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)																
(b)																
(c)																

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**

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

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

COGNOMS:

NOM:

Problema 2. (2 puntos)

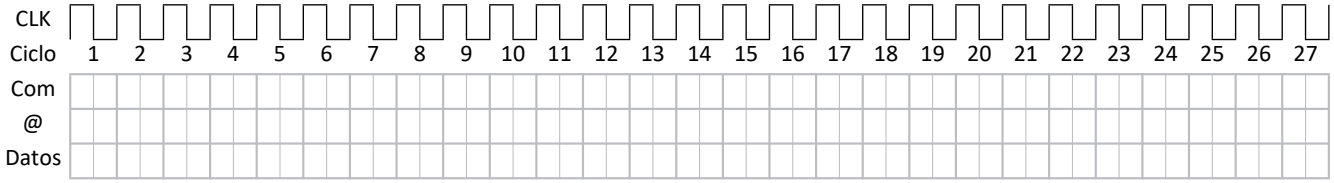
Una **CPU** está conectada a una cache de instrucciones (**$I**) y una cache de datos (**$D**). El conjunto formado por **CPU**+**$I**+**$D** esta conectado a una memoria principal formada por un único módulo DIMM estándar de 16 GBytes. Este DIMM tiene 8 chips de memoria **DDR**-SDRAM (**Double Data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

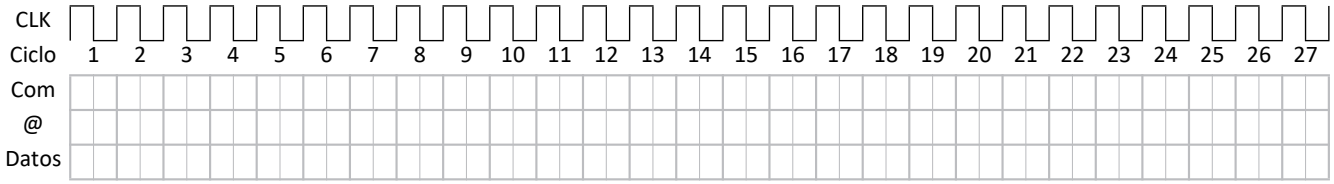
Bloque	A	B	C	D
Banco	0	0	1	1
Página	10	10	10	25

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.

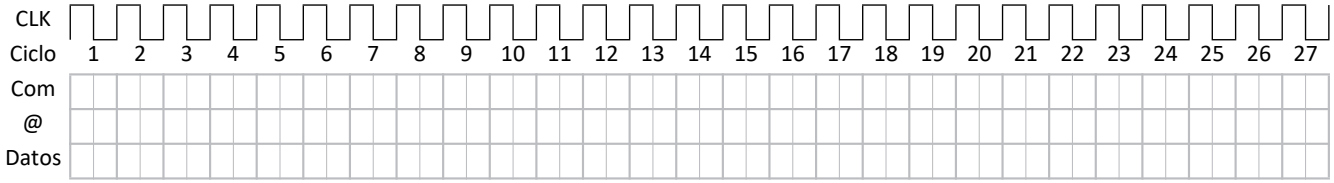
a) **Rellena** el siguiente cronograma para la lectura de los bloques A y B.



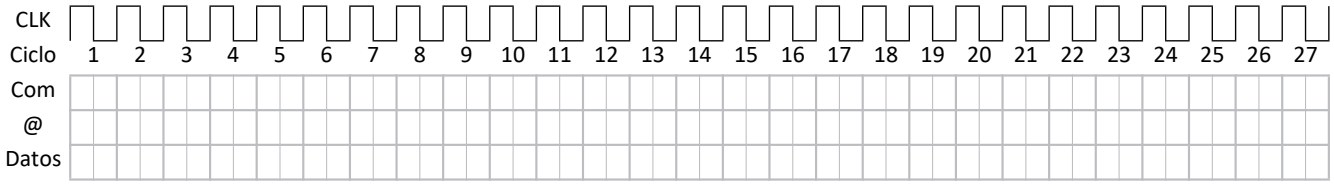
b) **Rellena** el siguiente cronograma para la lectura de los bloques A y C.



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



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).

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).

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.

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**.

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.

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.

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

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.

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.

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

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

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.