COGNOMS:

NOM: DNI/NIE:

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.

Problema 1. (3.5 puntos)

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

```
typedef struct {
  int t;
  short s[5];
  int u;
} st1;
```

```
typedef struct {
  st1 r[5];
  int n;
} st2;
```

```
void examen(st1 *p, st2 *q) {
    short v1, v2;
    st1 aux;
       . . .
}
```

a) **Dibuja** cómo quedarían almacenadas en memoria las estructuras **st1** y **st2**, indicando claramente los desplazamientos respecto al inicio, el tamaño de todos los campos y el tamaño de los structs.

b) **Dibuja** el bloque de activación de la rutina examen, indicando claramente los desplazamientos respecto a **%ebp** y el tamaño de todos los campos.

c) **Traduce** a ensamblador x86 la instrucción **aux.s[v2]=3;** `que se encuentra dentro de la subrutina`

Sea un computador con una memoria cache con capacidad para 8 líneas de 2 bytes cada una, y que está organizada asociativamente en 4 conjuntos. Por su parte, la memoria principal se encuentra organizada en bytes y posee una capacidad de 32 bytes.

d) **Indica** cuáles son los bits de toda dirección de memoria principal (@MP) que se han de utilizar para determinar el conjunto al que pertenece una línea de cache, cuáles a la etiqueta, y cuáles al byte dentro de la línea de cache. Justifica tu respuesta.

La política de escritura de la cache es copy back, write-allocate. La política de reemplazo es FIFO, y el primer acceso en fallo a un conjunto de cache se guarda en la vía 0. Sabemos que inicialmente la memoria cache está vacía y que para cualquier posición **@** de memoria principal su contenido es **2@** (e.g. el contenido de la posición 6 es 12). Dada la siguiente secuencia de instrucciones ejecutada por el procesador:

```
movb 17, %ah - movb 6, %al - movb $100, 8 - movb $101, 16 - movb 9, %bh - movb 25, %bl
```

e) **Dibuja** el contenido final de la memoria cache y de las posiciones de memoria principal involucradas. **Indica** también el contenido final de los registros %ah, %al, %bh y %bl. Para la cache debes dar el número de conjunto y los contenidos de todos los campos de cada línea involucrada (etiqueta, datos y bits de estado). Pon en decimal los valores de cada byte de datos.

El tiempo de acceso a la memoria cache es de 1 ciclo (Tsa). Para leer o escribir un bloque en la memoria principal se emplean 8 ciclos. El tiempo dedicado a realizar los accesos a memoria de dicha secuencia de operaciones es de 100 ns.

f) **Calcula** a qué frecuencia está funcionando el procesador que ejecuta la secuencia de instrucciones del apartado e), e indica cuál es la tasa de aciertos y fallos que se obtiene.

COGNOMS:

NOM: DNI/NIE:

Problema 2. (3 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;  // linea 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

`for (i=0, j=0; j<N; j++){`
`  sum = sum + v[i];`
`  i = i + step;`
`}`
	`for (i=0, j=0; j<N; j++){`
	`  if (i >= limite) i = 0;`
	`  sum = sum + v[i];`
	`  i = i + line_size;`
	`}`
		`for (i=0, j=0; j<N; j++){`
		`  if ((j % limite)==0) i = 0;`
		`  sum = sum+ v[i];`
		`  i = i + cache_size;`
		`}`

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*1024	4*1024	5*1024	6*1024	8*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. (3.5 puntos)

Se ha ejecutado un programa P en un sistema con un solo procesador y un solo disco D (un PC de sobremesa que denominaremos PC1) y se ha visto 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 11% del tiempo de la ejecución de P en el PC1.
- Fase 2: Código PARALELIZABLE, ocupa el 64% del tiempo de la ejecución de P en el PC1. Se puede paralelizar perfectamente sin que haya overhead de comunicación/sincronización.
- Fase 3: Código de E/S que pasa todo su tiempo accediendo al disco D, ocupa el 25% 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 32 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.

b) **Calcula** cuántos procesadores serían necesarios para obtener un speed-up de 3 en el PC2.

Al PC2 con 32 procesadores le añadimos un sistema RAID5 formado por discos iguales que el disco D del PC1. Esta configuración permite aumentar el ancho de banda máximo de la E/S en 5x en el caso de las lecturas, y en 4x en el caso de las escrituras, cuando los accesos, tanto para lecturas como para escrituras, son secuenciales. A este sistema le llamamos PC3. Sabemos que todos los accesos a disco del programa P son secuenciales, y que el 100% de los accesos a disco son lecturas.

c) **Calcula** el máximo speed-up que podría conseguirse al ejecutar el programa P con el PC3 respecto PC1.

d) **Calcula** cuántos discos tiene el sistema RAID5 del PC3. Si no se especifica claramente cómo se han realizado los cálculos, el apartado se valorará con un 0.

Otra opción que se ha barajado para mejorar el rendimiento del sistema es añadir un RAID6 de 8 discos como el D en lugar del sistema RAID5 del PC3. El disco D tiene una capacidad de 1 Terabyte y un ancho de banda de 400 MBytes/s.

e) **Describe** las principales características de este sistema RAID6 , **dibujando** un esquema de cómo se distribuyen los datos y especificando el tipo de entrelazado, número de Terabytes de información redundante, número mínimo de discos que han de fallar para que el sistema deje de ser operativo, ancho de banda **máximo** de las lecturas en acceso secuencial y ancho de banda (en general) de las escrituras en acceso aleatorio.

La CPU del sistema (en todos los casos) está alimentada con una tensión de 1,2 V, tiene una memoria cache de datos de primer nivel L1D 2-asociativa con acceso paralelo a etiquetas y datos. La capacidad de la cache es de 128 KB y el tamaño de bloque de cache de 64 bytes. Las direcciones físicas son de 48 bits. La corriente de fugas de la memoria RAM estática es de 3 micro Amperios por bit. Para simplificar el problema solo tendremos en cuenta las memorias de etiquetas y datos (ignoraremos por tanto los bits V, D, etc ...)

f) **Calcula** la potencia media estática (debida a fugas) de la cache.
Pista: calcula el tamaño en bits de las memorias de datos y etiquetas.

La energía consumida durante un acceso la memoria de etiquetas es de 5 nJ (nanoJoules) por vía, y la consumida durante un acceso a la memoria de datos es de 25 nJ por vía. El consumo del resto de componentes de la cache es despreciable.

g) **Calcula** la energía dinámica consumida al realizar una lectura en acierto a la cache.