COGNOMS:

NOM:

IMPORTANTE leer atentamente antes de empezar el examen: Escriba los apellidos y el nombre antes de empezar el examen. Escriba un solo carácter por recuadro, en mayúsculas y lo más claramente posible. Es importante que no haya tachones ni borrones y que cada carácter quede enmarcado dentro de su recuadro sin llegar a tocar los bordes. Use un único cuadro en blanco para separar los apellidos y nombres compuestos si es el caso. No escriba fuera de los recuadros.

Problema 1. (3,4 puntos)

Un procesador funciona a una frecuencia de 1GHz y esta conectado a una cache de datos nivel 1 (L1) 4-asociativa de 16 KBytes de capacidad con bloques de 64 bytes. La cache no tiene ninguna de las mejoras vistas en clase. En caso de acierto, los accesos a la cache tardan 1 ciclo y en caso de fallo tenemos una penalización media de 20 ciclos. Hemos determinado que la cache se encuentra en el camino crítico del procesador, por lo que la frecuencia (y por tanto el tiempo de ciclo) del procesador vienen determinados por el diseño de la cache.

En la siguiente tabla, las filas indican posibles mejoras (vistas en clase) que podríamos introducir en L1. Las columnas indican algunos parámetros de rendimiento en los que estas mejoras podrían influir, los tiempos tanto de acceso como de penalización se consideraran en **ns** y no en ciclos ya que el tiempo de ciclo puede cambiar. Cada mejora será considerada de forma individual, sin cambiar ningún otro parámetro de la cache. Para indicar el impacto de las mejoras en los parámetros de rendimiento rellenad la tabla usando la siguiente nomenclatura: (**M**) Mejora, (**NA**) No Afecta, (**E**) Empeora.

a) **Indica** el impacto de las mejoras en los parámetros de rendimiento.

	Parámetro de rendimiento			
Mejora/cambio	Frecuencia del procesador	Tiempo (ns) de acceso a la cache L1 en caso de acierto	Tasa de fallos de la cache L1	Tiempo (ns) medio de penalización en caso de fallo
Cambiar a cache 4-asociativa de 8 KBytes	M	M	E	NA
Añadir una cache L2 8-asociativa de 32 KBytes	NA	NA	NA	M
Cambiar a cache 8-asociativa de 16 KBytes	E	E	M	NA
Añadir continuación anticipada	NA	NA	NA	M
Segmentar la cache en 4 etapas	M	E	NA	NA
Añadir un mecanismo de prefetch hardware	NA	NA	M	M
Hacer la cache no bloqueante	NA	NA	NA	M
Poner un predictor de via	M	M	NA	E

Nota 1: El apartado vale 1.5 puntos, cada fallo o cuadro en blanco resta 0.1 puntos en el apartado (mínimo 0).

Nota 2: La decisión de si un cambio mejora, empeora o no afecta a un parámetro debéis basarla en el caso general. Por ejemplo: aunque en determinadas circunstancias un predictor de vía podría empeorar el consumo de energía, en general lo mejorará.

Vistas todas las posibilidades decidimos evaluar si vale la pena usar un sistema con memoria virtual, sin cache ni TLB que es más sencillo. Sabemos que el sistema tiene un CPI ideal (suponiendo que cada acceso a memoria principal se pudiese resolver en 1 ciclo) de 6 ciclos y supondremos una tabla de páginas uninivel que es capaz de resolver todas las traducciones en un solo acceso. Supondremos también que no hay fallos de página.

Supongamos el siguiente código que se ejecuta en 0,418s. Las etiquetas "V" y "bucle" están alineadas a 1MB y sabemos que todas las instrucciones x86 se codifican usando entre 1 y 12 bytes:

```
        movl $1000000, %ecx
        movl $0, %eax
bucle:  addl V(,%ecx,4), %eax
        decl %ecx
        jnz bucle
```

b) **Calcula** el número de accesos físicos a memoria principal producidos al ejecutar el bucle.

Instrucciones ejecutadas = 3 * 1000000 = 3000000
Accesos a memoria = 4 (3 ins + 1 datos) * 2 (1 tabla páginas + 1 @física) * 1000000 = 8000000 accesos

c) ¿**Cuál** es el tiempo de acceso a la memoria principal?

Ciclos programa = 3000000 * 6 + 8000000 * (ta-1) = 418000000 = 18000000 + 50*8000000 -> ta -1 = 50 -> ta = 51 ciclos

Vista la ineficiencia decidimos utilizar 2 caches (y 2 TLBs) separados de instrucciones y datos. Cada TLB tiene 8 entradas, la cache de datos es la original (4-asociativa de 16 KBytes de capacidad con bloques de 64 bytes) y la cache de instrucciones es de la misma capacidad y tamaño de linea pero de acceso directo. Las páginas del sistema son de 8 KB.

d) ¿**Cuántos** fallos de cache de instrucciones y datos dará en total el código anterior?

2 fallos de instrucciones
(un fallo las instrucciones antes del bucle y otro las instrucciones del bucle)

1000000 / 16 de datos = 62502 fallos de datos

e) ¿**Cuántos** fallos de TLB de datos dará en total el código anterior?

1000000 / 2048 -> 489 fallos de TLB

COGNOMS:

NOM:

Problema 2. (3.2 puntos)

Una **CPU x86** 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 16 GBytes. Este DIMM tiene 8 chips de memoria **DDR**-SDRAM (**Double Data Rate** Synchronous DRAM) de 1 byte de ancho cada uno. La DDR-SDRAM tiene 16 bancos. 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 3 ciclos, la latencia de columna de 4 ciclos y la latencia de precarga de 2 ciclos. Es posible que el conjunto **CPU+$I+$D** solicite múltiples bloques a la DDR (por ejemplo porque se produzca un fallo simultáneamente en **$I** y en **$D**). El controlador de memoria envía los comandos necesarios a la DDR-SDRAM de forma que los bloques sean transferidos lo más rápidamente posible y se maximice el ancho de banda. Los 24 bits de direcciones físicas se han mapeado de la siguiente forma:

Fila	Banco	Columna	Chip
23:16	15:12	11:3	2:0

Dadas las siguientes direcciones físicas de bloque:
A: 0xaa0100
B: 0xaa0200
C: 0xaa1b00
D: 0xbb1580

Rellena los siguientes cronogramas para la lectura de varios bloques de 64 bytes en función de la ubicación de los bloques involucrados de forma que se minimice el tiempo total. 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 previamente.

a) **Rellena** el siguiente cronograma para la lectura de los bloques A y B.

CLK																											
Ciclo	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
Com	A	C				R	D					R	D								P	R	-	-			
@	@F					@C						@C															
Datos												D	D	D	D	D	D	D	D	D	D	D	D	D			

b) **Rellena** el siguiente cronograma para la lectura de los bloques A y C.

CLK																											
Ciclo	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
Com	A	C				R	D	A	C			R	D					P	R	-	-			P	R	-	-
@	@F					@C		@F				@C															
Datos												D	D	D	D	D	D	D	D	D	D	D	D	D			

c) **Rellena** el siguiente cronograma para la lectura de los bloques A y D.

CLK																											
Ciclo	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
Com	A	C				R	D	A	C			R	D					P	R	-	-			P	R	-	-
@	@F					@C		@F				@C															
Datos												D	D	D	D	D	D	D	D	D	D	D	D	D			

d) **Rellena** el siguiente cronograma para la lectura de los bloques C y D

CLK																											
Ciclo	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
Com	A	C				R	D					P	R	-	-	A	C			R	D					P	R	-	-
@	@F					@C										@F				@C									
Datos								D	D	D	D	D	D	D								D	D	D	D	D	D	D	

Queremos estudiar el efecto de la búsqueda y descodificación de instrucciones en este procesador x86 con un conocido *benchmark*. Dicho procesador es un procesador segmentado, superescalar y con ejecución fuera de orden. En los procesadores segmentados pueden transcurrir decenas de ciclos hasta que se calcula la dirección destino del salto y el procesador decide si el salto se producirá (*taken*) o por el contrario se seguirá ejecutando en secuencia (*not taken*). Hemos obtenido que el 20% de las instrucciones dinámicas son saltos, y que el 75% de los saltos son *taken*. También hemos obtenido que, en un procesador ideal donde los saltos no penalizan, el CPI es de 0,3 ciclos/instrucción.

El procesador tiene un predictor de saltos sencillo que predice todos los saltos como *not taken*. Los saltos *not taken* no incurren en ninguna penalización. Sin embargo, los saltos *taken* incurren en una penalización de 18 ciclos.

e) **Calcula** el CPI del procesador con predictor de saltos sencillo.

CPI = 0,3 c/i + 0,2 saltos/ins * 0,75 taken/salto * 18 ciclos/taken = 3 ciclos/instrucción

Ante la perdida de rendimiento debida a los saltos, se ha decidido incorporar un predictor de saltos mas sofisticado. Este predictor tiene una tasa de aciertos del 97,5%. La penalización por fallo de predicción sigue siendo de 18 ciclos.

f) **Calcula** el CPI del procesador con predictor de saltos sofisticado.

CPI = 0,3 c/i + 0,2 saltos/ins * 0,025 fallos/salto * 18 ciclos/fallo = 0,39 ciclos/instrucción

Los procesadores superescalares leen varias instrucciones cada vez que acceden a la cache de instrucciones, por lo que el número de accesos a la cache es menor que el numero de instrucciones ejecutadas. Sabemos que en el *benchmark* se ejecutan $10x10^9$ instrucciones dinámicas y que se realizan $4x10^9$ accesos a la cache de instrucciones, es decir, se realizan sólo 0,4 accesos a la cache por instrucción. Nuestro procesador traduce las instrucciones x86 a microoperaciones RISC (uops). Cada acceso a la cache consume 2,5 nJ (sea acierto o fallo) y se consumen 2 nJ por cada instrucción x86 descodificada y traducida a uops.

g) **Calcula** la energía consumida por la búsqueda y traducción de las instrucciones x86.

E = $10x10^9$ instr * $2x10^{-9}$ J/instr + $4x10^9$ accesos * $2,5x10^{-9}$ J/accesos = 30 J

Se ha decidido introducir una cache de micro-ops (**uops**) donde se guardan las uops que ya han sido traducidas para que en caso de acierto no sea necesario descodificar y traducir de nuevo las instrucciones x86. Se ha medido que se generan $15x10^9$ uops dinámicas y se realizan $4x10^9$ accesos a la cache de uops. Gracias a esto, el número de accesos a la cache de instrucciones se ha reducido a $0,4x10^9$ accesos y el número de instrucciones x86 descodificadas se ha reducido a $1x10^9$ instrucciones. Un acceso a la cache de uops consume 1,75 nJ.

h) **Calcula** la energía consumida por el sistema con cache de uops.

E = $1x10^9$ instr * $2x10^{-9}$ J/instr + $0,4x10^9$ accesos * $2,5x10^{-9}$ J/acceso + $4x10^9$ accesos * $1,75x10^{-9}$ J/acceso= 10 J

COGNOMS:

NOM:

Problema 3. (3.4 puntos)

La empresa **Can60**, dispone de un servidor formado por los componentes mostrados en la tabla siguiente. La tabla también muestra el número de componentes de cada tipo y el tiempo medio hasta fallo (MTTF) de cada componente:

Componente	Fuente alimentación	CPU	Placa base	DIMMs	Disco duro
Nº	1	1	1	4	1
MTTF (horas)	200.000	1.000.000	200.000	1.000.000	100.000

a) **Calcula** el MTTF del sistema suponiendo que este falla si falla alguno de los componentes.

1/MTTF = 1/2e5 + 1/1e6 + 1/2e5 + 4/1e6 + 1/e5 = 1/40000
MTTF= 40000 horas

El disco duro del servidor es de tipo: **disco1**, pero la empresa posee inventariados en uno de sus 'desordenados' almacenes 23 discos más. En total, contando con el **disco1** del servidor, disponemos de 24 discos físicos: la mitad son de tipo: **disco1** y la otra mitad de tipo: **disco2**. El **disco1** inventariado posee un ancho de banda efectivo de 250 MBytes/s. El ancho de banda efectivo para un **disco2** es de 200 MBytes/s. Todos los discos poseen la misma capacidad. El MTTR de un disco físico cualquiera es de 10 horas.

En este servidor base (1 **CPU** + 1 **disco1**) ejecutamos una aplicación A que consta de dos fases bien diferenciadas:

- **Cálculo**: donde se usa exclusivamente la CPU, y es parcialmente paralelizable. Esta fase supone el 80% del tiempo de ejecución.
- **Entrada/Salida (E/S)**: que prácticamente no usa la CPU, y pasa todo el tiempo accediendo al sistema de almacenamiento para leer y escribir datos. Sabemos que 3/5 del tiempo se realizan lecturas secuenciales, y 2/5 del tiempo se realizan escrituras aleatorias.

Para mejorar el rendimiento y fiabilidad de la parte de E/S de la aplicación A, nos planteamos sustituir el disco1 por alguna configuración de discos lógicos RAID. En base a las características de los dos tipos de discos y la aplicación A, hemos decidido probar las siguientes configuraciones:

- un RAID 5 con los discos1 (discos con mayor ancho de banda efectivo).
- un RAID 6 con los discos2 (configuración con más fiabilidad).

Del RAID 6 con los discos2, sabemos que el MTTF de esta configuración es un valor que está entre $70x10^6$ y $80x10^6$ horas.

b) **Calcula** el MTTF de un RAID 5 con los discos1. **Justifica** tu respuesta indicando la expresión matemática que utilizas para realizar el cálculo.

MTTF = MTTFd^2/(N*(N-1)*MTTR) = 100.000^2/12*11*10 = 7.575.757 horas ---> (peor fiabilidad que RAID6)

Para acabar de decidirnos, evaluaremos algunas características más de las dos configuraciones.

Consideraremos las siguientes definiciones:

- **Ndi:** número de discos físicos del tipo i (Nd1 para disco1, Nd2 para disco2).
- **% útil:** el porcentaje de la capacidad total que es capacidad útil.
- **BW RD Secuencial:** ancho de banda efectivo para el caso de accesos de lectura secuenciales.
- **BW WR Aleatoria:** ancho de banda efectivo para el caso de accesos de escritura aleatorios.

c) **Completa** la siguiente tabla, y **justifica** con una expresión matemática cada uno de los cálculos que has realizado.

	RAID 5 Nd1 = 12	expresión	RAID 6 Nd2 = 12	expresión
% útil	91,6%	(Nd1 - 1)/Nd1 * 100	83,3%	(Nd2 - 2)/Nd2 * 100
BW RD Secuencial	3 GB/s	250MB/s * Nd1	2,4 GB/s	200MB/s * Nd2
BW WR Aleatoria	750MB/s	250MB/s * (Nd1/4)	400MB/s	200MB/s * (Nd2/6)

Elegimos sustituir el disco1 del servidor base (1 **CPU** + 1 **disco1**) por la configuración **RAID 5** con **12 discos1**. Llamaremos sistema E/S_RAID al formado por este RAID 5 (1 **CPU** + 1 **RAID 5**).

d) **Calcula** la ganancia en tiempo (speed-up) de toda la parte de E/S respecto del servidor base al ejecutar la aplicación A con el nuevo sistema E/S_RAID.

3/5 son lecturas secuenciales (0.6) y 2/5 son escrituras aleatorias (0.4)

gR = 12 y gW = 3

Ses_RAID = 1/(0.6/gR + 0.4/gW) = 1/(11/60) = 60/11 = 5.45

Al ejecutar A en el servidor base, la fase de Cálculo, la única donde se realizan cálculos en coma flotante, realiza $250x10^{12}$ operaciones en coma flotante, y la velocidad de ejecución es de 5 GFLOPS. La fase de Cálculo se divide en dos partes:

- una zona secuencial que corresponde al 2% del tiempo total de ejecución, y que no puede paralelizarse.
- una zona completamente paralelizable que corresponde al 98% del tiempo total de ejecución.

Para mejorar el rendimiento de la parte de Cálculo que es paralelizable, decidimos sustituir la única CPU del servidor base por 8 CPUs. Hemos visto que no es posible realizar una paralelización perfecta debido al tiempo de sincronización entre los distintos procesos. Se calcula que este tiempo de sincronización tiene una componente de penalización fija de 425 segundos, y otra penalización variable de 350 segundos por cada CPU que se añade al sistema.

e) **Calcula** el tiempo de ejecución de la fase de Cálculo, teniendo en cuenta la sincronización, utilizando 8 CPUs.

Tcalculo (1 CPU) = $250x10^{12}$ / $5000x10^{6}$ = 50000 segundos

Tcalculo (8 CPUs) = 50000x0.02 + (50000x0.98)/8 + 425 + 350(8-1) = 10000 segundos ---> Scalculo = 5

Finalmente la configuración seleccionada para el servidor consiste en un multiprocesador con el número optimo de CPUs y un sistema de almacenamiento con 4 discos **SSD**. Respecto del servidor base (1 **CPU** + 1 **disco1**) hemos obtenido ganancias de 6.4 para la fase de E/S y de 5.3 para la fase de cálculo al ejecutar A en la nueva configuración de servidor. La aplicación tiene 10^5 instrucciones estáticas y ejecuta $500x10^{12}$ instrucciones dinámicas.

f) **Calcula** los MIPS al ejecutar la aplicación en el servidor base. **Calcula** los MIPS y MFLOPS al ejecutar la aplicación en el servidor del multiprocesador con número óptimo de CPUs + sistema de E/S con 4 **SSD**.

servidor base: Texe = 50000/0.8 = 62500 segundos

MIPS = $500x10^{12}$ instr / $62.5x10^{3}$ segundos * $1M/10^{6}$ = 8000 MIPS

servidor n_optimoCPUs y 4 SSD: Scalculo = 5.3 y Ses_RAID_nuevo = 6.4 ---> S = 1/(0.8/5.3 + 0.2/6.4) = 5.5

MIPS = 8000 MIPS x 5.5 = 44000 MIPS

MFLOPS = ($250x10^{12}$ / $62.5x10^{3}$segundos * $1M/10^{6}$) x 5.5 = 4000 MFLOPS x 5.5 = 22 GFLOPS

COGNOMS:

NOM:

IMPORTANTE leer atentamente antes de empezar el examen: Escriba los apellidos y el nombre antes de empezar el examen. Escriba un solo carácter por recuadro, en mayúsculas y lo más claramente posible. Es importante que no haya tachones ni borrones y que cada carácter quede enmarcado dentro de su recuadro sin llegar a tocar los bordes. Use un único cuadro en blanco para separar los apellidos y nombres compuestos si es el caso. No escriba fuera de los recuadros.

Problema 1. (2 puntos)

Una **CPU** está conectada a una cache y una memoria principal formada por un único módulo DIMM estándar de 16 GBytes. Este DIMM tiene 8 chips de memoria **DDR**-SDRAM (**Double Data Rate** Synchronous DRAM) de 1 byte de ancho cada uno. La DDR-SDRAM tiene 8 bancos. 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 3 ciclos, la latencia de columna de 4 ciclos y la latencia de precarga de 2 ciclos. Cuando se solicitan múltiples bloques a la DDR-SDRAM, el controlador de memoria envía los comandos necesarios a la DDR-SDRAM de forma que los bloques sean transferidos lo más rápidamente posible.

La siguiente tabla muestra en qué banco y qué página de DRAM (fila) se encuentran los bloques etiquetados con las letras A B C D E.

Bloque	A	B	C	D	E
Banco	0	0	1	1	5
Página	10	10	10	25	40

Rellena los siguientes cronogramas para la lectura de varios bloques de 64 bytes (**sin cambiar el orden de los accesos**), en función de la ubicación de los bloques involucrados de forma que se minimice el tiempo total. 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 previamente y que al final deben quedan todas cerradas.

a) **Rellena** el siguiente cronograma para la lectura de los bloques AB.

Ciclo	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
Com	A C			R D				R D								P R	- -										
@	@ F			@ C				@ C																			
Datos								D D	D D	D D	D D	D D	D D	D D	D D												

b) **Rellena** el siguiente cronograma para la lectura de los bloques AC.

Ciclo	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
Com	A C			R D	A C			R D				P R	- -			P R	- -										
@	@ F			@ C	@ F			@ C																			
Datos								D D	D D	D D	D D	D D	D D	D D	D D												

c) **Rellena** el siguiente cronograma para la lectura de los bloques CD.

Ciclo	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
Com	A C			R D								P R	- -	A C			R D								P R	- -	
@	@ F			@ C										@ F			@ C										
Datos								D D	D D	D D	D D										D D	D D	D D	D D			

d) **Rellena** el siguiente cronograma para la lectura de los bloques ADBE.

Ciclo	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
Com	A C			R D	A C			R D				R D	A C			R D	P R	- -		P R	- -			P R	- -		
@	@ F			@ C	@ F			@ C				@ C	@ F														
Datos								D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D				

COGNOMS:

NOM:

Problema 2. (3 puntos)

Un procesador funciona a una frecuencia de 1GHz y esta conectado a una cache de datos nivel 1 (L1) 2-asociativa de 16 KBytes de capacidad con bloques de 64 bytes. La cache no tiene ninguna de las mejoras vistas en clase. En caso de acierto, los accesos a la cache tardan 1 ciclo y en caso de fallo tenemos una penalización media de 20 ciclos. Hemos determinado que la cache se encuentra en el camino crítico del procesador, por lo que la frecuencia (y por tanto el tiempo de ciclo) del procesador vienen determinados por el diseño de la cache.

En la siguiente tabla, las filas indican posibles mejoras (vistas en clase) que podríamos introducir en L1. Las columnas indican algunos parámetros de rendimiento en los que estas mejoras podrían influir, los tiempos tanto de acceso como de penalización se consideraran en **ns** y no en ciclos ya que el tiempo de ciclo puede cambiar. Cada mejora será considerada de forma individual, sin cambiar ningún otro parámetro de la cache. Para indicar el impacto de las mejoras en los parámetros de rendimiento rellenad la tabla usando la siguiente nomenclatura: (**M**) Mejora, (**NA**) No Afecta, (**E**) Empeora.

a) **Indica** el impacto de las mejoras en los parámetros de rendimiento.

	Parámetro de rendimiento			
Mejora/cambio	Frecuencia del procesador	Tiempo (ns) de acceso a la cache L1 en caso de acierto	Tasa de fallos de la cache L1	Tiempo (ns) medio de penalización en caso de fallo
Cambiar a cache Directa de 16 KBytes	M	M	E	NA
Añadir una cache L2 de 256 KBytes	NA	NA	NA	M
Aumentar el tamaño de la cache a 32 KBytes	E	E	M	NA
Añadir continuación anticipada	NA	NA	NA	M
Segmentar la cache en 2 etapas	M	E	NA	NA
Organizar la cache en 4 bancos	M	M	NA	NA
Añadir un mecanismo de prefetch hardware	NA	NA	M	M
Hacer la cache no bloqueante	NA	NA	NA	M

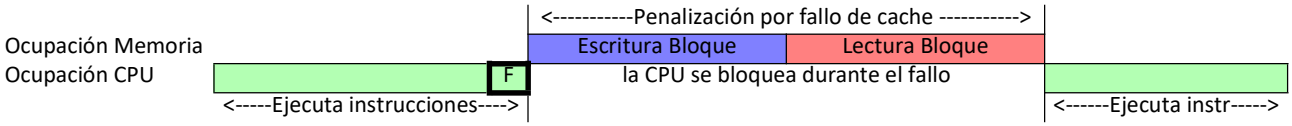
Nota 1: El apartado vale 1.5 puntos, cada fallo o cuadro en blanco resta 0.1 puntos en el apartado (mínimo 0).

Nota 2: La decisión de si un cambio mejora, empeora o no afecta a un parámetro debéis basarla en el caso general. Por ejemplo: aunque en determinadas circunstancias un predictor de vía podría empeorar el consumo de energía, en general lo mejorará.

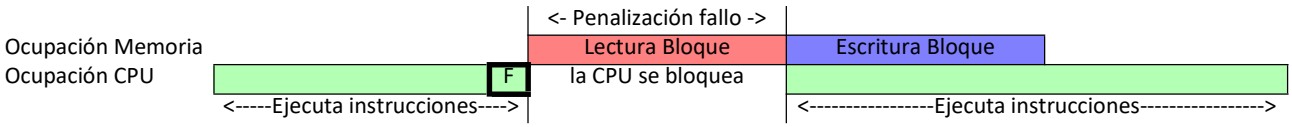
Finalmente se ha decidido estudiar el impacto de las políticas de escritura.

Se ha simulado la ejecución de un programa P en un procesador que denominaremos IDEAL. En este procesador IDEAL no hay ninguna penalización por fallo de cache. De esta simulación se ha obtenido que el programa P se ha ejecutado en $5x10^9$ ciclos durante los que ha ejecutado $2x10^9$ instrucciones, de las que $500x10^6$ son instrucciones de acceso a datos (Load/Store) y se han producido $50x10^6$ fallos en la cache de datos (los fallos en la cache de instrucciones son negligibles, con lo que los ignoraremos durante todo el problema).

El mismo programa lo ejecutamos en un procesador real con las mismas características que el IDEAL con la única diferencia que en caso de fallo en la cache de datos se bloquea la ejecución de instrucciones durante un cierto numero de ciclos hasta que se resuelve el fallo. La cache de datos sigue una política de escritura COPY BACK + WRITE ALLOCATE. En caso de fallo, si la línea reemplazada ha sido modificada, la penalización es de 30 ciclos, mientras que si no lo ha sido es sólo de 15 ciclos. Se sabe que durante la ejecución de P, en media 1/3 de los bloques reemplazados han sido modificados (dirty bit = 1). La siguiente figura muestra el cronograma de un fallo en que la línea reemplazada ha sido modificada.



Una posible mejora (vista en clase de teoría) consiste en incorporar un buffer para almacenar el bloque reemplazado, de forma que podamos leer primero el bloque que ha provocado el fallo y a continuación escribir en memoria el bloque reemplazado. Esta implementación permite que el procesador pueda seguir ejecutando instrucciones y la cache pueda ser accedida durante la escritura del bloque reemplazado a memoria, tal como muestra la siguiente figura.

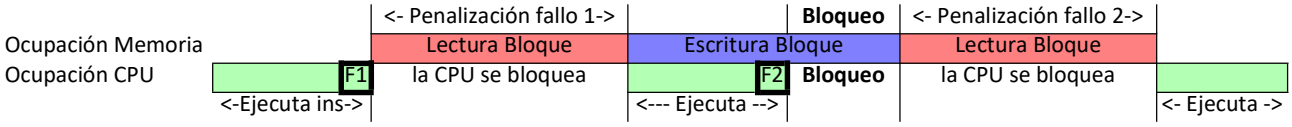


De momento supondremos la situación ideal (aunque no realista) en que nunca se produce un fallo de cache mientras se realiza la escritura del bloque reemplazado. A este procesador lo denominaremos procesador B

b) **Calculad** el numero de ciclos necesario para ejecutar P en el procesador B

penalización por fallo = 15 ciclos
$ciclos_B = ciclos_{IDEAL}$ + #fallos * Penalización per fallo
$ciclos_B = 5x10^9$ ciclos + $50x10^6$ fallos * 15 ciclos/fallo = $5,75\ x10^9$ ciclos

En la implementación real (que denominamos procesador C) dispondremos de un buffer de una sola entrada que nos permite tener como máximo una escritura pendiente. Si durante la escritura del bloque causada por un fallo (F1) se produce un segundo fallo (F2), hay que esperar a que la jerarquía de memoria complete la escritura del bloque antes de que pueda empezar a servir el siguiente fallo, con lo que el procesador se bloqueará por unos ciclos adicionales (**Bloqueo**), tal como muestra la siguiente figura (en el ejemplo suponemos que el siguiente fallo reemplaza un bloque no modificado).



Durante la fase en que la CPU ejecuta instrucciones, todos los ciclos tienen la misma probabilidad de que se produzca un nuevo fallo y sabemos que el número medio de ciclos en que la CPU ejecuta instrucciones (sin contar los bloqueos) entre dos fallos es de 100 ciclos.

c) **Calculad** la probabilidad de que se produzca un segundo fallo durante la escritura de un bloque reemplazado

probabilidad de fallar en un ciclo es p = 1/100 (inversa del tiempo medio entre fallos)
probabilidad de tener un fallo en 15 ciclos (intervalo de servicio de F1) es 1 - probabilidad de no fallar en ningún ciclo (repetimos un proceso independiente 15 veces con probabilidad p)

P(fallo en el intervalo) = $1 - (1 - p)^{15} = 1-(1-1/100)^{15} = 0,14$

Hemos medido que, si se produce un segundo fallo durante el intervalo de escritura de un bloque anterior, en media la CPU se bloquea durante 7 ciclos (correspondientes al intervalo **Bloqueo** de la figura anterior) antes de que se pueda iniciar la lectura del bloque que ha causado el segundo fallo.

d) **Calculad** el numero de ciclos necesario para ejecutar P en el procesador C

penalización por bloqueo = 7 ciclos
Solo hay bloqueo con probabilidad P si el bloque ha sido modificado

$ciclos_C = ciclos_B$ + #fallos * bloques modificados *probabilidad bloqueo * Penalización media por bloqueo
$ciclos_C = 5,75\ x10^9$ ciclos + $50x10^6$ fallos * 1/3 * 0,14 bloqueos/fallo * 7 ciclos/bloqueo = $5,77\ x10^9$ ciclos

COGNOMS:

NOM:

Problema 3. (1.5 puntos)

Queremos estudiar el efecto de la búsqueda y decodificación de instrucciones en el rendimiento de un procesador segmentado superescalar con ejecución fuera de orden. Para ello, simulamos la ejecución de un programa de prueba P en un simulador parametrizable. Para simplificar el problema, nos centraremos exclusivamente en los accesos a instrucciones, y supondremos que en los accesos a datos nunca se produce un fallo al acceder a la cache de datos.

En los procesadores segmentados pueden transcurrir decenas de ciclos hasta que se calcula la dirección destino del salto y el procesador decide si el salto se realizará (*taken*) o por el contrario el programa seguirá ejecutándose en secuencia (*not taken*). La alternativa más simple consiste en ejecutar por defecto las instrucciones de forma secuencial, con lo que los saltos *not taken* no incurren en ninguna penalización. Sin embargo, en nuestro caso los saltos *taken* incurren en una penalización de 10 ciclos. A este procesador le llamaremos PSeq.

Con el simulador hemos obtenido que el 30% de las instrucciones dinámicas del programa P son saltos, y que el 80% de los saltos son *taken*. También hemos obtenido que, en un procesador ideal donde los saltos *taken* no penalizan, el CPI sería de 0,6 ciclos/instrucción.

a) **Calcula** el CPI del procesador PSeq.

CPI = 0,6 c/i + 0,3 saltos/ins * 0,80 taken/salto * 10 ciclos/taken = 3 ciclos/instrucción

Para mejorar el rendimiento del procesador se ha decidido incorporar un predictor de saltos al procesador PSeq, con el que se ha obtenido un CPI de 0,93 ciclos/instrucción. En este diseño, la penalización debida a un salto mal predicho es de 22 ciclos, mientras que los saltos bien predichos no tienen penalización.

b) **Calcula** la tasa de fallos del predictor de saltos.

0,93 c/i = 0,6 i/c + 0,3 saltos/ins * TF fallos/salto * 22 ciclos/fallo
TF = 0,05 fallos/salto

Finalmente, se ha optado por usar un predictor de saltos más sofisticado que tiene una tasa de aciertos del 98% con el que se obtiene un CPI de 0,8 ciclos/instrucción. Hasta ahora no hemos considerado los fallos en la cache de instrucciones. Los procesadores superescalares leen varias instrucciones cada vez que acceden a la cache de instrucciones, por lo que el número de accesos a la cache es menor que el numero de instrucciones ejecutadas. Sabemos que en P se ejecutan $10x10^9$ instrucciones dinámicas y que se realizan $5x10^9$ accesos a la cache de instrucciones, es decir, se realizan sólo un 50% de accesos a la cache por instrucción. Sabemos además que la cache de instrucciones tiene una tasa de fallos de 0,04 fallos/acceso y la penalización es de 100 ciclos/fallo.

c) **Calcula** los ciclos perdidos debidos a los fallos en la cache de instrucciones y el CPI real del procesador.

Ciclos = $5x10^9$ accesos * 0,04 fallos/acceso * 100 ciclos/fallo = $20x10^9$ ciclos
CPI = 0,8 c/i + $20x10^9$ ciclos / $10x10^9$ instrucciones = 2,8 ciclos/instrucción

COGNOMS:

NOM:

Problema 4. (3,5 puntos)

La empresa ACME nos ha contratado para reducir el tiempo de ejecución de una aplicación. Hemos ejecutado dicha aplicación en un ordenador PC1 con un solo procesador y un solo disco, y hemos comprobado que la aplicación ejecuta de forma iterativa tres fases bien diferenciadas, tal como se ve en la figura:

```
for (;;) {
  LeerDatos();
  ProcesarDatos();
  EscribirResultados();
}
```

• **Fase 1:** LeerDatos(), únicamente se hacen lecturas desde disco. Ocupa el 25% de toda la ejecución en el PC1.

• **Fase 2:** ProcesarDatos(), parte intensiva en cálculo que es totalmente paralelizable. Ocupa el 60% de la ejecución en el PC1.

• **Fase 3:** EscribirResultados(), los datos recién calculados se escriben a disco. Ocupa el 15% de la ejecución en el PC1.

El tiempo de ejecución de la aplicación en el PC1 es de 120 horas. Queremos estudiar diferentes alternativas que nos permitan reducir este tiempo.

a) **Calcula** el Speedup máximo que podríamos obtener si ejecutáramos este programa en un supercomputador con un número infinito de procesadores y un único disco.

S = 1 / (1-0.6) = 2,5

b) **Calcula** con cuántos procesadores tenemos que ejecutar el programa para obtener un Speedup total de 2.

El número de procesadores es igual al speed up que se obtendrá en la fase 2
S = 1 / (1-0.6+0,6/p) = 2 --> p = 6

Dado que gran parte de la aplicación puede paralelizarse completamente, ofrecemos a la empresa reemplazar el ordenador PC1 por un nuevo ordenador PC6 con un procesador multi-core de 6 núcleos.

Hemos comprobado que el 80% de la Fase 2 se dedica a hacer operaciones matemáticas entre varias matrices de enteros de 4 bytes en las que cada cálculo es independiente de los demás, así que decidimos transformar el código en esas regiones para poder utilizar instrucciones SIMD. La arquitectura de los núcleos del procesador nos permite utilizar registros xmm de 128 bits.

c) **Calcula** el Speedup de la Fase 2 después de aplicar la transformación en el código para usar instrucciones SIMD y de ejecutarlo en el PC6. **Calcula** el tiempo de ejecución de la aplicación después de estas mejoras.

$S_{fase2+SIMD}$ = 1 / (0.2/6+0.8/(6*4)) = 15

S = 1 / (1-0.6+0.6/15) = 2,27
T = 120h / 2,27 = 52,86 h

Para reducir el tiempo de las fases de lectura/escritura de datos necesitaremos un sistema de almacenamiento que nos proporcione mayor ancho de banda que el actual. Además, queremos añadir algún mecanismo de tolerancia a fallos en disco. Valoramos utilizar los sistemas de almacenamiento RAID 10 (con mirror doble), RAID 5 y RAID 6. La aplicación necesita un total de 24 terabytes de almacenamiento.

Para implementar los diversos sistemas RAID disponemos de un único modelo de disco duro DD de 4TB con un ancho de banda de 100MB/s, tanto en lectura como en escritura.

d) **Calcula** cuántos discos duros DD necesitaríamos para implementar cada uno de los niveles de RAID 10 (mirror doble), RAID 5 y RAID 6 si queremos almacenar al menos 24TB útiles

RAID 10 -> 24 TB * 2 mirrors / 4 TB/disco = 12 discos
RAID 5 -> 24 TB / 4 TB/disco + 1 paridad = 7 discos
RAID 6 -> 24 TB / 4 TB/disco + 2 paridad = 8 discos

e) Después de explicar a la empresa ACME las diferencias entre los tres niveles de RAID, la empresa se decanta por montar un RAID5 con 8 discos duros DD. Dibuja el esquema del RAID 5 que nos ha pedido la empresa indicando claramente el tipo de entrelazado, cómo se distribuye la paridad entre los discos, y el ancho de banda máximo que se conseguirá en el RAID5 en lectura secuencial, en escritura secuencial y en escritura aleatoria.

Entrelazado a nivel de tira,
AB lectura secuencial = 8 discos * 100 MB/s /disco = 800 MB/s
AB escritura secuencial = 7 discos * 100 MB/s /disco = 700 MB/s
AB escritura aleatoria = 8 discos * 100 MB/s /disco / 4 accesos/escritura= 200 MB/s

f) **Calcula** el Speedup y el tiempo total que tarda la aplicación respecto a ejecutarla en el PC1 al ejecutarla en el PC6 + SIMD con un RAID 5 de 8 discos), teniendo en cuenta que todos los accesos a disco son secuenciales.

S_{f1} = 800 / 100 = 8

S_{f2} =15

S_{f3} = 700 / 100 = 7

S = 1 / (0.25/8 + 0.6/15 + 0.15/7) = 10,79

T = 120h / 10,79 = 11,12h

COGNOMS:

NOM:

IMPORTANTE leer atentamente antes de empezar el examen: Escriba los apellidos y el nombre antes de empezar el examen. Escriba un solo carácter por recuadro, en mayúsculas y lo más claramente posible. Es importante que no haya tachones ni borrones y que cada carácter quede enmarcado dentro de su recuadro sin llegar a tocar los bordes. Use un único cuadro en blanco para separar los apellidos y nombres compuestos si es el caso. No escriba fuera de los recuadros.

Problema 1. (2,5 puntos)

En un procesador Q1 con direcciones de 32 bits, el camino crítico, y por tanto el tiempo de ciclo, está limitado por la memoria cache de datos. El tiempo de retardo de los componentes de la memoria cache de datos se desglosa de la siguiente forma:

Componente	Tiempo
Memoria de etiquetas	0,32 ns
Comparación de etiquetas y (en caso necesario) **selección de vía**	0,18 ns
Memoria de datos y selección de byte de la linea	0,45 ns
Mux de vía de datos: selecciona el dato de la vía correspondiente (cuando sea necesario)	0,15 ns
Registro de desacoplo (cuando sea necesario)	0,05 ns

Queremos analizar 2 configuraciones para la cache de datos, todas ellas con 32KB de capacidad y lineas de 16 bytes:

- C1: Cache de mapeo directo con acceso PARALELO a etiquetas y datos.
- C2: Cache asociativa por conjuntos de dos vías SEGMENTADA en 2 etapas (el tiempo de acceso a la cache son 2 ciclos de procesador).

a) **Calcula** el tiempo de ciclo de la cache de datos y el tiempo total de un acceso para las diferentes versiones del procesador Q1 con las caches C1 y C2, usando la distribución más adecuada de los componentes por etapas.

C1: Tc= MAX(0,32+0,18; 0,45)= 0,5ns Tsa= 1 ciclo

C2: Tc= MAX(0,32+0,18+0,05; 0,45+0,15+0,05)= 0,65ns Tsa= 2 ciclos

b) **Calcula** la frecuencia de reloj para las diferentes versiones del procesador Q1 con las caches C1y C2.

F= 1/Tc

C1: 1/0,5ns= 2GHz

C2: 1/0,65ns= 1,538GHz

Un programa P que ejecuta $2{,}5x10^9$ instrucciones tiene un 50% de instrucciones aritméticas, un 20% de instrucciones de salto y un 30% de instrucciones de acceso a memoria (Load/Store). Las instrucciones aritméticas tardan 4 ciclos, las de salto 3 y las de memoria 5 ciclos + los ciclos del acceso a la cache.

c) **Calcula** el CPI del programa P para los procesadores con C1 y C2 suponiendo que nunca hay fallos en la cache de datos.

CPI_C1= 0,5*4 + 0,2*3 + 0,3*(5+1)= 4,4

CPI_C2= 0,5*4 + 0,2*3 + 0,3*(5+2)= 4,7

Sabemos que el programa P tiene un 10% de fallos con la cache de datos C1 y un 6% con la C2. Además, el tiempo de penalización medio por fallo en ambos casos es de 60 ciclos.

d) **Calcula** el speedup en tiempo de ejecución de C2 sobre C1 en % teniendo en cuenta la jerarquía de memoria completa.

$T=I*(CPI_{id}+CPI_{mem})*Tc$

T_C1= 2,5*10^9*(4,4 + 0,3*0,1*60)*0,5*10^(-9)= 7,75s
T_C2= 2,5*10^9*(4,7 + 0,3*0,06*60)*0,65*10^(-9)= 9,39s

SpeedUp= 7,75/9,39= 0,825 ==> Slowdown de 1/0,825 ==> 21,16% Slowdown o -17,5% de Speedup

Se hace una implementación multibanco de la cache C2, organizada en 4 bancos 2-asociativos, con entrelazado a nivel de bloque.

e) **Indica** cómo se desglosarían los bits de una dirección entre bits de Etiqueta, selección de Conjunto, Banco y Byte.

Etiqueta <31:14>	Conjunto <13:6>	Banco <5:4>	Byte <3:0>

4 bancos ==> 2 bits para seleccionar banco.
Entrelazado a nivel de bloque ==> 2 bits de menos peso del conjunto para seleccionar el banco.

COGNOMS:

NOM:

Problema 2. (2.5 puntos)

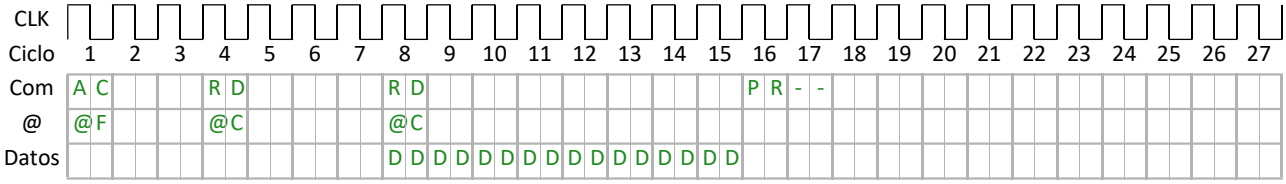
Una **CPU** está conectada a una cache de instrucciones (**$I**) y una cache de datos (**$D**). El conjunto formado por **CPU**+**$I**+**$D** esta conectado a una memoria principal formada por un único módulo DIMM estándar de 16 GBytes. Este DIMM tiene 8 chips de memoria **DDR**-SDRAM (**Double Data Rate** Synchronous DRAM) de 1 byte de ancho cada uno. La DDR-SDRAM tiene 2 bancos. 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 3 ciclos, la latencia de columna de 4 ciclos y la latencia de precarga de 2 ciclos. Es posible que el conjunto **CPU**+**$I**+**$D** solicite múltiples bloques a la DDR (por ejemplo porque se produzca un fallo simultáneamente en **$I** y en **$D**). El controlador de memoria envía los comandos necesarios a la DDR-SDRAM de forma que los bloques sean transferidos lo más rápidamente posible y se maximice el ancho de banda.

La siguiente tabla muestra en qué banco y qué página de DRAM (fila) se encuentran los bloques etiquetados con las letras A B C D.

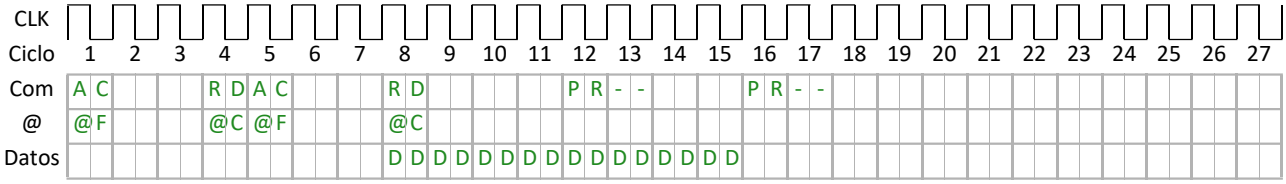
Bloque	A	B	C	D
Banco	0	0	1	1
Página	10	10	10	25

Rellena los siguientes cronogramas para la lectura de varios bloques de 64 bytes (en el orden que se indica), en función de la ubicación de los bloques involucrados de forma que se minimice el tiempo total. 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 previamente.

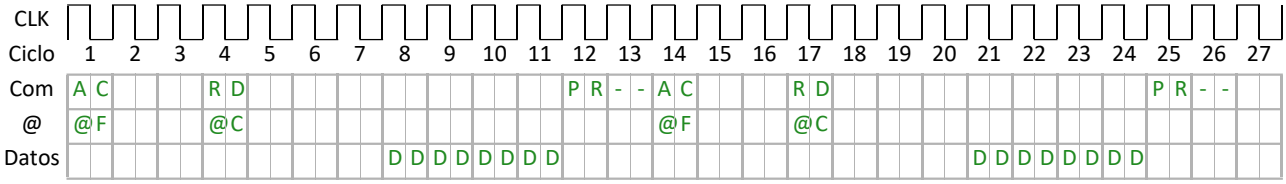
a) **Rellena** el siguiente cronograma para la lectura de los bloques AB.



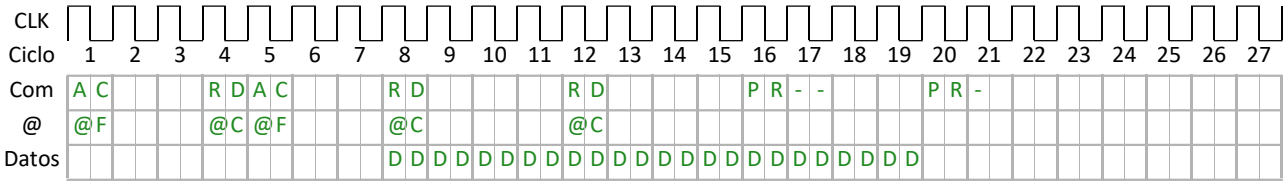
b) **Rellena** el siguiente cronograma para la lectura de los bloques AC.



c) **Rellena** el siguiente cronograma para la lectura de los bloques CD



d) **Rellena** el siguiente cronograma para la lectura de los bloques ADB.



En esta CPU ejecutamos un programa, en el que se detecta que el bucle **`for`** del siguiente fragmento de código consume la mayor parte de el tiempo de ejecución:

```
/* Variables globales */
float A[1024*1024]; float B[1024*1024]; /* un float ocupa 4 bytes */
.....
/* codigo */
for (i=0;i<1024*1024; i++)
   A[i] = A[i] + B[i];
```

Un análisis detallado muestra que los fallos en la cache de instrucciones son despreciables, pero que los fallos en la cache de datos son excesivos (mas del 60%). Sabemos que la cache de datos es de mapeo directo, con bloques de 64 bytes y política de escritura copy back + write allocate. Mediante el uso del debugger hemos averiguado que el compilador almacena las variables A y B consecutivas en memoria y que se encuentran respectivamente en las direcciones 0x00400000 y 0x00800000

e) **Indica** a qué son debidos los fallos. **Realiza** una optimización de código, de las vistas en clase, que minimice los fallos en la cache de datos.

Los fallos son debidos a conflictos entre el vector A y el B. A y B se reemplazan mutuamente en cada iteración por lo que tendremos fallo tanto en A como en B en todas las iteraciones. Aprovechamos la localidad temporal de A pero no la espacial (ni en A ni en B).

Se pueden minimizar los fallos haciendo padding, el padding tiene que ser como mínimo del tamaño de bloque:

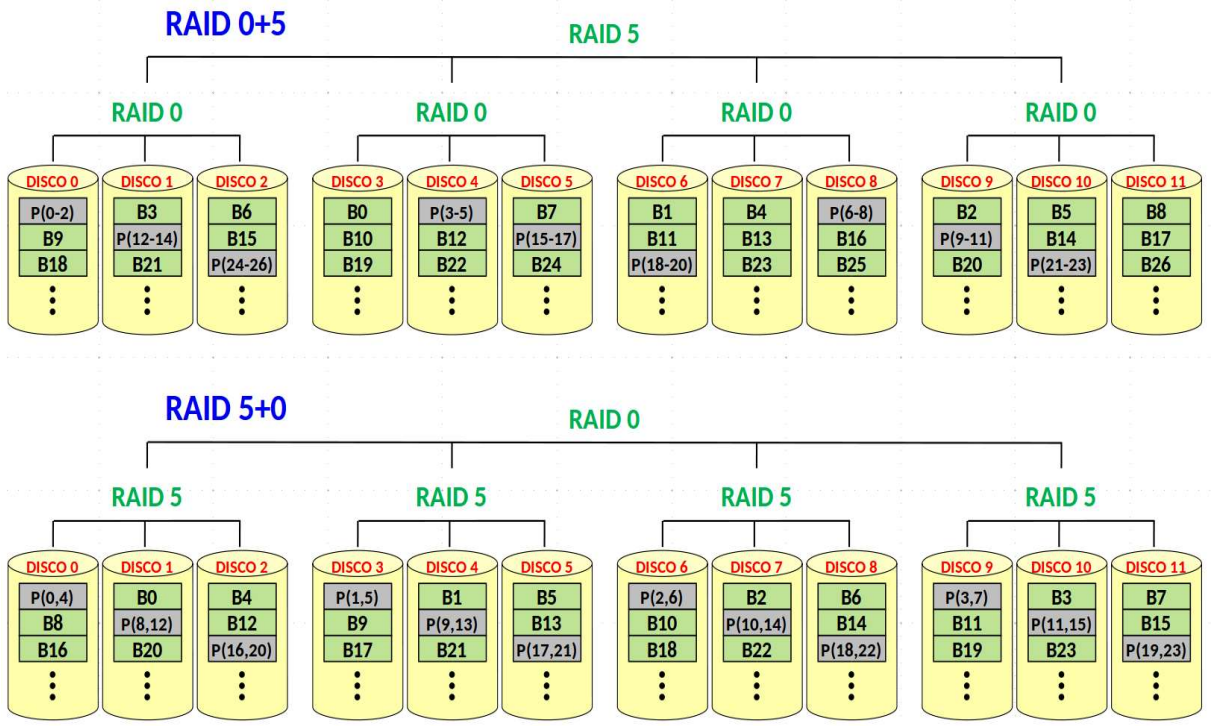
```
float A[1024*1024]; float pad[16]; float B[1024*1024];
```

COGNOMS: | | | | | | | | | | | | | | | | | | | | | | | | | | |

NOM: | | | | | | | | | | | | | | |

Problema 3. (2.5 puntos)

La figura muestra las 2 posibles combinaciones vistas en clase para montar un disco lógico multi-RAID combinando RAID 0 y RAID 5. Se tratan del RAID 05 y RAID 50 que, para un caso general, se forman con N discos y G grupos de dG discos por grupo. De la figura observamos cómo la paridad que introduce RAID 5 se calcula de diferente forma en RAID 05 que en RAID 50. Esto afecta a la capacidad útil y a la fiabilidad que ofrece cada sistema.



a) **Indica** la expresión general simplificada para calcular la capacidad útil de información que se puede almacenar en RAID 50 y en RAID 05. Debes utilizar para ello las variables N, G, dG y llamar T_d al tamaño en GBytes de cada disco. Indica el resultado en GBytes.

RAID 05: $N * T_d - dG * T_d = (N - dG) * T_d$ GBytes

RAID 50: $N * T_d - 1 * G * T_d = (N - G) * T_d$ GBytes

Para las dos configuraciones del caso particular de la figura se cumple que **N = 12, G = 4 y dG = 3**. Sin embargo, las dos poseen distinta capacidad útil, y RAID 50 es el que proporciona mejor tolerancia a fallos.

b) El RAID 50 de la figura puede seguir funcionando aunque fallen 4 discos. **Marca** con una cruz en la tabla siguiente 4 discos que podrían fallar y el RAID 50 seguir funcionando. Utiliza la fila de la tabla etiquetada con **(b)**.

c) El mismo RAID 50 de la figura puede dejar de funcionar si fallan 2 discos. **Marca** con una cruz en la tabla siguiente 2 discos que deberían fallar para que el RAID 50 no funcione. Utiliza la fila de la tabla etiquetada con **(c)**.

	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**
(b)	X			X				X			X	
(c)	X	X										

Montamos un disco lógico con la configuración RAID 50 con valores **N = 15, G = 3 y dG = 5**. El $MTTF_{\_d}$ de un disco es de 60.000 horas y el tiempo de reemplazar un disco y reconstruir la información es MTTR = 30 horas. Sabemos que $MTTF_{\_RAID\ 50}$ coincide con el MTTF de uno de sus grupos ($MTTF_{\_grupo}$) dividido por el número de grupos (G).

d) **Escribe** la expresión general del $MTTF_{\_RAID\ 50}$ suponiendo que los discos son el único componente que puede fallar. Debes utilizar para ello las variables N, G, dG, MTTF_d y MTTR. **Calcula** el valor de este $MTTF_{\_RAID\ 50}$ para los valores proporcionados.

$MTTF_{\_RAID\ 50} = MTTF_{\_grupo} / G$ y 1 grupo = RAID 5 con 5 discos (dG = 5)

$MTTF_{\_RAID\ 50} = (MTTF_{\_d} / dG) * (MTTF_{\_d} / ((dG - 1) * MTTR)) * (1 / G) = MTTF_{\_d}\ ^2 / (dG * (dG -1) * G * MTTR)$

$MTTF_{\_RAID\ 50} = 60.000\ ^2 / (5 * (5 - 1) * 3 * 30) = 2.000.000$ horas

Queremos evaluar el rendimiento de utilizar esta misma configuración RAID 50 con **N = 15, G = 3 y dG = 5**. Utilizamos para ello discos de 1 TByte, tamaño de sector de 512 Bytes, y un mismo ancho de banda de 256 MB/s por disco tanto para leer como para escribir un bloque de datos. Un bloque de datos está formado por 50000 sectores consecutivos. Ejecutamos una aplicación formada por 4 fases (en las fases 1, 3 y 4 sólo consideramos el tiempo de la transferencia):

- La fase 1 lee de disco los datos de entrada formados por 100 bloques de datos distribuidos entre todos los discos.
- La fase 2 realiza los cálculos, con un tiempo de ejecución de 4 segundos.
- La fase 3 escribe a disco 50 bloques de datos realizando **escrituras secuenciales**.
- La fase 4 escribe a disco 50 bloques de datos realizando **escrituras aleatorias**, distribuidas uniformemente entre todos los discos.

e) **Calcula** el tiempo de ejecución de nuestra aplicación si utilizamos un único disco. **Calcula** también el tiempo de ejecución de la aplicación cuando usamos el RAID 50 de **N = 15, G = 3 y dG = 5**.

bloque de datos = 50000 sectores = 25,6 MB

fase 1: 100 bloques * 25,6 MB = 2,56 GB ---> 2,56 GB / 256 MB/s = 10 s

fase 3: 50 bloques * 25,6 MB = 1,28 GB ---> 1,28 GB / 256 MB/s = 5 s

fase 4: igual que fase 3 = 5 s

Total con un único disco: 10 + 4 + 5 + 5 = 24 s

fase 1: 2,56 GB / (15 * 256 MB/s) = 10 s / 15 = 0,66 s

fase 3: 1,28 GB / ((15 - 3) * 256 MB/s) = 5 s / 12 = 0,41 s

fase 4: 1,28 GB / ((15 / 4) * 256 MB/s) = 5 s / 3,75 = 1,33 s

Total con RAID 50: 0,66 + 4 + 0,41 + 1,33 = 6,4 s

COGNOMS:

NOM:

Problema 4. (2.5 puntos)

Queremos evaluar un servidor web multi-thread en un sistema con un multiprocesador. El programa, que se ejecuta de forma permanente, se compone de dos threads de servicio y dos threads multimedia. Los threads de servicio se dedicarán a atender las peticiones HTTP y, de forma inherente, estarán siempre activos el 100% del tiempo. Los threads multimedia estarán dedicados al tratamiento de imágenes, necesario para atender las peticiones de los usuarios, y estos estarán activos sólo cuando se requiera según la carga del servidor, que mediremos en una media de $3.6 \cdot 10^{13}$ operaciones de coma flotante por hora a repartir entre los dos threads multimedia. Esta carga será constante en todo el problema.

Dado que tenemos cuatro threads en total, evaluaremos configuraciones con cuatro cores, en el que asumiremos que siempre tenemos un único thread asignado a cada core. Por simplicidad, podremos hablar de cores de servicio y cores multimedia. Para un prodesador de 4 cores tendremos 2 cores de servivio y 2 cores multimedia.

a) **Calcula** en GFLOPS el rendimiento mínimo que debe tener un procesador para poder ejecutar toda la carga de trabajo del servidor. **Calcula** los GFLOPS mínimos por cada core multimedia .

GFLOPS total = $3.6*10^{13}$ flop * (1 gflop/ 10^9 flop) / 3600 s = 10 GFLOPS

GFLOPS por core = 10 GFLOPS / 2 = 5 GFLOPS

Evaluamos el programa en un multiprocesador CISC que llamaremos C1. Este multiprocesador se compone de cuatro cores idénticos. Cada core trabaja a una frecuencia de 1.6 GHz, tiene una corriente de fuga de 2 A, se alimenta a un voltaje de 1.5 V y tiene una carga capacitiva equivalente de 5 nF. El consumo debido a cortocircuito es despreciable.

b) **Calcula** la potencia disipada por el chip C1 al ejecutar el servicio web.

P_{fugas} = I*V = 2 A * 1.5 V = 3 W por core
P_{comm} = $C*V^2*F$ = 5 nF * $(1.5\ V)^2$ * 1.6 GHz = 18 W por core
P_{C1} = (3 W + 18 W) * 4 cores = 84 W

El chip C1 no implementa escalado de frecuencia, es decir que cada core mantiene el voltaje y la frecuencia independientemente de que el thread esté activo o no. La evaluación de este sistema indica que:

- Cada uno de los threads de servicio está activo el 100% del tiempo.
- Con la carga de trabajo que va a soportar el servidor, vemos que cada uno de los threads multimedia está activo sólo el 60% del tiempo.

Disponemos de un procesador más avanzado que llamaremos C2, con las mismas características que C1 pero que implementa voltaje dinámico y escalado de frecuencia. Este procesador puede configurar individualmente cada core para trabajar en modo bajo consumo, normal (como C1), o turbo. En bajo consumo, un core del chip C2 reduce su voltaje a 0.8 V y su frecuencia a 1.2 GHz. En modo turbo, un core incrementa su voltaje a 2 V y su frecuencia a 1.8 GHz.

Ejecutamos la aplicación en el chip C2 y vemos que los cores de servicio pueden estar todo el tiempo en modo bajo consumo, aunque sus threads asociados estén siempre activos. También observamos que los cores multimedia van cambiando entre bajo consumo y turbo en función de si su thread asociado está activo; y, debido al aumento de frecuencia del modo turbo, también vemos que el tiempo que los threads multimedia están activos es menor.

c) **Calcula**, en porcentaje, cuánto tiempo está en modo turbo cada uno de los cores multimedia en el chip C2 al ejecutar el servicio web (un core multimedia estará en modo turbo sólo cuando su thread asociado esté activo).

Aumenta la frencuencia de 1.6 GHz a 1.8 GHz:
S = 1.8 / 1.6 = 1.125
60% / 1.125 = 53.3 %

d) **Calcula** la potencia media disipada por el chip C2 mientras se ejecuta el servicio web.

$P_{fugas\ bajo\ consumo}$ = 2 A * 0.8 V = 1.6 W
$P_{comm\ bajo\ consumo}$ = 5 nF * $(0.8\ V)^2$ * 1.2 GHz = 3.84 W
$P_{fugas\ turbo}$ = 2 A * 2 V = 4 W
$P_{comm\ turbo}$ = 5 nF * $(2\ V)^2$ * 1.8 GHz = 36 W

P_{C2} = (1.6+3.84) * (2 + 2*(1-0.533)) + (4+36)*2*0.533 = 58.6 W

Queremos analizar el rendimiento de la aplicación en un procesador RISC. Es común en estos procesadores tener cores heterogéneos y activar uno u otro en función de la carga de trabajo con el fin de reducir el consumo. ARM llamó a esta tecnología big.LITTLE, y en un principio el control era completamente hardware, y o bien se activaban los cores big, o bien los LITTLE. Ahora, los diseños más modernos ya permiten que el S.O. sea consciente de los cores heterogéneos y puede asociar un thread concreto a un core específico según una serie de análisis e inferencias. Creemos que esta tecnología nos puede ir muy bien porque los threads de servicio podrían ejecutarse perfectamente en cores sencillos, como los LITTLE, y los thread multimedia en los cores que ofrecen más rendimiento.

Estudiamos el mercado y podemos elegir entre estos chips RISC que implementan multi-procesamiento heterogéneo:

Chip	Número de cores	Consumo total	Rendimiendo máximo por core big
R1	2 big, 2 LITTLE	1.5 W	2 GFLOPS
R2	2 big, 2 LITTLE	2 W	2.5 GFLOPS
R3	4 big, 4 LITTLE	5 W	3 GFLOPS

Estos modelos nos ofrecen un consumo mucho menor en comparación a los anteriores, pero a cambio el rendimiento de los cores big (que irán destinados a ejecutar los thread multimedia) también se ve reducido. Asumiendo la misma carga de peticiones que hemos analizado anteriormente, hemos de asegurarnos que los threads multimedia podrán ejecutar la misma carga de trabajo. Debido a ello, consideramos también algún modelo con cuatro cores big y asumiremos que la tarea multimedia es perfectamente paralelizable y que no conlleva penalización de sincronización. Además, para estos chips asumiremos que siempre podemos llegar a los GFLOPS del rendimiento máximo y que no hay voltaje dinámico, y por tanto el consumo será siempre estable.

e) **Justifica** si alguno de los chips RISC podría servirnos para la aplicación, y elige cuál (R1, R2, R3, o ninguno). Si lo hubiera, el chip idóneo será aquel con mejor rendimiento por consumo. **Calcula** los GFLOPS/w del chip.

Mínimo necesitamos 10 GFLOPS, el único chip que nos servirá será R3, que nos ofrece 3 x 4 = 12 GFLOPS.

GFLOPS/w = 12 GFLOPS / 5 W = 2.4

Tras analizar el algoritmo multimedia, observamos que durante el 10% del tiempo se ejecuta una rutina que ejecuta las operaciones óptimas, pero durante el otro 90% del tiempo se ejecuta otra rutina que con otro algoritmo de cálculo podría optimizarse para hacer el mismo trabajo con menos operaciones. Queremos estudiar cuánto habría que mejorar el algoritmo de esta rutina para poder ejecutar nuestro programa en el chip R1, consiguiendo así un consumo mínimo.

f) **Calcula** la ganancia que debemos aplicar al algoritmo no optimizado para poder ejecutar la aplicación en el chip R1.

Si el algoritmo actual necesita 10 GFLOPS de rendimiento y queremos ejecutarlo en un chip que ofrece 4 GFLOPS, necesitamos una ganancia global de 10 / 4 = 2.5.

2.5 = 1 / (0.1 + 0.9/s) -> s = 3

COGNOMS:

NOM:

IMPORTANTE leer atentamente antes de empezar el examen: Escriba los apellidos y el nombre antes de empezar el examen. Escriba un solo carácter por recuadro, en mayúsculas y lo más claramente posible. Es importante que no haya tachones ni borrones y que cada carácter quede enmarcado dentro de su recuadro sin llegar a tocar los bordes. Use un único cuadro en blanco para separar los apellidos y nombres compuestos si es el caso. No escriba fuera de los recuadros.

Problema 1. (3 puntos)

Disponemos de un procesador de señal (DSP) que ejecuta el siguiente kernel (obsérvese que es un bucle infinito). **Las variables cambian de valor en cada iteración** ya que están mapeadas a diversos sensores/actuadores que las actualizan constantemente. Por ello **deben leerse y escribirse en cada iteración**.

```
while (true) {
    A=(F+B*C)-(D*E);
}
```

Las dos alternativas de las que disponemos son un procesador de tipo Acumulador y otro de tipo Memoria/Memoria. La descripción del ISA de ambos procesadores es la siguiente:

		Acumulador		**Memoria / Memoria**	
Tipo ins.	Opcodes	Ejemplo	Operación	Ejemplo	Operación
Aritmética	add, mul, sub, div	sub A	ACC=ACC-A	sub A, B, C	C=A-B
Memoria	load,store	load A	ACC=A	--	--
Salto	br	br Loop	PC=PC+despl	br Loop	PC=PC+despl

El siguiente código muestra la traducción a ensamblador del procesador Memoria/Memoria del kernel anterior. NO es posible optimizar el código haciendo loads/stores fuera del bucle dado que los datos son distintos en cada iteración.

```
Loop:                                   Instrucciones dinámicas/iteración: 5
  mul D,E,r1
  mul B,C,r2
  add F,r2,r2
  sub r2,r1,A
  br Loop
```

a) **Traduce** el bucle a ensamblador del procesador Acumulador usando el menor número de instrucciones posible. Podéis usar variables temporales tmp1, tmp2, ... si lo necesitáis. Escribid cuantas instrucciones dinámicas se ejecutan en cada iteración del bucle.

```
Loop:                                   Instrucciones dinámicas/iteración: 9
  load D
  mul E
  store tmp1
  load B
  mul C
  add F
  sub tmp1
  store A
  br Loop
```

Sabemos que la memoria de instrucciones es el cuello de botella del sistema y por tanto el rendimiento del procesador estará únicamente limitado por el ancho de banda con dicha memoria. La memoria de instrucciones es capaz de ofrecer, para el kernel anterior, un ancho de banda sostenido de 18 GB/s. Cada instrucción del procesador Acumulador ocupa 4 bytes y cada instrucción del procesador Memoria/Memoria ocupa 8 bytes.

b) **Justifica** cuantitativamente cuál es el procesador capaz de ejecutar el código más rápidamente y **calcula** cuál es su ganancia con respecto al más lento para el código dado (Pista: calculad cuántas iteraciones por segundo puede hacer cada procesador)**.**

Acumulador: es capaz de realizar 18E9bytes/s / 36 bytes/iter = 500E6 iter/s

Mem/Mem : 18E9bytes/s / 40 bytes/iter = 450E6 iter/s

El procesador acumulador será más rápido y su ganancia será de:
S= 500E6/450E6 = 1,11 con respecto al Mem/Mem.

Este DSP está conectado a una memoria principal formada por 1 DIMM de memoria DDR con 8 chips de 1 byte por DIMM y cada chip contiene un único banco. La latencia de fila es de 3 ciclos, la latencia de columna es de 2 ciclos y la latencia de precarga es de 1 ciclo.

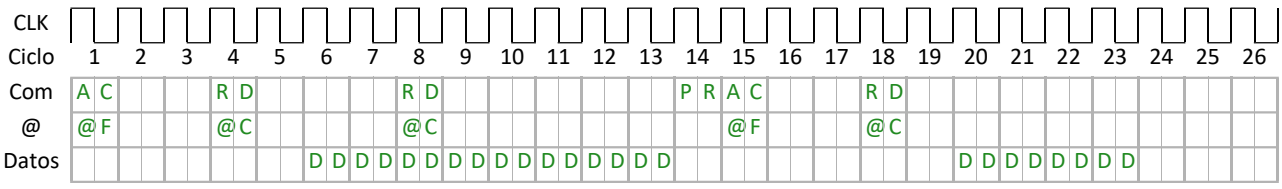
A la memoria DDR se realiza la siguiente secuencia de 3 accesos en los que se lee un bloque de 64 bytes en cada acceso: página X, página X, página Y

El sistema tiene un controlador de memoria que no cierra la página después de cada acceso. En caso de que un acceso se realice sobre una página abierta, no es necesario abrirla. Sin embargo si el acceso se realiza sobre una página distinta, tenemos que cerrar la página anterior y abrir la página que se desea acceder.

Para indicar la ocupación de los distintos recursos utilizaremos la siguiente nomenclatura:

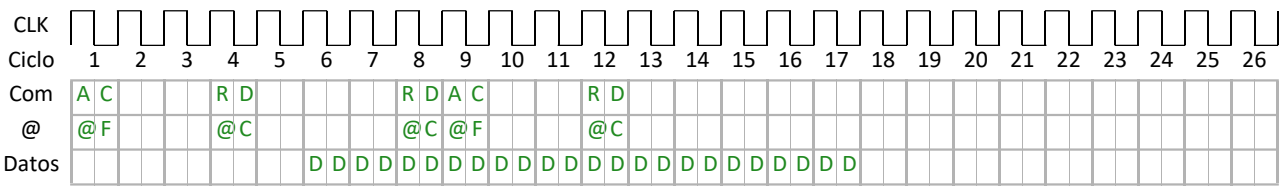
- AC: ciclo en que se envía el comando ACTIVE
- RD: ciclo en que se envía el comando READ
- PR: ciclo en que se envía el comando PRECHARGE
- @F: ciclo en que se envía la dirección de fila
- @C: ciclo en que se envía la dirección de columna
- D: transmisión de un paquete de datos

c) **Rellena** el siguiente cronograma, indicando la ocupación de los distintos recursos del sistema con **1 banco** por chip (inicialmente no hay ninguna página abierta), de forma que la secuencia se realice en el número mínimo de ciclos:



Una mejora adicional consiste en usar chips con múltiples bancos cada uno, de forma que solo se cierra página si accedemos a una página distinta a la abierta en el mismo banco. Sabemos que las páginas X e Y se encuentran en bancos distintos.

d) **Rellena** el siguiente cronograma, indicando la ocupación de los distintos recursos del sistema con **2 bancos** por chip (inicialmente no hay ninguna página abierta), de forma que la secuencia se realice en el número mínimo de ciclos:



COGNOMS:

NOM:

Problema 2. (3,5 puntos)

Tenemos un procesador conectado directamente a la memoria principal para los datos y las instrucciones, que llamaremos procesador **SIN**. En media, las instrucciones tienen un CPI de 181 ciclos/instr (incluyendo tanto el acceso a la memoria de instrucciones como la de datos). En este procesador, ejecutamos un código que tiene un 10% de las instrucciones que realizan 1 acceso a memoria de datos y un 20% de las instrucciones que realizan 2 accesos a la memoria de datos. Todas las instrucciones acceden a la memoria de instrucciones.

a) **Calcula** cuántos accesos a memoria por instrucción realiza el procesador **SIN** en media.

N = Número instrucciones

Núm accesos= Núm_accesos_inst + Núm_accesos_datos = 1*N + 0,1*1*N + 0,2*2*N

Accesos/Inst= 1,5*N/N= 1,5

El CPI del procesador si la memoria fuera ideal (1 ciclo de acceso siempre), es de 1 ciclo, a este procesador lo llamaremos **IDEAL**.

b) **Calcula** la penalización por acceso a memoria del procesador **SIN** respecto al **IDEAL**.

CPI = 181=1 + 1,5*Pmem

Pmem=120

Los ingenieros de computadores han decidido introducir una nueva versión de la memoria principal, una cache de instrucciones y una cache de datos. La latencia de acceso a las caches es de 1 ciclo. Si todos los accesos que realiza una instrucción son acierto en cache, esta se ejecuta con un CPI de 1 como en el procesador **IDEAL**, por lo que el CPI ideal (siempre acertamos en la cache) es de 1 ciclo. El tiempo de penalización en caso de fallo es de 60 ciclos para ambas caches. La tasa de fallos de la cache de instrucciones es de un 10%. A este procesador lo llamaremos procesador **CON**. Para simplificar el problema, asumir que en el procesador **CON** todos los accesos a datos son lecturas.

c) **Calcula** el CPI del procesador **CON** en función de la tasa de fallos de la cache de datos (t_f):

CPIcon = 1 + tf_inst*60+accesos_datos/inst*tf_datos*60

CPIcon= 1 + 0,1*60 + 0,5*tf*60= 7 + 30*tf

Como tenemos varios diseños de cache de datos con distintas tasas de fallos (t_f), debemos escoger el diseño que nos de el rendimiento deseado. En nuestro caso queremos que este código se ejecute 10 veces más rápido en el procesador **CON** que en el procesador **SIN**. Asumir que ambos funcionan a la misma frecuencia.

d) **Calcula** la mayor tasa de fallos posible de la cache datos para que el código se ejecute 10 veces más rápido en el procesador **CON** respecto al procesador **SIN**:

Tsin=N*CPIsin*Tc ; Tcon=N*CPIcon*Tc

Tsin/Tcon=10=CPIsin/CPIcon

10=181/(7+30*tf) => tf= 0,37

Después de un largo debate entre los ingenieros, la cache de datos implementada es write-through y write-no-allocate. Esta cache tiene una tasa de fallos de 0,2 tanto para lecturas como para escrituras. La tasa de fallos de la cache de instrucciones continua siendo de un 10%. Todas las instrucciones tienen un consumo base de 10nJ. Además, si acceden a cualquier cache tiene un consumo por acceso (escritura o lectura) de 30nJ. Un acceso (tanto de bloque como de palabra) a la memoria principal de datos consume 400nJ. En el código analizado, un 40% de los accesos a memoria de datos son escrituras. A este procesador lo llamaremos procesador **WT**.

e) **Calcula** la ganancia en energía por instrucción del procesador **WT** respecto al procesador **SIN**.

Esin = Ebase + accesos/inst*400 = 10 + 1,5*400=10+600=610nJ

Ewt = Ebase + Ecache-(inst+datos) + Einst-mem + Edatos-lecturas-mem + Edatos-escrituras_mem=

Ewt = Ebase + accesos/inst*Ecache + accesos_inst/inst*Emem + accesos_datos/inst*lecturas/accesos_datos*tfallo*Emem + accesos_datos/inst*escrituras/accesos_datos*Emem =

=10 + 1,5*30 + 0,1*400 + 0,5*0,6*0,2*400 + 0,5*0,4*400 = 10+ 45+40+24+80=199nJ

Esin/Ewt=610/199=3,06x

El procesador **WT** funciona a una frecuencia de 2GHz y tiene un CPI de 13 ciclos/instrucción.

f) **Calcula** la potencia en Watios del procesador **WT** si la frecuencia de operación es de 2GHz

N= Numero de instrucciones

Tc=1/f=0.5ns

P=E/T = (Ewt * N)/(N*CPIwt*Tc) = Ewt/(CPIcwt*Tc)

$P=199*10^{-9}/(13*0,5*10^{-9}) = 30,62W$

Debido al impacto en el rendimiento de los accesos a memoria principal, los diseñadores quieren introducir un buffer de escrituras entre la cache de datos y la memoria principal. Este buffer funciona como una cola y va guardando **todas** las escrituras que se realizan. Una vez se escribe en el buffer, el procesador sigue ejecutando el código (no se espera a que se escriba en memoria). Cuando la memoria está libre, el buffer escribe en la memoria principal. En el caso de una lectura de la memoria principal, primero se mira si los datos están en el buffer, si no lo están, se accede -después- a la memoria principal. Para simplificar el problema asumiremos que el buffer no se llena nunca y por tanto el procesador no se bloqueará por falta de espacio en el buffer. A este procesador lo llamaremos **BUFFER**.

Sólo un 10% de las lecturas encuentran el dato en el buffer. Cada acceso al buffer (lectura o escritura) consume 50nJ.

g) **Calcula** la energía media por instrucción del procesador **BUFFER**:

Ewt = Ebase + Ecache-(inst+datos) + Einst-mem + Edatos-lecturas-buffer +Edatos-lecturas-mem + Edatos-escrituras_buffer-i-mem=

Ebuffer= Ebase + accesos/inst*Ecache + accesos_inst/inst*Emem + accesos_datos/inst*lecturas/accesos_datos*tfallo*Ebuffer + accesos_datos/inst*lecturas/accesos_datos*fallo_lectura_buffer/escrutras*Emem + accesos_datos/inst*escrituras/accesos_datos*(Ebuffer+Emem)=

= 10 + 1,5*30 + 0,1*400 + 0,5*0,6*0,2*50+ 0,5*0,6*0,2*0,9*400 + 0,5*0,4*(50+400) =

= 209.6nJ

h) **Calcula** la latencia máxima en ciclos del buffer para que el procesador **BUFFER** sobrepase los 165 MIPS.

Lmem= 60 (latencia memoria)
Lbuffer = (latencia buffer)

CPIbuf = CPIbase + Pinst-mem + Pdatos-lecturas-buffer + Pdatos-lecturas-mem +Pdatos-escrituras-buffer=

CPIbuf = CPIbase + tfallo_inst*Lmem + accesos_datos/inst*lecturas/accesos_datos*tfallocache*Lbuffer + accesos_datos/inst*lecturas/accesos_datos*tfallocache*tfallobuffer*Lmem + accesos_datos/inst*escrituras/accesos_datos*Lbuffer = 1 + 0,1*60 + 0,5*0,6*0,2*lat + 0,5*0,6*0,2*0,9*60+0,5*0,4*lat= 10,24+0,26*lat

$MIPS = 1/(CPI*Tc)=165*10^{6}=1/(CPI*0,5*10^{-9})$ => CPI<=12,12 => lat= 7

COGNOMS:

NOM:

Problema 3. (3,5 puntos)

Queremos evaluar el rendimiento de un programa P en un sistema, que llamaremos PC1, con un solo procesador single-core y un solo disco duro D. Hemos tomado algunos valores de rendimiento ejecutando la aplicación en el PC1 y vemos que el programa se ejecuta en dos fases bien diferenciadas:

• Fase 1, o Fase de Lectura, en el que lee un bloque de datos de 300 GB del disco D almacenado en posiciones consecutivas. El disco D tiene un ancho de banda de 200 MB/s.

• Fase 2, o Fase de Cálculo, en el que se realizan $45 \cdot 10^{12}$ operaciones de coma flotante. El PC1 ejecuta esta fase a una velocidad de cómputo de 3 GFLOPS.

En todo el problema, no vamos a considerar los tiempos de búsqueda ni latencia de los discos.

a) **Calcula** el tiempo de ejecución de las Fases de Lectura y Cálculo, y el tiempo total de la aplicación en el PC1.

$T_{lectura}$ = 300 GB / 200 MB/s = 1500 s
$T_{cálculo}$ = $45 \cdot 10^{12}$ flops / $3 \cdot 10^{9}$ flops/s = 15000 s
$T_{PC1} = T_{lectura} + T_{cálculo}$ = 16500 s

Con el objeto de reducir el tiempo del programa, se baraja la opción de sustituir el procesador del PC1 por un sistema multiprocesador, que llamaremos PC2. Sin embargo, calculamos que solo el 80% de la Fase de Cálculo es paralelizable y que debido a la paralelización del código estaremos introduciendo una sobrecarga por la sincronización necesaria entre procesos. Estimamos que la sobrecarga introducida será de 15 segundos por cada procesador que se añada.

b) **Calcula** el número ideal de procesadores que debería tener el PC2 para obtener una ganancia máxima. **Calcula** la ganancia obtenida respecto al PC1. (Pista: recuerda que una manera de obtener el máximo de una función es encontrar la primera derivada, igualar a 0 y resolver)

$T_{secuencial} = T_{lectura} + T_{cálculo} \cdot 0,2$ = 4500 s
$T_{paralelo} = T_{cálculo} \cdot 0,8$ = 12000 s
T_{PC2} = 4500 + 12000/N + 15·(N-1)
Derivando e igualando a 0 --> N ≈ 28

T_{PC2} = 5333,57 s
G = 16500 / 5333,57 = 3,09

Después de acudir al mercado, decidimos optar por una propuesta más económica y adquirir un multiprocesador que, junto al disco D, montaremos en un nuevo sistema que llamaremos PC3. En este sistema, calculamos que la aplicación completa se ejecutará 3 veces más rápido que en el PC1.

Sabemos que la Fase de Cálculo del programa P tiene $25 \cdot 10^{3}$ instrucciones estáticas y ejecuta $60 \cdot 10^{12}$ instrucciones dinámicas, de las cuales $50 \cdot 10^{12}$ son instrucciones de coma flotante.

c) **Calcula** los MIPS y GFLOPS de la Fase de Cálculo al ejecutar el programa P en el PC3 sabiendo que este ejecutará un 20% de instrucciones más (en la fase de cálculo) respecto al PC1 debido a la sincronización.

$T_{cálculo\_PC3}$ = 16500/3 - 1500 = 4000 s
MIPS = $(1,2 \cdot 60 \cdot 10^{12}$ instr. / $10^{6}) / 4000$ s = 18000 MIPS
GFLOPS = $(45 \cdot 10^{12}$ flops / $10^{9}) / 4000$ s = 11,25 GFLOPS

Otra opción que se ha barajado para mejorar el rendimiento del sistema PC1 es añadir un RAID de discos en lugar del disco duro D. Un RAID nos permite paralelizar la Fase de Lectura, ya que en esta fase hay suficientes accesos para saturar el ancho de banda de todos los discos, además de añadir un mecanismo de tolerancia a fallos en disco. Para ello, disponemos de 6 discos iguales con ancho de banda de 200 MB/s y un sistema RAID que puede configurarse como **RAID 10** o **RAID 5**.

d) **Describe** las principales características de cada uno de estos sistemas RAID, dibujando un esquema de cómo se distribuyen los datos y especificando el tipo de entrelazado, el porcentaje de información redundante, el número de discos que han de fallar para que el sistema deje de ser operativo, el ancho de banda **máximo** de las lecturas y el ancho de banda **máximo** de las escrituras.
NOTA: Considerad el mejor de los casos entre accesos secuenciales y aleatorios.

RAID 10:
Entrelazado a nivel de tira. El 50% de la información es redundante. Es fiable ante el fallo de un disco cualquiera, pero podrían fallar hasta tres discos si están en mirrors distintos. El ancho de banda máximo de lecturas es el de leer de los 6 discos (6 x 200 MB/s = 1,2 GB/s), mientras que el de las escrituras es sólo la mitad porque se ha de escribir en cada disco y en su mirror (600 MB/s).

RAID 5:
Entrelazado a nivel de tira. Tiene la paridad distribuida entre todos los discos, de forma que en total 1/6 de la información es redundante. Es fiable ante el fallo de un disco cualquiera. El ancho de banda máximo de lecturas es el de leer de los 6 discos (6 x 200 MB/s = 1,2 GB/s). En el mejor caso, el ancho de banda de las escrituras sería el de escribir en 5 discos (en el sexto se escribiría la paridad de los otros 5), y por lo tanto 5 x 200 MB/s =1 GB/s.

Decidimos configurar el sistema de discos como **RAID 5** y montarlo en el PC3. A este sistema le llamamos PC4.

e) **Calcula** la ganancia al ejecutar el programa P en el PC4 respecto al PC1.

$T_{lectura\_PC4}$ = 300 GB / 1,2 GB/s = 250 s
$T_{PC4} = T_{lectura\_PC4} + T_{cálculo\_PC3}$ = 250 + 4000 = 4250 s
G = 16500 / 4250 = 3,88

COGNOMS:

NOM:

IMPORTANTE leer atentamente antes de empezar el examen: Escriba los apellidos y el nombre antes de empezar el examen. Escriba un solo carácter por recuadro, en mayúsculas y lo más claramente posible. Es importante que no haya tachones ni borrones y que cada carácter quede enmarcado dentro de su recuadro sin llegar a tocar los bordes. Use un único cuadro en blanco para separar los apellidos y nombres compuestos si es el caso. No escriba fuera de los recuadros.

Problema 1. (2,5 puntos)

Hemos simulado la ejecución de un programa en una CPU con un sistema de memoria en donde todos los accesos a memoria tardan 1 ciclo sean aciertos o fallos (denominaremos **CPU_{IDEAL}** a esta combinación simulada) y hemos obtenido que el programa se ejecuta en $12{,}5x10^9$ ciclos, ejecuta $5x10^9$ instrucciones, realiza $6x10^9$ accesos a memoria y de estos, $500x10^6$ son fallos de cache. La CPU funciona a 2,5GHz.

a) **Calcula** el CPI, el numero de accesos por instrucción, la tasa de fallos, el tiempo de ejecución del programa en la **CPU_{IDEAL}** y el tiempo medio entre accesos en ciclos.

CPI = $12{,}5x10^9$ ciclos / $5x10^9$ instrucciones = **2,5 c/i**

Accesos por instrucción = $6x10^9$ accesos / $5x10^9$ instrucciones = **1,2 a/i**

Tasa de fallos = $500x10^6$ fallos / $6x10^9$ accesos = 0,083 f/a = **8,33%**

Tiempo ejecución = $12{,}5x10^9$ ciclos / $2{,}5x10^9$ Hz = **5 s**

Tiempo medio entre accesos = $12{,}5x10^9$ ciclos / $6x10^9$ accesos = **2,083 ciclos**

Queremos integrar esta CPU con una cache unificada (instrucciones+datos) de mapeo directo. Esta cache no está segmentada y su tiempo de acceso es de 0,9 ns. Obsérvese que el tiempo de acceso es mayor que el tiempo de ciclo del procesador, por lo que al acceder a cache, el procesador se bloquea durante unos ciclos, y por tanto se produce una pequeña penalización respecto a la **CPU_{IDEAL}** (incluso en caso de acierto). En caso de que el acceso sea un fallo de cache, hay una penalización adicional de 20 ciclos más.

b) **Calcula** los ciclos de penalización en caso de acierto.

Ciclos penalización acierto = $\lceil 0{,}9x10^{-9}$ s * 2,5 $x10^9$ Hz $\rceil$ - 1 ciclo = 2 ciclos

NOTA: Para evitar la propagación de errores entre apartados, independientemente de la respuesta correcta del apartado anterior, a partir de aquí, supondremos que la respuesta correcta al ejercicio anterior: **ciclos de penalización por acierto es 1** y, por lo tanto, los ciclos de penalización por fallo (totales sobre la **CPU_{IDEAL}**) son 21.

c) **Calcula** el tiempo de ejecución cuando ejecutamos el programa con la cache unificada.

ciclos = $12{,}5x10^9$ ciclos + 1 ciclos/acceso * $6x10^9$ accesos + 20 ciclos/fallo * $500x10^6$ fallos = $28{,}5x10^9$ ciclos
Texe = $28{,}5x10^9$ ciclos / $2{,}5x10^9$ Hz = **11,4 s**

Nuestra CPU es capaz de continuar ejecutando instrucciones mientras se accede la a cache, sin embargo en el apartado c) bloqueamos la CPU en cada acceso para evitar lanzar un segundo acceso a la cache antes de que acabe el acceso anterior. Una posible mejora, que denominaremos control de bloqueos de cache, consiste en no bloquear la CPU en cada acceso, sino solamente si se inicia un acceso antes de que el anterior haya terminado. La CPU no soporta loads no bloqueantes, por lo que en caso de fallo siempre bloquearemos la CPU mientras se trae el bloque del siguiente nivel de la jerarquía (ciclos de penalización adicional). Sabemos que la probabilidad de realizar un acceso es la misma en todos los ciclos y es independiente de lo sucedido en ciclos anteriores. Durante los ciclos que no está bloqueada, la CPU se comporta exactamente igual que en el caso ideal por lo que el número medio de ciclos entre dos accesos será el mismo.

d) **Calcula** la probabilidad de acceder a memoria en un ciclo determinado y la probabilidad de que al realizar un acceso la cache esté ocupada.

Probabilidad acceso en 1 ciclo =

1/tiempo medio = 1/2,083 = **0,48**

Probabilidad de un acceso con cache ocupada =

la cache sólo puede estar ocupada durante un ciclo (por el acceso anterior) -> **0,48**

NOTA: Para evitar la propagación de errores entre apartados, independientemente de la respuesta correcta del apartado anterior, a partir de aquí supondremos que la respuesta correcta al ejercicio anterior: **probabilidad de acceso con la cache ocupada es 0,4**.

e) **Calcula** el tiempo de ejecución cuando ejecutamos el programa en la CPU con control de bloqueos de cache.

ciclos = $12{,}5 \times 10^9$ ciclos + 1 ciclos/acceso * 6×10^9 accesos * 0,4 + 20 ciclos/fallo * 500×10^6 fallos = $24{,}9 \times 10^9$ ciclos
Texe = $24{,}9 \times 10^9$ ciclos / $2{,}5 \times 10^9$ Hz = **9,96 s**

En una cache organizada en bancos el acceso a cada banco es independiente, por lo que es posible acceder a un banco aunque otro este ocupado. Si organizamos nuestra cache en 4 bancos, una posible mejora, que denominaremos control de bloqueos de banco, consiste en bloquear la CPU solamente en caso de que accedamos a un banco ocupado. En nuestro caso, sabemos que en cada acceso la probabilidad de acceder a cualquiera de los 4 bancos es la misma, y que es independiente de los accesos anteriores. Como en el caso anterior, la CPU no soporta loads no bloqueantes, por lo que en caso de fallo siempre bloquearemos la CPU mientras se trae el bloque del siguiente nivel de la jerarquía (ciclos de penalización adicional).

f) **Calcula** la probabilidad de que al realizar un acceso el banco accedido esté ocupado.

probabilidad de acceso con cache ocupada = 0,4
probabilidad de acceder mismo banco = 1/4

probabilidad de acceder a banco ocupado = 1/4*0,4 = **0,1**

COGNOMS:

NOM:

Problema 2. (2.5 puntos)

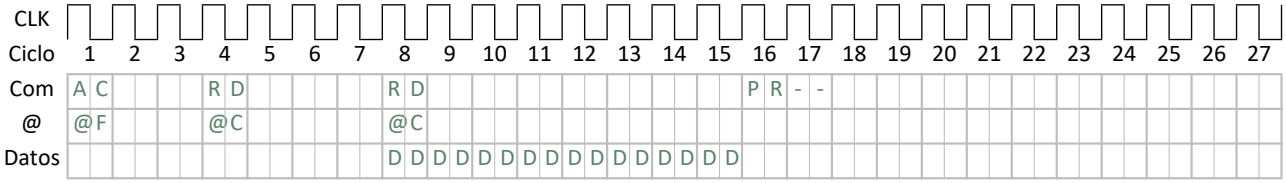
Una **CPU** está conectada a una cache de instrucciones (**$I**) y una cache de datos (**$D**). El conjunto formado por **CPU**+**$I**+**$D** esta conectado a una memoria principal formada por un único módulo DIMM estándar de 16 GBytes. Este DIMM tiene 8 chips de memoria **DDR**-SDRAM (**Double Data Rate** Synchronous DRAM) de 1 byte de ancho cada uno. La DDR-SDRAM tiene 2 bancos. 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 3 ciclos, la latencia de columna de 4 ciclos y la latencia de precarga de 2 ciclos. Es posible que el conjunto **CPU**+**$I**+**$D** solicite múltiples bloques a la DDR (por ejemplo porque se produzca un fallo simultáneamente en **$I** y en **$D**). El controlador de memoria envía los comandos necesarios a la DDR-SDRAM de forma que los bloques sean transferidos lo más rápidamente posible y se maximice el ancho de banda.

La siguiente tabla muestra en qué banco y qué página de DRAM (fila) se encuentran los bloques etiquetados con las letras A B C D.

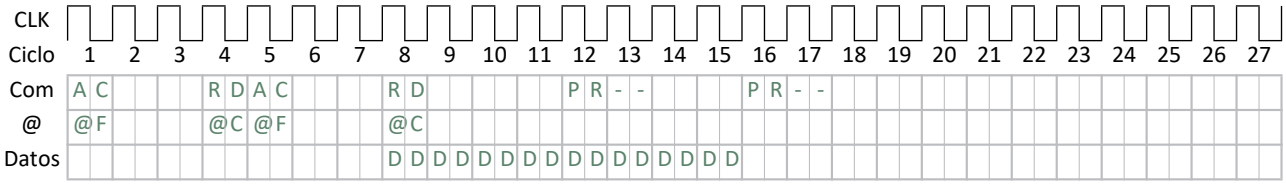
Bloque	A	B	C	D
Banco	0	0	1	1
Página	10	10	10	25

Rellena los siguientes cronogramas para la lectura de varios bloques de 64 bytes en función de la ubicación de los bloques involucrados de forma que se minimice el tiempo total. 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 previamente.

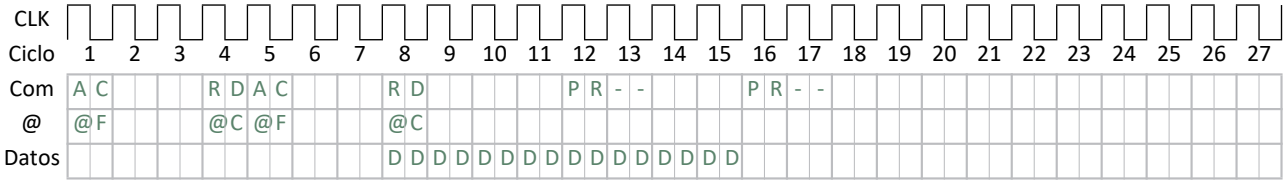
a) **Rellena** el siguiente cronograma para la lectura de los bloques A y B.



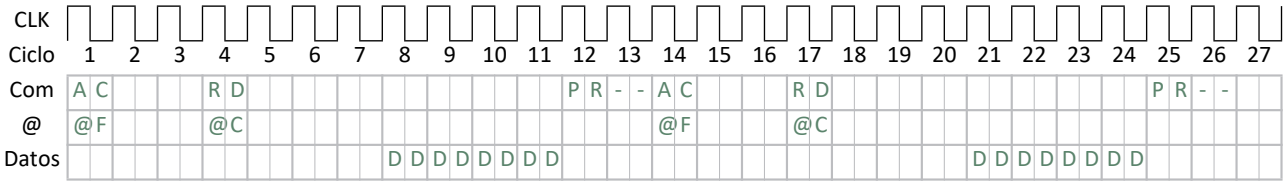
b) **Rellena** el siguiente cronograma para la lectura de los bloques A y C.



c) **Rellena** el siguiente cronograma para la lectura de los bloques A y D.



d) **Rellena** el siguiente cronograma para la lectura de los bloques C y D



Con un simulador hemos simulado una versión ideal de dicha CPU en que cada acceso a memoria tarda 1 ciclo. Para una aplicación A hemos obtenido los siguientes datos: $2x10^9$ instrucciones ejecutadas, $1x10^9$ accesos a datos, $CPI_{ideal} = 2$ c/i.

A la implementación real de dicha CPU la llamaremos procesador P. En el procesador P la cache de datos (**$D**) es 2 asociativa e incorpora un predictor de vía. Al ejecutar la aplicación A, el predictor de vía tiene una tasa de aciertos del 76% y la cache de datos tiene una tasa de fallos del 4%. En caso de que el predictor acierte la vía no hay penalización respecto al procesador ideal, si hay fallo de predictor pero acierto de cache se incurre en un ciclo de penalización, y finalmente, si es fallo de predictor y también de cache, la penalización es de 25 ciclos. Respecto a la cache de instrucciones, apenas se producen fallos por lo que se puede considerar que se comporta igual que en la CPU ideal.

e) **Calcula** en cuantos ciclos se ejecuta la aplicación A en el procesador P.

76% acierto predictor ----> No penaliza
20% fallo predictor - acierto cache ---> 1 ciclo de penalización
4% fallo predictor - fallo cache ---> 25 ciclos de penalización

Ciclos = ciclos ideal + ciclos (extra) accesos datos =
= I * CPI ideal + Accesosdatos * (m_predictor * tp_pfedictor + m_cache * tp_fallo) =
= $2x10^9$i * 2 c/i + $1x10^9$ a * (0,2 fp/a * 1 c/fp + 0,04 fc/a * 25 c/fc) = **$5,2x10^9$ ciclos**

Queremos saber la energía de conmutación consumida por los accesos a datos del procesador P. Ignoraremos por tanto la potencia disipada por fugas así como la potencia de conmutación del procesador, y también la del predictor, que tiene un impacto despreciable. Sabemos que cada vez que se accede una vía de la cache de datos se consumen 5 nJ (nanojoules) y cada vez que hay un fallo en la cache de datos se consumen 50 nJ adicionales.

f) **Calcula** la energía consumida por los accesos a datos del procesador P al ejecutar la aplicación A

Siempre se accede una via (5 nJ)
24% Fallo de predictor ademas se accede la otra vía -> 5nJ adicionales
4% Fallo de cache 50 nJ adicionales (60 nJ en total)

Energia = $1x10^9$accesos * (1 * $5x10^{-9}$ J + 0,24 * $5x10^{-9}$ J + 0,04 * $50x10^{-9}$ J) = **8,2 Joules**

Otra:

Energia = $1x10^9$accesos * (0,76 * $5x10^{-9}$ J + 0,20 * $10x10^{-9}$ J + 0,04 * $60x10^{-9}$ J) = **8,2 Joules**

COGNOMS:

NOM:

Problema 3. (2.5 puntos)

Tenemos un disco duro que gira a una velocidad de rotación (número de vueltas por minuto) de 12000 r.p.m. y tiene 1000 sectores por pista. El tiempo medio de búsqueda del disco (*seek*) es de 1 ms. El tiempo de posicionamiento promedio (*latency*) siempre es el mismo, independientemente de dónde esté situado el cabezal y coincide con el tiempo que se tarda en dar media vuelta. El ancho de banda de transferencia del disco es de 150MBytes/s. De todos los bytes que forman un sector del disco, 238 bytes corresponden a información de control, y el resto de bytes del sector es información de datos.

a) **Calcula** el tiempo de transferencia de un sector. Este tiempo sólo depende de la velocidad de rotación del disco y de la densidad de la pista. Asumir que el cabezal ya está situado al principio del sector.

```
velocidad de rotación = 12000 vueltas/60s = 200 vueltas por segundo
1/velocidad de rotación = 0.005s = 5ms para recorrer todos los sectores de una
pista (dar una vuelta).
5ms/1000sectores = 0.005ms = 5µs tiempo de transferencia para leer un sector
```

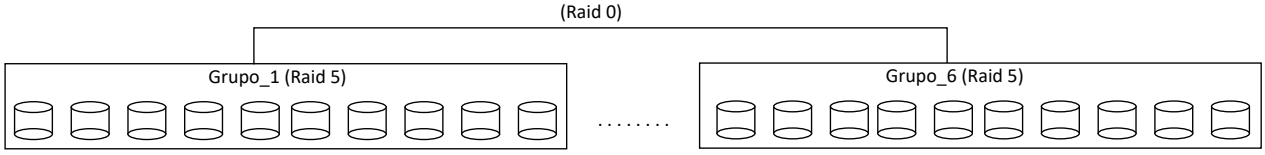
b) **Calcula** el tiempo que transcurre hasta que se lee el último byte de un fichero que está almacenado en 50 sectores consecutivos de una pista, y en otros 30 sectores consecutivos de otra pista.

```
t_medio de búsqueda + t_de posicionamiento promedio + (t_de transferencia de
un sector x número de sectores)

t_de posicionamiento promedio = 5ms/2 = 2.5ms
t_de transferencia de un sector = 5µs = 0.005ms

50 sectores = 1ms + 2.5ms + (0.005ms x 50) = 1 + 2.5 + 0.25 = 3.75ms
30 sectores = 1ms + 2.5ms + (0.005ms x 30) = 1 + 2.5 + 0.15 = 3.65ms
Tiempo para leer el fichero = 3.75ms + 3.65ms = 7.4ms
```

c) **Calcula** el tamaño de un sector (bytes de datos + bytes de control).

```
ancho de banda de transferencia = 150MBytes/s
t_de transferencia de un sector = 5µs = 0.005ms

Tamaño de un sector = 150MBytes/s x 0.005ms = 750 bytes "brutos"
Tamaño de los datos = 750 - 238 bytes = 512 bytes
```

Disponemos de 60 discos físicos con capacidad de 1 TByte por disco y queremos montar con ellos un disco lógico con la configuración RAID50 con 6 grupos de 10 discos en cada grupo:



d) **Calcula** la cantidad de información útil que puede almacenar este sistema RAID50. **Calcula** el porcentaje de información redundante que hay en este sistema RAID50.

```
G=6, N=60 y 10 discos en cada G

Información útil = (10 - 1)xG = 54 discos x 1 TByte = 54 TBytes

Porcentaje de información redundante = (1xG)/N = 1/10 = 10%
```

Cada disco tiene un tiempo medio entre fallos (MTTF_disco) de 100.000 horas y el tiempo de recuperar la información en caso de tener que reemplazar un disco (MTTR) es de 10 horas. El tiempo medio entre fallos para un sistema multi-RAID como RAID50 (MTTF_RAID50) coincide con el tiempo medio entre fallos de uno de sus grupos (MTTF_grupo) dividido por el número de grupos (G). El cálculo del MTTF_RAID50 se hará en base al cálculo de MTTF_grupo:

$$MTTF\_RAID50 = \frac{MTTF\_grupo}{G}$$

e) **Escribe** la expresión general del tiempo medio entre fallos de un grupo de discos (MTTF_grupo) para la organización RAID50 suponiendo que los discos son el único componente que puede fallar. **Calcula** el valor de MTTF_grupo con los datos proporcionados.

```
1 Grupo = RAID5 con 10 discos (N=10)

En RAID5 una vez que falla un disco, el sistema deja de funcionar cuando falla
un segundo disco durante el tiempo de recuperación del disco averiado (MTTR)

MTTF_grupo = MTTF_disco/N x MTTF_disco/(N-1)xMTTR = MTTF_disco^2/10x9xMTTR =
(10^5 horas)^2 / 10x9x10 horas = 111.11x10^5 horas
```

COGNOMS:

NOM:

Problema 4. (2.5 puntos)

Tenemos que implementar un servidor de páginas web usando algunos de los siguientes componentes:

- Procesador RISC superescalar fuera de orden (RISC) que consume 60W a una frecuencia de 2GHz.
- Procesador VLIW de ancho 4 (VLIW) que consume 20W a una frecuencia de 1 GHz. Al compilar, medimos que cada instrucción del VLIW (las llamaremos Iv) equivale en media a 3 instrucciones del RISC en cualquier parte del programa.
- Discos duros de 5TB, con un ancho de banda de 200 MBytes/s por disco y un consumo de 10W por disco.

El kernel del programa a ejecutar en el servidor tiene 3 fases diferenciadas:

1) Fase secuencial. Ejecuta $1{,}2x10^9$ instrucciones dinámicas en el RISC.
2) Fase paralela de cálculo. Por simplicidad, supondremos que esta fase es **perfectamente paralelizable**. En esta fase se ejecutan en el RISC $2{,}4x10^9$ instrucciones y se realizan $5x10^9$ operaciones de coma flotante usando instrucciones SIMD. Esta es la única fase con operaciones de coma flotante del programa.
3) Fase de Entrada/salida La velocidad de esta fase está siempre limitada por el rendimiento del sistema de disco, y la cantidad de instrucciones que ejecuta se considera despreciable a efectos de cálculo. Todas las operaciones de disco son lecturas secuenciales.

Para estimar el rendimiento que podemos obtener, se realiza una ejecución secuencial del programa en un sistema (llamado SR) que dispone de un solo procesador RISC y un único disco. Una instancia del programa tarda 2s en ejecutarse. La fase 1 tarda el 20% de dicho tiempo, la fase 2 el 60% y la fase 3 el 20% restante.

a) **Calcula** el IPC (instrucciones por ciclo) en el SR de las la fase 1, la fase 2 y de todo el programa.

Fase 1: 0,4s -> 0,8E9 ciclos -> 1,2E9/0,8E9 = 1,5 Ins/ciclo

Fase 2: 1,2s -> 2,4E9ciclos -> 2,4E9/2,4E9 = 1 Ins/ciclo

Total: 4E9 ciclos -> 3,6E9/4E9 =0,9 Ins/ciclo

b) **Calcula** los MFLOPS efectivos de la fase 1, la fase 2 y de todo el programa en el SR.

MFLOPS F1=0

MFLOPS F2=5E9/1,2=4167

MFLOPS = 5E9/2=2500

Con el objeto de reducir el tiempo de ejecución del programa, se baraja la opción de substituir el procesador RISC por un sistema multiprocesador de 12 procesadores RISC idénticos al original, manteniendo igual el resto del sistema. A este sistema multiprocesador le llamaremos MPSR.

c) **Calcula** el máximo speed-up que podría conseguirse al ejecutar el programa con el MPSR en vez de en el SR.

Sea T el tiempo de ejecutar el programa en SR y T' el tiempo de ejecutar el kernel en MPSR en el caso de que no se pierda ningún tiempo en la sincronización (speed-up máximo).
T' = 0,2T + 0,6T/12 + 0,2T = 0,45T
Speed-up máximo= T/T'=T/0,45T=2,22

Medimos el IvPC (instrucciones del VLIW, Iv, por ciclo) y obtenemos una medida para la fase 1 de 0,5 IvPC y para la fase 2 una media de 1 IvPC.

d) **Calcula** el tiempo de cada una de las tres fases del programa en el VLIW con un único disco (a esta configuración la llamaremos SV).

Fase 1 IPCeq= 0,5*3 = 1,5 -> 1,2E9 /1,5 /1E9 = 0,8 s

Fase 2 IPCeq= 1*3 = 3 -> 2,4E9 / 3 / 1E9 = 0,8 s

Fase 3 = 0,4 s (no varía)

El sistema de disco solo consume energía durante la fase 3 (los discos están apagados durante las fases 1 y 2). Sin embargo, al menos un procesador debe estar encendido durante todas las fases (incluida la fase 3).

e) **Calcula** la ganancia en energía al ejecutar una instancia del programa en el SV respecto el SR. Da el resultado en porcentaje.

ERISC = 60*0,4+60*1,2+70*0,4 = 24+72+28= 124 J

EVLIW = 20*0,8+20*0,8+30*0,4 = 16 + 16 + 12 = 44 J

G = 124/44 = 2,82 -> 182%

Decidimos evaluar la posibilidad de implementar dos sistemas multiprocesador, uno con procesadores RISC y otro con procesadores VLIW, ambos usando un RAID 01 de 10 discos. Por razones de coste, tenemos un límite de consumo instantáneo de 160 W. En cada fase se pueden apagar los componentes que no se usen, a excepción de 1 procesador que tiene que estar siempre encendido.

f) **Calcula** el número máximo de componentes del sistema multiprocesador en cada alternativa y la potencia media consumida en cada caso.

AR -> 2 RISC y 10 discos -> (0,4*60+(1,2/2)*120+(0,4/10) * 160)/1,04 = 102,4/1,04=98,46W

AV-> 8 VLIW y 10 discos -> (0,8*20+(0,8/8)*160+(0,4/10) * 120)/0,94 = 36,8/0,94 = 39,15W

COGNOMS:

NOM:

IMPORTANTE leer atentamente antes de empezar el examen: Escriba los apellidos y el nombre antes de empezar el examen. Escriba un solo carácter por recuadro, en mayúsculas y lo más claramente posible. Es importante que no haya tachones ni borrones y que cada carácter quede enmarcado dentro de su recuadro sin llegar a tocar los bordes. Use un único cuadro en blanco para separar los apellidos y nombres compuestos si es el caso. No escriba fuera de los recuadros.

Problema 1. (2 puntos)

Dado el siguiente código escrito en ensamblador del x86:

```
       xorl %esi, %esi
       movl $0, %ebx
for:   cmpl $256*1024, %esi
       jge end
(a)    movl (%ebx, %esi, 4), %eax
       shll $2, %eax
(b)    addl %eax, 4*1024(%ebx, %esi, 4)
(c)    addl 12*1024(%ebx, %esi, 4), %eax
       addl $1024, %esi
       jmp for
end:
```

Suponiendo que la memoria virtual utiliza páginas de tamaño 4K y que se dispone de un TLB de 3 entradas completamente asociativo con reemplazo LRU, responde a las siguientes preguntas:

a) Para cada uno de los accesos etiquetados como (a), (b) y (c), **indica** a qué página de la memoria virtual se accede en cada una de las 16 primeras iteraciones del bucle

iteración	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
(a)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(b)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(c)	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

b) **Indica** mediante una F (fallo) o una A (acierto), cuáles de los accesos a memoria ejecutados en las 16 primeras iteraciones son fallo de TLB (F) y cuales son acierto de TLB (A)

iteración	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
a read	F	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
b read	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
b write	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
c read	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

c) Calcula la cantidad de aciertos de TLB en TODO el bucle

256 iteraciones

Acceso (a): 1 fallo y 255 aciertos
Acceso (b): 256 fallos y 0 aciertos para leer operando en memoria y 256 aciertos para escribir resultado
Acceso (c): 256 fallos y 0 aciertos para leer operando en memoria

Total aciertos: 255 + 0 + 256 + 0 = 511 aciertos

d) Calcula la cantidad de fallos de TLB en TODO el bucle

Total fallos: 1 + 256 +0 + 256 = 513 fallos

COGNOMS:

NOM:

Problema 2. (2 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 16 GBytes. Este DIMM tiene 8 chips de memoria **DDR**-SDRAM (**Double Data Rate** Synchronous DRAM) de 1 byte de ancho cada uno. 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 3 ciclos, la latencia de columna de 4 ciclos y la latencia de precarga de 2 ciclos. Es posible que el conjunto **CPU**+**$I**+**$D** solicite múltiples bloques a la DDR (por ejemplo porque se produzca un fallo simultáneamente en **$I** y en **$D**). El controlador de memoria envía los comandos necesarios a la DDR-SDRAM de forma que los bloques sean transferidos lo más rápidamente posible y se maximice el ancho de banda.

La siguiente tabla muestra en que banco y que pàgina de DRAM (fila) se encuentran los bloques etiquetados con las letras A B C D.

Bloque	A	B	C	D
Banco	0	0	1	1
Página	10	10	10	25

Rellena los siguientes cronogramas para la lectura de varios bloques de 64 bytes en función de la ubicación de los bloques involucrados de forma que se minimice el tiempo total. 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 previamente.

a) **Rellena** el siguiente cronograma para la lectura de los bloques A y B.

CLK																											
Ciclo	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
Com	A	C			R	D				R	D					P	R	-	-								
@	@F				@C					@C																	
Datos								D	D	D	D	D	D	D	D	D	D	D	D	D	D						

b) **Rellena** el siguiente cronograma para la lectura de los bloques A y C.

CLK																											
Ciclo	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
Com	A	C			R	D	A	C		R	D			P	R	-	-			P	R	-	-				
@	@F				@C		@F			@C																	
Datos									D	D	D	D	D	D	D	D	D	D	D	D	D						

c) **Rellena** el siguiente cronograma para la lectura de los bloques A y D.

CLK																											
Ciclo	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
Com	A	C			R	D	A	C		R	D			P	R	-	-			P	R	-	-				
@	@F				@C		@F			@C																	
Datos									D	D	D	D	D	D	D	D	D	D	D	D	D						

d) **Rellena** el siguiente cronograma para la lectura de los bloques C y D

CLK																											
Ciclo	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
Com	A	C			R	D						P	R	-	-	A	C			R	D					P	R	-	-
@	@F				@C										@F					@C									
Datos								D	D	D	D	D	D	D	D							D	D	D	D	D	D	D	D

COGNOMS:

NOM:

Problema 3. (3 puntos)

En una **CPU** ejecutamos un programa (X) que realiza 10^9 accesos a datos. Esta **CPU** está conectada a una cache de datos **$D** con políticas de escritura **copy back** + **write allocate**. La siguiente tabla muestra algunos datos obtenidos al ejecutar el programa X:

Característica	$D
Tasa de fallos (m)	10%
Consumo de energía en caso de acierto (Ea)	4 nJ
Penalización en consumo de energía en caso de fallo al reemplazar un bloque no modificado (Epf)	20 nJ
Penalización en consumo de energía en caso de fallo al reemplazar un bloque modificado (EpfM)	40 nJ
Porcentaje de bloques modificados (pm)	25%

a) **Calcula** la energía consumida por los accesos a datos del programa X (da el resultado en Joules).

E = Accesos * (Ea + m*(pm*EpfM + (1-pm)*Epf)) =
=10^9 a * (4 nJ/a + 0,1 f/a * (0,25 * 40 nJ/fallo + 0,75 * 20 nJ/fallo)) = 6,5 Joules

Sabemos que la cache L1 es 2-asociativa y que el acceso a una vía consume 2 nJ. Se desea añadir un predictor de vía a la cache. El predictor de vía seleccionado tiene una tasa de aciertos del 80% para el programa X. El consumo del predictor de vía es insignificante.

b) **Calcula** la energia ahorrada por los accesos a datos del programa X gracias al predictor de vía (da el resultado en Joules).

80% acierto predictor -> 1 via, ahorramos 2 nJ
10% fallo predictor - acierto cache -> 2 vias no ahorramos nada
10% fallo predictor - fallo de cache -> 2 vias no ahorramos nada

E = 10^9 a * 0,8 ap/a* 2 nJ/ap = 1,6 Joules

Todos los accesos del programa X son de 4 bytes y los bloques de cache de **$D** son todos de 64 bytes.

c) **Calcula** cuantos bytes lee **$D** de memoria principal y cuantos bytes escribe **$D** en memoria principal.

Bytes leidos = 1e9 a * 0,1 f/a * 64 bytes/f = 6,4e9 bytes leidos

Bytes escritos $D = 1e9 a * 0,1 f/a * 0,25 escr/f * 64 bytes/escr = 1,6e9 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 la cache de instrucciones **$I** (por lo que no provoca fallos), las variables i, N y suma están en registros y **$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 de 8 Kbytes.

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

- En la 1a ejecución el bucle tarda 55.000 ciclos.
- En la 2a ejecución el bucle tarda 30.000 ciclos.

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

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

25000 ciclos penalización / 125 fallos = 200 ciclos penalización/fallo

Deseamos ejecutar una sola copia del mismo fragmento de código para **N muy grande** (el vector recorrido es mucho mayor que el tamaño de cache).

e) **Calcula** en función de N los ciclos que tarda el fragmento de código anterior.

30N + 200/8 * N = 55N ciclos

Alt:
55000 ciclos / 1000 iteraciones = 55 ciclos/iteración
solo localidad espacial -> Para N iteraciones -> 55*N ciclos

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).

f) **Calcula** el número máximo de ciclos que puede durar un *prefetch* para que el bucle se ejecute en 40*N ciclos.

Se hace un prefetch cada 8 iteraciones
40 ciclos/iteración * 8 iteraciones/prefecth = 320 ciclos/prefetch

g) ¿Es posible ejecutar el bucle en menos de 40*N haciendo el *prefetch* más rápido? (justifica la respuesta)

SI -> 30 ciclos por iteración es lo que podemos conseguir sin fallos.

Si el prefetch tarda 30*8 = 240 ciclos o menos no hay fallos

COGNOMS:

NOM:

Problema 4. (3 puntos)

Se ha ejecutado un programa P en un PC de sobremesa que tiene un solo procesador y un solo disco duro D (sistema que denominaremos PC1) y se ha calculado que su tiempo de ejecución es de T horas. Para poder estimar el rendimiento del programa P, hemos medido (en el PC1) que el programa se ejecuta en tres fases bien diferenciadas:

Fase 1: Código SECUENCIAL que no puede paralelizarse, ocupa el 18% del tiempo de la ejecución de P en el PC1.

Fase 2: Código PARALELIZABLE, ocupa el 48% del tiempo de la ejecución de P en el PC1.

Fase 3: Código de E/S que pasa todo su tiempo accediendo al disco D, ocupa el 34% del tiempo de la ejecución de P en el PC1.

Con el objeto de reducir el tiempo de ejecución del programa, se baraja la opción de substituir el procesador del PC1 por un sistema multiprocesador de 16 procesadores idénticos al del PC1, manteniendo igual el resto del sistema. A este sistema multiprocesador le llamaremos PC2.

a) **Calcula** el máximo speed-up que podría conseguirse al ejecutar el programa P con el PC2 en el caso de que no se pierda tiempo en la sincronización entre procesadores.

Sea T el tiempo de ejecutar P en PC1, y T' el tiempo de ejecutar P en PC2.
T' = 0,18T + 0,48T/16 + 0,34T = 0,55T
Speed-up máximo= T/T'=T/0,55T=1,818

Sabemos que el programa P tiene 10^5 instrucciones estáticas y ejecuta 10^{17} instrucciones dinámicas. En la Fase 2, la única que realiza cálculos en coma flotante, se ejecutan $800x10^{14}$ instrucciones, de las cuales $500x10^{14}$ son instrucciones de coma flotante que realizan un total de $200x10^{14}$ operaciones de coma flotante en simple precisión.

b) **Calcula** los MIPS y MFLOPS al ejecutar el programa P en el PC1 si la ejecución tarda $2x10^7$ segundos.

PC1
MIPS= 10^{17} instrucciones /($2*10^7$) segundos * $1M/10^6=5*10^3$ MIPS
MFLOPS = $200 * 10^{14}$ operaciones/($2 *10^7$)segundos * $1M/10^6= 10^3$MFLOPS

c) **Calcula** la ganancia máxima en MIPS y MFLOPS al ejecutar el programa P en el PC2 en vez de en PC1.

El programa ejecuta el mismo número de instrucciones y realiza el mismo número de operaciones en coma flotante tanto en PC1 como en PC2, pero en PC2 lo hace en 1,818 veces menos tiempo (máximo speed up).
Por lo tanto, la ganancia máxima en ambos casos es de 1,818

Otra opción que se ha barajado para mejorar el rendimiento del sistema PC1 es añadir un RAID de discos en lugar del disco duro D. El ancho de banda del disco D es de 250GBytes/s. A este sistema le llamamos **PC3.** El RAID nos permite paralelizar la Fase 3, ya que en esta fase hay suficientes accesos como para saturar el ancho de banda de todos los discos del RAID. El RAID de discos del que disponemos tiene 6 discos y puede configurarse como **RAID 10** o **RAID 6**.

d) **Describe** las principales características de cada uno de estos sistemas RAID, dibujando un esquema de cómo se distribuyen los datos y especificando el tipo de entrelazado, el porcentaje de información redundante, el numero de discos que han de fallar para que el sistema deje de ser operativo, el ancho de banda **máximo** de las lecturas y el ancho de banda **máximo** de las escrituras.
NOTA: Considerar el mejor de los casos entre accesos secuenciales y aleatorios

RAID 10
Entrelazado a nivel de tira. El 50% de la información es redundante. Es fiable ante el fallo de un disco cualquiera, pero podrían fallar hasta tres discos si están en mirrors distintos. El ancho de banda máximo de lecturas es el de leer de los 6 discos (6*250GB/s=1500GB/s), mientras que el de las escrituras es sólo la mitad porque se ha de escribir en cada disco y en su mirror (750 GB/s).

RAID 6
Entrelazado a nivel de tira. Tiene dos tiras redundantes para cada grupo de tiras: la paridad P y un segundo nivel Q de redundancia basado en códigos Reed-Solomon. Ambas tiras redundantes están distribuida entre todos los discos, de forma que en total 2/6 de la información es redundante (33,33%). Es fiable ante el fallo de dos discos cualquiera. El ancho de banda máximo de lecturas es el de leer de los 6 discos (6*250GB/s=1500GB/s). En el mejor caso, el ancho de banda de las escrituras sería el de escribir en 4 discos (en los otros dos se escribiría la paridad P y la función Q de los otros 4), y por lo tanto 4*250GB/s = 1000GB/s.

RAID 10 RAID 6

Decidimos configurar el sistema de discos como RAID 6.

e) Al PC2 le montamos el RAID 6 de 6 discos en lugar del disco duro D. A este sistema le llamamos **PC4**. **Calcula** el speed-up máximo del PC4 **sobre el PC2** asumiendo que todos los accesos a disco son lecturas secuenciales.

Con RAID 6 podemos leer de los 6 discos a la vez, por lo que el tiempo de la Fase 3 se verá reducido en 1/6.
Sea T el tiempo de ejecutar P en PC2 con el disco D, y T' el tiempo de ejecutar P en PC2 con el RAID6.
En primer lugar, normalizaremos el tiempo obtenido en el paratado a).
T = (0,18T + 0,03T + 0,34T)/0,55 = 0,327T+0,055T+0,618T

Por tanto, la Fase 1 gasta el 32,7% del tiempo, la Fase 2 el 5,5% del tiempo y la Fase 3 el 61,8% del tiempo
Como la Fase 3 se reduce en 1/6, tenemos que el tiempo total del PC4:

T ' = 0,327T+0,055T+0,618/6 T = 0,485T y por lo tanto el máximo speed.up que se puede obtener es:
speed-up= T/T'=T/0,485T=2,06