

COGNOMS:	<input type="text"/>		
NOM:	<input type="text"/>	DNI/NIE:	<input type="text"/>

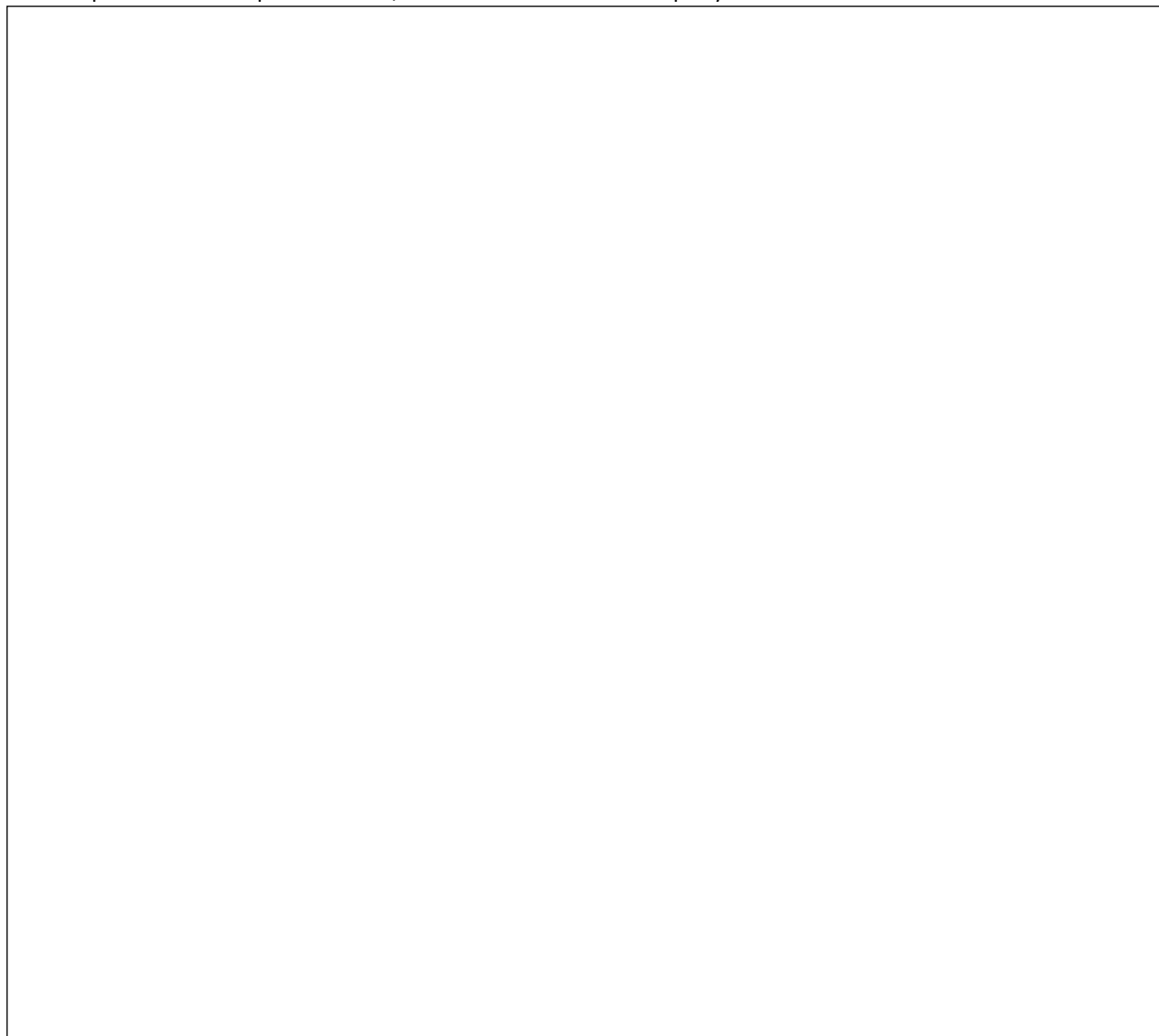
IMPORTANTE leer atentamente antes de empezar el examen: Escriba los apellidos, el nombre y el DNI/NIE antes de empezar el examen. Escriba un solo carácter por recuadro, en mayúsculas y lo más claramente posible. Es importante que no haya tachones ni borrones y que cada carácter quede enmarcado dentro 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, todo lo que haya fuera de ellos es ignorado. La identificación del alumno se realiza de forma automática, no seguir correctamente estas instrucciones puede comportar no tener nota.

Problema 1. (3 puntos)

Dado el siguiente código escrito en C, que compilamos para un sistema linux de 32 bits:

```
typedef struct {  
    char a;  
    short b[4];  
    double c;  
    double *d;  
} s1;  
  
typedef struct {  
    short e;  
    char f;  
    s1 g[10];  
} s2;
```

- a) **Dibuja** cómo quedarían almacenadas en memoria las estructuras **s1** y **s2**, indicando claramente los desplazamientos respecto al inicio, el tamaño de todos los campos y el tamaño de los structs.



- b) **Escribe** UNA SECUENCIA MÍNIMA DE INSTRUCCIONES que permita mover **x.g[y].b[1]** al registro **%ax**, siendo **x** una variable de tipo **s2** cuya dirección está almacenada en **%ecx**, e **y** una variable entera almacenada en **%ebx**. **Indica** claramente la expresión aritmética utilizada para el cálculo de la dirección.

- c) **Define** de forma CLARA Y CONCISA los conceptos de latencia y ancho de banda y sus unidades de medida.

COGNOMS:

NOM:

 DNI/NIE:

Problema 2. (3,6 puntos)

Queremos programar un simulador de cache, similar al de la práctica 5, con las siguientes características: direcciones de 32 bits, mapeo directo, tamaño de cache 16 Kbytes, tamaño de bloque 64 bytes. La rutina a programar es:

void reference (unsigned int address)

a) **Escribe** 4 líneas de código en C para calcular las siguientes variables locales a partir del parámetro **address**

```
unsigned int byte;      // posición del byte dentro del bloque
unsigned int bloque_m;  // bloque de memoria
unsigned int linea_mc;  // línea de cache donde se mapea dicho bloque
unsigned int tag;       // etiqueta asociada a dicho bloque
```


Un estudiante ha declarado las siguientes estructuras de datos globales:

```
unsigned int tags[256]; // vector correspondiente a la memoria de etiquetas
unsigned int v[256];    // vector correspondiente a los bits de validez
```

b) **Escribe** un fragmento de código en C para calcular la variable local booleana **miss** cuando se accede a la dirección **address**. Puedes usar las variables del apartado a). Se valorará la brevedad y sencillez del código.

En la práctica 6, para el simulador de la cache **Copy back + Write Allocate** (con las mismas características que en los apartados anteriores), el mismo estudiante ha reutilizado la práctica 5 añadiendo la siguiente estructura de datos global:

```
unsigned int d[256]; // vector correspondiente a los dirty bits
```

Recordemos asimismo que la rutina a programar en la práctica 6 es:

```
void reference (unsigned int address, unsigned int LE) // LE=0 lectura, LE=1 escritura
```

c) **Escribe** un fragmento de código en C para actualizar el dirty bit correspondiente. Puedes usar las variables de los apartados a) y b). Se valorará la brevedad y sencillez del código.

Hemos ejecutado los siguientes fragmentos de código, prácticamente idénticos a los usados en la práctica 7, en un procesador con un sólo nivel de cache de datos.

Código A	Código B	Código C
<pre>for (i=0, j=0; j<N; j++){ sum = sum + v[i]; i = i + step; }</pre>	<pre>for (i=0, j=0; j<N; j++){ if (i >= limite) i = 0; sum = sum + v[i]; i = i + line_size; }</pre>	<pre>for (i=0, j=0; j<N; j++){ if ((j % limite)==0) i = 0; sum = sum+ v[i]; i = i + cache_size; }</pre>

Cada elemento de $v[i]$ ocupa 1 byte.

$line_size$ es el tamaño de bloque en bytes.

$cache_size$ es el tamaño de la cache en bytes.

$N = 1000000$

El vector V es suficientemente grande para que ningún acceso exceda el tamaño del vector.

En este procesador no es posible contar directamente el número de fallos en la cache de datos, pero hemos medido el tiempo de ejecución de cada uno de los códigos. El tiempo de ejecución de cada código es $T_{total} = T_{ideal} + T_{fallos}$ donde T_{ideal} es el tiempo que tardaría el código si no hubiese fallos de cache y T_{fallos} es el tiempo adicional debido a los fallos de cache. T_{ideal} puede ser distinto según el código. La siguiente tabla muestra el tiempo de ejecución (T_{total}) en milisegundos en función de las variables $step$ y $limite$.

Código A	step	4	8	16	32	64
	T_{total}	15 ms	20 ms	30 ms	50 ms	50 ms
Código B	limite	$2 \cdot 1024$	$4 \cdot 1024$	$5 \cdot 1024$	$6 \cdot 1024$	$8 \cdot 1024$
	T_{total}	15 ms	15 ms	42 ms	55 ms	55 ms
Código C	limite	1	2	3	4	5
	T_{total}	20 ms	20 ms	60 ms	60 ms	60 ms

La precisión es de 1 ms, por lo que los tiempos son aproximados y pequeñas variaciones en el número de fallos pueden dar como resultado el mismo tiempo. En caso que sea necesario asume que el algoritmo de reemplazo es LRU.

d) **Calcula** el tamaño de línea de la cache. **Justifica** la respuesta.

e) **Calcula** el tamaño de la cache. **Justifica** la respuesta.

f) **Calcula** la asociatividad de la cache. **Justifica** la respuesta.

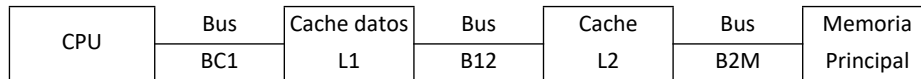
COGNOMS:

NOM:

 DNI/NIE:

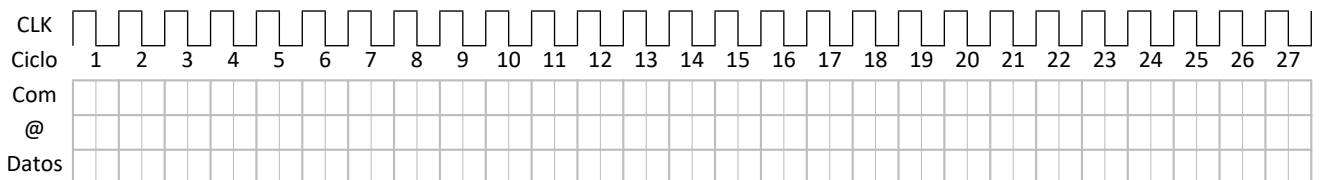
Problema 3. (3,4 puntos)

Disponemos de la jerarquía de memoria (solo datos, ignoramos las instrucciones) mostrada en la siguiente figura:



La memoria principal está formada por un único módulo DIMM estándar de 4 Gbytes. Este DIMM tiene 8 chips de memoria **DDR-SDRAM (Double Data Rate Synchronous DRAM)** de un byte de ancho cada uno. 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.

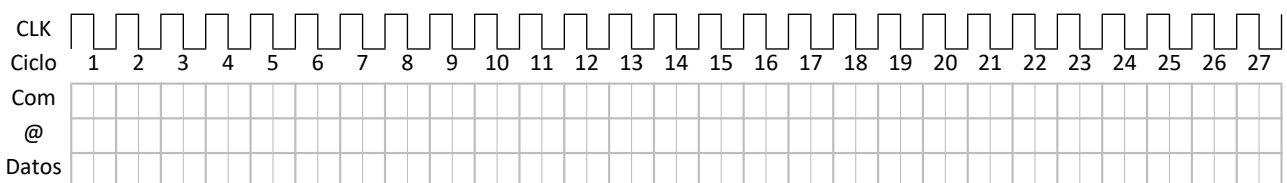
- a) **Rellena** el siguiente cronograma indicando la ocupación de los distintos recursos (bus de datos, bus de direcciones, bus de comandos) para una operación de lectura de un bloque de 64 bytes de la memoria **DDR**.



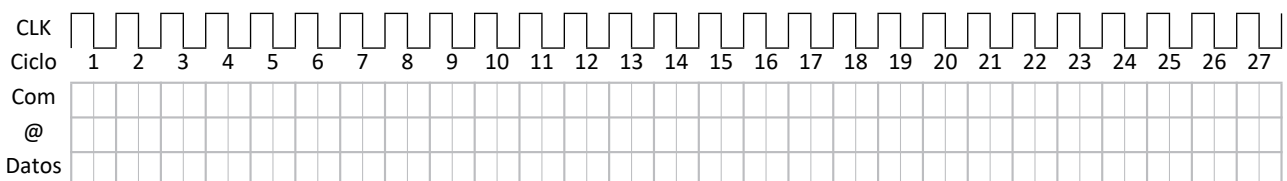
Cuando en la L2, que es copy-back, se reemplaza un bloque modificado, primero se tiene que escribir el bloque reemplazado en la memoria principal y a continuación leer el bloque que se necesita. En este caso, nuestro controlador de memoria envía los comandos necesarios a la DDR-SDRAM de forma que ambos accesos sean servidos lo más rápidamente posible. Suponiendo que inicialmente no hay ninguna página abierta. Rellena los siguientes cronogramas, indicando la ocupación de los distintos recursos, para una operación de escritura seguida de una lectura, de forma que se maximice el ancho de banda útil del bus.

Observación importante: El bus de datos, a diferencia del de direcciones y el de comandos, realiza transmisiones en ambos sentidos. En las memorias reales no es posible cambiar el sentido de la transmisión de forma inmediata. Desde que se reciben los últimos datos de una escritura hasta que se envían los primeros de la siguiente lectura (y también viceversa) debe transcurrir un ciclo como mínimo. Ten en cuenta esta restricción en los siguientes cronogramas.

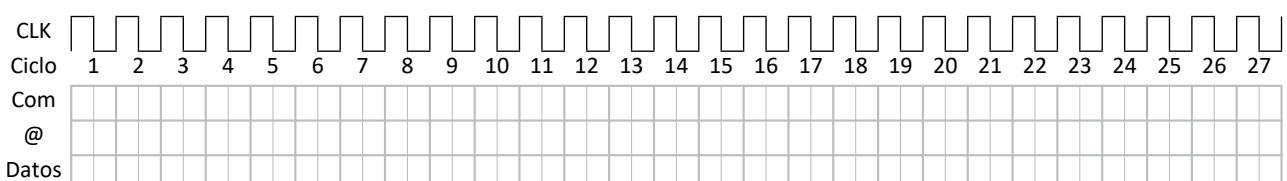
- b) Ambos bloques están ubicados en el mismo banco pero en páginas distintas.



- c) Ambos bloques están ubicados en la misma página.



- d) Ambos bloques están ubicados en bancos distintos.



El mapeo de las direcciones de memoria a bancos, filas y columnas puede variar el comportamiento del acceso a una misma estructura de datos guardada en memoria.

Asume que sólo se está accediendo a un vector de enteros ($v[N]$) de 1GB de tamaño total. Por simplicidad, asume que se accede a la dirección $0x00000000$ y se espera el comportamiento descrito en cada pregunta.

- e) **¿Cómo** deberían distribuirse los bits de la dirección para que páginas de DRAM contiguas se mapeen en el mismo banco? Ordena los bits de chip, columna, fila y banco según su mapeo en la dirección (no importa cuantos bits forman cada uno de los campos).

Mayor peso			Menor peso

- f) **Cómo** deberían repartirse los bits de la dirección para que páginas de DRAM contiguas se mapeen en bancos distintos? Ordena los bits de chip, columna, fila y banco según su mapeo en la dirección (no importa cuantos bits forman cada uno de los campos).

Mayor peso			Menor peso