

Tipos de Memoria de Semiconductores

■ **Memoria Estática** (SRAM, Static RAM): Cada celda de memoria equivale a 1 biestable (6-8 transistores).

En comparación con las DRAM:

- son **rápidas**,
- tienen un **alto consumo**,
- **pequeñas** (poca capacidad) y
- **caras**.

→ Memoria Cache

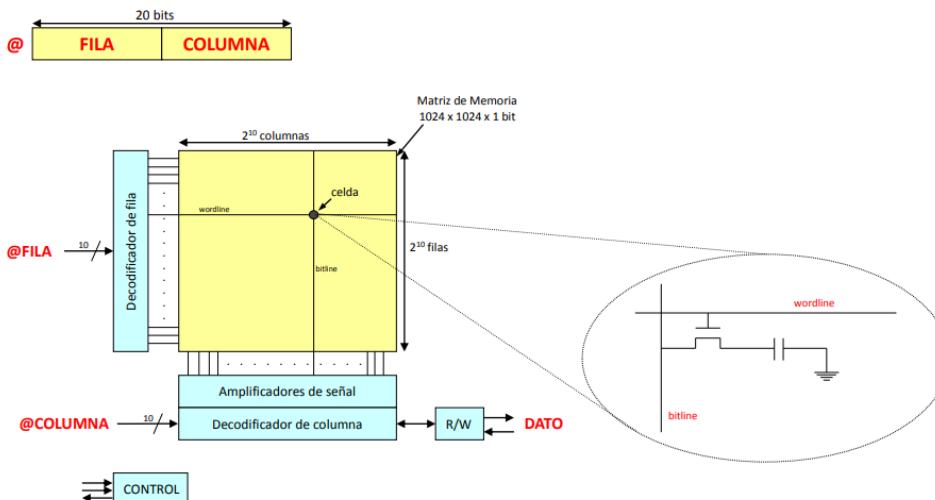
■ **Memoria Dinámica** (DRAM, Dynamic RAM): Cada celda se comporta como un condensador (1-1.x transistores).

En comparación con las SRAM:

- son **lentas**,
- tienen un **bajo consumo**,
- **grandes** (mucha capacidad) y
- **baratas**.
- Problema del refresco.

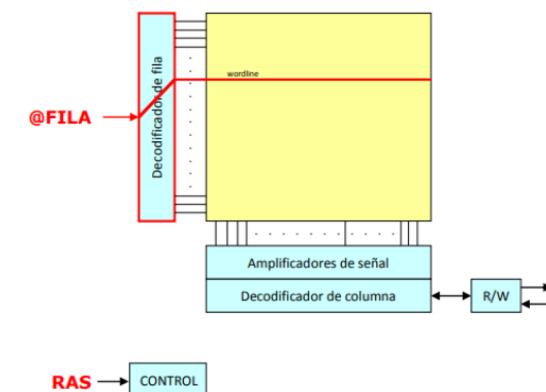
→ Memoria Principal

Estructura interna de una DRAM



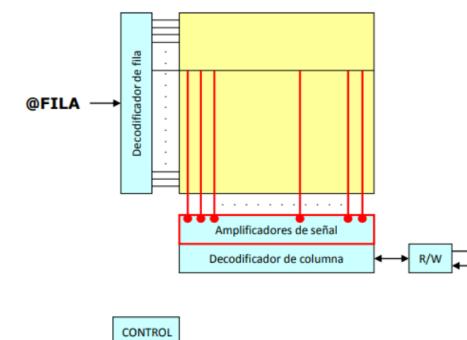
■ Una operación de lectura:

1) Decodificar @FILA, se activa la señal **wordline**

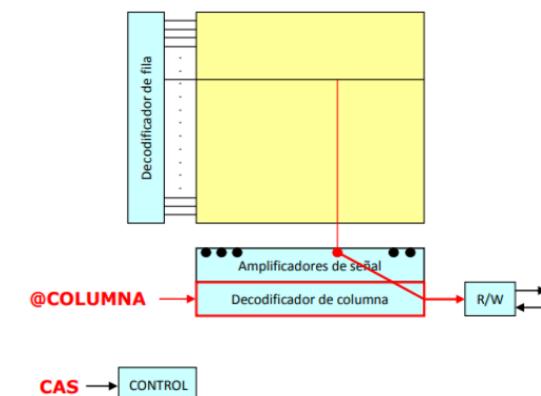


2) Se accede a todas las celdas de la fila,

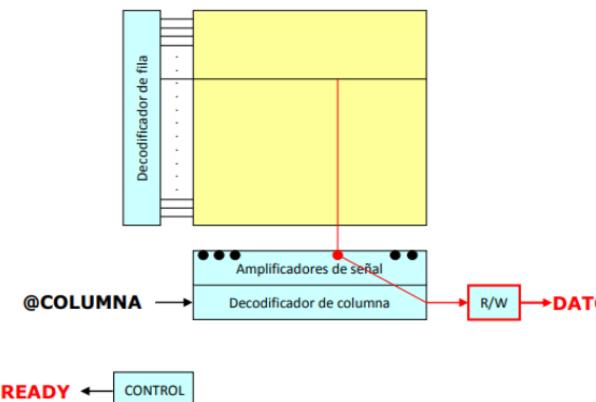
los datos de toda la fila se envían a los amplificadores de señal y se recupera la tensión (el dato está en un condensador que se va descargando poco a poco).



3) Decodificar @COLUMNNA, se selecciona una **bitline** y se envía el dato al buffer R/W.

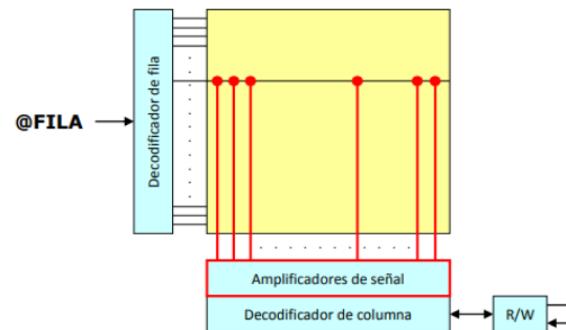


4) Se envía el dato al exterior desde el buffer R/W.



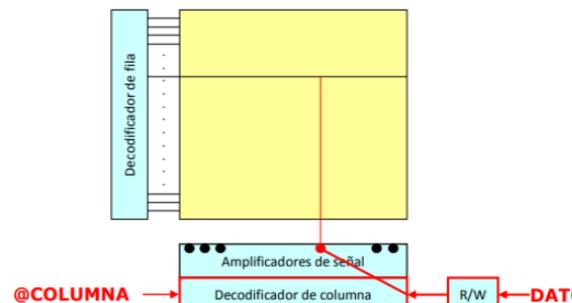
5) La lectura es destructiva,

hay que reescribir la celda (y toda la fila) para recuperar el valor original y precargar los bitlines para el siguiente acceso a memoria.



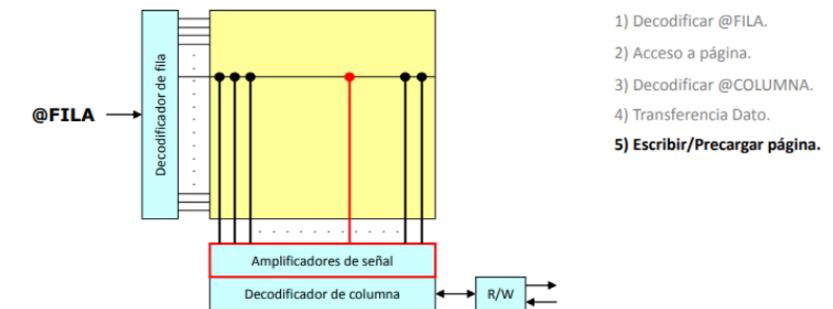
■ Una operación de escritura:

3 y 4) Prácticamente igual,
la única diferencia es que la celda se reescribe con el dato que entra por el buffer R/W.



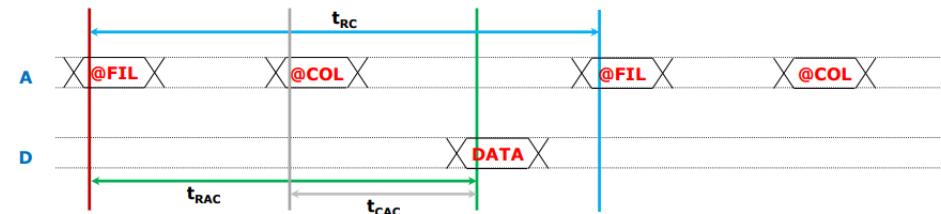
- 1) Decodificar @FILA.
- 2) Acceso a página.
- 3) Decodificar @COLUMNNA.
- 4) Transferencia Dato.**

5) Hay que reescribir la celda con el nuevo valor (y el resto de la fila con el valor original).



- 1) Decodificar @FILA.
- 2) Acceso a página.
- 3) Decodificar @COLUMNNA.
- 4) Transferencia Dato.
- 5) Escribir/Precargar página.**

Valores principales en una lectura a DRAM



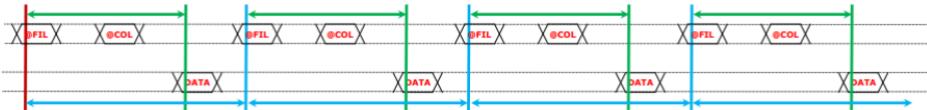
- 1. Tiempo de acceso (t_{RAC}):**
retardo máximo desde que se suministra la dirección de fila hasta que se obtiene el dato
→ **latencia de memoria**.
- 2. Tiempo de ciclo (t_{RC}):**
intervalo de tiempo mínimo entre dos accesos consecutivos a memoria
→ **ancho de banda**.
- 3. Tiempo de acceso a columna (t_{CAC}):**
retardo máximo desde que se suministra la dirección de columna hasta que se obtiene el dato.

cost negar una línea caché : $T = t_{RC} \cdot 4 = 60\text{ns} \cdot 4 = 240\text{ns}$

Ample banda = $\frac{32\text{B}}{240\text{ns}} = 133.3 \text{ MB/s}$

■ Leer una línea de cache: 4 lecturas de memoria principal

- Tiempo de acceso: 50ns
- Tiempo de ciclo: 60ns



■ Coste de leer una línea de cache:

- Tiempo total: Tiempo de ciclo * 4: 240 ns
- Ancho de banda: $32B / 240ns = 133,33 \text{ MB/s}$

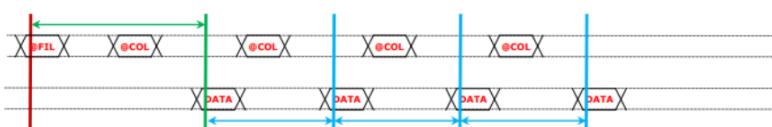
→ Están en la misma fila del array de memoria (en posiciones consecutivas)

■ Idea Fundamental: una vez accedida la fila, se puede acceder a varias columnas simplemente cambiando la @COL.

→ Aprovechamos la **localidad espacial**

■ Leer una línea de cache: 4 lecturas de memoria principal

- Tiempo de acceso: 50ns
- Latencia de columna: 30ns



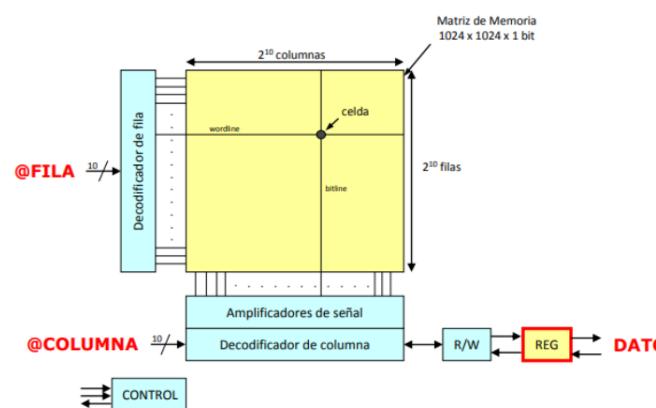
■ Coste de leer una línea de cache:

- Tiempo total: Tiempo de acceso + Latencia de columna * 3 : 140 ns
- Ancho de banda: $32B / 140ns = 228,57 \text{ MB/s}$
- Ancho de banda de pico: $8B / 30ns = 266,67 \text{ MB/s}$

■ Problema: hay que esperar a que el dato sea leído antes de enviar la nueva @COL.

■ Solución: se añade un registro en la salida de datos

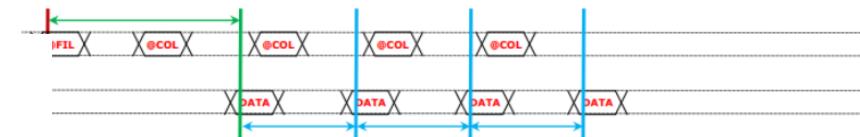
→ se puede solapar el acceso a los datos con el envío de la nueva @COL



■ Idea Fundamental: Como el dato está almacenado en un registro, antes de acabar el envío del dato ya podemos enviar la @COL.

■ Leer una línea de cache: 4 lecturas de memoria principal

- Tiempo de acceso: 50ns
- Latencia de columna: 20ns

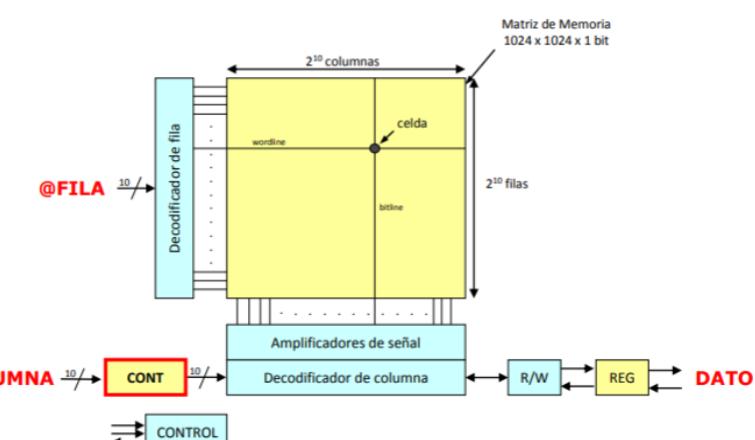


■ Coste de leer una línea de cache:

- Tiempo total: Tiempo de acceso + Latencia de columna * 3 : 110 ns
- Ancho de banda: $32B / 110ns = 290,91 \text{ MB/s}$
- Ancho de banda de pico: $8B / 20ns = 400 \text{ MB/s}$

■ Mejora:

Añadir un contador para que genere de forma automática @COL+1, @COL+2 y @COL+3.

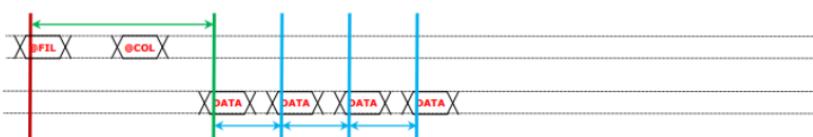


Idea Fundamental:

Sólo hay que enviar la @COL una vez, del resto se encarga la propia memoria.

Leer una línea de cache: 4 lecturas de memoria principal

- Tiempo de acceso: 50ns
- Latencia de columna: 15ns



Coste de leer una línea de cache:

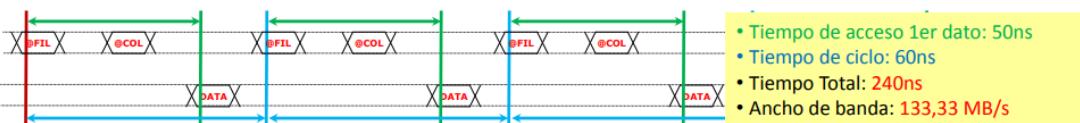
- Tiempo total: Tiempo de acceso + Latencia de columna * 3 : 95 ns
- Ancho de banda: $32B / 95ns = 336,84 \text{ MB/s}$
- Ancho de banda de pico: $8B / 15\text{ns} = 533,33 \text{ MB/s}$

La latencia del primer dato se mantiene.

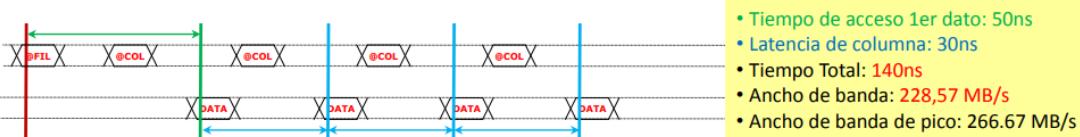
Mejora la latencia de los siguientes datos simplemente añadiendo un contador.

En este punto se llega al límite de rendimiento de una memoria ASÍNCRONA.

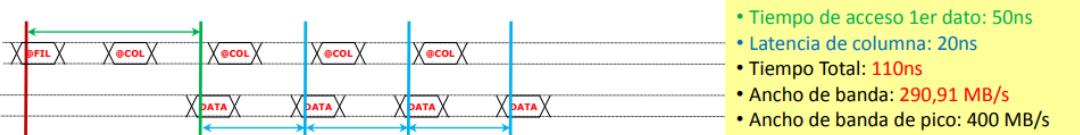
Resumen de las optimizaciones



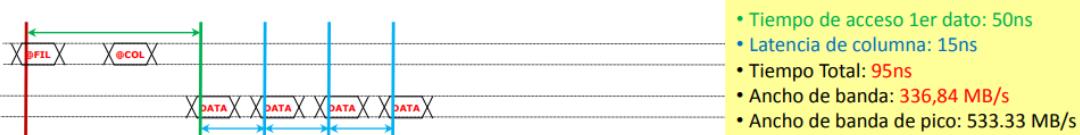
- Tiempo de acceso 1er dato: 50ns
- Tiempo de ciclo: 60ns
- Tiempo Total: 240ns
- Ancho de banda: 133,33 MB/s



- Tiempo de acceso 1er dato: 50ns
- Latencia de columna: 30ns
- Tiempo Total: 140ns
- Ancho de banda: 228,57 MB/s
- Ancho de banda de pico: 266.67 MB/s



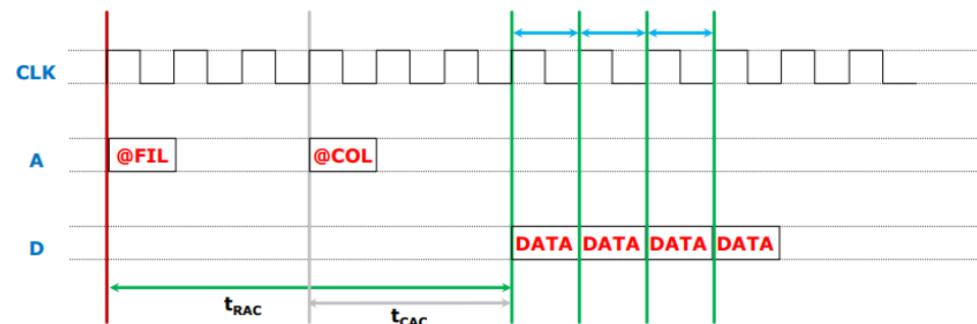
- Tiempo de acceso 1er dato: 50ns
- Latencia de columna: 20ns
- Tiempo Total: 110ns
- Ancho de banda: 290,91 MB/s
- Ancho de banda de pico: 400 MB/s



- Tiempo de acceso 1er dato: 50ns
- Latencia de columna: 15ns
- Tiempo Total: 95ns
- Ancho de banda: 336,84 MB/s
- Ancho de banda de pico: 533,33 MB/s

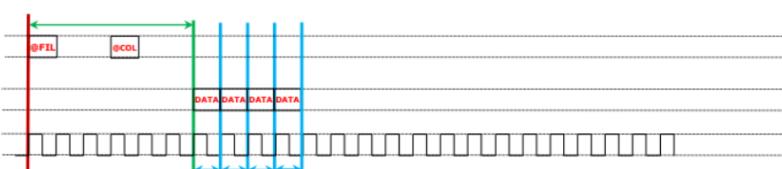
Solución: Memoria SÍNCRONA

- El protocolo se simplifica, se reducen los problemas de ruido
- Se puede aumentar la frecuencia de funcionamiento



Leer una línea de cache: 4 lecturas de memoria principal

- Tiempo de acceso: 48ns (6 ciclos)
- Tiempo de transferencia: 8ns (1 ciclo)

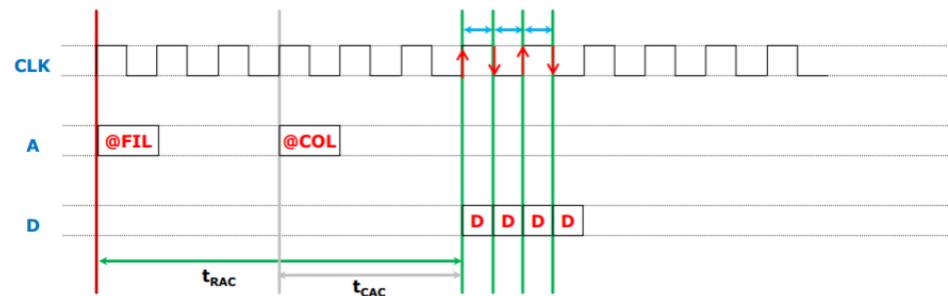


Coste de leer una línea de cache:

- Tiempo total: Tiempo de acceso + Tiempo de transferencia * 4 = 80ns
- Ancho de banda: $32B / 80ns = 400 \text{ MB/s}$
- Ancho de banda de pico: $8B / 8\text{ns} = 1 \text{ GB/s}$

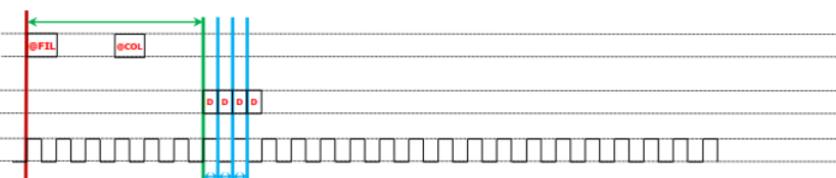
DDR SDRAM

DDR SDRAM: Double Data Rate Synchronous Dynamic Random-Access Memory



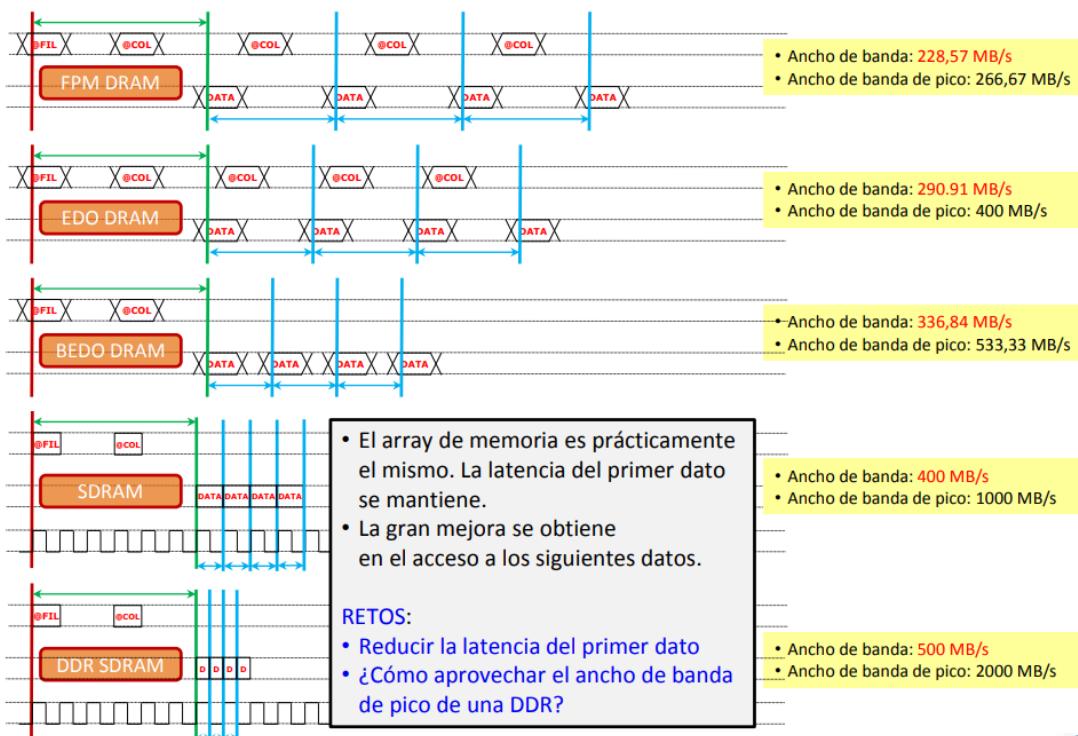
■ Leer una línea de cache: 4 lecturas de memoria principal

- Tiempo de acceso: 48ns (6 ciclos)
- Tiempo de transferencia : 4ns (1/2 ciclo)



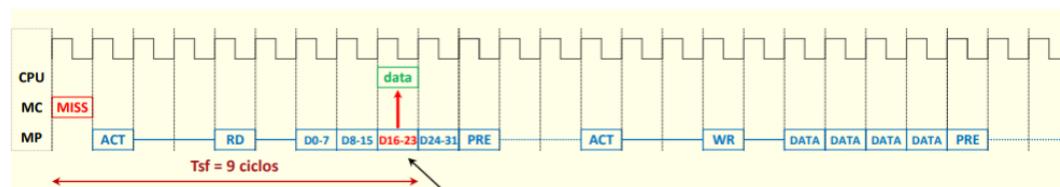
■ Coste de leer una línea de cache:

- Tiempo total: Tiempo de acceso + Tiempo de transferencia * 4 = 64ns
- Ancho de banda: $32B / 64 \text{ ns} = 500 \text{ MB/s}$
- Ancho de banda de pico: $8B / 4\text{ns} = 2 \text{ GB/s}$



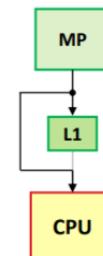
■ Continuación Anticipada (*Early Restart*):

en cuanto llega el dato que provoca el fallo, se envía al procesador

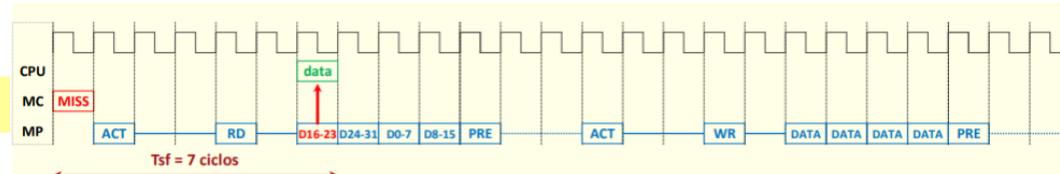


- Copy Back + Write Allocate
- 32 bytes por línea
- **MISS con REEMPLAZO BLOQUE SUCIO**
- Fallo en el byte 16
- ACT: 3 ciclos
- WR, RD: 2 ciclos
- PRE: 3 ciclos
- Transferencia: 8B por ciclo

Cuando llega a la MC el dato que provoca el fallo, se envía simultáneamente a la CPU.



■ Transferencia en desorden: se envía en primer lugar la palabra que ha provocado el fallo.



- Copy Back + Write Allocate
- 32 bytes por línea
- **MISS con REEMPLAZO BLOQUE SUCIO**
- Fallo en el byte 16
- ACT: 3 ciclos
- WR, RD: 2 ciclos
- PRE: 3 ciclos
- Transferencia: 8B por ciclo

En todos los casos, y para que estos mecanismos sean efectivos, cuando se pasa el dato al procesador y mientras se acaba de servir el fallo, la MC ha de ser capaz de recibir nuevos accesos (Cache no bloqueante).

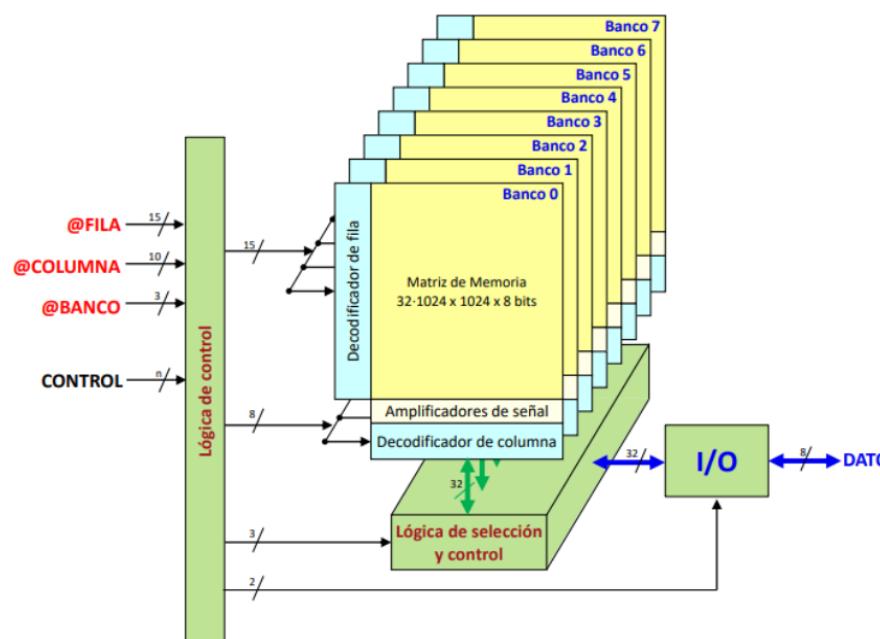
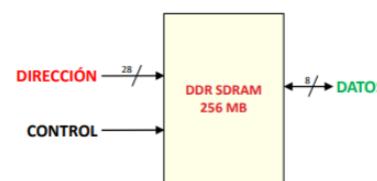
Estructura actual de una DRAM

■ Una característica fundamental de las DRAM actuales es que están divididas en **bancos**.

Los bancos permiten:

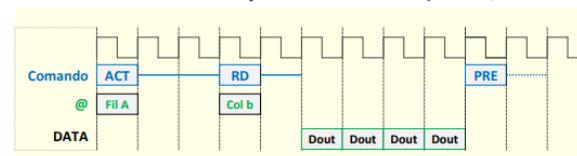
- Realizar accesos concurrentes a diferentes bancos
- Ocultar la precarga
- Ocultar el refresco
- Tener arrays de memoria más pequeños:
 - ✓ Tiempo de acceso ↓
 - ✓ Consumo de energía ↓

- Por ejemplo, una DDR3 de 256 MB tiene:
- 8 bancos de 32 MB
 - Cada banco tiene 32 K filas por 1K columnas de 1 byte.

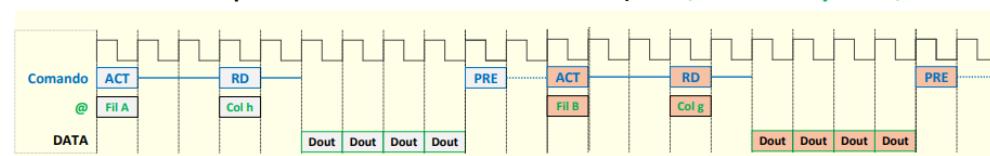


Accesos a una DRAM con 1 banco

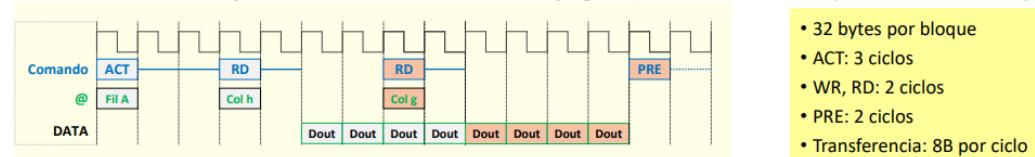
Lectura de un bloque de memoria (Fila A, columna b)



Lectura de 2 bloques de memoria en Filas diferentes (Fila A, columna h y Fila B, columna g)



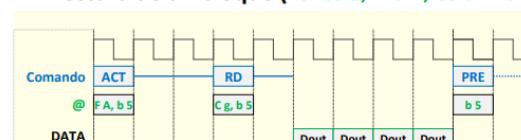
Lectura de 2 bloques de memoria en la misma página (Fila A, columna h y Fila A, columna g)



- 32 bytes por bloque
- ACT: 3 ciclos
- WR, RD: 2 ciclos
- PRE: 2 ciclos
- Transferencia: 8B por ciclo

Accesos a una DRAM con múltiples bancos

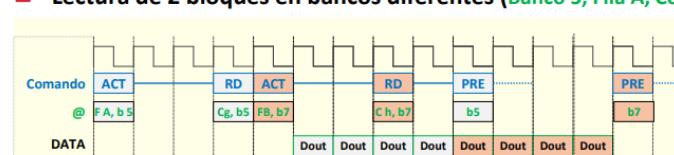
Lectura de un bloque (Banco 5, Fila A, Columna g)



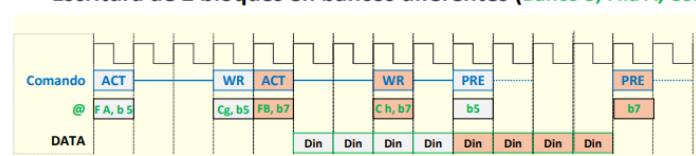
En una DRAM real existen bastantes restricciones que no hemos tenido en cuenta, p.e.:

- Entre una lectura y una escritura hay que esperar N ciclos.
- Entre 2 ACT consecutivos a bancos diferentes han de pasar M ciclos.

Lectura de 2 bloques en bancos diferentes (Banco 5, Fila A, Columna g y Banco 7, Fila B, columna g)



Escritura de 2 bloques en bancos diferentes (Banco 5, Fila A, Columna g y Banco 7, Fila B, columna g)

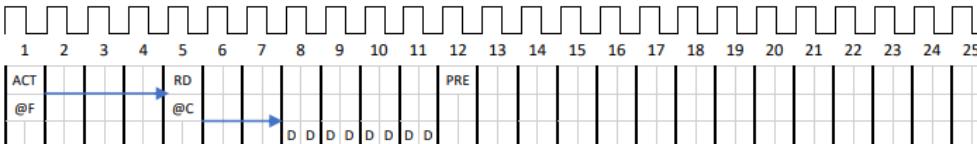


- 32 bytes por bloque
- ACT: 3 ciclos
- WR, RD: 2 ciclos
- PRE: 2 ciclos
- Transferencia: 8B por ciclo

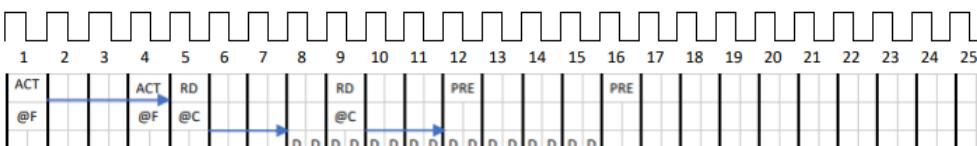
Acceso a una DRAM

Ejemplo con un DIMM de 8 chips de DDR-SDRAM (Double Data Rate Synchronous DRAM). El DIMM está configurado para leer y escribir ráfagas de 64 bytes. Latencia de fila de 4 ciclos, latencia de columna de 2 ciclos y latencia de precarga de 1 ciclo. (Cada medio ciclo se leen 8 bytes)

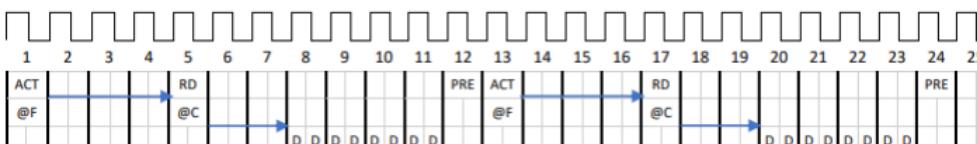
Lectura de un bloque de 64 bytes de la DDR



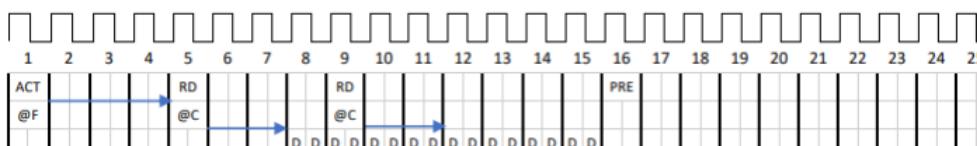
Lectura de dos bloques ubicados en distintos bancos



Lectura de dos bloques ubicados en el mismo banco pero en páginas distintas



Lectura de dos bloques ubicados en el mismo banco y en la misma página



■ Nuestra memoria tiene una capacidad de 8GB (33 bits de dirección)

■ En nuestra Memoria tenemos:

- Canales (2): 1 bit para el canal
- DIMMs (2 por canal): 1 bit para el DIMM
- Chips (8 por DIMM): 3 bits para el chip
- Bancos (8 por chip): 3 bits para el banco
- Filas (32·1024 por Banco): 15 bits para la fila
- Columnas (1024 por Fila): 10 bits para la columna

■ Existen muchas posibilidades válidas:

Canal	DIMM	Banco	Fila	Columna	Chip
Canal	DIMM		Fila	Banco	Columna
Canal			Fila	DIMM	Banco
Canal			Fila		Columna

¡ El rendimiento del Sistema de Memoria está muy influenciado por esta decisión !

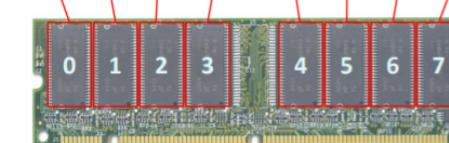
■ Es imprescindible que los bits bajos de la dirección seleccionen el chip dentro del DIMM

→ MEMORIA ENTRELAZADA

■ Por ejemplo, las direcciones en el DIMM 0 del Canal 0 se distribuyen así:

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
2 ³¹ -8	2 ³¹ -7	2 ³¹ -6	2 ³¹ -5	2 ³¹ -4	2 ³¹ -3	2 ³¹ -2	2 ³¹ -1

La memoria entrelazada permite realizar accesos de 64 bits de ancho.



Canal, DIMM, Banco, Fila, Columna Chip

Posiciones consecutivas de memoria se encuentran en el mismo «canal, DIMM, banco, fila y columna» de chips diferentes.

■ Todos los accesos a Memoria Principal leen (o escriben) bloques de memoria.

bloque byte en el bloque

■ Las DRAM actuales permiten leer (o escribir) de forma muy eficiente ráfagas de datos.

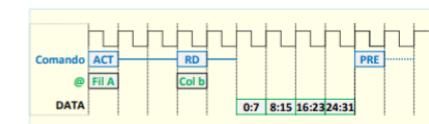
■ Para que esas ráfagas coincidan con bloques de memoria es imprescindible que el campo correspondiente al byte dentro del bloque corresponda a los bits que seleccionan chip y columna_L.

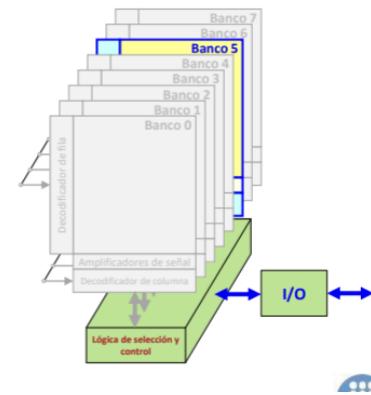
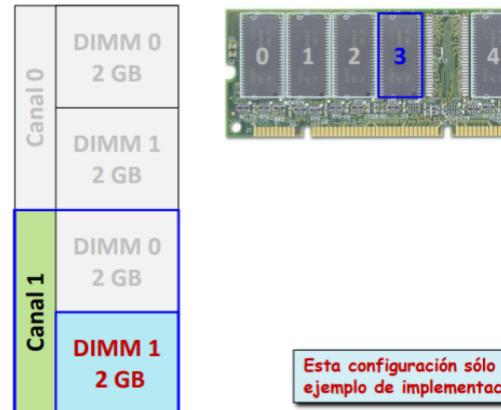
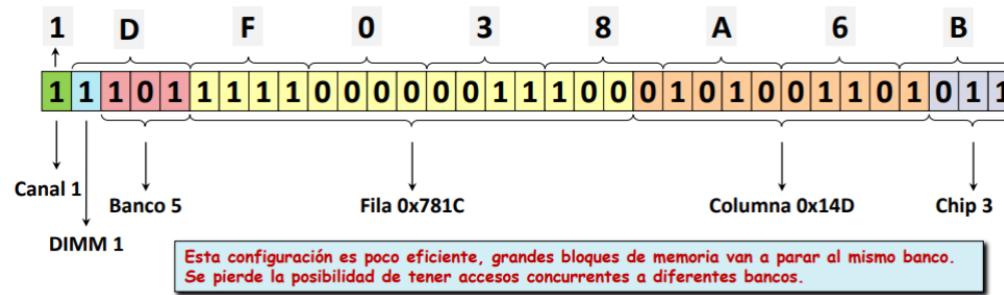
#bloque=(Canal, DIMM, Banco, Fila, Columna_H) Col_L Chip

■ Por ejemplo, con bloques de 32 bytes, el bloque 0 se encuentra en el canal 0, DIMM 0:

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
2 ³¹ -8	2 ³¹ -7	2 ³¹ -6	2 ³¹ -5	2 ³¹ -4	2 ³¹ -3	2 ³¹ -2	2 ³¹ -1

Para leer un bloque de 32 bytes necesitamos una ráfaga de 4 accesos.



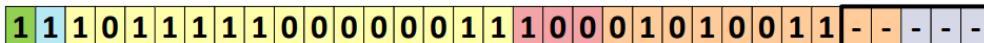


Lectura línea

- La transmisión de datos entre MP y MC se hace por bloques completos
- Ejemplo: 32 bytes por bloque, acceso provocado por un fallo en la dirección 0x1DF038A6B



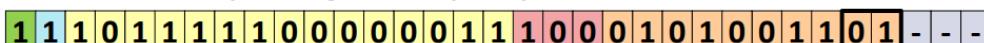
- Hay que leer todos los bytes del bloque:



- Si transferimos el bloque en orden hay que iniciar el acceso con la siguiente dirección:



- Si transferimos en primer lugar el dato que ha provocado el fallo, la dirección es:



Problema del refresco en una DRAM

- Cada celda de una DRAM equivale a un condensador, como tal, va perdiendo carga de forma exponencial:
 - Corriente de fuga.
 - La lectura es destrutiva.

- Para evitar la perdida de información almacenada en las celdas, la carga debe regenerarse periódicamente

→ REFRESCO

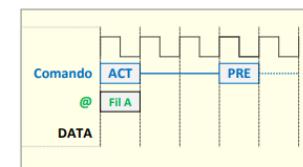
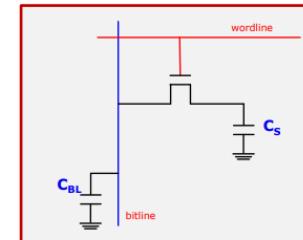
- El refresco hay que aplicarlo a todas las celdas de la memoria. Se aplica fila a fila.

- El refresco de la fila A se puede hacer con un ACT y una PRE.

- Existen comandos especiales para hacer el refresco.

- Refrescan todos los bancos a la vez.
- La propia memoria controla qué fila hay que refrescar.

- Actualmente el refresco es una operación estandarizada establecida por el JEDEC (Junction Electronic Devices Engineering Council)
 - Periodo de refresco de 64ms.
 - Si tenemos 32-1024 filas, en media, hay que lanzar una operación de refresco cada 2μs.



Detección y Corrección de Errores

- Al acceder a memoria se pueden producir errores:

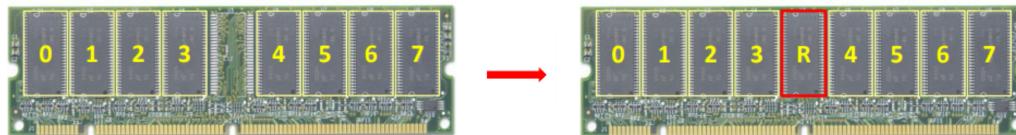


- Estos errores se deben a diversas causas:

- Interferencias eléctricas o magnéticas (**transitorios**)
- Problemas de transmisión de datos (**transitorios**)
- Problemas hardware (**permanentes**), como en cualquier otro chip y por las mismas causas.

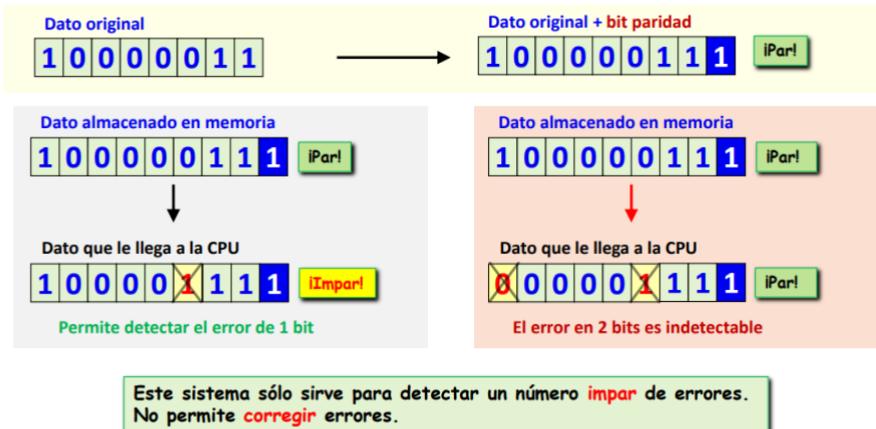
- Los errores permanentes sólo se resuelven sustituyendo el elemento dañado.
- Los fabricantes de memoria diseñan sistemas que permiten detectar y en algunos casos corregir los errores transitorios.

→ AÑADIR INFORMACIÓN REDUNDANTE



Bit de Paridad

- El sistema más simple es añadir 1 bit, de tal forma que cada porción de información tenga SIEMPRE un número impar (o par) de 1's.



Código Hamming (7, 3): 4 bits de datos y 3 redundantes

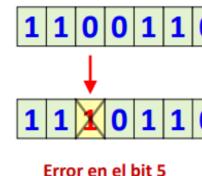
- Bits redundantes: 1, 2, 4
- Bits de datos: 3, 5, 6 y 7

7	6	5	4	3	2	1	
111	110	101	100	011	010	001	
b3	b2	b1	p2	b0	p1	p0	Palabra de 7 bits
111	-	101	-	011	-	001	
b3	b1	b0	p0				p0 = b3 xor b1 xor b0
111	110	-	-	011	010	-	p1 = b3 xor b2 xor b0
b3	b2	-	-	b0	p1	-	
111	110	101	100	-	-	-	p2 = b3 xor b2 xor b1
b3	b2	b1	p2	-	-	-	

7	6	5	4	3	2	1	
1	1	0	p2	1	p1	p0	Palabra de 7 bits
1	-	0	-	1	-	0	$p0 = 1 \text{ xor } 0 \text{ xor } 1 = 0$
1	1	-	-	1	1	-	$p1 = 1 \text{ xor } 1 \text{ xor } 1 = 1$
1	1	0	0	-	-	-	$p2 = 1 \text{ xor } 1 \text{ xor } 0 = 0$

Posibles implementaciones		
Bits Paridad	Bits Datos	total
3	4	7
4	11	15
5	26	31
6	57	63
7	120	127
m	$2^m - m - 1$	$2^m - 1$

- Permite corregir un error en 1 bit.



b3	b2	b1	p2	b0	p1	p0	
7	6	5	4	3	2	1	Test Hamming
1	1	1	0	1	1	0	
1	-	1	-	1	-	0	$T0 = p0 \text{ xor } b3 \text{ xor } b1 \text{ xor } b0 = 0 \wedge 1 \wedge 1 \wedge 1 = 1$
1	1	-	-	1	1	-	$T1 = p1 \text{ xor } b3 \text{ xor } b2 \text{ xor } b0 = 1 \wedge 1 \wedge 1 \wedge 0 = 0$
1	1	1	0	-	-	-	$T2 = p2 \text{ xor } b3 \text{ xor } b2 \text{ xor } b1 = 0 \wedge 1 \wedge 1 \wedge 1 = 1$

El test Hamming da **T2T1T0=101**, indicando que el error está en el bit 5.

Si **T2T1T0=000** no ha habido ningún error.

Código Hamming extendido (7, 3) + 1 bit de paridad.

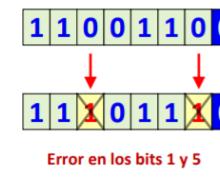
- El código Hamming no funciona cuando se produce un error en 2 bits.



b3	b2	b1	p2	b0	p1	p0	
7	6	5	4	3	2	1	Test Hamming
1	1	1	0	1	1	1	
1	-	1	-	1	-	1	$T0 = p0 \text{ xor } b3 \text{ xor } b1 \text{ xor } b0 = 1 \wedge 1 \wedge 1 \wedge 0 = 0$
1	1	-	-	1	1	-	$T1 = p1 \text{ xor } b3 \text{ xor } b2 \text{ xor } b0 = 1 \wedge 1 \wedge 1 \wedge 0 = 0$
1	1	1	0	-	-	-	$T2 = p2 \text{ xor } b3 \text{ xor } b2 \text{ xor } b1 = 0 \wedge 1 \wedge 1 \wedge 1 = 1$

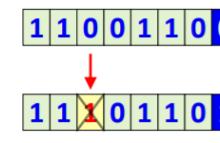
El test Hamming da **T2T1T0=100**, indicando que hay un error en el bit 4 (¿?).

- Añadiendo un bit de paridad es posible detectar un error en 2 bits o corregir un error de 1 bit.



No hay error de paridad.
El test Hamming da **T2T1T0=100**
Esto implica que se ha producido un error en 2 bits.

No se puede recuperar.



Error de paridad.
El test Hamming da **T2T1T0=101**
Esto implica que se ha producido un error en el bit 5.

Se puede recuperar.

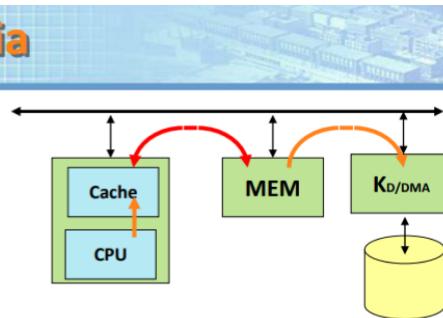
Memoria ECC (Error Correction Code)

- La memoria que incluye estos mecanismos se utiliza exclusivamente en servidores y estaciones de trabajo de gama alta.
- Basado en códigos de Hamming. A cada porción de datos se añade información redundante que permite detectar y corregir errores.
- El código ECC más utilizado en memorias es el SEC-DED (Single Error Correction – Double Error Detection). Permite corregir 1 error o detectar 2 en la misma porción de datos.
- En DIMMs de memoria se suele utilizar un código extendido (71, 64) + bit de paridad. 72 bits totales, con 64 bits de datos y 8 redundantes. Perfecto para utilizar en un DIMM con 9 chips de 8 bits de ancho.

DMA y Jerarquía de Memoria

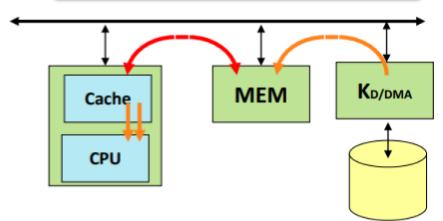
Problema de COHERENCIA 1

1. La CPU escribe un dato X en la cache.
2. Una vez escrito, el dato permanece en cache hasta que la línea sea substituida.
3. Se programa una **escritura en disco** del bloque de memoria donde estaba X (X tiene un valor distinto en Memoria Principal que en Memoria Cache con lo que escribimos un **VALOR INCORRECTO**).



Existen diversas soluciones para este problema:

- Usar una cache **Write Through**
- Que el K_{DMA} sólo pueda acceder a zonas de MP **NO-CACHEABLES**
- Vaciar la cache cada vez que se lanza una operación con el K_{DMA}



Existen diversas soluciones para este problema:

- Que el K_{DMA} sólo pueda acceder a zonas de MP **NO-CACHEABLES**
- Vaciar la cache cada vez que se lanza una operación con el K_{DMA}

Problema de COHERENCIA 2

1. La CPU lee un dato X y lo trae a la cache.
2. Una vez leído el dato, el dato sigue permaneciendo en cache hasta que la línea sea substituida.
3. Se programa una **lectura de disco** y se lee un bloque de datos que incluye la posición donde estaba X (X toma un nuevo valor en Memoria Principal).
4. La CPU lee X de nuevo, pero obtiene el valor almacenado en la cache y **NO el VALOR CORRECTO** que está en Memoria Principal.
 - Que la cache sea write through **NO SOLUCIONA** el problema.

SSD (Solid State Drive)

- Dispositivos de almacenamiento que utilizan memoria no volátil (FLASH) en vez de elementos mecánicos y soporte magnético.
- Utilizan NAND Flash
 - Acceso aleatorio a bloques y secuencial dentro de los mismos.
 - Número limitado de escrituras y borrados (entre 10^4 y 10^6).
 - Menor fiabilidad que las NOR Flash (usan ECC para aumentarla)

VENTAJAS

- Arranque rápido (no hay elementos mecánicos).
- Gran velocidad de lectura y escritura.
- Baja latencia de lectura y escritura.
- Menor consumo de energía y producción de calor.
- Sin ruido.
- Mejor MTTF que un disco duro.
- Seguridad: borrado seguro e irrecuperable.
- Rendimiento: El tiempo de "búsqueda" es constante.
- El rendimiento no se deteriora mientras el medio se llena. No necesita defragmentación
- Menor peso y tamaño que un disco duro tradicional de similar capacidad.

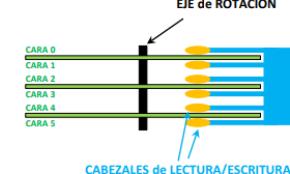
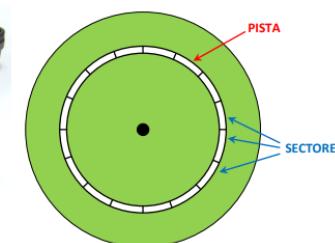
INCONVENIENTES

- Alto Precio.
- Información no recuperable después de un fallo físico.
- Baja Capacidad.
- Menor tiempo de vida total. Bajo número de ciclos de lectura y escritura.

Disco Duro

Dispositivo de Almacenamiento Masivo

- Gran Capacidad
- Memoria No volátil
- Tecnología Magnética
- Elemento Mecánico
- Organizado en:
 - Caras
 - Pistas (Cilindros)
 - Sectores
- Métodos de Acceso
 - **Modo CHS** (Cylinder Head Sector). Se accede a disco con especificaciones geométricas: Cilindro, Cara y Sector.
 - **Modo LBA** (Logical Block Addressing). Los sectores están numerados de 0 a N-1 (N sectores lógicos por disco). Se accede a disco utilizando 1 número de sector lógico. Requiere un mecanismo de traducción interno.



Rendimiento de un Disco Duro (parámetros)

- Average Seek Time: Coste en media de situar el cabezal en el cilindro al que se quiere acceder. Se mide en **milisegundos**.
- Latency: Coste en media de situar el cabezal sobre el sector al que se quiere acceder. Se mide en **milisegundos**. Depende de la velocidad de giro del disco duro (R.P.M.).
- R.P.M.: **Revoluciones por minuto**. En modo «normal» el disco está siempre girando. Valores típicos: 7.200 (PC), 4.200 (portátil), 10.000 y 15.000 (servidor).
- Average access time: Average Seek Time + Latency
- Transfer rate: Velocidad de transferencia, medida en **MB/s**. Velocidad típica: 100-300 MB/s.
- Cache: Memoria secundaria en donde se almacena temporalmente la información leída de disco (o pendiente de escribir, equivale a un buffer de escritura). Posibles estrategias de prefetch. Valores típicos: 8-64 **MB**.
- MTTF: Tiempo medio hasta fallo. Valores típicos: **1.000.000 – 1.600.000 horas**.
- Interfaz: Forma de comunicación entre el Disco Duro y el Computador.
Ejemplos actuales: **SATA, SCSI, ...**

Estructura de un sector

- La unidad mínima de acceso a disco es el **sector**.
- Un sector de disco tiene la siguiente estructura:



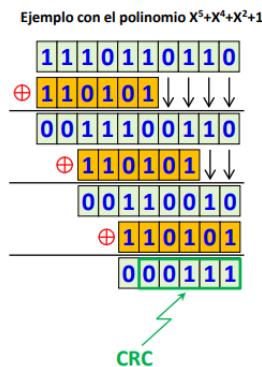
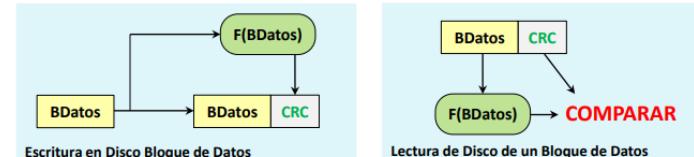
- GAP: marca física que indica la separación entre sectores (formateo bajo nivel)
- SYNC (Preamble): alrededor de 10 bytes que permite establecer la frecuencia y amplitud con las que se ha grabado la información.
- @ (Address mark): identificación del sector, más información de estado.
- DATA: 512 bytes de datos codificados en RLL
- CRC: checksum para comprobar la integridad de los datos (alrededor de 10 bytes).
- ECC: información redundante para detectar y corregir errores (alrededor de 40 bytes).
- FP (Flush Pad): información interna para sincronización.

ECC (Error Correcting Code)

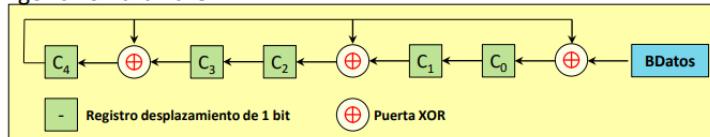
- Junto con cada sector de 512 bytes almacenado en disco se incluye información redundante para detectar y corregir errores.
- El algoritmo utilizado es el Reed-Solomon (1).
- La eficacia del algoritmo ECC depende del número de bits redundantes que se almacenen.
 - Incluir más bits de ECC implica que el sistema es más robusto, pero significa menos sectores por pista.
 - Un sistema más robusto permite aumentar la densidad de datos.
 - Más bits de ECC implica que el controlador correspondiente ha de ser más potente.

CRC (Cyclic Redundancy Check)

- Algoritmo utilizado para comprobar la integridad de un bloque de información.
- Es un algoritmo derivado de la división de polinomios con XOR, en vez de división.
- Discos duros usan **CRC-16-CCITT**: $X^{16}+X^{12}+X^5+1$



- Algoritmo hardware:



«División» del polinomio $X^5+X^4+X^2+1$
Usado en redes RDSI

S.M.A.R.T.

S.M.A.R.T. (Self Monitoring Analysis and Reporting Technology)

- Muchos de los problemas de funcionamiento de un disco duro no ocurren de repente.
- La mayoría son resultado de la degradación (lenta) de los componentes mecánicos o electrónicos.
- La tecnología S.M.A.R.T. monitoriza diferentes parámetros del disco (¡decenas!) con el objetivo de anticipar los problemas, predecir cuando un disco está en situación de riesgo, y avisar al usuario para que lo reemplace.
- La versión actual no sólo predice errores, también corrige algunos.
 - Reubicación de sectores erróneos.
 - Los discos vienen con sectores adicionales (no contabilizados)
 - No se detectan «BAD SECTORS» (el disco duro los reasigna).
- Los «avisos de S.M.A.R.T.» están correlacionados con altas probabilidades de fallo:
 - Después del primer aviso, los discos tienen 39 veces más posibilidades de fallar en los siguientes 60 días, que los discos sin avisos (1).
 - Aunque muchos fallos de disco se producen sin aviso previo.

Introducción RAID

RAID (Redundant Array of Inexpensive / Independent Disks)

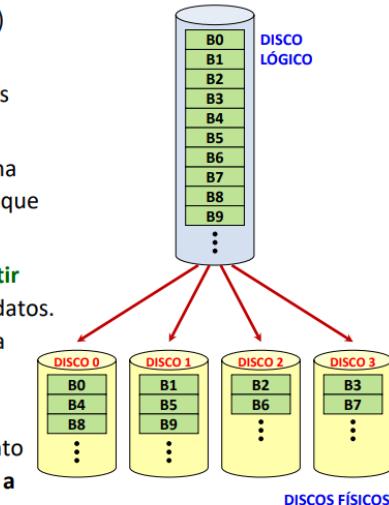
- Esquema estandarizado de múltiples discos que la industria ha adoptado para almacenar grandes cantidades de datos.
- **Objetivos**
 1. Aumentar el rendimiento (ancho de banda)
 2. Aumentar la capacidad (muchos discos)
 3. Aumentar la fiabilidad (*reliability*) de los datos (tolerancia a fallos) en los sistemas de almacenamiento masivo (Discos Magnéticos)
- **La idea detrás de RAID:** Usar múltiples discos (más capacidad) que operen independientemente y en paralelo para incrementar el ancho de banda y mejorar la fiabilidad.
 - Ficheros distribuidos a través de múltiples discos. El ancho de banda aumenta con el número de discos
 - Se pueden añadir esquemas de redundancia y corrección de errores para mejorar la fiabilidad de los datos.
- **Fiabilidad**
 - La tecnología RAID **protege los datos contra el fallo de una unidad de disco duro**. Si se produce un fallo, RAID mantiene el servidor activo y en funcionamiento hasta que se sustituya la unidad defectuosa
 - Los sistemas RAID (excepto RAID 0) suponen la **pérdida de parte de la capacidad de almacenamiento** de los discos, para conseguir la redundancia o almacenar los datos de paridad
 - Los sistemas RAID profesionales deben incluir los elementos críticos **por duplicado**: fuentes de alimentación y ventiladores redundantes. De poco sirve disponer de un sistema tolerante al fallo de un disco si después falla por ejemplo una fuente de alimentación que provoca la caída del sistema

Introducción: conceptos básicos

- RAID es un **conjunto de unidades físicas** de disco vistos por el sistema operativo como **una única unidad lógica**.
- Los datos se distribuyen de forma entrelazada a través de las unidades físicas. Son posibles distintos niveles de entrelazado:
 - No entrelazado
 - Entrelazado a nivel de tira (*stripe*): cada fichero se divide en bloques llamados tiras que se distribuyen entre los discos. El tamaño típico de las tiras puede ir de 2 a 512 Kbytes
 - Entrelazado a nivel de byte
 - Entrelazado a nivel de bit
- La capacidad de los discos redundantes se usa para almacenar información que garantice la **recuperación de los datos** en caso de fallo del disco.
- Técnicas de redundancia de datos
 - No redundancia
 - *Mirroring*
 - Paridad
 - Códigos *hamming* horizontales
 - Códigos *Reed-Solomon*

RAID 0: Disk Striping

- Distribuye los datos con **entrelazado a nivel de tira (*stripe*)**
- También conocido como "**separación o fraccionamiento (*Striping*)**". Los datos se desglosan en pequeños segmentos y se distribuyen entre los discos del *array*
- Las **unidades de disco conectadas en paralelo** permiten una transferencia simultánea de datos a/de todos ellos, con lo que se obtiene un gran ancho de banda.
- Este nivel de RAID **NO ofrece tolerancia al fallo**. Al **no existir redundancia**, RAID 0 no ofrece ninguna protección de los datos. El fallo de cualquier disco del array tiene como resultado la pérdida de los datos y es necesario restaurarlos desde una copia de seguridad
- Este esquema es **aconsejable** en aplicaciones de tratamiento de imágenes, audio o video. En general, **acceso secuencial a ficheros de gran tamaño**. [Siempre y cuando los datos no sean críticos, o sean fácilmente recuperables o generados.]

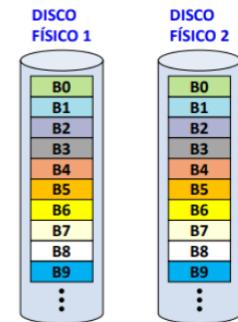


$$MTTF_{RAID\ 0} = MTTF_{disco}/\#discos$$

RAID 1: Mirroring

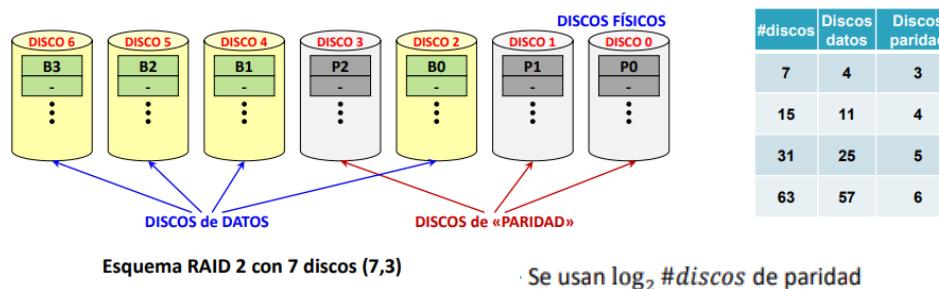
Mirror = ESPEJO

- Utiliza **discos adicionales** sobre los que se realiza una **copia exacta de los datos**
- **Se duplican todos los datos**. De esta manera se asegura la integridad de los datos y la tolerancia al fallo pues, en caso de avería, el RAID sigue trabajando con los discos no dañados sin detener el sistema
- RAID 1 ofrece una excelente **disponibilidad** de los datos mediante la redundancia total de los mismos.
- Los datos se pueden leer desde cualquiera de las copias
- Las escrituras son algo más lentas: se ha de escribir en las dos copias.
- RAID 1 es una alternativa costosa para los grandes sistemas, ya que duplica el coste de los discos



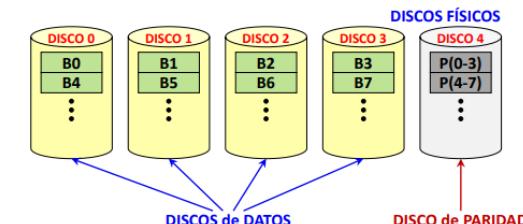
RAID 2: Redundancia a través del código Hamming

- Distribuye los datos con **entrelazado a nivel de bit**
- La operación de E/S accede al mismo sector de **todos los discos en paralelo**
- Adapta la **misma técnica ECC** que se usa para detectar y corregir errores en **DRAM**
- El código ECC se intercala a través de varios discos a nivel de bit. El método empleado es el *Hamming*
- Permite detectar y corregir 1 disco que falla o bien detectar que fallan 2 discos.
- Es un esquema teórico que no se utiliza.**
- El resto de sistemas de corrección (RAID 3 a 6) se basan en tener información de qué disco/s falla/n
- Se dedica un espacio considerable para información redundante.



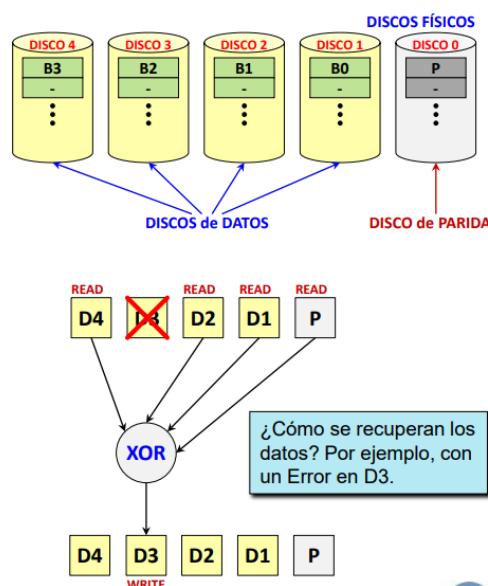
RAID 4: Acceso Independiente con un disco dedicado a paridad

- Es similar a RAID 3 pero con **entrelazado a nivel de tira**
- En RAID 4 se **puede acceder a los discos de forma individual**.
- Basa su tolerancia al fallo en la utilización de un disco dedicado a guardar la información de paridad calculada a partir de los datos guardados en los otros discos.
- El disco de paridad es el cuello de botella del sistema**
- En caso de avería de cualquiera de las unidades de disco, la información se puede reconstruir en tiempo real mediante la realización de una operación lógica XOR a nivel de tira.
- Sólo se usa **1 disco** para paridad



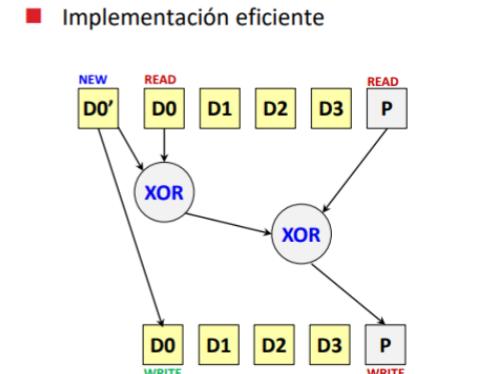
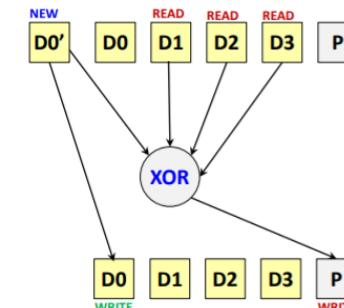
RAID 3: Acceso síncrono con un disco dedicado a paridad

- Distribuye los datos con **entrelazado a nivel de byte (bit en H&P)**
- Dedica **un único disco** al almacenamiento de información de **paridad**
- La información de ECC del disco (*Error Checking and Correction*) se usa para detectar errores. La recuperación de datos se consigue calculando la OR exclusiva (XOR) de la información registrada en los otros discos
- La operación de E/S accede al mismo sector de **todos los discos en paralelo**
- Su rendimiento de transacción es pobre porque todos los discos del conjunto han de operar conjuntamente



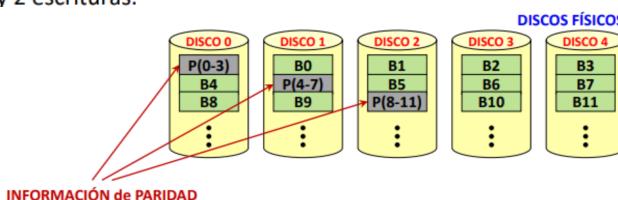
Las escrituras son costosas

- Implementación intuitiva
 - Implementación eficiente
- Con N discos, necesita N-2 lecturas y 2 escrituras.**
- Cuello de botella: SIEMPRE se ESCRIBE en el disco P.**



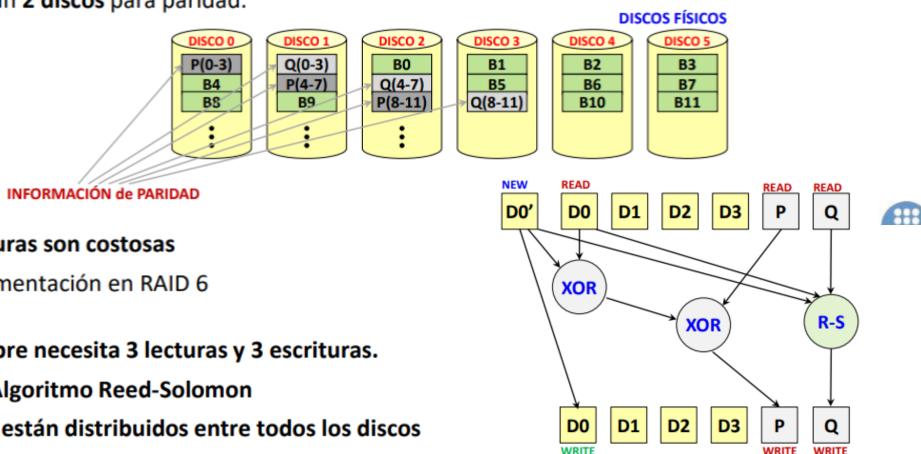
RAID 5: Acceso independiente con paridad distribuida

- Organizado de forma similar al RAID 4. La diferencia es que **los bloques de paridad están distribuidos entre todos los discos**
- Este esquema **evita el cuello de botella que hay en el disco de paridad** de RAID 4.
- RAID 5 no asigna un disco específico a esta misión, sino **un bloque alternativo de cada disco**.
- Distribuir la función de comprobación entre todos los discos disminuye el cuello de botella
- Con una cantidad suficiente de discos puede llegar a eliminarse completamente el cuello de botella, proporcionando una velocidad equivalente a un RAID 0.
- Se puede acceder a los discos de forma **independiente, en paralelo**.
- Una escritura requiere 2 lecturas y 2 escrituras.
- Sólo se usa **1 disco** para paridad.



RAID 6: Acceso independiente con doble paridad

- Similar al RAID 5, pero incluye un **segundo esquema de redundancia distribuido** por los distintos discos. Ofrece tolerancia extremadamente alta a fallos (dos niveles de redundancia)
- Pocos ejemplos comerciales. Su **coste de implementación es mayor** al de otros niveles RAID. Las controladoras que soportan esta doble paridad son más complejas y caras.
- Las dos tiras redundantes son normalmente llamadas P y Q
 - P es la tira de paridad como en RAID 5
 - Q es el segundo nivel de redundancia basado en códigos Reed-Solomon
- Permite recuperar información aunque fallen hasta 2 discos
- Se usan **2 discos** para paridad.



Fiabilidad RAIDs

- El tiempo hasta fallo de **1 disco** se aproxima a una **distribución exponencial** donde:
 - p = probabilidad de que se produzca un fallo
 - $\lambda = 1/MTTF$ (failure rate)
 - t = tiempo transcurrido
- ¿Cuál es el **tiempo medio hasta fallo** de un RAID 0 con N discos?
 - Si falla un solo disco el sistema falla.
 - $MTTF_{RAID\ 0} = MTTF_{disco} / N$
- ¿Cuál es el **tiempo medio hasta fallo** de otros RAID con N discos?
 - RAID 3, 4, 5 y 1 (caso particular N=2): Si falla un disco, el sistema sigue operativo.
 - En estos 4 casos, si durante el intervalo MTTR (tiempo de cambiar disco + tiempo de reconstruir la información) falla un segundo disco entonces falla el sistema.
 - RAID 2: no se usa
 - RAID 6: El sistema falla si falla un tercer disco
- En un RAID 3,4,5 el sistema falla si falla un segundo disco durante el intervalo MTTR

- $MTTR$ = tiempo de cambiar disco + tiempo de reconstruir la información
- $MTTF_N = MTTF_{disco}/N$ tiempo medio hasta fallo para N discos
- $MTTF_{N-1} = MTTF_{disco}/(N - 1)$ tiempo medio hasta fallo para N-1 discos
- Probabilidad P_2 de que falle un segundo disco (de N-1 discos que nos quedan):

$$P_2 = 1 - e^{-\lambda t} = 1 - e^{-\frac{MTTR}{MTTF_{N-1}}}$$

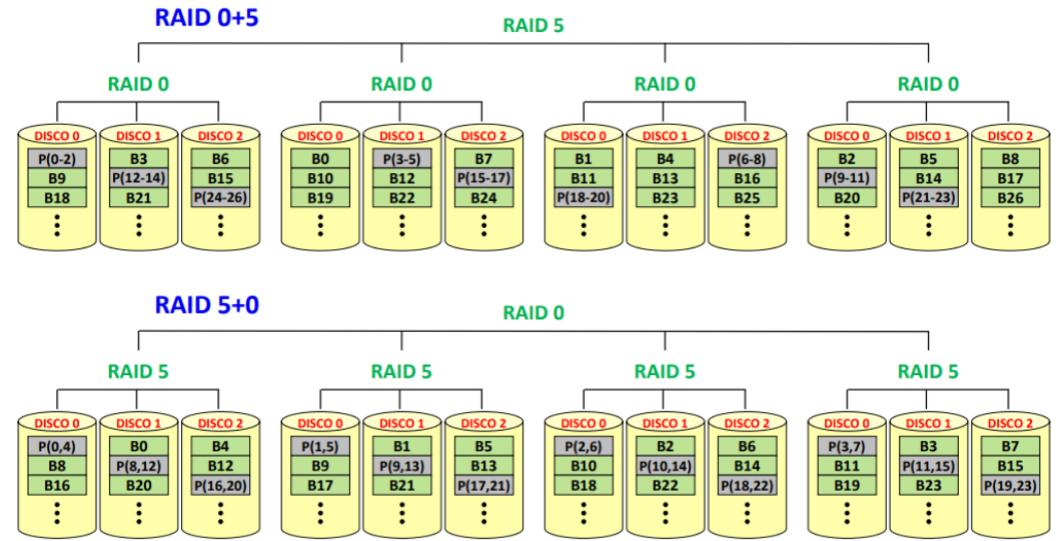
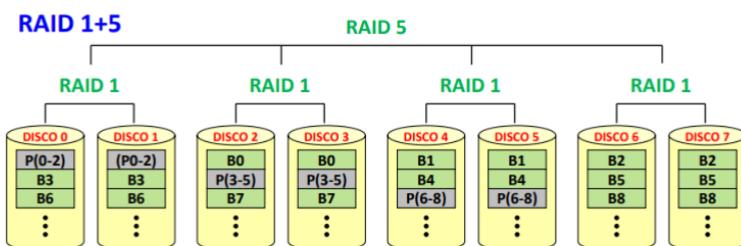
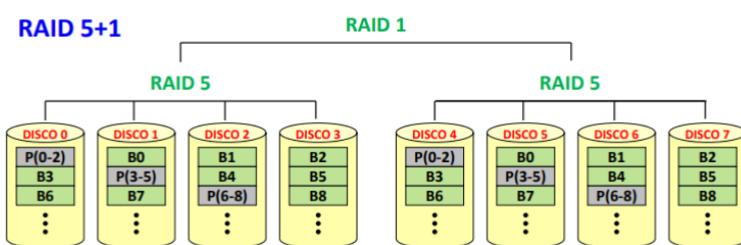
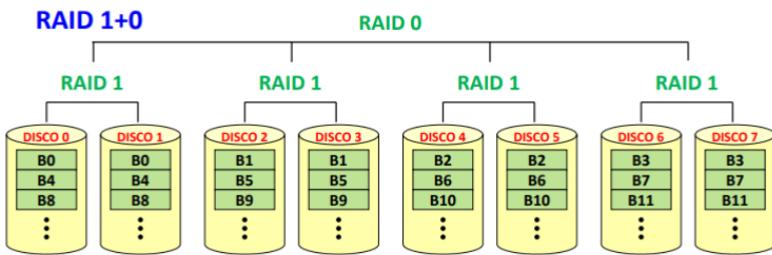
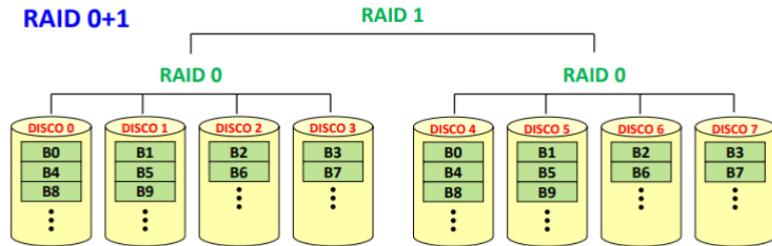
- Cada vez que falla un disco (de N discos) han transcurrido en media $MTTF_N$ horas, por tanto:

$$MTTF_{RAID\ 5} = MTTF_N \times \frac{MTTF_{N-1}}{MTTR} = \frac{MTTF_{disco}^2}{N \times (N - 1) \times MTTR}$$

- **Ejemplos:**
 - $MTTF_{disco} = 100.000$ horas
 - $MTTR = 10$ horas
 - $N = 10$ discos
- **RAID 0**
 - $MTTF_{RAID\ 0} = 100.000/10 = 10.000$ horas
- **RAID 5**
 - $MTTF_{RAID\ 5} = 100.000^2 / (10 \times 9 \times 10) = 11.111.111$ horas
- **RAID 1 (N = 2 discos)**
 - $MTTF_{RAID\ 1} = 100.000^2 / (2 \times 1 \times 10) = 500.000.000$ horas
 - ¡Atención!: no es comparable al resto, sólo usamos 2 discos
- **RAID 6**
 - $MTTF_{RAID\ 6} = 13.888.888.889$ horas
 - Ejercicio: deducir la expresión para RAID 6

- Esquemas multinivel comerciales
 - RAID 0+1 (RAID 01) y RAID 1+0 (RAID 10)
 - RAID 0+5 (RAID 05) y RAID 5+0 (RAID 50)
 - RAID 1+5 (RAID 15) y RAID 5+1 (RAID 51)

- El orden es importante: RAID X+Y ≠ RAID Y+X
 - RAID X+Y consiste en crear grupos de discos con RAID X y después tratar estos grupos como discos individuales para crear un array con RAID Y



Lecturas / Escrituras

Lecturas secuenciales → ancho de banda efectivo por disco * discos físicos (discos totales)

Lecturas aleatorias → ancho de banda efectivo por disco * discos físicos (discos totales)

Escrituras secuenciales → ancho de banda efectivo por disco * discos de datos

Escrituras aleatorias → ancho de banda efectivo por disco / coste escritura (depende del RAID)

Almacenamiento externo

$$\# \frac{\text{pistas}}{\text{plato}} = \frac{\text{superficie}_{\text{plato}}}{\text{distancia entre pistas}}$$

$$Capacidad_{disco} = \#platos * \# \frac{pistas}{plato} * \# \frac{sectores}{pista} * \# \frac{bytes}{sector}$$

$$Velocidad_{disco} = \frac{rpm}{60} * \# \frac{\text{sectores}}{\text{pista}} * \frac{\text{bytes}}{\text{sector}}$$

$$\text{tiempo transferencia de un sector} = \frac{\# \text{ bytes}}{\text{sector}} \cdot \frac{1}{\text{velocidad}_\text{disco}}$$

#operandos (memoria)	Tipo de Arquitectura	Ejemplo: A = A + B
0 (0)	Pila	PUSH A PUSH B ADD POP A
1 (1)	Acumulador	LOAD B ADD A STORE A
2 (1)	General Purpose Register (GPR)	Registro /Memoria MOV B,R1 ADD R1,A
3 (0)		Load / Store LOAD A,R1 LOAD B,R2 ADD R1,R2,R3 STORE R3,A
3 (3)		Memoria / Memoria ADD A,B,A

Máquina de PILA

- Sin operandos explícitos a excepción de las instrucciones de salto y acceso a memoria.

Aritméticas, lógicas y comparación	ADD, SUB, MUL, DIV, AND, OR, XOR, CMPge, CMPEq, ...	pila[TOP+1] ← pila[TOP] op pila[TOP+1] TOP ← TOP+1
Acceso Memoria	PUSH @	TOP ← TOP-1 pila[TOP] ← M[@]
	POP @	M[@] ← pila[TOP] TOP ← TOP+1
Salto	Bcond \$	if (pila[TOP]) PC ← PC + despl TOP ← TOP+1
	BR \$	PC ← PC + despl

- Procesadores antiguos en los que la memoria era un recurso escaso. Máquina Virtual Java.
- La pila es un recurso hardware (equivalente al banco registros coma flotante x86)
- Instrucciones muy cortas, buena densidad de código.
- ↓ La pila no se puede acceder aleatoriamente, es difícil generar código eficiente.
- ↓ La pila puede ser un cuello de botella.

Máquina de ACUMULADOR

- Dispone de un acumulador como operando implícito de todas las operaciones

Aritméticas, lógicas y comparación	ADD @ AND @ CMPge @	ACC ← ACC op M[@]
Acceso Memoria	LOAD @	ACC ← M[@]
	STORE @	M[@] ← ACC

Salto	Bcond \$	if (ACC) PC ← PC + despl
	BR \$	PC ← PC + despl

- Acumulador es un término arcaico para referirse a registro
- ↑ Estado interno de la máquina mínimo (1 registro)
- ↑ Instrucciones cortas
- ↓ Tráfico con memoria muy elevado

Arquitecturas con Registros de Propósito General (GPR, General Purpose Register)

- ↑ Modelo más general para la generación de código, código eficiente
- ↑ Acceso rápido a los registros
- ↑ Utilización eficiente de los registros por el compilador
- ↑ Reducción del tráfico con memoria
- ↓ Todos los operandos son explícitos, algunas instrucciones pueden ser muy grandes.

Máquina de Registro / Memoria

- Procesador con dos operandos explícitos, uno de ellos puede estar en memoria.

Aritméticas, lógicas y comparación	ADD op1, op2 AND op1, op2 CMP op1, op2	op2 ← op2 op op1
Salto	Bcond \$	if (cond) PC ← PC + despl
	BR \$	PC ← PC + despl

- Uno de los operandos es simultáneamente fuente y destino.
- Aparecen instrucciones de movimiento de datos.
- ↑ Los datos son accesibles sin lectura previa de memoria
- ↓ Los operandos no son equivalentes (un operando fuente se modifica)
- ↓ Codificar una dirección y un registro puede limitar el número de registros
- ↓ Los ciclos por instrucción varían dependiendo del tipo de acceso (de 0 a 2 accesos a memoria en 1 única instrucción).

Máquina de Memoria/Memoria

- Procesador con 3 operandos explícitos, cualquiera de ellos puede estar en memoria.

Aritméticas, lógicas y comparación	ADD op1,op2,op3 AND op1,op2,op3 CMP op1,op2,op3	op3 \leftarrow op2 op op1
Salto	Bcond \$ BR \$	if (cond) PC \leftarrow PC + despl PC \leftarrow PC + despl

- ↑ Código muy compacto
- ↑ No hace falta utilizar registros para variables temporales
- ↓ Diferentes tamaños de instrucción \Rightarrow Dificulta la búsqueda y decodificación de instrucciones
- ↓ Diferentes cargas de trabajo por instrucción
- ↓ La memoria se convierte en el cuello de botella

Tipos de Modos de Direcciónamiento

■ Modo registro.

- El operando se encuentra en uno de los registros del procesador
- ↑ Acceso rápido
- ↑ Pocos bits
- ↓ Se necesitan instrucciones para mover datos con memoria
- Ejemplo: `movl %edx, %eax`

■ Modo inmediato.

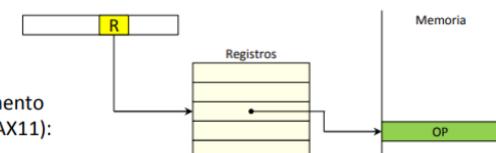
- El operando se encuentra en la instrucción
- Muy útil para trabajar con constantes
- Ejemplo: `movl $34, %eax`

■ Modo Absoluto (Directo).

- La dirección del operando se encuentra en la instrucción
- ↓ Se necesitan muchos bits para codificar la dirección
- Ejemplo: `movl 1242451, %eax`

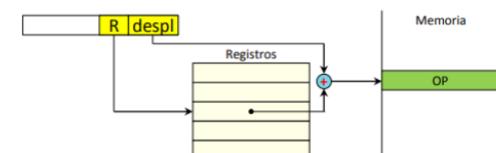
■ Modo registro indirecto.

- La instrucción codifica el registro que contiene la dirección del operando
- ↑ Pocos bits
- Ejemplo: `movl (%ebx), %eax`
- ↑ Posibilidad de autoincremento / autodecremento para facilitar accesos a arrays, pila, etc. (p.e. VAX11):
 - ✓ Autoincremento: `MOVL R1, (R2) +`
 - ✓ Autodecremento: `MOVL R1, -(R2)`



■ Modo Base + desplazamiento

- La dirección se calcula utilizando un registro y un desplazamiento (\pm)
- Si el desplazamiento es cero, equivale al modo registro indirecto
- Ejemplo: `movl -24(%ebp), %eax`
- ↓ Cálculo de la dirección



Máquina Load/Store

- Procesador con 3 operandos explícitos en registros.

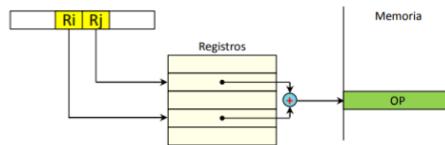
Aritméticas, lógicas y comparación	ADD Ri, Rj, Rk AND Ri, Rj, Rk CMP Ri, Rj, Rk	Rk \leftarrow Ri op Rj
Memoria	LOAD D(Rj),Rk	Rk \leftarrow M[Rj+D]
	STORE Rk,D(Rj)	M[Rj+D] \leftarrow Rk

- Todas las operaciones aritméticas se realizan sobre registros.
- Necesita instrucciones específicas para acceder a memoria (load / store)
- ↑ Instrucciones codificadas de forma fija \Rightarrow Facilita la búsqueda y decodificación de instrucciones
- ↑ Generación de código sencilla (el compilador tiene pocas alternativas)
- ↑ Todas las instrucciones tardan tiempos parecidos.
- ↓ Hacen falta más instrucciones, p.e. las utilizadas para acceder a memoria
- ↓ Formato fijo \Rightarrow Desperdicio de memoria



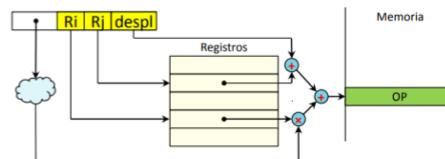
■ Modo indexado.

- La dirección efectiva se obtiene sumando el contenido de dos registros
 - Pocos bits
 - Útil en el acceso a vectores
 - Cálculo de la dirección
- Ejemplo: `movl (%ebx, %esi), %eax`



■ Modo escalado

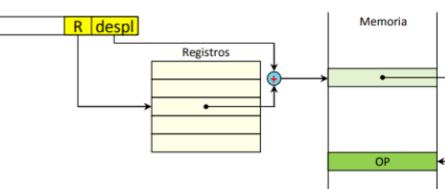
- Especialmente útil para acceder a vectores.
- El escalado depende de la instrucción
- Cálculo de la dirección
- Ejemplo: `movl -44(%ebx, %esi, 4), %eax`



Modos de Direccionamiento “combinados”

■ Post - indirección

- El dato obtenido con los modos anteriores es la dirección del operando.
- Ejemplo: `Base + desplazamiento post-indirecto`
- Ejemplo: `MOVL @32(SP), R3`



■ Post - escalado

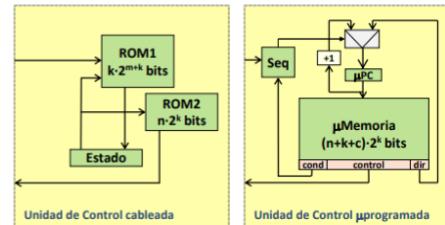
- El VAX11 permitía combinar el modo escalado (indexado en terminología de DIGITAL) a cualquier otro.
- Ejemplo: `MOVL @32(SP)[R7], R3`



Unidad de Control Cableada vs μprogramada

■ Unidad de control cableada

- X estados, $k = \log_2 X$
- Entradas: m bits (IR + cond)
- Salidas: n bits (control de la UP)
- ROM1: $k \cdot 2^{m+k}$ bits para calcular el siguiente estado
- ROM2: $n \cdot 2^k$ bits para obtener las señales de control asociadas a cada estado.



■ Unidad de control μprogramada

- X μinstrucciones (estados), $k = \log_2 X$
- Entradas: m bits (IR + cond)
- Salidas: n bits (control de la UP)
- Cond: c bits para seleccionar la siguiente μinstrucción
- μmemoria: $(n+k+c) \cdot 2^k$ bits para obtener las señales de control asociadas a cada estado.
- Seq: 2^{c+m} bits para seleccionar la siguiente μinstrucción

- **Ejemplo:**
 - 1024 estados, $k = 10$,
 - Entradas: 14 bits ($m = 14$)
 - Salidas: 24 bits ($n = 24$)
 - Cond: 8 condiciones ($c = 3$)
- **Coste UC cableada**
 - ROM1: $10 \cdot 2^{14} = 167.772.160$ bits
 - ROM2: $24 \cdot 2^{10} = 24.576$ bits
- **Coste UC μprogramada**
 - μmemoria: $37 \cdot 2^{10} = 37.888$ bits
 - Seq: $2^{17} = 131.072$ bits

Ejemplos de traducción de x86 a μops

ADDL \$17, 36(%EBX, %EDX, 4)

```
R1 ← M[EBX+EDX*4+36]
R2 ← R1+17
M[EBX+EDX*4+36] ← R2
```

RET

```
EIP ← M[ESP]
ESP ← ESP+4
```

CALL \$1

```
ESP ← ESP-4
M[ESP] ← EIP
EIP ← EIP+despl. ($1)
```

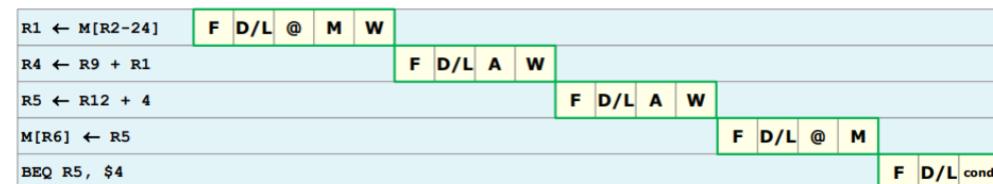
PUSHL 100(%EBX,8)

```
R1 ← M[EBX*8+100]
ESP ← ESP-4
M[ESP] ← R1
```

Procesador Secuencial

- Las instrucciones se ejecutan de forma secuencial.

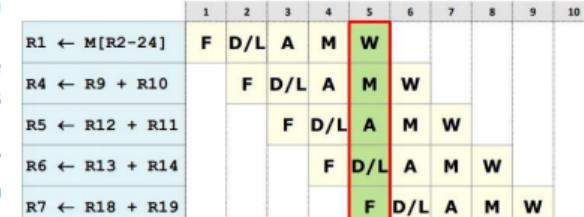
- Las instrucciones pueden tener tiempos de ejecución diferentes, dependiendo de su complejidad:



Ejecución Segmentada

Hay que tener en cuenta :

- Que no se produzcan errores de lecturas incorrectas, es decir, cuando un valor se lee antes de ser modificado por una instrucción anterior
- Que no se ejecuten instrucciones que han tenido un salto anterior a estas, es decir, si antes teníamos una instrucción de salto, hay que procurar que las instrucciones posteriores a esta, no se empiecen a ejecutar.
- Las instrucciones (datos + control) se mueven por el pipeline.
- Al inicio de ciclo, todas las etapas empiezan a funcionar con los datos que hay en los registros de desacoplamiento en la entrada de la etapa.
- Al final de ciclo, las señales de salida de cada etapa se almacenan en los registros de desacoplamiento.
- En un ciclo determinado se está ejecutando una instrucción diferente en cada una de las etapas.
- Prácticamente, con el hardware necesario para ejecutar 1 instrucción, estamos ejecutando concurrentemente 5 instrucciones independientes.



Límites a la Segmentación

VLIW

Riesgos de Datos

	1	2	3	4	5	6	7
I ₁ : R1 ← M[R2+X]	F	D/L	A	M	W		
I ₂ : R4 ← R9 + R1	F	D/L	A	M	W		

La instrucción I₁ calcula R1 y es utilizado en la instrucción I₂. I₁ escribe R1 en el ciclo 5, I₂ lee el registro R1 en el ciclo 3 ⇒ **I₂ lee un valor incorrecto.**

Riesgos de Control

	1	2	3	4	5	6	7
I ₁ : BEQ R1, \$4	F	D/L	A	M	W		
I ₂ : R4 ← R9 + R10	F	D/L	A	M	W		
I ₃ : R5 ← R12 + R11		F	D/L	A	M	W	

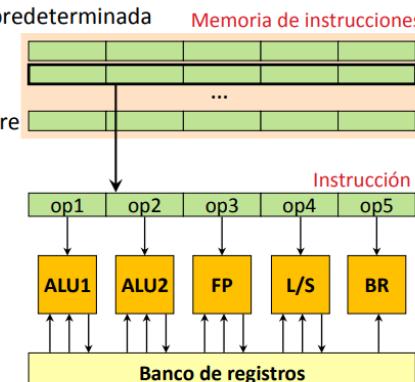
La instrucción I₁ evalúa la condición de salto en el ciclo 3 (etapa A). Hasta ese ciclo no sabemos si el salto será efectivo. Si el salto es efectivo ⇒ **Se ha iniciado la ejecución de 2 instrucciones erróneamente (I₂ e I₃)**.

Riesgos Estructurales

- Se producen cuando un único recurso se intenta utilizar por 2 instrucciones diferentes.
- El procesador segmentado que hemos presentado está libre de riesgos estructurales:
 - ✓ El Banco de Registros permite 2 lecturas y 1 escritura en el mismo ciclo
 - ✓ Dispone de Memorias independientes para instrucciones (MI) y datos (MD).

VLIW: Very Long Instruction Word (Arquitecturas con tamaño de instrucción muy grande)

- Objetivo: explotar ILP (Instruction Level Parallelism), CPI < 1.
- Una instrucción especifica múltiples operaciones independientes
- Cada operación se ejecuta en una unidad funcional predeterminedada
- La planificación de las instrucciones es estática: realizada por el compilador
- La planificación estática permite usar menos hardware de control y una arquitectura más simple:
 - Mayor frecuencia
 - Menor consumo
 - Menor coste
 - Más espacio para recursos (registros, UFs, caches, ...)
 - Mayor facilidad de verificación
- El procesador sigue estando segmentado
- Un porcentaje muy grande de los procesadores que se fabrican son embedded y de éstos muchos son VLIW



VLIW

- El compilador decide cuándo y dónde se ejecuta una operación

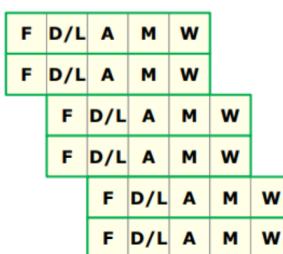
Superescalar

- El hardware decide ...

Procesadores Superescalares

Objetivo CPI < 1

- Dispone de **múltiples Unidades Funcionales**
- Permite **iniciar más de una instrucción (u operación) por ciclo**
- Sigue siendo un procesador segmentado
- Las instrucciones pueden tener tiempos de ejecución diferentes
- La mayoría de los procesadores de propósito general actuales son superescalares



Procesadores Superescalares OoO

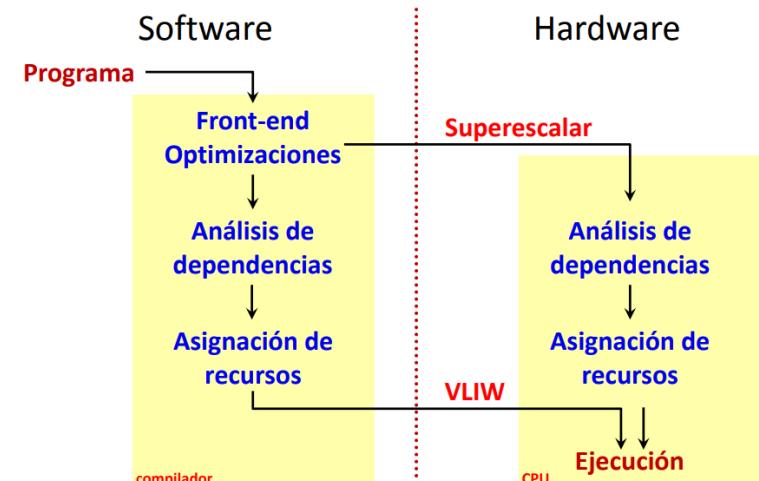
Objetivo CPI < 1

- Muchos procesadores superescalares permiten la ejecución fuera de orden (*OoO Processors, Out of Order Processors*)
- Las instrucciones se leen en orden, pero pueden ejecutarse en desorden
- Las instrucciones se bloquean si sus operandos no están disponibles
- Las instrucciones inician su ejecución cuando tiene sus operandos disponibles y la correspondiente U.F. está libre.

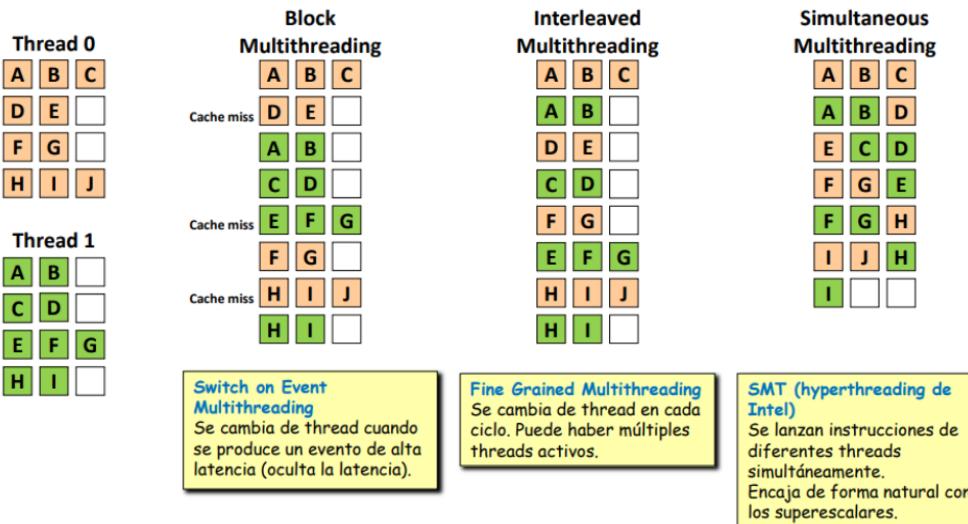
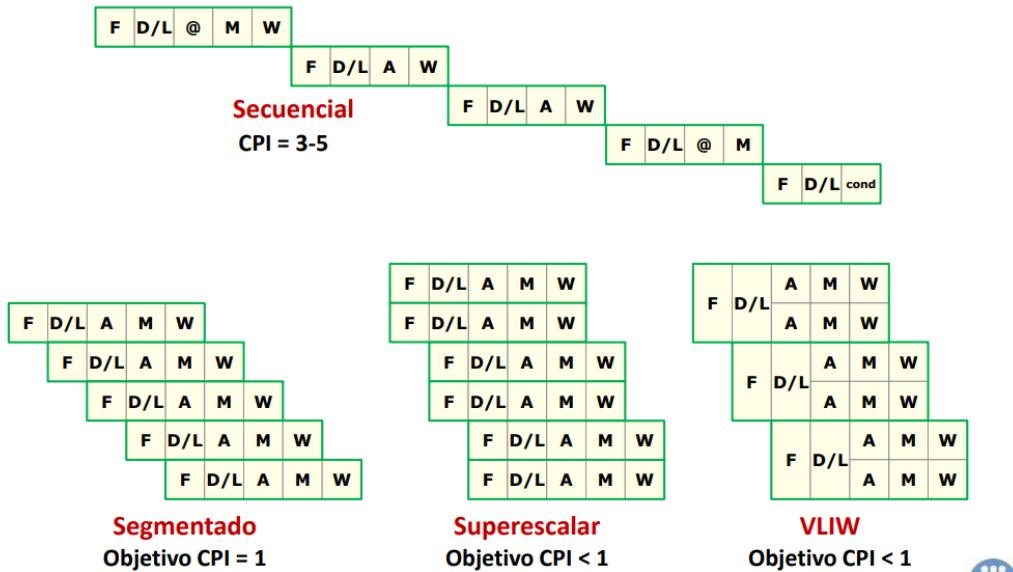


Intel C

Compilación para VLIW vs para Superescalar

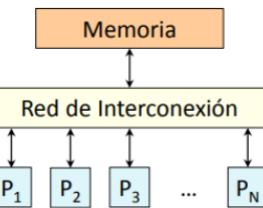


Resumiendo ILP



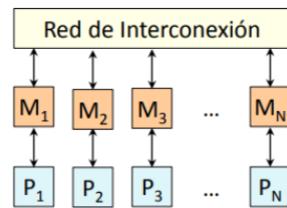
Multiprocesadores: Organización

Multiprocesadores con Memoria Compartida



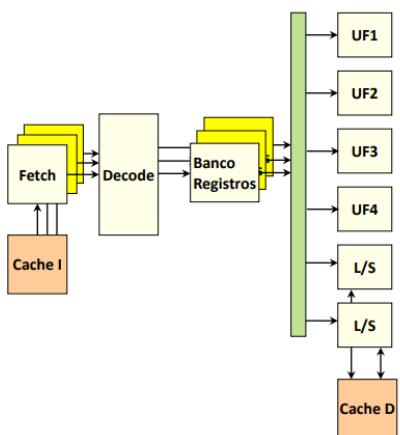
- Un procesador actual dispone de múltiples unidades funcionales.
- Se intenta iniciar P instrucciones/operaciones por ciclo.
- Los programas no siempre disponen de suficiente ILP.
- Una forma de aprovechar el hardware disponible es intentar ejecutar instrucciones de threads diferentes.
- Sólo es necesario multiplicar alguno de los elementos hardware: Fetch y Banco de Registros. Hay que mantener el estado de cada thread en ejecución.
- El SO ve tantas CPUs lógicas como estados puede mantener la CPU.

Multiprocesadores con Memoria Distribuida



- Existe un único espacio de direcciones compartido por todos los procesadores.
- La red de interconexión permite a cualquier procesador acceder a cualquier posición de memoria
- Los procesadores sólo pueden acceder a su memoria local.
- La red de interconexión permite a cualquier procesador comunicarse con cualquiera de los procesadores del sistema.

Multithreading

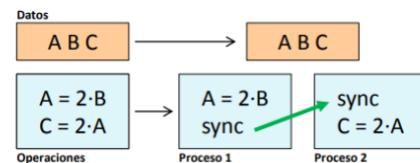


- Son diferentes unidades funcionales dentro de un procesador que permiten ejecutar múltiples instrucciones/operaciones por ciclo.
- Un procesador actual dispone de múltiples unidades funcionales.
- Se intenta iniciar P instrucciones/operaciones por ciclo.
- Los programas no siempre disponen de suficiente ILP.
- Una forma de aprovechar el hardware disponible es intentar ejecutar instrucciones de threads diferentes.
- Sólo es necesario multiplicar alguno de los elementos hardware: Fetch y Banco de Registros. Hay que mantener el estado de cada thread en ejecución.
- El SO ve tantas CPUs lógicas como estados puede mantener la CPU.

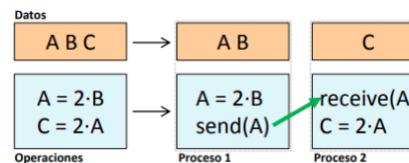
Son diferentes unidades funcionales dentro de un procesador que permiten ejecutar múltiples instrucciones/operaciones por ciclo.

Multiprocesadores: Programación

Modelo de variables compartidas



Modelo de paso de mensajes



- Las operaciones se dividen en procesos
- Los datos son compartidos por los procesos
- Se requieren primitivas de sincronización:
 - Señalización
 - Acceso Exclusivo

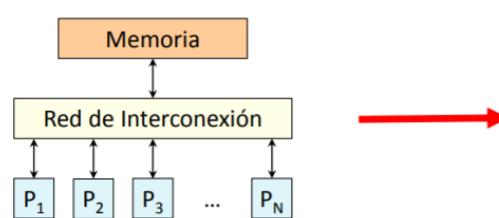
Situación Ideal



El modelo de programación y la organización

Organización	Modelo de Programación	
	Variables Compartidas	Paso de Mensajes
Memoria Compartida	Combinación natural Poco Escalable Programación fácil	Poco Escalable Programación difícil
Memoria Distribuida	Programación fácil Escalable Implementación difícil	Combinación natural Escalable Programación difícil

- La red de interconexión aumenta la latencia con memoria
- El uso de **memorias cache** locales de gran capacidad es **imprescindible**

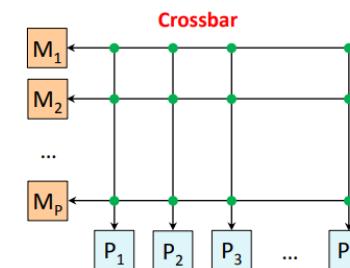
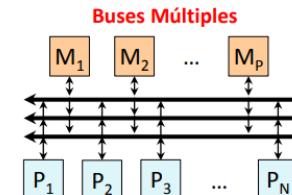
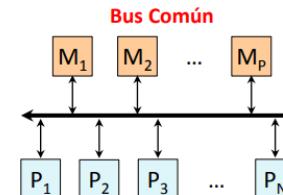


- Un sistema de memoria es coherente si cualquier lectura de un dato devuelve el último valor escrito sobre esa posición de memoria. Existen 2 temas a tener en cuenta:

- Coherencia
- Consistencia

Redes de Interconexión

- Elemento fundamental en el rendimiento de un multiprocesador



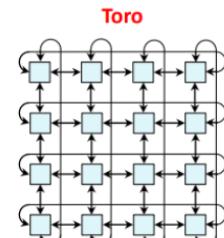
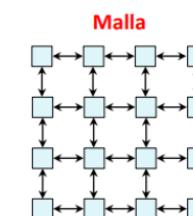
Bus Común
• Barato
• Ancho de banda bajo
Crossbar
• Caro (para muchos procesadores)
• Ancho de banda alto
Buses Múltiples
• Compromiso entre bus común y crossbar.

- La red de interconexión también es fundamental en los multiprocesadores con memoria distribuida

Redes de Acceso a Memoria Uniforme (UMA)

- Cualquier pareja de nodos se comunican con igual coste de comunicación
- Redes poco escalables

- Los multiprocesadores con memoria distribuida pueden utilizar conexiones punto a punto.



Redes de Acceso a Memoria No Uniforme (NUMA)

- El coste de la comunicación entre dos nodos depende de la posición relativa de los nodos en la red.
- Redes Escalables.