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

Organización del Computador 2
Recuperatorio del Primer Parcial – 30/06/2015

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)

Considerar la estructura `bajaLista` que representa una lista de strings almacenados en el campo `nombre` de sus respectivos nodos. La lista se implementa a partir de enlazar de forma simple los nodos entre sí. La única estructura que compone a la lista será la siguiente:

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

(a) **(28p)** Implementar en ASM la función `filtrarBajaLista`, que recibe una *bajaLista*, una función de comparación y un string de comparación, y modifica la lista tal que sólo quedan en ella los nodos que cumplen con la condición dada por la función y el string de comparación. La aridad de la función es:

```
void filtrarBajaLista( nodo** n, bool(*f)(char*,char*), char* nombreCmp )
```

Se deberán borrar aquellos nodos tales que la función `f`, aplicada a su *nombre* y a *nombreCmp*, devuelva *falso*.

Importante:
- En la lista resultante los nodos deberán mantener el mismo orden relativo que en la original.
- El puntero a la función `f` pasada por parámetro representa una función que recibe dos strings y determina, con algún cierto criterio, si la comparación es cierta entre ambos strings, devolviendo `true` o `false`. El tipo `bool` está representado como un `byte` que vale 0 ó 1.
- Asuma que cuenta con la implementación de la función `void nodoBorrar( nodo *n )` que libera la memoria utilizada por el nodo correctamente.
- La lista puede ser vacía.

(b) **(12p)** Implementar en ASM la función `void nodoBorrar( nodo *n )` utilizada en el ítem anterior.

Ej. 2. (40 puntos)

En Orga2 llamamos `sumMatricialConSigno` a una función muy extraña que dada una matriz cuadrada de dimensión n calcula dos sumatorias con los elementos de la misma. Por un lado, la sumatoria de los elementos positivos de la matriz entre sí y, por el otro, la sumatoria de los elementos negativos de la matriz entre sí. Esta función recibe como parámetros la matriz M a procesar, su dimensión n y dos punteros donde guardar los resultados de ambas sumatorias. El prototipo de la función es el siguiente:

```
void sumMatricialConSigno( char* M, unsigned int n, short* sumPositiva, short* sumNegativa )
```

Como se puede ver en la aridad de la función las sumatorias buscadas no necesariamente entrarán en un `char` y se las debe devolver como `short`.

(a) **(15p)** Implementar la porción de código ASM necesario dentro de un ciclo para el cálculo simultáneo con SIMD sólamente de la `sumPositiva`. Asuma que ya tiene en `XMM0` los primeros 16 bytes de la matriz.

(b) **(15p)** Implementar la porción de código ASM necesario dentro de un ciclo para el cálculo simultáneo con SIMD de `sumPositiva` y `sumNegativa`. Asuma que ya tiene en `XMM0` los primeros 16 bytes de la matriz. Tome como base la implementación del punto anterior y realice las modificaciones necesarias para procesar ahora ambas sumatorias.

(c) **(10p)** Implementar el código completo ASM de `sumMatricialConSigno` con SIMD. Utilice la implementación del punto anterior. Puede hacer referencia al código que necesite del punto anterior sin escribirlo todo nuevamente.

Importante:

- Para cada operación que use registros XMM debe indicar el estado del registro destino mediante un dibujo de su contenido.
- El valor de la dimensión de la matriz n es siempre mayor que 0 y múltiplo de 16.

Ej. 3. (20 puntos)

Se cuenta con una lista doblemente enlazada de números que respeta la siguiente estructura:

```
typedef struct {
    numero* siguiente;
    numero* anterior;
    unsigned short n;
} __attribute__((__packed__)) numero;
```

La lista almacena los valores de un vector que se desea pasar como parámetro a la función: `void dameVector(unsigned short* numeros[])`, esta toma un arreglo de punteros a `short` terminado en `null`.

Se busca implementar la función `void ahiVaElVector(numero* listaNumeros)` que se encarga de convertir la lista en un vector de números y llamar a `dameVector` con el vector resultante como parámetro.

(a) **(20p)** Escribir en ASM la función `ahiVaElVector` considerando que la única memoria auxiliar disponible es la pila.

Importante:

- No está permitido utilizar memoria dinámica.
- Los elementos del vector deben ubicarse en el mismo orden que los elementos de la lista.