COGNOMS: . . . . . NOM: . . . . . .

3er Control Arquitectura de Computadors **Curs 2016-2017 Q1**

- **Temps 3 hores**
- **Poseu clarament amb LLETRES MAJÚSCULES a cada full els cognoms i el nom**

Problema 1. (3 puntos)

Se quiere diseñar una memoria cache de datos con políticas de escritura *write through* y *write NO allocate*:

Se han obtenido por simulación las siguientes medidas para un determinado programa:

- porcentaje de escrituras (sobre el total de accesos): 20%
- tasa de aciertos: 0,9

La memoria cache es de mapeo directo y se leen etiquetas y datos en paralelo. En caso de fallo de lectura, el bloque de MP se escribe en la MC y posteriormente el dato se envía a la CPU desde la MC. El tiempo de acceso (Tsa) a memoria cache (MC) es de 10 ns tanto para lectura como escritura. El tiempo de acceso a memoria principal (MP) para escribir una palabra es de 90 ns. Para leer o escribir un bloque en la MP se emplean 130 ns.

a) **Calcula** el tiempo empleado en realizar 1000 accesos consecutivos

Lectura: 0,80*1000 = 800 accesos

Número de aciertos: 0,9 aciertos/acc*0,80*1000 acc = 720 aciertos

Número de fallos: 0,1 fallos/acc x 0,80*1000 acc = 80 fallos

Tiempo aciertos: 720 acc*10x10^{-9} seg/acc = 7,2*10^{-6} segundos

Tiempo fallos: 80 acc*(10+130+10)*10^{-9} seg/acc = 12*10^{-6} segundos

Escritura: 0,20*1000 = 200 acc. Tiempo escrituras: 200 acc*90*10^{-9} seg/acc = 18 * 10^{-6} segundos

Tiempo total: 7,2*10^{-6} segundos + 12*10^{-6} segundos + 18*10^{-6} segundos = 37,2*10^{-6} segundos

---

Forma 2 de hacerlo:

T = 1000 accesos * Tma

1000*Tma = 1000*(Tsa + m_lec * Tpf_lecturas + m_esc * Tpf_escrituras) =

= 1000*(10ns + 0,1*0,80 * (130ns + 10ns) + 0,20 * (90ns - 10ns)) =

= 1000*(10ns + 11,2ns + 16ns) = 37,2*10^{-6} segundos

Forma 3 de hacerlo:

Aciertos: Lectura - 0,80 * 0,9 * 10ns/acceso * 1000 accesos = 7,2* 10^{-6} segundos

Escrituras - 0,9 * 0,20 * 90ns/acceso * 1000 accesos = 16,2* 10^{-6} segundos

Fallos: Lectura - 0,80 * 0,1 * (10+130+10 ns/acceso) * 1000 accesos = 12* 10^{-6} segundos

Escrituras - 0,1 * 0,20 * 90ns/acceso * 1000 accesos = 1,8* 10^{-6} segundos

Tiempo total: 7,2*10^{-6} s + 16,2*10^{-6} s + 12*10^{-6} s + 1,8*10^{-6} s = 37,2*10^{-6} segundos

Dado el siguiente código escrito en ensamblador del x86:

```
        movl $0, %ebx
        movl $0, %esi
   for:
        cmpl $1024*1000, %esi
        jge end

    (a) movl (%ebx, %esi, 4), %eax
    (b) addl 2*8*1024(%ebx, %esi, 4), %eax
    (c) movl %eax, 3*8*1024(%ebx, %esi, 4)

        addl $1, %esi
        jmp for
   end:
```

El código se ejecuta en un sistema con memoria cache y memoria virtual. Queremos estudiar el comportamineto de los accesos a datos en este sistema. La memoria cache de datos es *Write Through + Write No Allocate*, 2-asociativa con reemplazo LRU, tamaño 8 KB y 16 bytes por bloque. Responde a las siguientes preguntas:

b) **Calcula,** para cada uno de los accesos etiquetados como (a, b, c), el conjunto de la memoria cache al que se accede en cada una de las 13 primeras iteraciones del bucle

iteración	0	1	2	3	4	5	6	7	8	9	10	11	12
a	0	0	0	0	1	1	1	1	2	2	2	2	3
b	0	0	0	0	1	1	1	1	2	2	2	2	3
c	0	0	0	0	1	1	1	1	2	2	2	2	3

Calcula la cantidad de aciertos y de fallos de cache, en todo el código.

El bucle se ejecuta 1024*1000 veces, y se produce el siguiente patrón de aciertos/fallos cada 4 iteraciones:
a) 1F 1A 1A 1A
b) 1F 1A 1A 1A
c) 1F 1F 1F 1F

Aciertos = (1024 * 1000 / 4) * (3+3) = 1536000

Fallos = (1024 * 1000 / 4) * (1+1+4) = 1536000

La memoria virtual utiliza páginas de tamaño 8KB y disponemos de un TLB de 4 entradas y reemplazo LRU.

c) **Indica,** para cada uno de los accesos indicados (etiquetas a, b, c), a qué página de la memoria virtual se accede en cada una de las siguientes iteraciones del bucle (recuerda que los accesos son a 4 bytes).

iteración	0	1*1024	2*1024	3*1024	4*1024	5*1024	6*1024	7*1024	8*1024	9*1024
a	0	0	1	1	2	2	3	3	4	4
b	2	2	3	3	4	4	5	5	6	6
c	3	3	4	4	5	5	6	6	7	7

Calcula la cantidad de aciertos y de fallos de TLB, en todo el código.

Se falla en el TLB cada vez que se accede a una nueva página, el resto de accesos a esa página son simepre aciertos.
Hay 1024 accesos a cada página de cada instrucción (a,b, c). Cada instrucción accede a 500 páginas y en el bucle se accede a (1024*1000*4+3*8*1024)/8192 = 503 páginas distintas.
a) falla al acceder las paginas 0 y 1, el resto reusa las que accede b)
b) falla al acceder la pagina 2, el resto reusa las que accede c)
c) falla al acceder las 500 paginas

Fallos = 503
Aciertos = 1024*1000*3 - 503 = 3071497

COGNOMS: . . . . . NOM: . . . . . .

3er Control Arquitectura de Computadors **Curs 2016-2017 Q1**

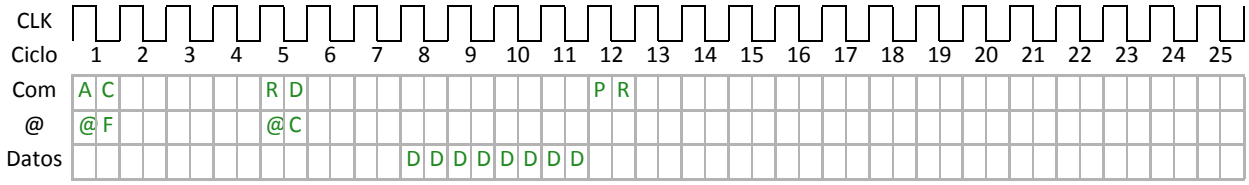
- **Temps 3 hores**
- **Poseu clarament amb LLETRES MAJÚSCULES a cada full els cognoms i el nom**

Problema 2. (4 puntos)

Una **CPU** está conectada a una cache de instrucciones (**$I**) y una cache de datos (**$D**). El conjunto formado por **CPU**+**$I**+**$D** esta conectado a una memoria principal formada por un único módulo DIMM estándar de 4 GBytes. Este DIMM tiene 8 chips de memoria **DDR**-SDRAM (**Double Data Rate** Synchronous DRAM). El DIMM esta configurado para leer/escribir ráfagas de 64 bytes (justo el tamaño de bloque de las caches). La latencia de fila es de 4 ciclos, la latencia de columna de 3 ciclos y la latencia de precarga de 1 ciclo.

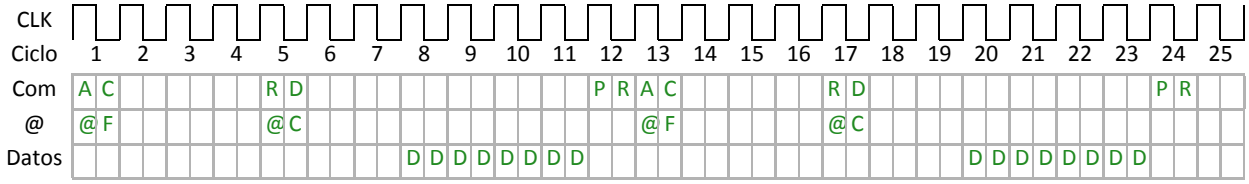
En los siguientes cronogramas, indica la ocupación de los distintos recursos de la memoria DDR: bus de datos, bus de direcciones y bus de comandos. En todos los cronogramas supondremos que no hay ninguna página de DRAM abierta.

a) **Rellena** el siguiente cronograma para una lectura de un bloque de 64 bytes de la DDR.

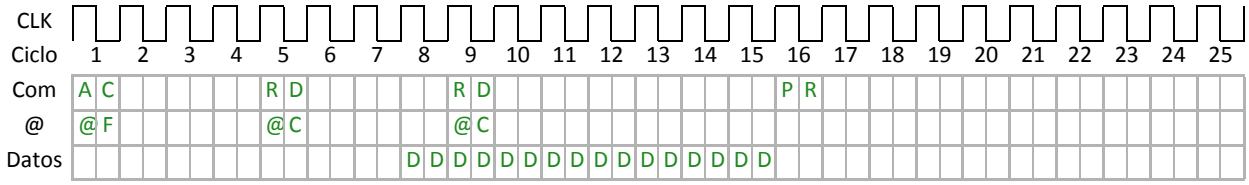


En ocasiones, es posible que el conjunto **CPU**+**$I**+**$D** solicite múltiples bloques a la DDR (por ejemplo porque se produzca un fallo simultaneamente en **$I** y en **$D**). El controlador de memoria envía los comandos necesarios a la DDR-SDRAM de forma que ambos bloques sean transferidos lo más rápidamente posible y se maximice el ancho de banda. Rellena los siguientes cronogramas para la lectura de dos bloques de 64 bytes en función de la ubicación de los dos bloques involucrados. El objetivo es minimizar el tiempo total.

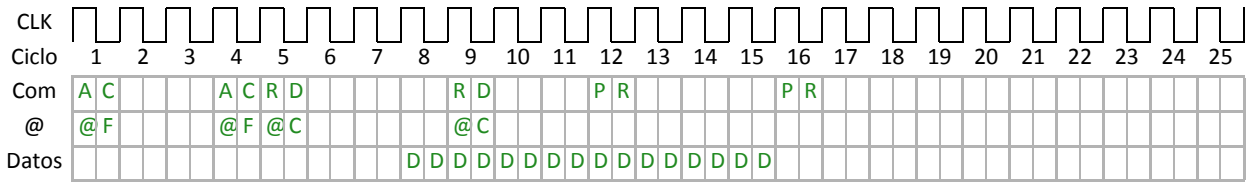
b) Ambos bloques están ubicados en el mismo banco pero en páginas distintas.



c) Ambos bloques están ubicados en la misma página .



d) Ambos bloques están ubicados en bancos distintos.



El conjunto **CPU**+**$I**+**$D** funciona a una frecuencia interna mucho mayor que la memoria SDRAM, por lo que los ciclos de los apartados siguientes no se corresponden a los cronogramas anteriores. Un ciclo de la SDRAM puede corresponder a multiples ciclos de CPU.

Un programa P realiza $5x10^9$ accesos a datos, todos de 8 bytes. Sabemos que **$D** tiene bloques de 64 bytes y políticas de escritura **copy back** + **write allocate**. Hemos medido que, durante la ejecución de P, **$D** tiene una tasa de fallos del 10% y que el 30% de los bloques reemplazados tenian el *dirty bit* a 1.

e) **Calcula** cuantos bytes lee **$D** desde la **DDR** y cuantos bytes escribe **$D** en la **DDR**.

Bytes leidos = 0,1 f/a * 5e9 i * 64 bytes/f = 32e9 bytes leidos

Bytes escritos = 0,1 f/a * 0,3 escr/f * 5e9 i * 64 bytes/escr = 9,6e9 bytes escritos

Dado el siguiente fragmento de código:

```
for (i=0; i<N; i++)
  suma = suma + v[i]; // v[i] es un vector de doubles (8 bytes)
```

El código está almacenado en **$I**, las variables i, N y suma estan en registros y **$D** está inicialmente vacía. Los elementos del vector v son de 8 bytes y los bloques de **$D** son de 64 bytes. La capacidad de **$D** es mayor de 8 Kbytes.

Hemos ejecutado 2 veces consecutivas el mismo bucle (para **N = 1000**) y hemos medido los ciclos de CPU de ambas ejecuciones:

- En la 1a ejecución el bucle tarda 42.500 ciclos.
- En la 2a ejecución el bucle tarda 30.000 ciclos.

f) **Calcula** el tiempo de penalización medio (en ciclos) en caso de fallo en **$D**.

1a ejecucion: Cada 8 iteraciones 1 fallo -> 125 fallos
2a ejecución: reusa el vector -> no hay fallos

12500 ciclos penalización / 125 fallos = 100 ciclos penalización/fallo

A la cache **$D** le añadimos un mecanismo de *prefetch* hardware. Cuando se accede un bloque (i) se desencadena prefetch del bloque siguiente (i+1) siempre que el bloque (i+1) no se encuentre ya en la cache o no haya un *prefecth* previo del bloque (i+1) pendiente de completar (en ambos casos es innecesario hacer prefecth de nuevo). En esta variante de **$D** ejecutamos una única vez el bucle anterior con N muy grande (mucho mayor que el tamaño de cache).

g) **Calcula** el número máximo de ciclos que puede durar un *prefetch* para que el bucle se ejecute en 30*N ciclos. ¿Es posible ejecutar el bucle en menos de 30*N haciendo el *prefetch* más rápido?

Se hace un prefetch cada 8 iteraciones
30 ciclos/iteracion * 8 iteraciones/prefecth = 240 ciclos/prefetch

No es posible ejecutar en menos de 30*N ciclos -> 30 ciclos por iteración es lo que podemos conseguir sin fallos.

h) **Calcula** los ciclos que tardaria en ejecutarse el bucle suponiendo que tanto un fallo como un *prefetch* tardasen 400 ciclos en traer los datos a la cache.

400 ciclos -> el camino crítico es el prefetch -> 400 ciclos/prefetch / 8 iteraciones/prefecth = 50 ciclos

50*N ciclos

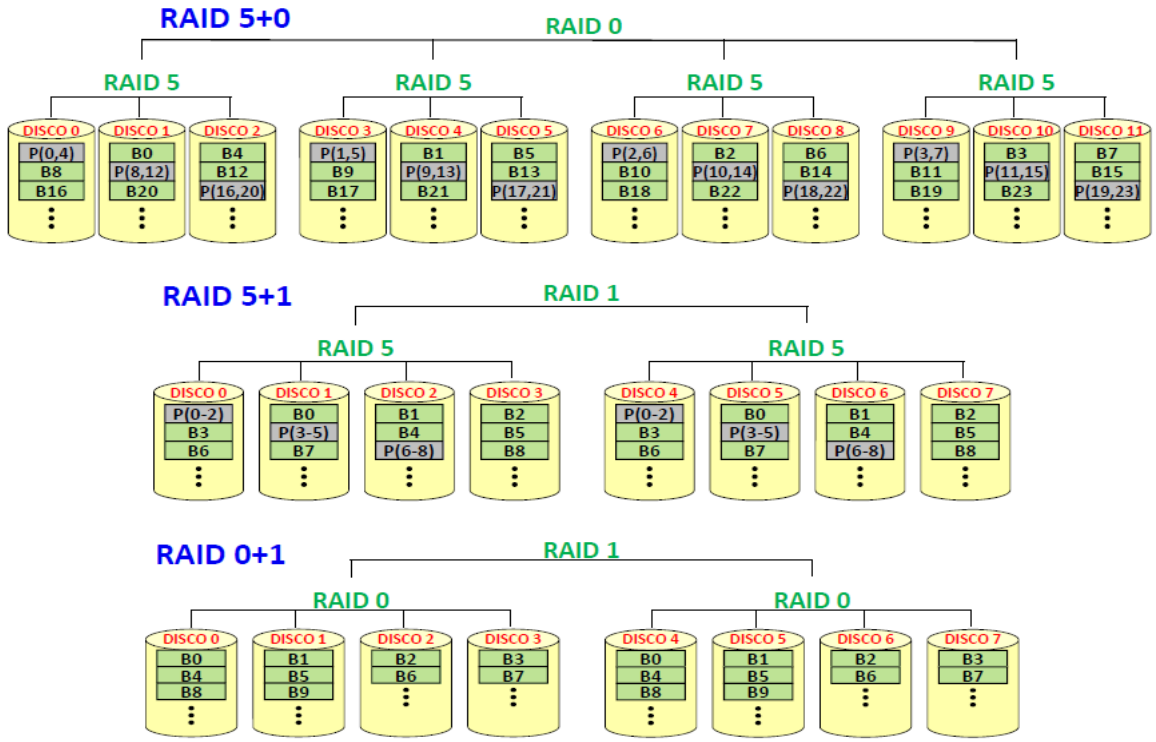
COGNOMS: . . . . . NOM: . . . . . .

3er Control Arquitectura de Computadors **Curs 2016-2017 Q1**

- **Temps 3 hores**
- **Poseu clarament amb LLETRES MAJÚSCULES a cada full els cognoms i el nom**

Problema 3. (3 puntos)

Determinadas configuraciones de RAID pueden seguir funcionando aunque algunos discos fallen. En la figura se muestran 3 configuraciones de RAID: RAID 5+0, RAID 5+1 y RAID 0+1.



a) El RAID 5+0 de la figura puede seguir funcionando aunque fallen 4 discos. Marca con una cruz en la tabla 4 discos qué podrían fallar y el RAID seguir funcionando.

b) El mismo RAID 5+0 de la figura puede dejar de funcionar si fallan 2 discos. Marca con una cruz en la tabla 2 discos qué deberían fallar para que el RAID no funcione.

	RAID 0											
	RAID 5			RAID 5			RAID 5			RAID 5		
	disco 0	disco 1	disco 2	disco 3	disco 4	disco 5	disco 6	disco 7	disco 8	disco 9	disco 10	disco 11
(a)	X			X				X			X	
(b)	X	X										

c) El RAID 5+1 de la figura puede seguir funcionando aunque fallen 5 discos. Marca con una cruz en la tabla 5 discos qué podrían fallar y el RAID seguir funcionando.

d) El mismo RAID 5+1 de la figura puede dejar de funcionar si fallan 4 discos. Marca con una cruz en la tabla 4 discos qué deberían fallar para que el RAID no funcione.

	RAID 1							
	RAID 5				RAID 5			
	disco 0	disco 1	disco 2	disco 3	disco 4	disco 5	disco 6	disco 7
(c)	X	X	X	X	X			
(d)	X	X			X	X		

e) El RAID 0+1 de la figura puede seguir funcionando aunque fallen 4 discos. Marca con una cruz en la tabla 4 discos que podrían fallar y el RAID seguir funcionando.

f) El mismo RAID 0+1 de la figura puede dejar de funcionar si fallan 2 discos. Marca con una cruz en la tabla 2 discos qué deberían fallar para que el RAID no funcione.

	RAID 1							
	RAID 0				**RAID 0**			
	disco 0	**disco 1**	**disco 2**	**disco 3**	**disco 4**	**disco 5**	**disco 6**	**disco 7**
(e)	X	X	X	X				
(f)	X				X			

Queremos evaluar el rendimiento de utilizar estos RAIDs, para ello vamos a utilizar discos de 1 TB de capacidad y 500 MB/s de ancho de banda. En la evaluación vamos a utilizar una aplicación con un porcentaje muy elevado de operaciones de E/S con disco. En esta aplicación pueden identificarse 4 fases (en las fases 1, 2 y 4 sólo tendremos en cuenta el coste de la transferencia de información):

- fase 1: En donde se ha de leer de disco los datos de entrada que ocupan 10 GB, distribuidos entre todos los discos.
- fase 2: Cálculo, con un tiempo de ejecución de 3s.
- fase 3: En donde se ha de escribir un bloque de datos de 5 GB, lo que permite realizar **escrituras secuenciales**.
- fase 4, En donde se han de realizar **escrituras aleatorias** de 5 GB de datos, distribuidas uniformemente entre todos los discos.

g) **Calcula** el tiempo de ejecución de nuestra aplicación si utilizamos un único disco.

fase 1: 10 GB / 500 MB/s = 20s

fase 3: 5GB / 500 MB/s = 10s

fase 4: 5GB/ 500 MB/s = 10s

Total: 20 + 3 + 10 + 10 = 43s

h) **Calcula** el tiempo de ejecución de nuestra aplicación en un RAID 5+1, organizado en 2 grupos de 10 discos cada uno.

fase 1: 10 GB / (20*500 MB/s) = 1s

fase 3: 5GB / (9*500 MB/s) = 1,1s

fase 4: 5GB/ ((10/4)*500 MB/s) = 4s

Total: 1 + 3 + 1,1 + 4 = 9,1s

i) **Calcula** el tiempo de ejecución de nuestra aplicación en un RAID 0+1, organizado en 2 grupos de 10 discos cada uno.

fase 1: 10 GB / (20*500 MB/s) = 1s

fase 3: 5GB / (10*500 MB/s) = 1s

fase 4: 5GB/ (10*500 MB/s) = 1s

Total: 1 + 3 + 1 + 1 = 6s