Nº Orden	Apellido y nombre	L.U.	Cantidad de hojas

Organización del Computador 2
Recuperatorio del Primer Parcial – 01/12/2015

1 (40)	2 (40)	3 (20)	

Normas generales

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Está permitido tener los manuales y los apuntes con las listas de instrucciones en el examen. Está prohibido compartir manuales o apuntes entre alumnos durante el examen.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

Ej. 1. (40 puntos)

En Orga2 llamamos `miraQueCoincidencia` al filtro que dadas dos imágenes de tamaño NxN en escala de grises nos devuelve otra de similares características, llamada `laCoincidencia`, donde sólo se encuentran aquellos pixeles que coinciden en ambas imágenes, los que no coinciden son puestos en blanco. La única característica distinta es que los pixeles de `laCoincidencia` son de tamaño `unsigned short` en lugar de `unsigned char`. Para ser más concretos la podemos definir de la siguiente forma:

$$miraQueCoincidencia[ij] = \begin{cases} A[ij] & \text{si } A[ij] = B[ij] \\ 255 & \text{si no} \end{cases}$$

(a) **(25p)** Implementar sólo la porción de código ensamblador necesario dentro de un ciclo para el cálculo simultáneo de pixeles de `miraQueCoincidencia` con SIMD, suponiendo que `RDI` y `RSI` contienen los punteros a las imágenes originales, y `R8` a la de destino.

(b) **(15p)** Utilizando el ejercicio anterior, escribir el código ensamblador de `miraQueCoincidencia` con SIMD. Puede simplemente indicar, donde lo necesite, *"aquí va la porción de código del item anterior"*. La aridad de la función es:

```
void miraQueCoincidencia( unsigned char *A, unsigned char *B, unsigned int N,
                          unsigned short *laCoincidencia )
```

Notas:

- Para cada operación que use registros XMM debe indicar el estado del registro destino mediante un dibujo de su contenido.
- Puede asumir que el tamaño N de las imágenes es múltiplo de 16.
- Se debe recorrer una sola vez cada imagen.

Ej. 2. (40 puntos)

Considerar una estructura que representa una lista de cosas, enlazadas de forma simple desde el último elemento al primero. La única estructura que compone a esta lista de cosas será la siguiente:

```
typedef struct {
    nodo* anterior;
    void* cosa;
} __attribute__((__packed__)) nodo;
```

(a) **(30p)** Implementar en código ensamblador la función `borrarCositas`, que recibe una lista de cosas, una función para analizar cosas, y modifica la lista tal que sólo quedan en ella los nodos cuyo análisis resulta negativo. La aridad de la función es:

```
void borrarCositas( nodo** n, bool(*analizarCosa)(void*) )
```

Se deberán borrar aquellos nodos tales que al aplicar su atributo "cosa" a la función `analizarCosa` recibida por parámetro, devuelva falso.

Importante:
- En la lista resultante los nodos deberán mantener el mismo orden relativo que en la original.
- El puntero a la función `analizarCosa` pasada por parámetro representa una función que recibe un `void*` y con algún cierto criterio devuelve `true` o `false`. El tipo `bool` está representado como un `byte` que vale 0 ó 1.
- Puede asumir que cuenta con la implementación de la función `void borrarNodo( nodo *n )` que libera toda la memoria utilizada por el nodo correctamente.
- La lista puede ser vacía.

(b) **(10p)** Implementar en código ensamblador la función void `borrarNodo( nodo *n )` utilizada en el ítem anterior.

Importante:
- Puede asumir que cuenta con la implementación de la función `void borrarCosa( void *c )` que libera toda la memoria utilizada por la cosa.

Ej. 3. (20 puntos)

Considerar la siguiente estructura que representa una lista circular de caracteres:

```
typedef struct {
    nodo* siguiente;
    char letra;
} __attribute__((__packed__)) nodo;
```

(a) **(20p)** Implementar en código ensamblador la función `imprimirLista`, que recibe una lista circular de caracteres e imprime todos los caracteres de la lista de forma contigua y, al mismo tiempo, retorna la cantidad total de elementos de la lista. La aridad de la función es:

```
int imprimirLista( nodo* l )
```

Importante:
- No se puede utilizar memoria dinámica.
- La cantidad n de elementos de la lista es $2 \leqslant n \leqslant 31$.
- Sólo se puede llamar a `printf` una sola vez.