaciones (cronogramas)**

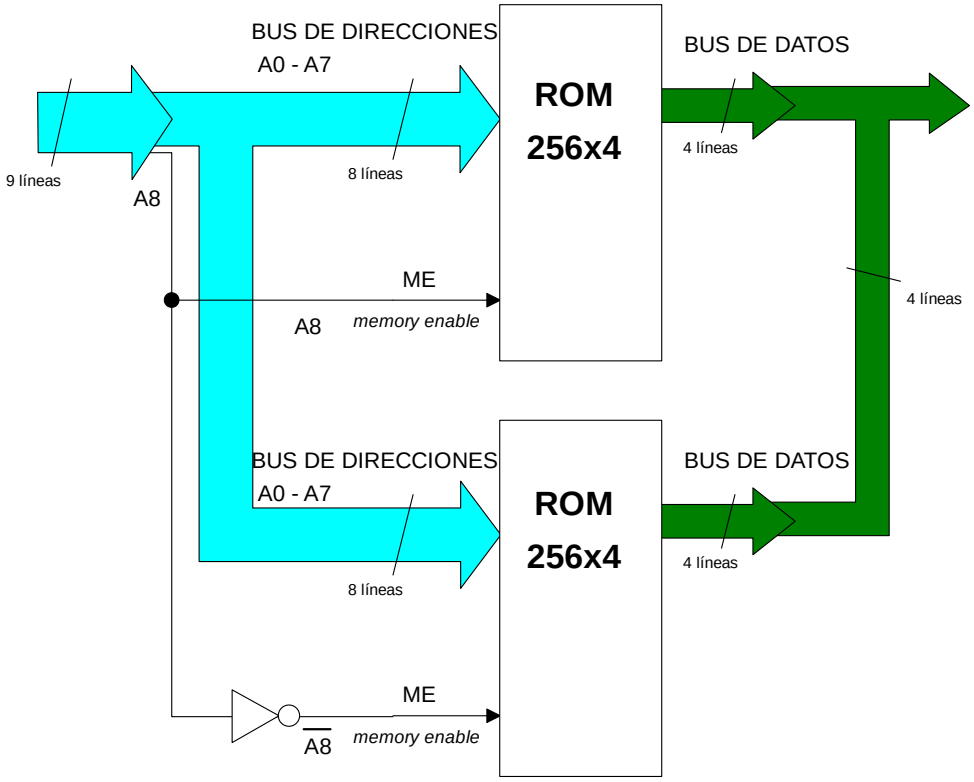
Figura tomada de Pedro de Miguel A. *Fundamentos de los computadores*. Paraninfo, 1999

c) FPM: los datos A, B y C están en la misma fila o bloque o página

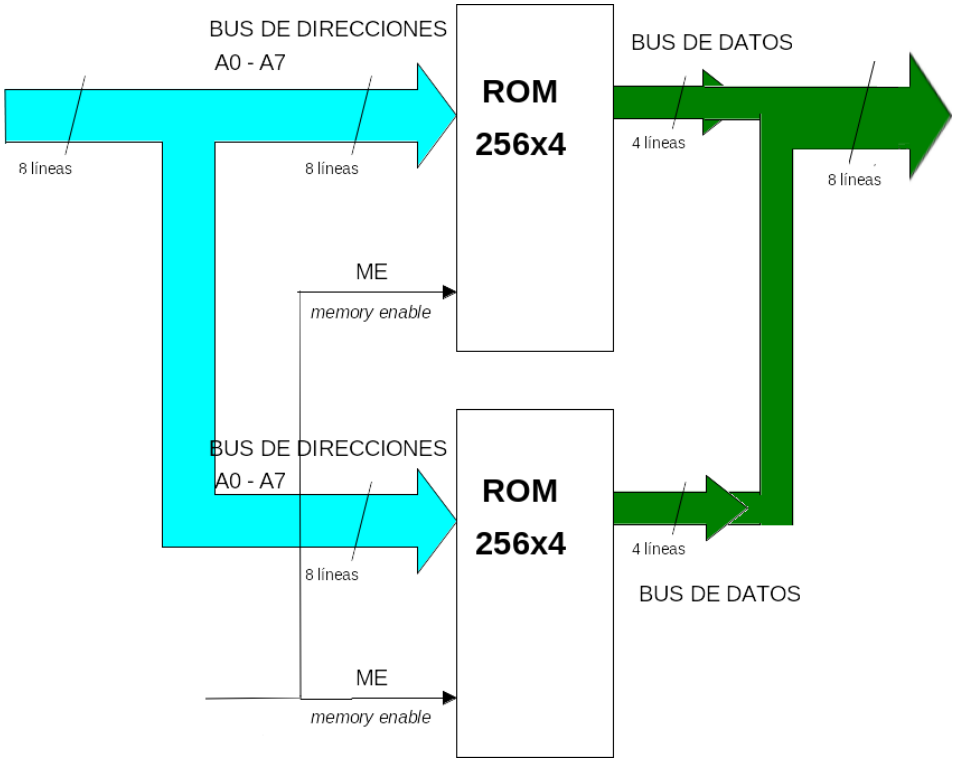
d) EDO: ídem; véase como el direccionamiento de B se solapa con la salida de A



- **Extensión de direcciones**
  - ➔ **2 pastillas de 256x4 dan lugar a una memoria de 512x4 añadiendo una línea más de direcciones que ataca a cada pastilla de manera complementaria**



- **Incremento de la anchura de palabra**
  - ➔ **2 pastillas de 256x4 dan lugar a una memoria de 256x8 colocando en paralelo la salida de ambas pastillas; esto aumenta la capacidad de transferencia**

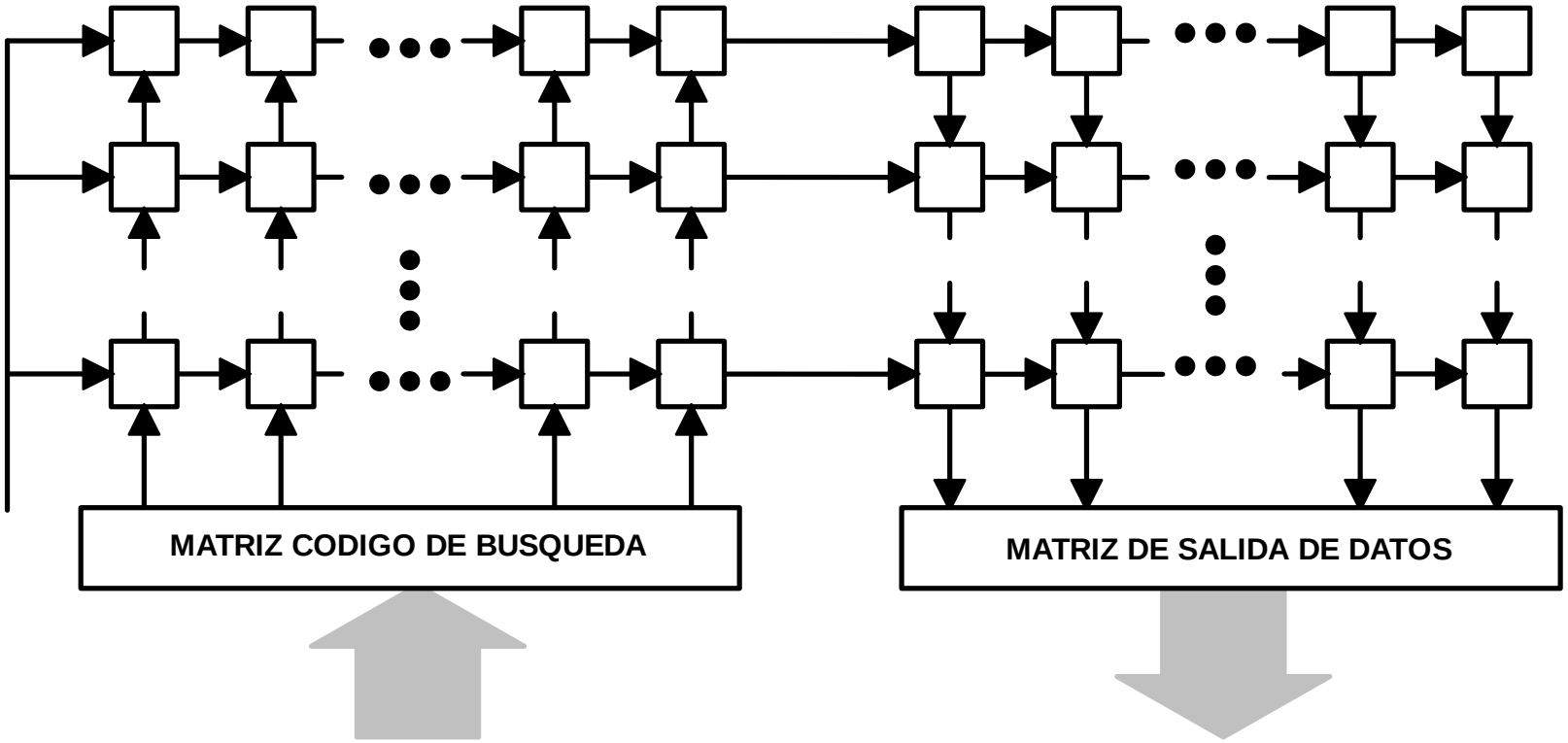


La memoria

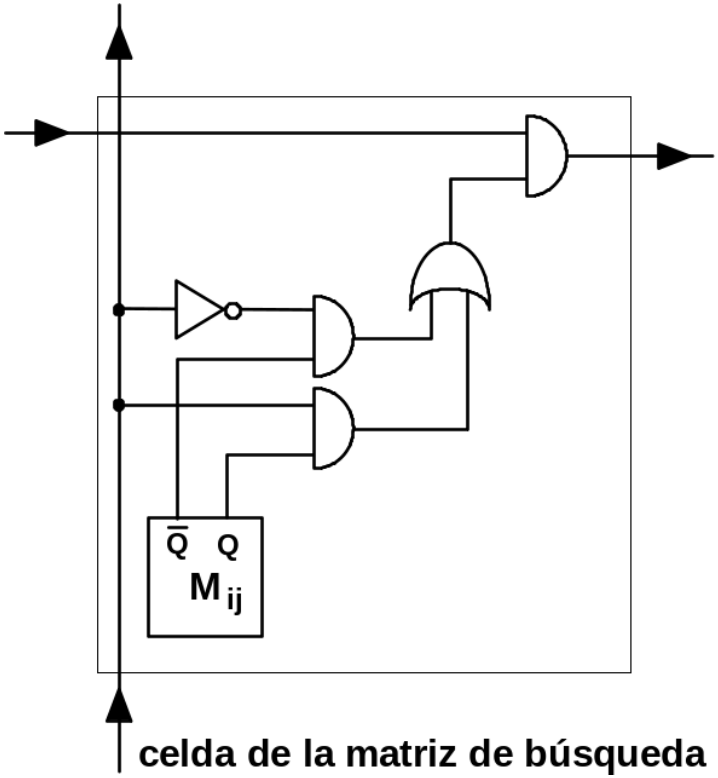
Índice

- Es una organización de memoria en la que no existe identificador del continente, no existen direcciones
- El acceso consiste en buscar un código entre todos sus datos
- La búsqueda se hace en paralelo de izquierda a derecha
- También se llama:
  - ➔ Memoria CAM (*Content Access Memory*)
  - ➔ Memoria de acceso por contenido

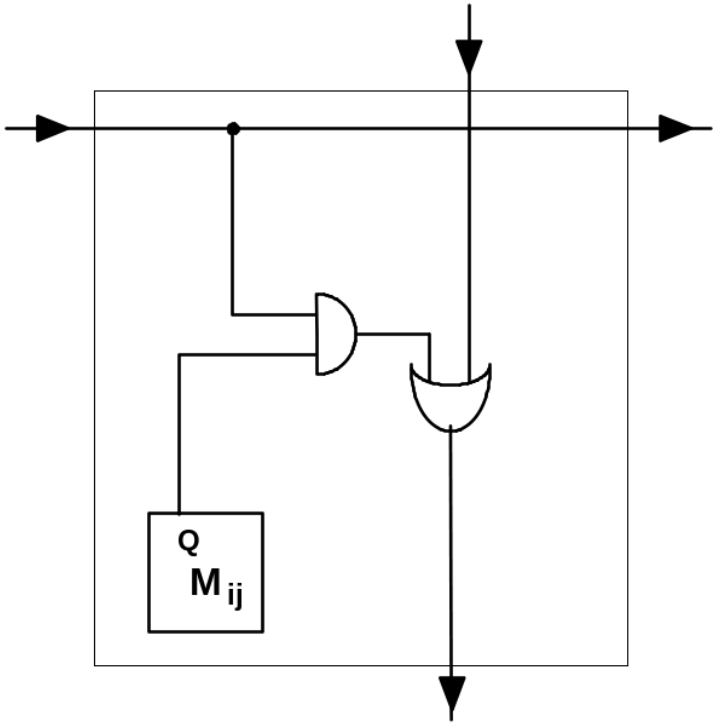
➔ **Estructura interna:**



➔ Celda de la matriz de búsqueda y celda de la matriz de datos



celda de la matriz de búsqueda

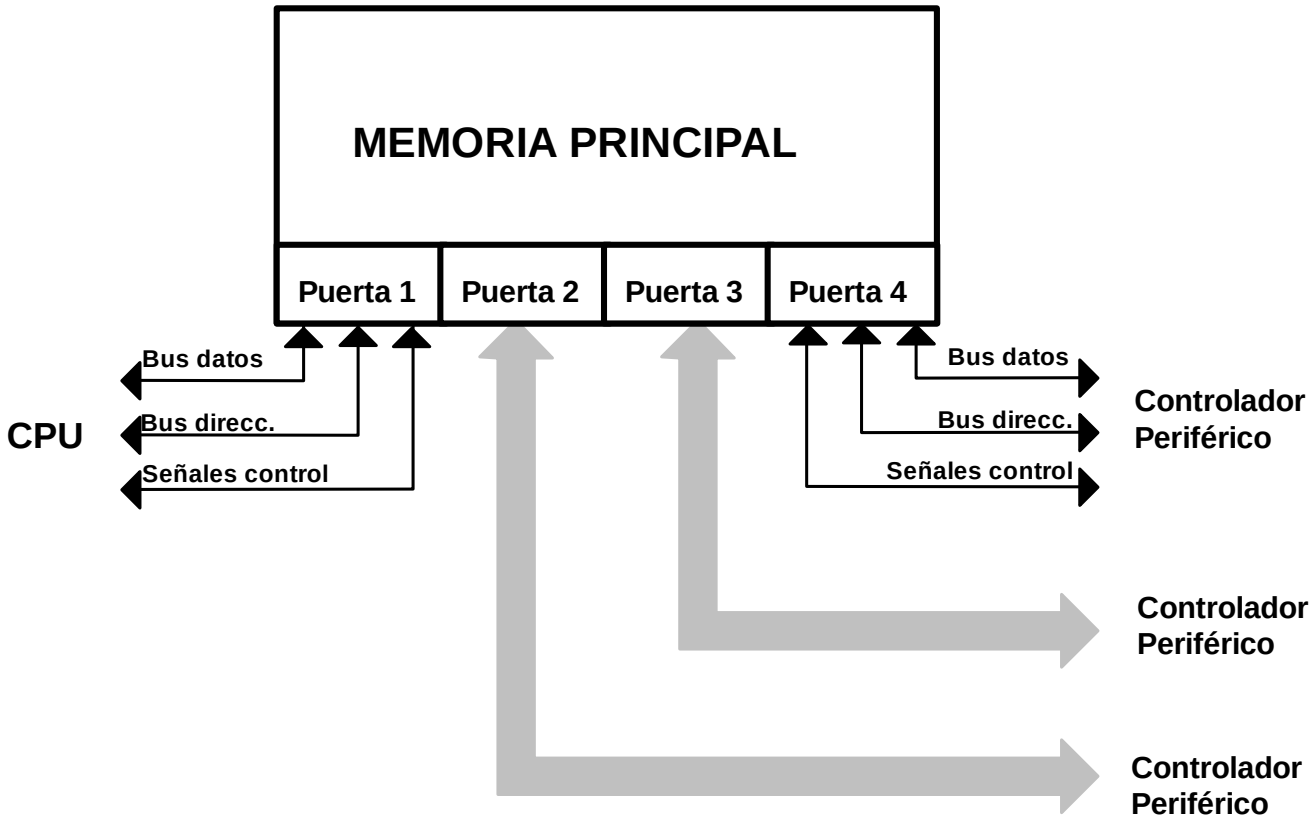


celda de la matriz de datos

- **Funcionamiento:**
  - ➔ **El código de búsqueda se compara simultáneamente en todas las filas de la matriz de búsqueda**
  - ➔ **Los emparejamientos se propagan de izquierda a derecha**
  - ➔ **La fila en la que se produzcan todos los emparejamientos será aquella que contenga el código de búsqueda y producirá un '1' en la señal de salida de fila**
  - ➔ **La señal de salida de fila habilita una fila de la matriz de datos que vuelca su código en la salida**

- **Usos típicos:**
  - Tablas de correspondencias
  - Tablas de traducción de direcciones de memoria virtual y caché

- Misma latencia de memoria
- Mayor ancho de banda



La memoria

2. Organizaciones alternativas
2.2. Memoria multipuerta

- **Dado que se realizan accesos simultáneos por todas las puertas, crece la capacidad de transferencia**

- **El problema que tiene es que dos o más puertas tengan que servir un acceso a la misma dirección de memoria**
  - En este caso se produce un conflicto que el *hardware* debe solucionar con la consiguiente pérdida de tiempo

- **Es una organización que está indicada siempre que aseguremos que la tasa de conflictos va a ser pequeña**

- **Uso típico:**
  - Memoria de vídeo (en este caso se suele llamar VRAM)
  - Durante un ciclo de vídeo
    - Una puerta envía un “frame” al monitor
    - La otra puerta recibe un “frame” desde el procesador
  - Al ciclo siguiente se permutan los accesos y nunca se produce conflicto de acceso a la misma dirección

La memoria

Índice

- **Memoria: cuello de botella con 2 facetas**

  - ➔ **Latencia de memoria**
    - ➔ Los tiempos de acceso a memoria son muy grandes

  - ➔ **Ancho de banda**
    - ➔ Capacidad de transferencia limitada

- **El problema no tiene una solución fácil:**
  - ➔ **Grande**
  - ➔ **Rápido**
  - ➔ **Barato**

  - ➔ **Dependencias circulares en las opciones de diseño:**
    - ➔ A mayor capacidad mayor tiempo de acceso
    - ➔ A mayor velocidad mayor coste
    - ➔ A mayor capacidad menor coste por bit

- **No se puede utilizar una única tecnología**

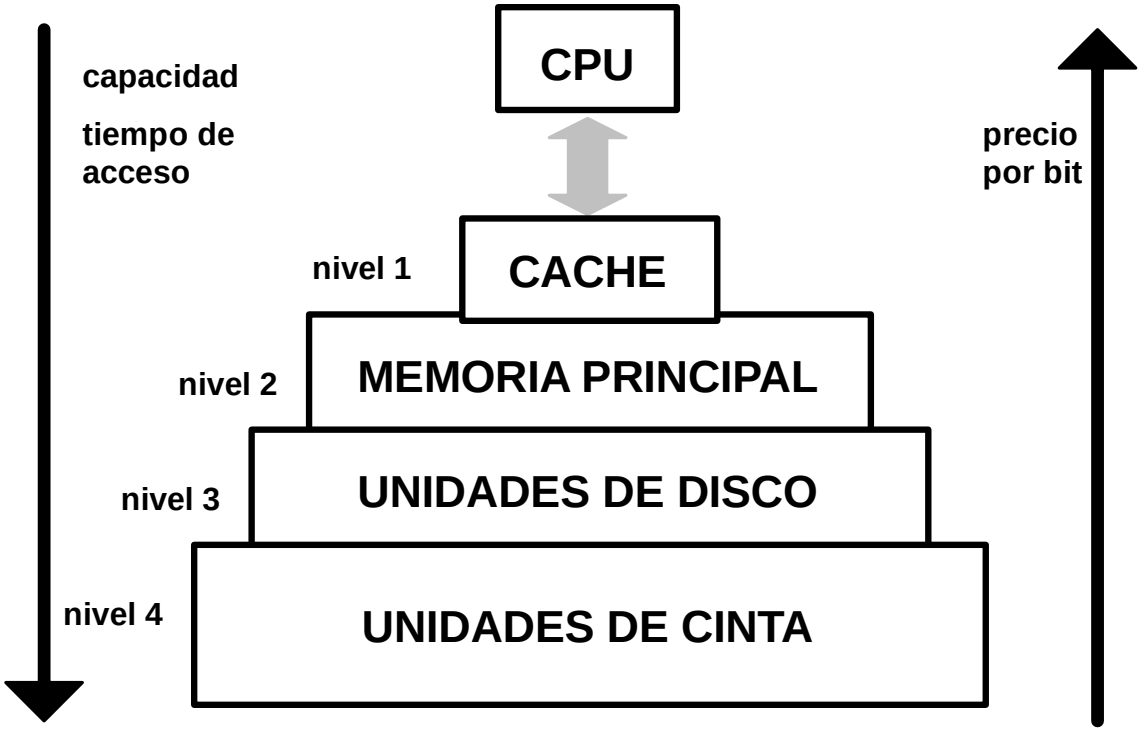
➔ **Solución ➔ jerarquía de memoria**

➔ **Combinación de tecnologías**

➔ Desde el punto de vista del procesador

- Hay una única memoria
- El tamaño es el más grande de todos
- El tiempo de acceso corresponde (casi) al de la tecnología más rápida

- Organización de memoria en varios niveles
  - ➔ Cada uno más pequeño, más rápido y más caro por byte que el anterior
  - ➔ Todos los datos de un nivel se encuentran en el inferior



- **Tamaños y tiempos de acceso:**

Jerarquía	Capacidad en bytes	Tiempo de acceso	Tipo	Sistema de control
Registros	512 bytes	10 ns	ECL	compilador
Memoria caché	128 KB	25 - 40 ns	SRAM	*hardware*
Memoria principal	512 MB	60 - 100 ns	DRAM	S. O.
Memoria auxiliar en disco	60-228 GB	12 - 20 ms	Magnético	S.O. /usuario
Memoria auxiliar en cinta	512 GB - 2 TB	minutos	Magnético	S.O. /usuario

- **La efectividad de la jerarquía de memoria se basa en el principio de localidad**

  ➔ **Los programas no acceden al código y a los datos de manera uniforme, aleatoria o "*equiprobable*"**

  ➔ **Los programas favorecen una parte del espacio de direcciones**

- **Localidad temporal → si se referencia un elemento, tenderá a ser referenciado de nuevo pronto**
  - En código: instrucciones del cuerpo del bucle
  - En datos: accesos iterativos

- **Localidad espacial → si se referencia un elemento, los elementos cercanos a él tenderán a ser referenciados también**
  - En código: ejecutada una instrucción se ejecutarán las cercanas
  - En datos: referenciado un elemento de un *array* se referenciarán los cercanos

- En cada momento la jerarquía se gestiona entre 2 niveles adyacentes
- La mínima unidad de información es el bloque

- Acierto (*hit*) → éxito en la búsqueda de un bloque en un nivel superior
- Fallo (*miss*) → fracaso en la búsqueda

- Frecuencia o tasa de aciertos: porcentaje de aciertos (*h*)
- Frecuencia o tasa de fallos: 1 – *h* = *f*

- **Lo importante es el rendimiento (velocidad de ejecución) no la tasa de aciertos**

- **Tiempos:**
  - Tiempo de acierto = tiempo de acceso al nivel superior
    - Incluye el tiempo de búsqueda (determinar acierto o no)
  - Penalización de fallo = tiempo de reemplazo + tiempo de entrega al dispositivo
    - Incluye el tiempo de acceso al nuevo bloque y el tiempo de transferencia
      - tiempo acceso → depende de la latencia
      - tiempor transferencia → depende del ancho de banda y del tamaño del bloque

➔ **Tiempo medio de acceso:**

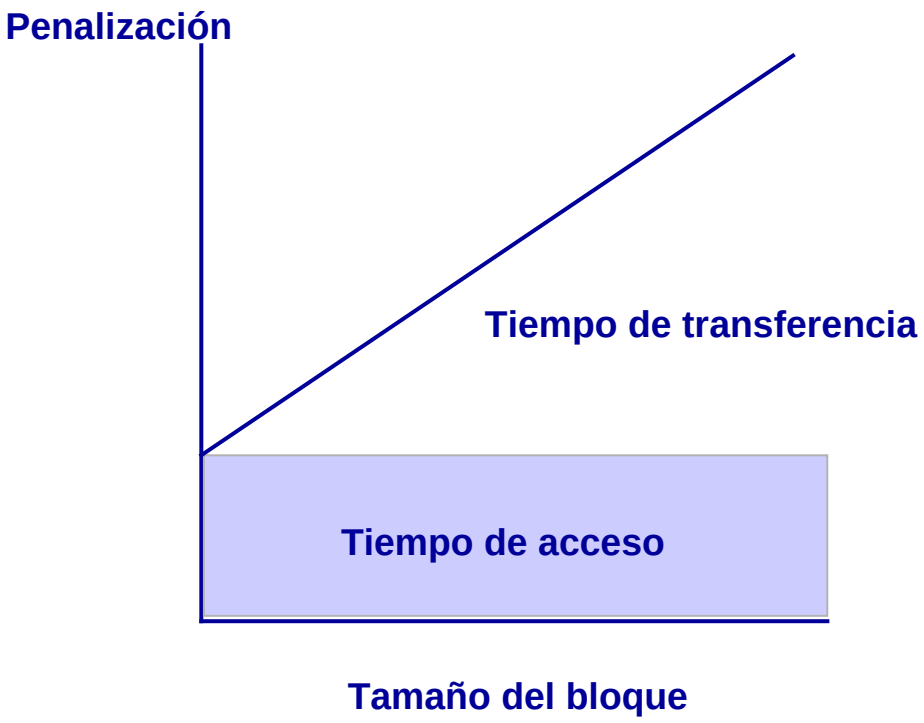
$$tiempo\,medio\,acceso\,memoria = tiempo\,acierto + tasa\ fallos \cdot penalización$$

$$t_m = t_a + (1 - h) \cdot p$$

➔ **La penalización depende de las características de la caché (tamaño del bloque, organización, etc.)**

➔ Penalización *vs.* Tamaño del bloque

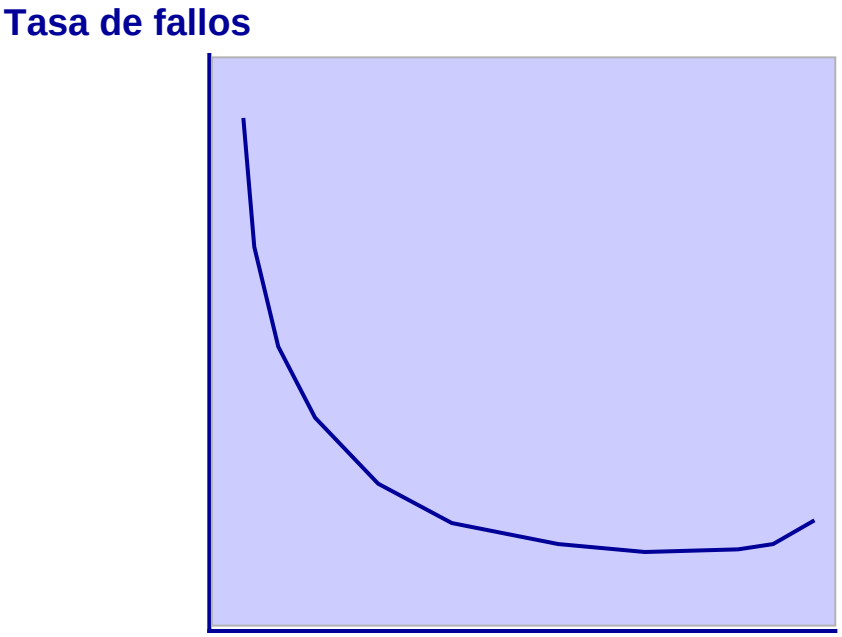
- tamaño de memoria constante



- **En los bloques grandes el tiempo de transferencia es grande**

➔ Tasa de fallos *vs.* Tamaño del bloque

- tamaño de memoria constante



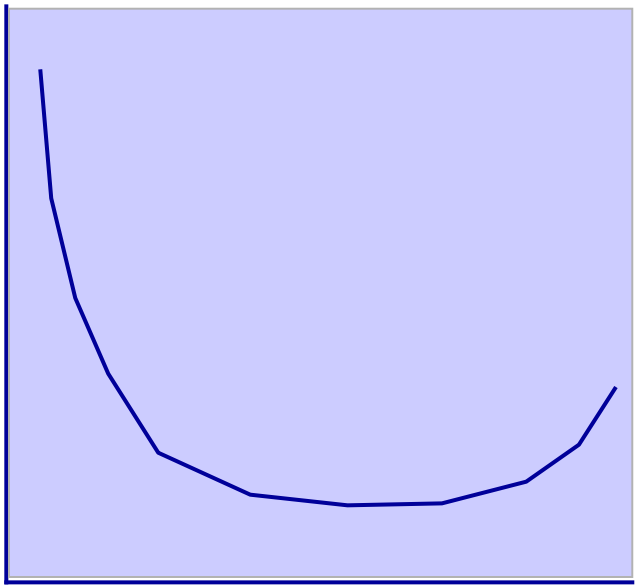
- **En los bloques muy pequeños enseguida se produce fallo**
- **Con bloques muy grandes la parte no utilizada resta capacidad para albergar el bloque requerido**

gráfica con forma de L

➔ Tiempo de acceso *vs.* Tamaño del bloque

- tamaño de memoria constante

Tiempo de acceso



Tamaño del bloque

- **Con bloques pequeños la tasa de fallos manda en la evaluación del tiempo de acceso**
- **Con bloques muy grandes manda el término de la penalización**

gráfica con forma de U

➔ **Si la penalización es muy costosa, el procesador es interrumpido (excepción por fallo) y utilizado en otro proceso**

➔ Implica:

- Salvar contexto y recuperar el proceso original
- Implementar protección de memoria

➔ Esta situación degrada el rendimiento; por eso es conveniente que la penalización sea pequeña

- **Dependiendo de los niveles involucrados y del coste temporal de**
  - las búsquedas;
  - los reemplazos; y
  - las transferencias de bloques

- **… la implementación puede ser:**
  - *Hardware* → caché
  - *Software* → memoria virtual

- **Las direcciones de memoria se dividen en campos que identifican:**
  - Un elemento del bloque (campo de menor peso)
  - Identificador de bloque (campo de mayor peso u orden superior)

Identificador de bloque

Dirección dentro del bloque

La memoria

3. Jerarquía de memoria
3.4. Coherencia

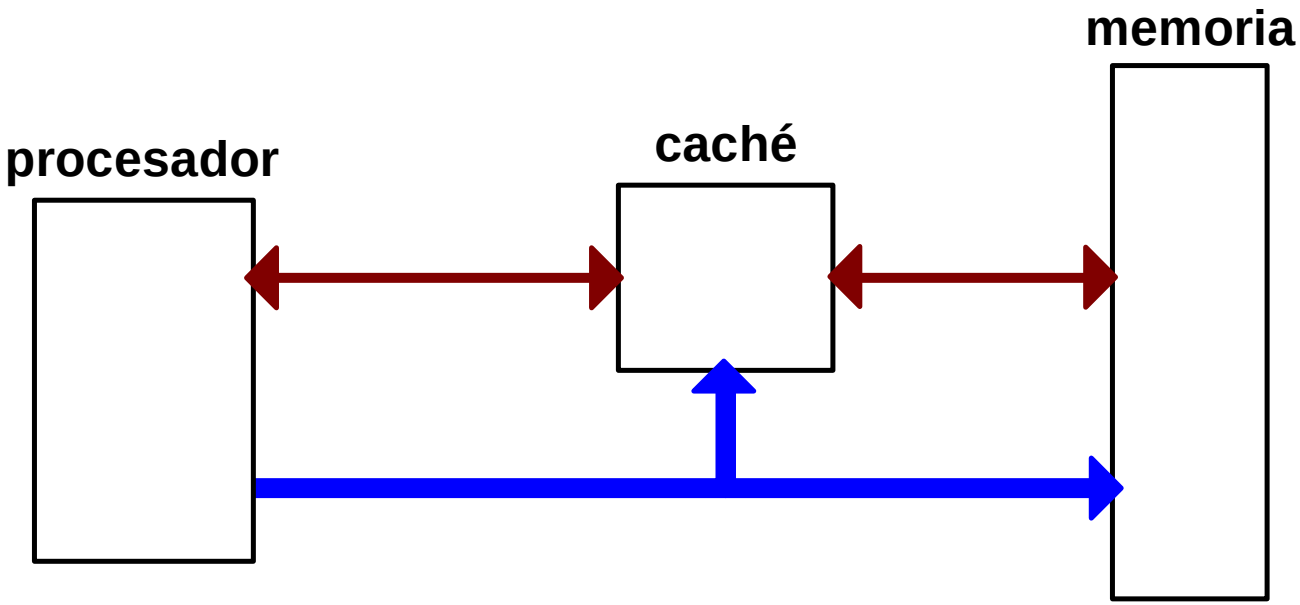
- Hay que asegurar la coherencia o consistencia de la información almacenada cuando se escribe en un bloque

- Todos los datos de un nivel se encuentran en el inferior

- Hay que diseñar sistemas que indiquen la validez o no de un bloque

Índice

- **Es una memoria auxiliar de alta velocidad, que se interpone entre la CPU y la memoria principal para acelerar el funcionamiento del procesador**



La memoria

4. Memoria caché

4.1. Introducción

- **Concepto de caché:**
  - ➔ **Wilkes, 1965**

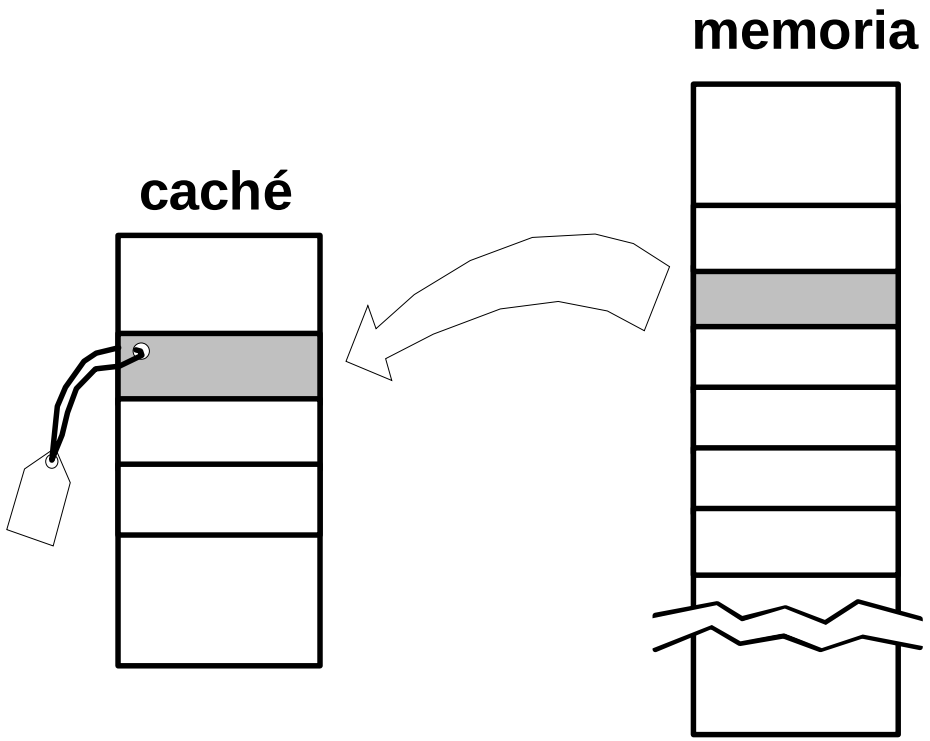
- **Primera máquina comercial con caché**
  - ➔ **IBM 360/85, 1968**

La memoria

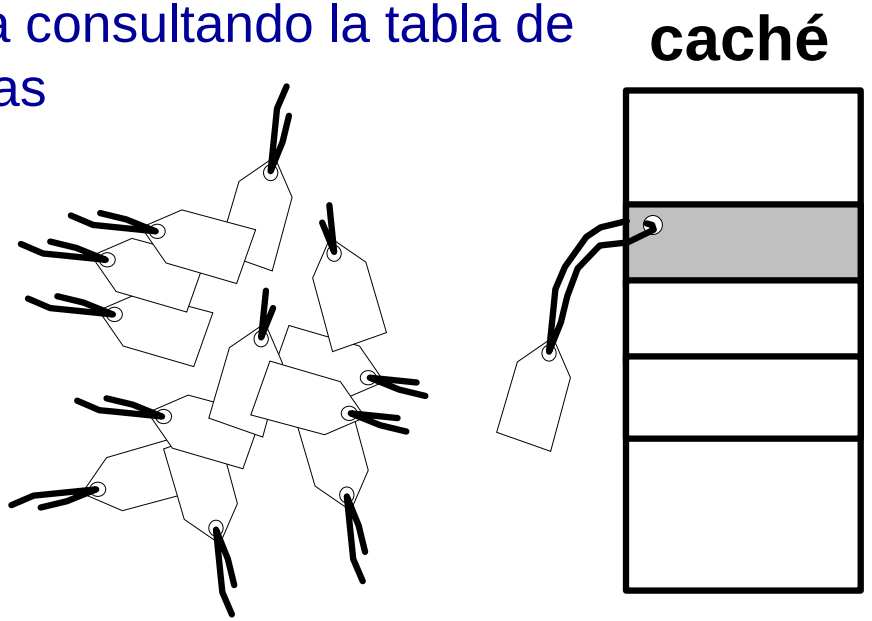
4. Memoria caché

4.1. Introducción

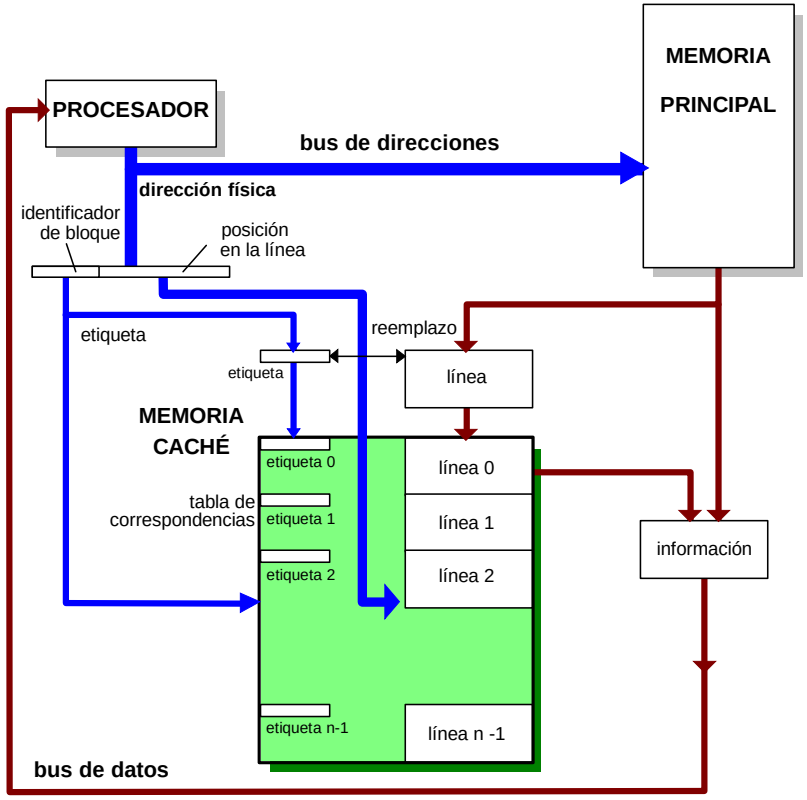
- La caché esta dividida en bloques de longitud fija llamados *líneas* que contienen temporalmente copias de los bloques de memoria principal
- Cada una de las líneas lleva asociada una etiqueta que la identifica



- **Dos zonas de memoria:**
  - **Las líneas; y**
  - **La tabla con las etiquetas de las líneas o *tabla de correspondencias***
    - La comprobación de si una posición está o no en la caché se realiza consultando la tabla de correspondencias



- **Secuencia de operaciones (I)**
  - ➔ **El procesador genera una dirección física de memoria y la envía a memoria caché**
  - ➔ **La memoria caché traduce la dirección física a dirección caché y comprueba si tiene la referencia**
    - ➔ Desde el punto de vista de la memoria caché, la dirección física contiene algún campo relativo a la etiqueta de la línea (identificador de línea) y otro relativo a la dirección dentro de la misma

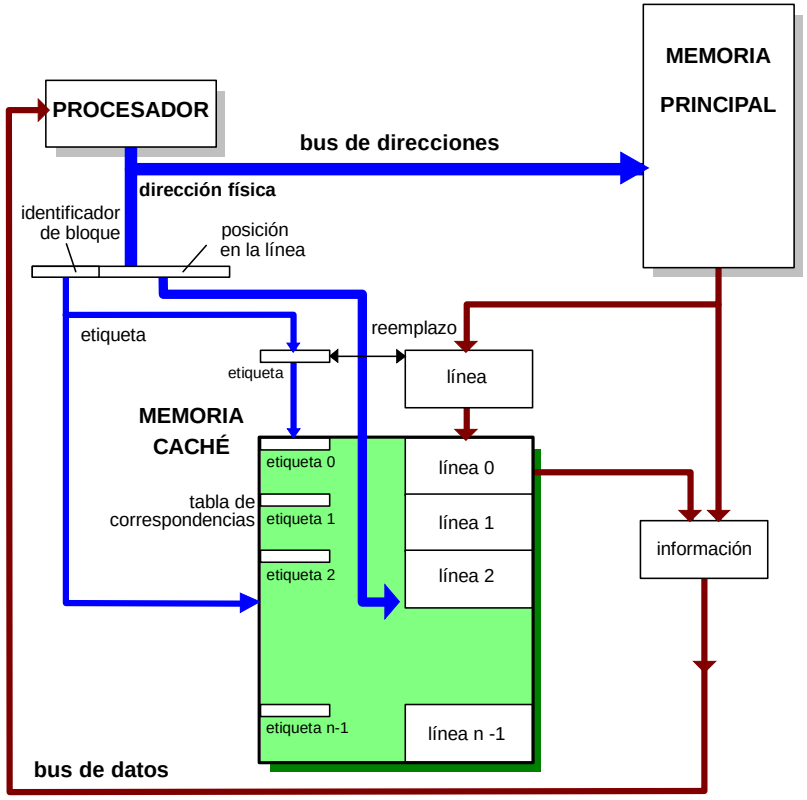


La memoria

4. Memoria caché

4.1. Introducción

- **Secuencia de operaciones (II)**
  - **En caso de acierto basta con referenciar la posición correspondiente de la línea**
  - **En caso de fallo, se accede a memoria principal para referenciar la posición y, dado que muy probablemente necesitaremos ese bloque, copiamos en caché la información de memoria principal, seleccionando una línea para sustituirla**
  - **Antes de sustituir la línea, si ha sido modificada, se almacena en memoria principal para salvar la coherencia**



La memoria

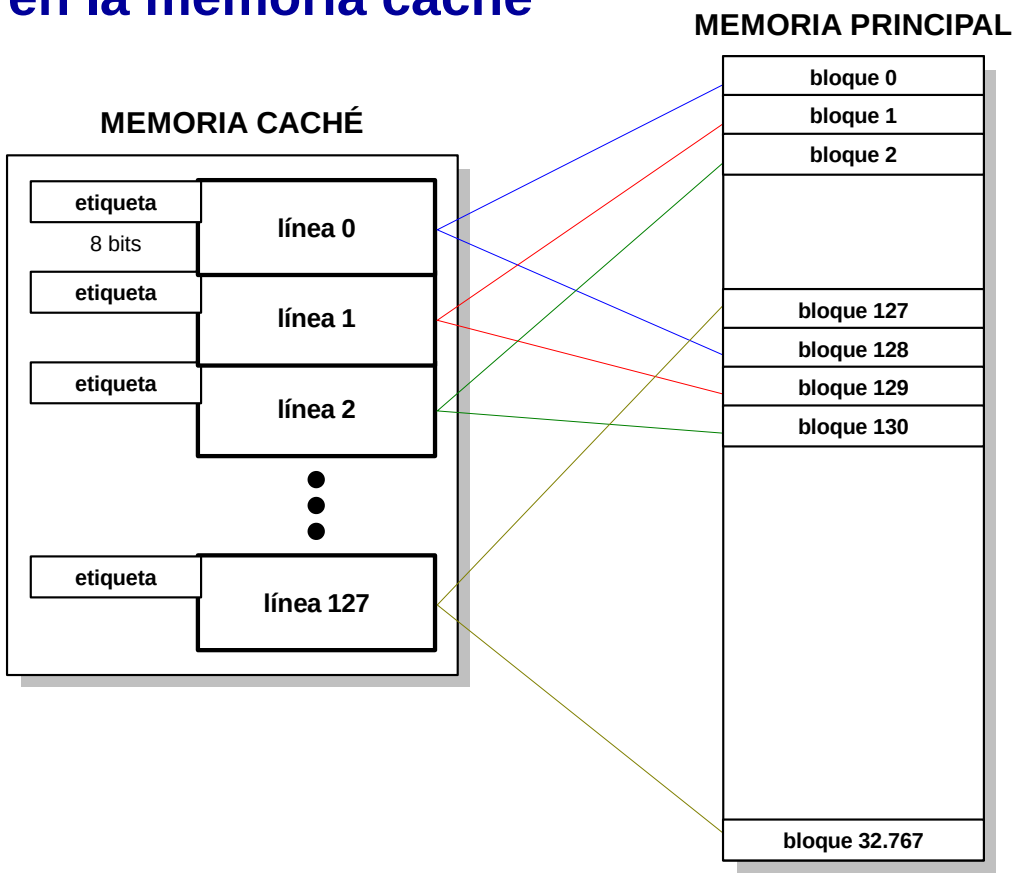
4. Memoria caché

4.1. Introducción

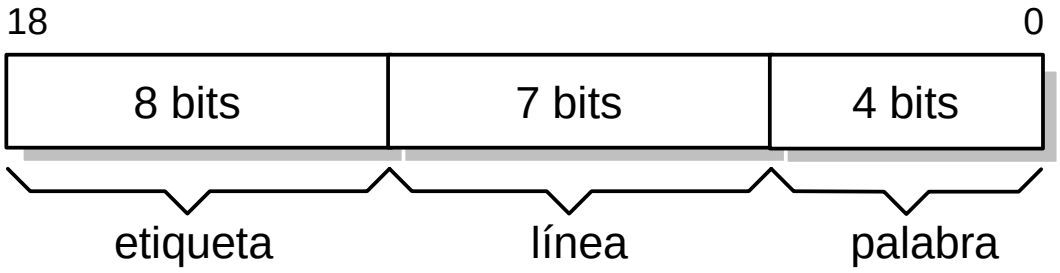
- **A la hora de diseñar la caché hemos de hacernos cuatro preguntas:**

- **Organización de caché**
  - **Para la transferencias de bloques entre memoria principal y memoria caché hay que seguir una determinada política de ubicación**
  - **Existen básicamente tres políticas de ubicación u organizaciones:**
    - correspondencia directa;
    - totalmente asociativa; y
    - asociativa por conjuntos.

- **Correspondencia directa (I)**
  - ➔ **Cada bloque de memoria principal solo puede cargarse en una línea determinada en la memoria caché**
    - ➔ Ejemplo:
      - caché de 2Kpalabras
      - líneas de 16 palabras
      - 128 líneas
      - memoria principal de 512Kpalabras
      - 32.768 bloques



- **Correspondencia directa (II)**
  - **Direcciones de 19 bits que se dividen en tres campos**



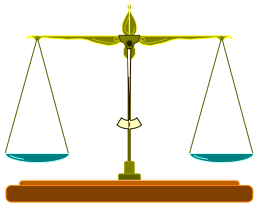
  - ***Palabra* direcciona una de las 16 palabras dentro de la línea**
  - ***Línea*, de 7 bits en el ejemplo, identifica una de entre las 128 de la caché; todas las direcciones que tengan este campo igual irán a colocarse en el mismo lugar físico (*i* MOD 128)**
  - ***Etiqueta* identifica uno de los 256 (2^8) bloques de memoria principal que pueden colocarse en la línea indicada**

- **Correspondencia directa (III)**

  - ➔ **El procesador genera una dirección física (19 bits) donde los 7 bits del campo de línea referencian una de las 128 de memoria caché**

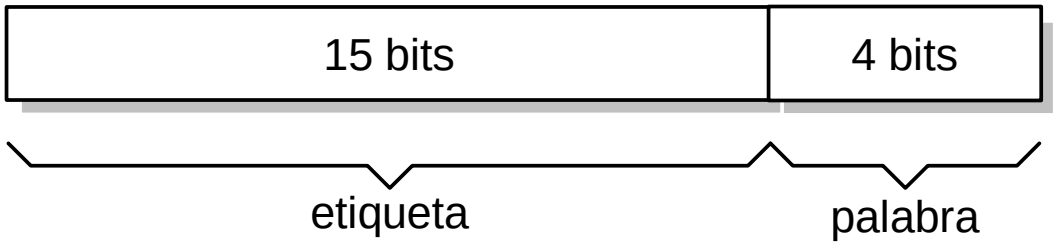
  - ➔ **Cada línea lleva asociado un registro de etiqueta que se compara con los 8 bits del campo correspondiente y, si coincide, se busca la palabra señalada por los bits del campo palabra dentro de la línea**

- **Ventajas e inconvenientes:**
  - ➔ **No existe una tabla de correspondencias propiamente dicha**
  - ➔ **La comparación de la etiqueta y el acceso a la línea pueden ser simultáneos minimizando el tiempo de identificación**



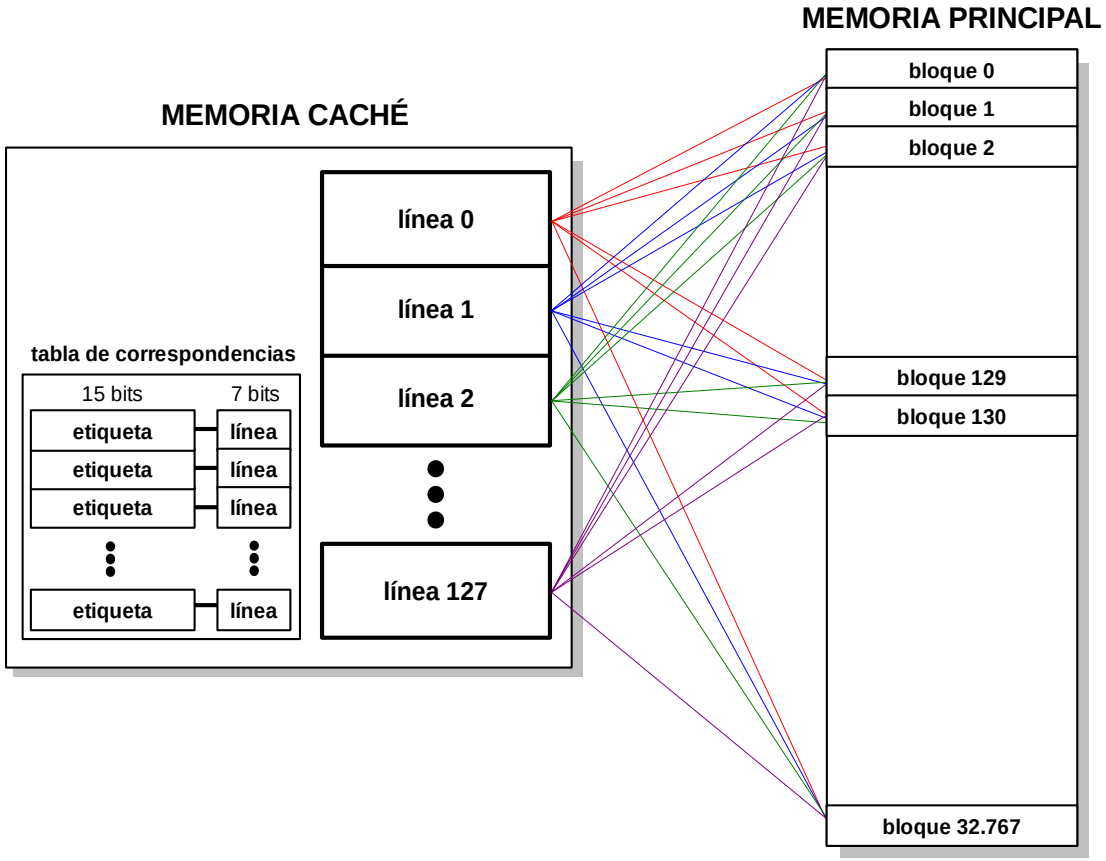
  - ➔ Tiene la ventaja de presentar una relación unívoca entre los bloques de memoria principal y las líneas de memoria caché por lo que se evita la búsqueda asociativa entre varias etiquetas
  - ➔ Esta política de ubicación no necesita ningún algoritmo de reemplazo
  - ➔ Una desventaja es que la tasa de aciertos disminuye cuando se referencian dos o más bloques alternativamente que necesitan la misma línea

- **Totalmente asociativa (I)**
  - ➔ **Un bloque de memoria principal puede colocarse en cualquier línea de la memoria caché**

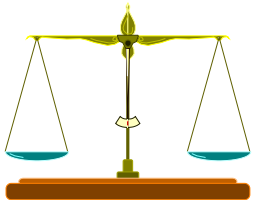


  - ➔ **El campo *etiqueta* de la dirección física debe ser capaz de nombrar cualquier bloque de memoria principal**
    - ➔ Para nuestro ejemplo, necesitará 15 bits puesto que 2^{15} da lugar a 32.768 (32K) posibles bloques

- **Totalmente asociativa (II)**

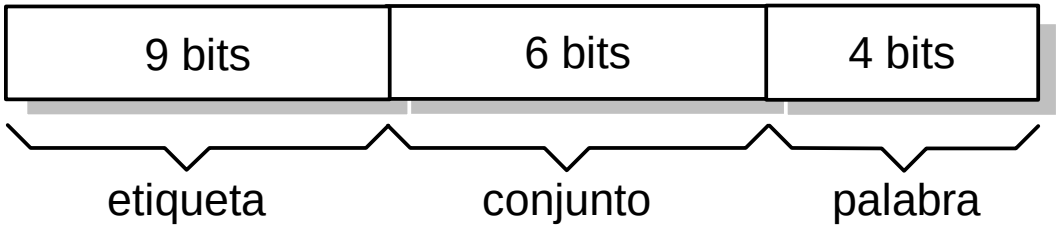


- **Ventajas e inconvenientes:**
  - **La etiqueta de la dirección debe compararse con todas las posibles entradas de la tabla de correspondencias con el consiguiente retardo**
  - **La comparación se realiza por búsqueda asociativa implementada en memoria asociativa (referencia por dato)**

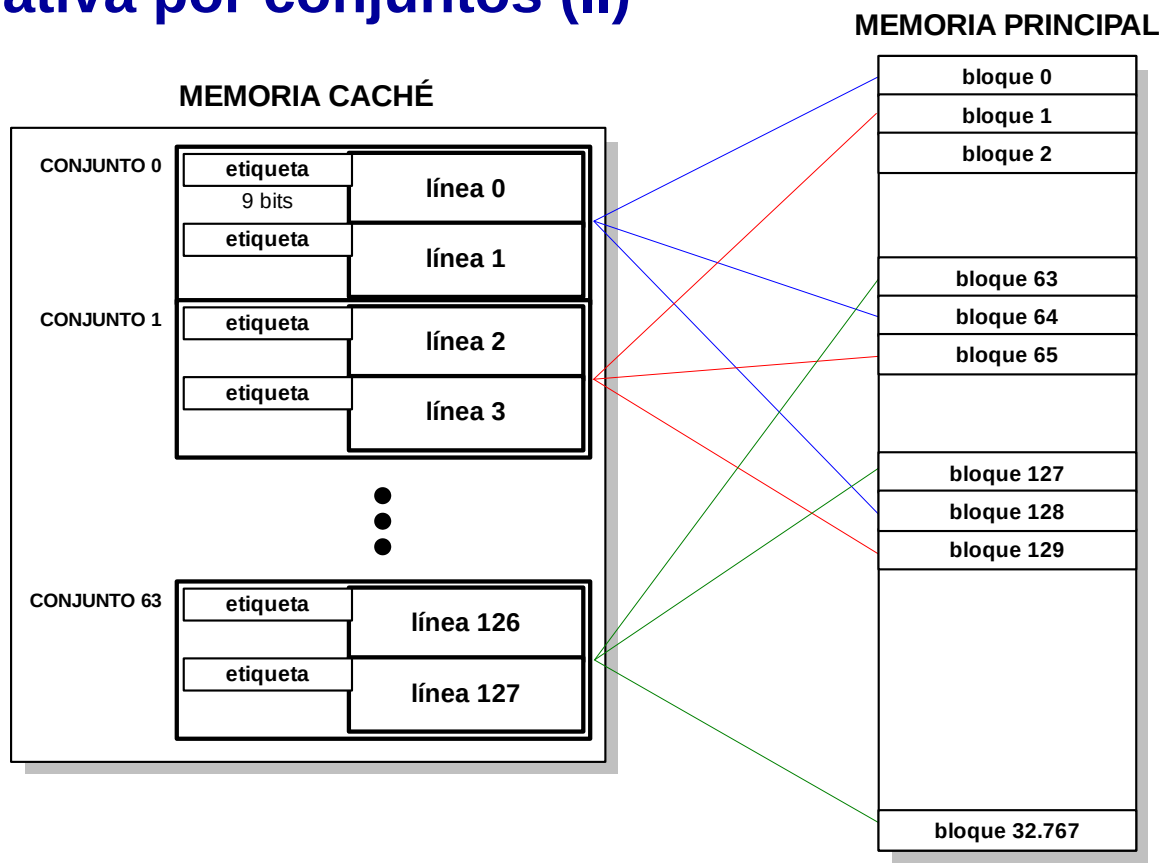


  - Esta es la organización con mejor tasa de aciertos ya que las líneas de caché contienen los bloques de memoria principal que más se están utilizando
  - Tiene como inconvenientes el tiempo de acceso grande (búsqueda asociativa), además de un alto coste en implementación (gran área de silicio que consume)
  - Necesita incluir algún criterio de reemplazo para determinar, en caso de fallo, qué línea va a ser sustituida

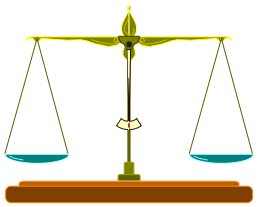
- **Asociativa por conjuntos (I)**
  - ➔ **Organización combinación de las dos anteriores que consiste en dividir el número total de líneas de caché en *C* conjuntos de *V* líneas cada uno y aplicar la correspondencia directa a nivel de conjunto**
  - ➔ **El campo *conjunto* especifica aquel sobre el que se ubica un bloque de principal**



- **Asociativa por conjuntos (II)**



- **Ventajas e inconvenientes:**
  - **El campo conjunto determina aquel sobre el que se ha de buscar asociativamente el bloque solicitado**
  - **Dado que el número de vías es pequeño comparado con el total de líneas, dicha búsqueda se hace sencilla y rápida**



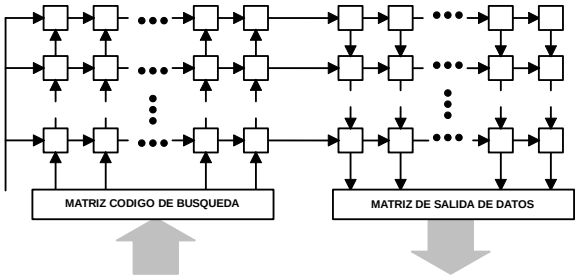
  - El coste de la búsqueda asociativa se reduce respecto a la anterior pero la tasa de aciertos es muy cercana
  - Dentro de cada conjunto se puede utilizar cualquier algoritmo de reemplazo
  - Resultados experimentales demuestran que conjuntos de 2 a 8 vías proporcionan rendimientos cercanos a la organización totalmente asociativa con un coste muy poco mayor al de la organización de correspondencia directa

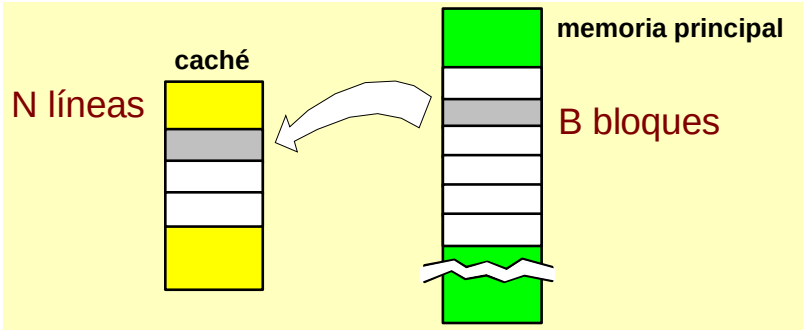
- **Comparativa de tipos de organización**

  - **Costes de implementación de las tablas de correspondencia**
    - Supone usar transistores, es decir, área de silicio

  - **Retardos debidos a las tablas de correspondencia**
    - Supone consumo de tiempo

- **Comparativa de costes (I)**
  - ➔ **Totalmente asociativa y asociativa por conjuntos necesitan tablas de correspondencias**
  - ➔ **Si la memoria principal está dividida en *B* bloques y la caché tiene *N* líneas**





  - ➔ **El tamaño de la memoria asociativa será:**

$$TAMAÑO = N \cdot (\log_2 B + \log_2 N)$$

➔ *N*: número de salidas de la tabla; *B*: número de entradas

- **Comparativa de costes (II)**
  - ➔ **Si organización es totalmente asociativa el tamaño será:**

$$TAMAÑO_{asoc\_total} = N \cdot (\log_2 B + \log_2 N)$$

  - ➔ **Asociativa por conjuntos de *V* vías tendré *N*/*V* conjuntos**
    - ➔ La tabla de cada conjunto ocupa:

$$TAMAÑO_{conj} = V \cdot (\log_2 \frac{B \cdot V}{N} + \log_2 V)$$

  - ➔ **El tamaño total de memoria asociativa será igual al resultado anterior multiplicado por el número de conjuntos (*N*/*V*)**

$$TAMAÑO_{asoc\_conj} = N \cdot (\log_2 \frac{B \cdot V}{N} + \log_2 V)$$

- **Comparativa de retardos (I)**
  - ➔ **El retardo de una memoria asociativa viene dado por la siguiente expresión:**

$$RETARDO = \log_2 B + N$$

  - ➔ donde *N* es el número de salidas de la tabla y *B* es el número de entradas

- **Comparativa de retardos (II)**
  - ➔ **Si la organización es totalmente asociativa el retardo será:**

$$RETARDO_{asoc\_total} = \log_2 B + N$$

  - ➔ **Si tenemos una memoria asociativa por conjuntos de *V* vías la expresión será la siguiente:**

$$RETARDO_{asoc\_conj} = \log_2 \frac{B \cdot V}{N} + V$$

4.2.1. Comparativa de costes y retardos

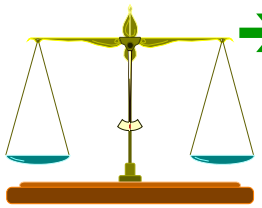
- **Espacio de soluciones para las tres organizaciones de caché**
  - ➔ **Si agrupamos las características de cada organización en una tabla tendremos**

	búsqueda asociativa	tiempo de identificación	algoritmo de reemplazo	tasa de aciertos	coste
correspondencia directa	⊘	0	⊘	↓	↓
totalmente asociativa	✓	↑	✓	↑	↑
asociativa por conjuntos	✓	↓	✓	↗	↘

- **Algoritmos de reemplazo (I)**
  - ➔ **Cuando se produce un fallo de caché hay que buscar el bloque referenciado en memoria principal y cargarlo en caché seleccionando una línea para quitar el bloque almacenado**
  - ➔ **Los mecanismos por los que se selecciona este bloque se conocen como *algoritmos de reemplazo***
  - ➔ **Estos algoritmos se implementan en *hardware***

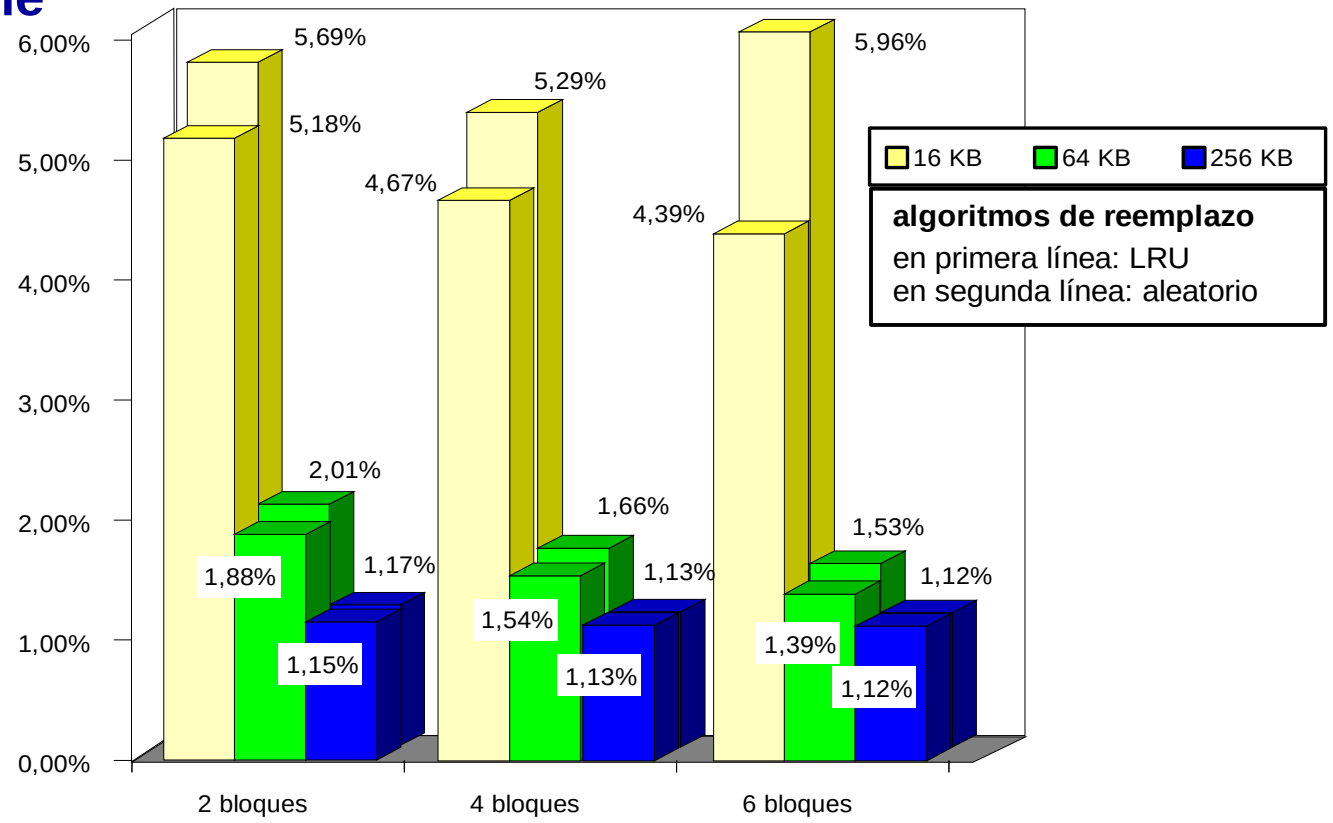
- **Algoritmos de reemplazo (II)**
  - **Los algoritmos de reemplazo de bloques mas utilizados son:**
    - Aleatorio: se sustituye cualquier línea seleccionada aleatoriamente
    - Primero en entrar-primero en salir (FIFO): el bloque que más tiempo lleva cargado en memoria caché es el desalojado
    - Menos recientemente usado (LRU): se sustituye el bloque que lleva más tiempo sin ser usado
      - Está basado en el principio de localidad temporal

- **Algoritmos de reemplazo (III)**
  - **La correspondencia directa no utiliza ningún algoritmo de reemplazo**



  - **El algoritmo que mejor rendimiento presenta es el LRU pero es el más costoso de implementar y el que más recursos necesita**
  - **El tipo de algoritmo tiene mayor importancia en las cachés pequeñas que en las grandes (al disponer de más bloques, la tasa de aciertos no se ve sensiblemente alterada por el tipo de reemplazo)**

- **Algoritmos de reemplazo (IV)**
  - ➔ **Frecuencia de fallos para distintas configuraciones de caché**



Gráfica tomada de *Arquitectura de Computadores. Un enfoque cuantitativo.* Hennessy-Patterson.

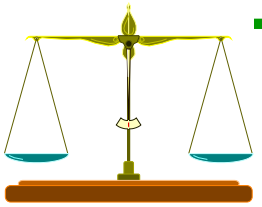
- **Algoritmos de reemplazo (V)**
  - ➔ **De la gráfica anterior se deduce que para tamaños de memoria caché grandes es irrelevante el algoritmo de reemplazo que se implemente**
  - ➔ **Cuando la memoria caché es pequeña entonces sí es visiblemente mejor el algoritmo LRU**

- **Políticas de escritura (I)**

  - ➔ **Las políticas de escritura, también llamadas políticas de actualización, determinan en qué instante se actualiza la información de memoria principal cuando se ha producido una escritura en memoria caché**

- **Políticas de escritura (II)**
  - **Existen dos políticas básicas para escribir en la caché:**
    - Escritura directa: la información se escribe en el bloque de caché y en el bloque de memoria de nivel inferior; así, la información en caché es siempre igual a la de memoria principal (*write through*)
    - Postescritura: la información se escribe sólo en el bloque de la caché y el bloque modificado de la caché se escribe en memoria principal sólo cuando es reemplazado (*copy back*)
      - Se asocia a cada línea un *bit de ensuciado* (*dirty bit*) que se activa solamente cuando se realiza una operación de escritura; de esta manera, a la hora de reemplazar, se comprueba el estado de este bit para evitar escribir si el bloque no ha sido modificado

- **Políticas de escritura (III)**



  - **La escritura directa es más sencilla desde el punto de vista de implementación *hardware* ya que no requiere bit de ensuciado**
  - **La escritura directa garantiza la coherencia en todo momento**
  - **La postescritura genera menos tráfico en el bus y opera a la velocidad de la caché sin que sea necesario incluir ningún retardo debido a la memoria principal**
  - **Ante posibles fallos *hardware* transitorios, es mejor utilizar la escritura directa ya que así toda la información contenida es válida y podemos reconstruir el contenido de la caché**

- **Si el acceso a memoria caché es para escritura y el bloque al que queremos acceder no se encuentra se produce un *fallo de escritura***

  - **¿Reemplazamos?**

  - **¿Escribimos y reemplazamos?**
    - Coherencia
    - Tráfico

- **Ante un fallo de escritura se puede actuar de dos formas:**
  - **Ubicar en escritura: el bloque se carga en memoria caché para ser escrito y luego se utiliza cualquier política de las vistas anteriormente (*with allocate*)**
  - **No ubicar en escritura: el bloque se modifica en el nivel inferior y no se carga en memoria caché (*with no allocate*)**
  - **La escritura directa suele estar acompañada de no ubicación ya que las posteriores referencias a ese bloque se han de escribir en memoria principal también y la postescritura suele ubicar el bloque, esperando que sea utilizado más tarde (localidad temporal)**

- **Tasa de aciertos:**

$$f_a = \frac{\text{nº aciertos}}{\text{nº accesos}} \cdot 100\%$$

- **Tasa de fallos:**

$$f_f = \frac{\text{nº fallos}}{\text{nº accesos}} \cdot 100\%$$

- **Relación:**

$$f_a = (1 - f_f)$$

- **Tiempo de acierto**

$$t_a$$

- **Tiempo de fallo**
  - ➔ **Se suele dar en función del tiempo de acierto**

$$t_f = t_a + p_f$$

  - ➔ La **penalización** p_f depende de:
    - Algoritmo de reemplazo
    - Latencia de memoria principal
    - Tamaño del bloque
    - Política de escritura

- **Tiempo medio de acceso**

$$t_m = t_a \cdot f_a + t_f \cdot f_f$$

$$t_m = t_a \cdot (1 - f_f) + (t_a + p_f) \cdot f_f$$

$$t_m = t_a + p_f \cdot f_f$$

- **Tiempo de programa**

$$t_p = r \cdot (CPI_{ejecución} + accesos\,mem_{instrucción} \cdot f_f \cdot p_f) \cdot periodo$$

$$fallos_{instrucción} = accesos\,mem_{instrucción} \cdot f_f$$

$$t_p = r \cdot (CPI_{ejecución} + fallos_{instrucción} \cdot p_f) \cdot periodo$$

$$t_p = r \cdot (CPI_{ejecución} + fallos_{instrucción} \cdot p_f) \cdot periodo$$

- **Si el CPI es pequeño el impacto de los fallos de caché representa un término importante**

- **Si el periodo (de CPU) es pequeño la penalización en ciclos se hace grande**

periodo CPU << periodo MEM

$$t_p = r \cdot (CPI_{ejecución} + fallos_{instrucción} \cdot p_f) \cdot periodo$$

- **Supongamos que el CPI de ejecución es 1 (muy buen dato)**
- **Supongamos que los fallos por instrucción son 1% (muy buen dato)**
- **En una máquina real la penalización va desde unas decenas de ciclos a algo más del millar; esto significa que el producto fallos x penalización puede ser hasta 10 veces el CPI de ejecución !!!!**

- **Enfoque alternativo**
  - ➔ **$f_a = h$ $f_f = m$ m = 1 – h**
  - ➔ **Tomamos en consideración los 2 niveles adyacentes de la jerarquía**
  - ➔ **Al tiempo de acceso a un nivel de la jerarquía de memoria hay que sumarle la penalización por fallos que viene representada por el tiempo de acceso al nivel inmediatamente inferior multiplicado por la probabilidad del fallo**

$$\bar{t} = t_{nivel1} + (1 - h) \cdot t_{nivel2}$$

- **Desagregación (I)**
  - ➔ **Tiempo de acceso**
    - ➔ Lecturas y escrituras
      - Al calcular el tiempo medio de acceso (t_a) a memoria hay que distinguir entre lectura y escritura debido a las posibles diferencias debidas a la política de escritura utilizada

$$\bar{t}_a = p_l \cdot \bar{t}_l + p_e \cdot \bar{t}_e$$

    - ➔ donde:
      - p_l es la probabilidad de que un acceso sea de lectura
      - p_e es la probabilidad de que un acceso sea de escritura
      - t_l es el tiempo medio que consume un acceso de lectura.
      - t_e es el tiempo medio que consume un acceso de escritura **(si la política de escritura es escritura directa el tiempo medio de escritura corresponde al de memoria principal)**

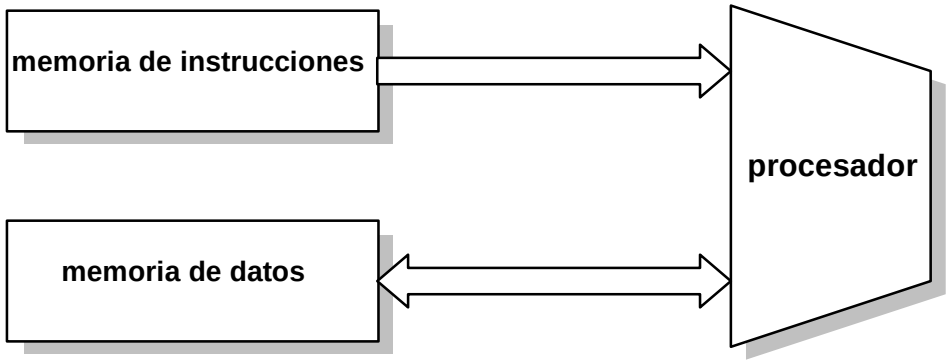
- **Desagregación (II)**
  - ➔ **Tiempo de acceso**
    - ➔ Flujo de instrucciones y flujo de datos

$$\bar{t}_a = \bar{t}_{a\,instrucciones} + \bar{t}_{a\,datos} = (p_l \cdot \bar{t}_l)_{instrucciones} + (p_l \cdot \bar{t}_l + p_e \cdot \bar{t}_e)_{datos}$$

    - ➔ Asumimos que el flujo de instrucciones no realiza escrituras

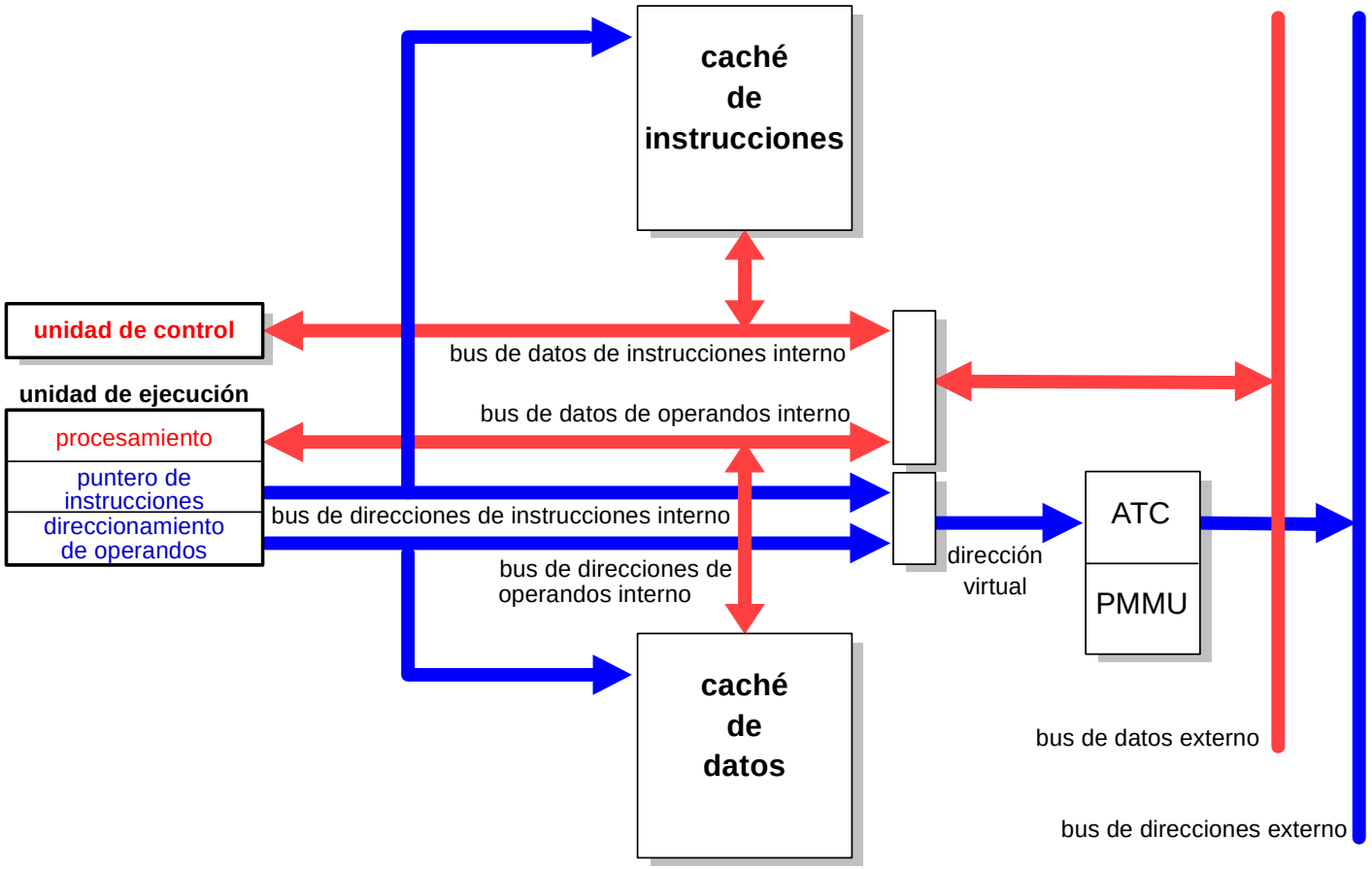
- **Organización del primer nivel de la jerarquía**
  - ➔ **El nivel de la jerarquía de memoria más cercano al procesador pueden distinguir entre flujo de instrucciones y flujo de datos al emitir una dirección**
  - ➔ **Esto permite construir dos tipos de organizaciones**
    - ➔ Caché separada para instrucciones y datos; o
    - ➔ Caché unificada

- **Caché separada en datos e instrucciones**
  - ➔ **Se conoce como arquitectura *Harvard***

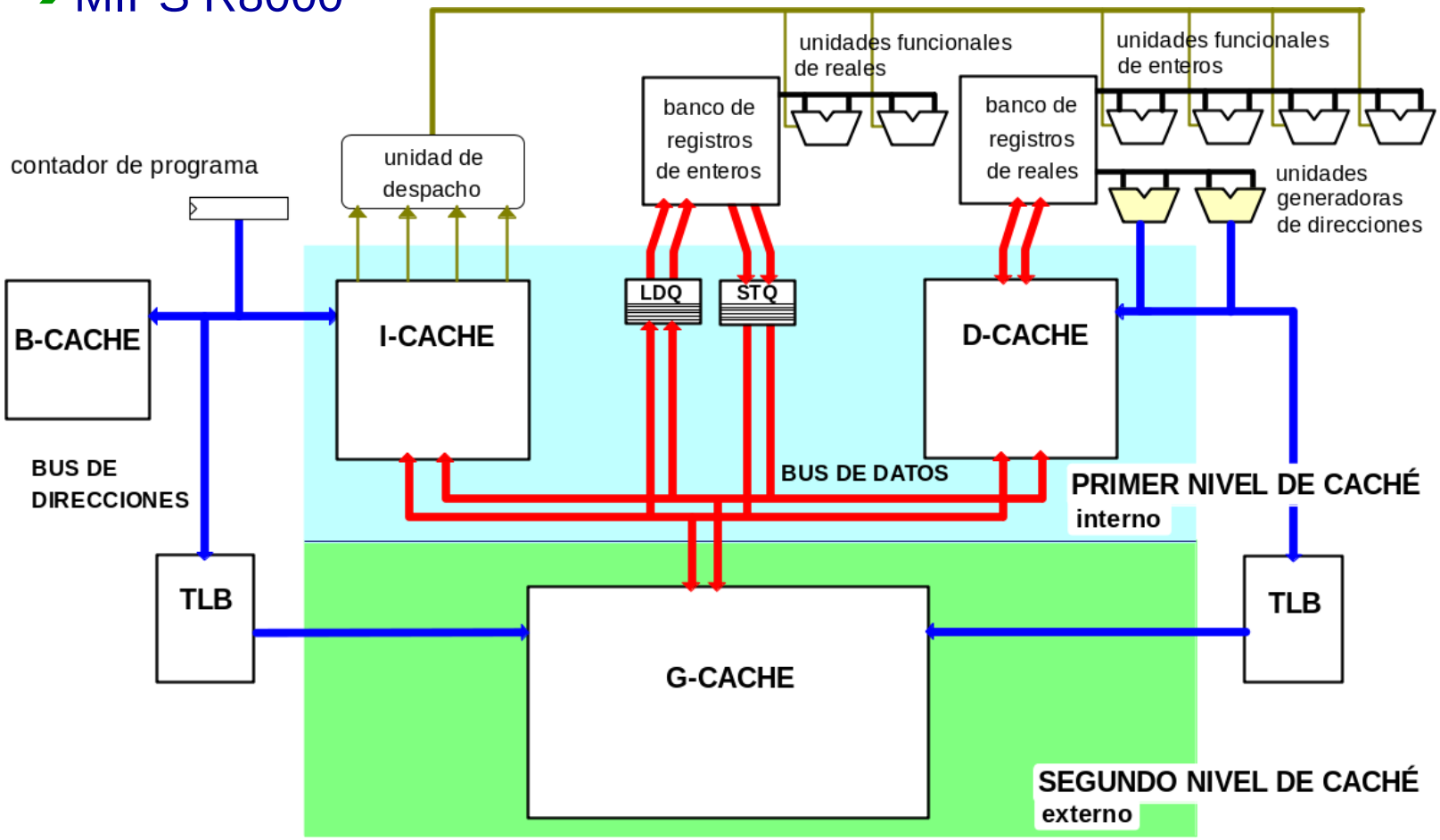


  - ➔ **Aumenta el ancho de banda**
  - ➔ **Ya que el comportamiento en flujo de instrucciones y datos es diferente se puede diseñar la caché optimizada para cada tipo de flujo**

- **Ejemplos de arquitectura *Harvard***
  - ➔ **MC68030 de Motorola**



- **Ejemplos de arquitectura *Harvard***
  - ➔ MIPS R8000



- **Fuentes de fallos de caché:**
  - **Las tres 'C'**
    - Forzosos (*Compulsory*) ➔ el primer acceso a un bloque no está en caché y debe ser traído (fallos de arranque frío o de primera referencia)
    - Capacidad (*Capacity*) ➔ si la caché no puede contener todos los bloques requeridos por el programa
    - Conflicto (*Conflict*) ➔ en asociativas por conjuntos y correspondencia directa (asociativa por conjuntos de 1 vía), si se requieren más bloques de los que caben en un conjunto (fallos de colisión)