

Nuestra CPU es capaz de continuar ejecutando instrucciones mientras se accede a la 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 =

$$1/\text{tiempo medio} = 1/2,083 = 0,48$$

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.

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

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.

$$\text{probabilidad de acceso con cache ocupada} = 0,4$$

$$\text{probabilidad de acceder mismo banco} = 1/4$$

$$\text{probabilidad de acceder a banco ocupado} = 1/4 * 0,4 = 0,1$$

[illegible][illegible]

Problema 2. (2.5 puntos)

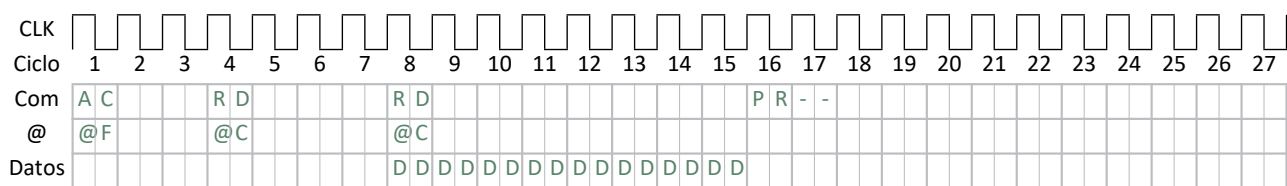
Una **CPU** está conectada a una cache de instrucciones (**\$I**) y una cache de datos (**\$D**). El conjunto formado por **CPU+\$I+\$D** está conectado a una memoria principal formada por un único módulo DIMM estándar de 16 GBytes. Este DIMM tiene 8 chips de memoria **DDR-SDRAM (Double Data Rate Synchronous DRAM)** de 1 byte de ancho cada uno. La DDR-SDRAM tiene 2 bancos. El DIMM está 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.

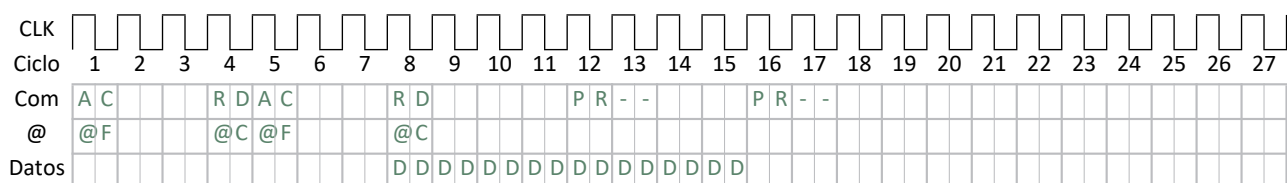
Bloque	A	B	C	D
Banco	0	0	1	1
Página	10	10	10	25

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.

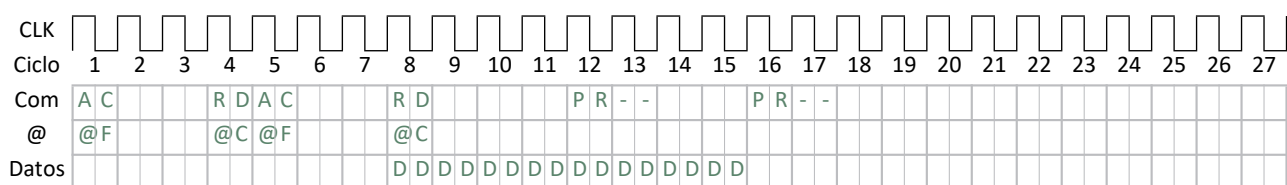
a) **Rellena** el siguiente cronograma para la lectura de los bloques A y B.



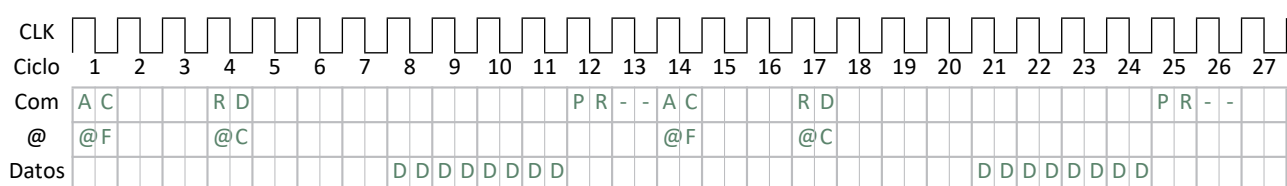
b) **Rellena** el siguiente cronograma para la lectura de los bloques A y C.



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: 2×10^9 instrucciones ejecutadas, 1×10^9 accesos a datos, $CPI_{ideal} = 2 \text{ c/i}$.

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 =

$= I * CPI_{ideal} + \text{Accesos datos} * (m_{predictor} * tp_{pfedictor} + m_{cache} * tp_{fallo}) =$

$= 2 \times 10^9 i * 2 \text{ c/i} + 1 \times 10^9 a * (0,2 \text{ fp/a} * 1 \text{ c/fp} + 0,04 \text{ fc/a} * 25 \text{ c/fc}) = \mathbf{5,2 \times 10^9 \text{ ciclos}}$

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 vía (5 nJ)

24% Fallo de predictor además se accede la otra vía -> 5nJ adicionales

4% Fallo de cache 50 nJ adicionales (60 nJ en total)

Energía = $1 \times 10^9 \text{ accesos} * (1 * 5 \times 10^{-9} \text{ J} + 0,24 * 5 \times 10^{-9} \text{ J} + 0,04 * 50 \times 10^{-9} \text{ J}) = \mathbf{8,2 \text{ Joules}}$

Otra:

Energía = $1 \times 10^9 \text{ accesos} * (0,76 * 5 \times 10^{-9} \text{ J} + 0,20 * 10 \times 10^{-9} \text{ J} + 0,04 * 60 \times 10^{-9} \text{ J}) = \mathbf{8,2 \text{ Joules}}$

COGNOMS:

[illegible]

NOM:

[illegible]

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.

$$t_{\text{medio de búsqueda}} + t_{\text{de posicionamiento promedio}} + (t_{\text{de transferencia de un sector}} \times \text{número de sectores})$$

$t_{\text{de posicionamiento promedio}} = 5\text{ms}/2 = 2.5\text{ms}$
 $t_{\text{de transferencia de un sector}} = 5\mu\text{s} = 0.005\text{ms}$

$$50 \text{ sectores} = 1\text{ms} + 2.5\text{ms} + (0.005\text{ms} \times 50) = 1 + 2.5 + 0.25 = 3.75\text{ms}$$
$$30 \text{ sectores} = 1\text{ms} + 2.5\text{ms} + (0.005\text{ms} \times 30) = 1 + 2.5 + 0.15 = 3.65\text{ms}$$

Tiempo para leer el fichero = 3.75ms + 3.65ms = 7.4ms

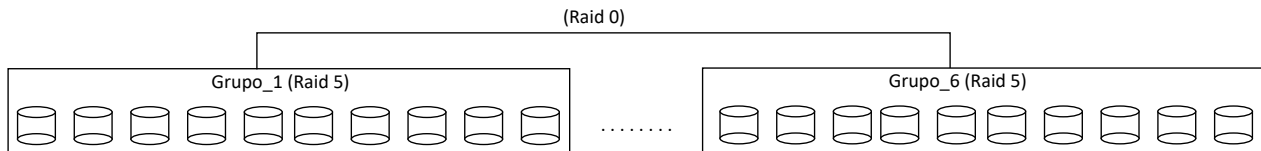
- c) **Calcula** el tamaño de un sector (bytes de datos + bytes de control).

```
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 = $(10 - 1) \times G = 54 \text{ discos} \times 1 \text{ TByte} = 54 \text{ TBytes}$

Porcentaje de información redundante = $(1 \times G) / N = 1/10 = 10\%$

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:

$$\text{MTTF_RAID50} = \frac{\text{MTTF_grupo}}{G}$$

- 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)

$\text{MTTF_grupo} = \text{MTTF_disco} / N \times \text{MTTF_disco} / (N-1) \times \text{MTTR} = \text{MTTF_disco}^2 / 10 \times 9 \times \text{MTTR} = (10^5 \text{ horas})^2 / 10 \times 9 \times 10 \text{ horas} = 111.11 \times 10^5 \text{ horas}$

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

- 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 = $0,5 * 3 = 1,5 \rightarrow 1,2E9 / 1,5 / 1E9 = 0,8 \text{ s}$
 Fase 2 IPCeq = $1 * 3 = 3 \rightarrow 2,4E9 / 3 / 1E9 = 0,8 \text{ s}$
 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 = $60 * 0,4 + 60 * 1,2 + 70 * 0,4 = 24 + 72 + 28 = 124 \text{ J}$
 EVLIW = $20 * 0,8 + 20 * 0,8 + 30 * 0,4 = 16 + 16 + 12 = 44 \text{ J}$
 $G = 124 / 44 = 2,82 \rightarrow 182\%$

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 \rightarrow 2 RISC y 10 discos $\rightarrow (0,4 * 60 + (1,2/2) * 120 + (0,4/10) * 160) / 1,04 = 102,4 / 1,04 = 98,46 \text{ W}$
 AV \rightarrow 8 VLIW y 10 discos $\rightarrow (0,8 * 20 + (0,8/8) * 160 + (0,4/10) * 120) / 0,94 = 36,8 / 0,94 = 39,15 \text{ W}$