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 $12{,}5 \times 10^9$ ciclos, ejecuta 5×10^9 instrucciones, realiza 6×10^9 accesos a memoria y de estos, 500×10^6 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 =

Accesos por instrucción =

Tasa de fallos =

Tiempo ejecución =

Tiempo medio entre accesos =

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 =

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.

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 =	
Probabilidad de un acceso con cache ocupada =	

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.

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.

COGNOMS:

NOM:

Problema 2. (2.5 puntos)

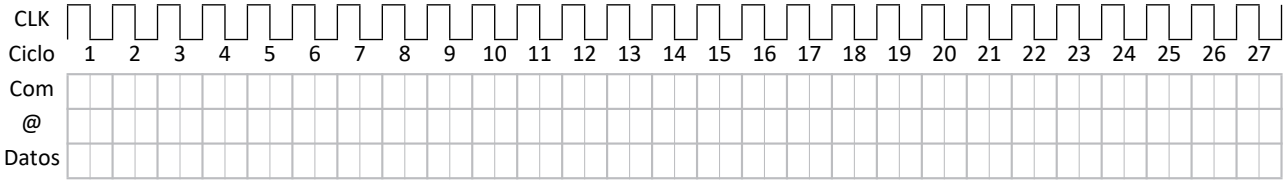
Una **CPU** está conectada a una cache de instrucciones (**$I**) y una cache de datos (**$D**). El conjunto formado por **CPU**+**$I**+**$D** esta conectado a una memoria principal formada por un único módulo DIMM estándar de 16 GBytes. Este DIMM tiene 8 chips de memoria **DDR**-SDRAM (**Double Data 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.

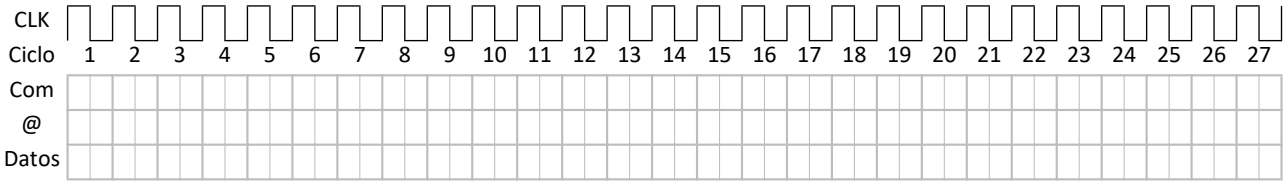
Bloque	A	B	C	D
Banco	0	0	1	1
Página	10	10	10	25

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.

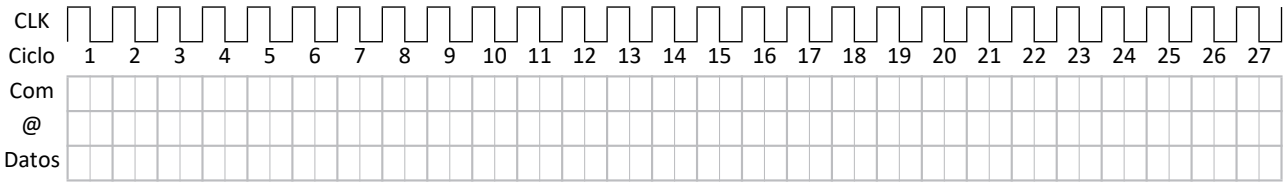
a) **Rellena** el siguiente cronograma para la lectura de los bloques A y B.



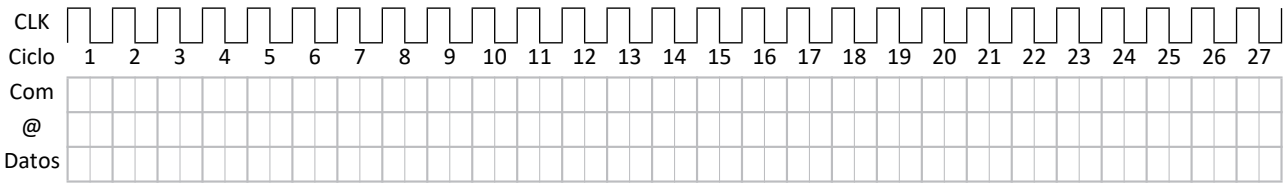
b) **Rellena** el siguiente cronograma para la lectura de los bloques A y C.



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.

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

COGNOMS:

NOM:

Problema 3. (2.5 puntos)

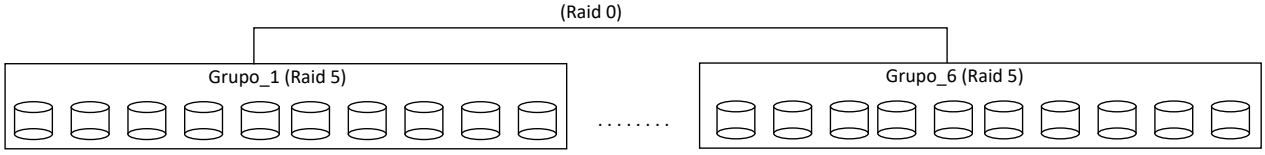
Tenemos un disco duro que gira a una velocidad de rotación (número de vueltas por minuto) de 12000 r.p.m. y tiene 1000 sectores por pista. El tiempo medio de búsqueda del disco (*seek*) es de 1 ms. El tiempo de posicionamiento promedio (*latency*) siempre es el mismo, independientemente de dónde esté 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.

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.

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

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.

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.

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 $1{,}2 \times 10^9$ 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 $2{,}4 \times 10^9$ instrucciones y se realizan 5×10^9 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.

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

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.

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).

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.

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.