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

Organización del Computador 2
Primer parcial – 5/10/2019

1 (40)	2 (30)	3 (30)	

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 una lista circular de nodos respetando la siguiente estructura:

```
typedef struct node {
    int      counter,
    void**   data,
    node_t*  next
} node_t;
```

Cada nodo contiene un puntero al siguiente (`next`), un doble puntero a `data` y un entero dado por `counter`; `data` representa un puntero a un arreglo de punteros de tamaño `counter`.

(10p) a. Implementar en ASM la función `int sonLosTresUltimosNull(void** data, int counter)` que toma un arreglo de punteros de longitud `counter` y devuelve `1` en caso de que los últimos tres punteros del arreglo sean *null*, y `0` en caso contrario.

(30p) b. Implementar en ASM la función `void filtrarSiTresUltimosSonNull(node_t** primero)` que toma un doble puntero a un nodo y borra de la lista los nodos cuyo `dato` evalúe a 1 utilizando la función anterior. Recordar actualizar el doble puntero al primer nodo de la lista, de ser necesario. Considerar que para borrar un nodo, se debe liberar la memoria del nodo mismo y el arreglo de datos, ambos utilizando la función `free`.

Ej. 2. (30 puntos)

La esteganografía consiste en el estudio y aplicación del ocultamiento de mensajes en objetos, posibilitando la comunicación de forma tal que pase el mensaje inadvertido para quienes desconozcan la técnica empleada.

Dada una imagen de `512x512` píxeles, representada mediante una matriz de `pixeles en escala de gris` de un byte cada uno, y un string de largo múltiplo de 16 y menor a 128 (contando el `caracter nulo`), se desea esconder el string en la imagen aplicando un mismo patrón por cada caracter. Para ello los 8 bits de cada byte del texto deberán ser distribuidos a lo largo de 4 píxeles consecutivos de la imagen, aplicándose de forma secuencial la técnica hasta haber finalizado el texto (ocultándose incluso el caracter nulo).

En caso de haberse procesado el texto por completo, deberá repetirse el ocultamiento del texto hasta finalizar la imagen.

A continuación se presenta el patrón a aplicar por cada byte del string:

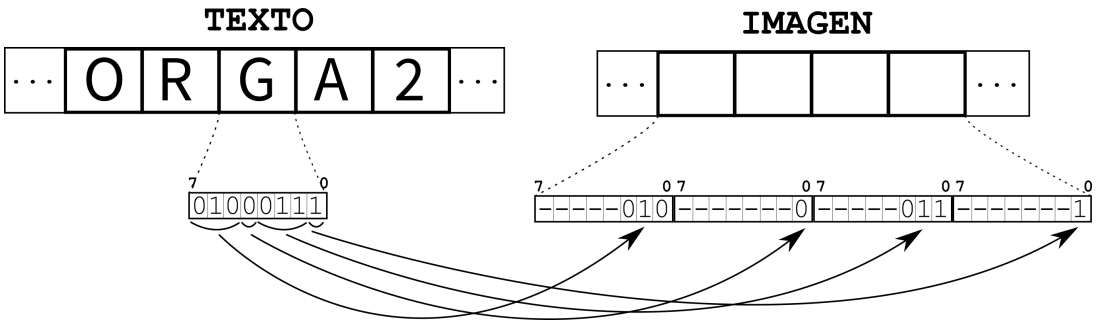

Figura 1: Ejemplo de texto escondido en una imagen.

Cada caracter del texto se encodifica en 4 píxeles consecutivos:

- Los bits 0-2 del primer píxel se sobrebrescriben con los bits 5-7 del caracter.
- El bit 0 del segundo píxel se sobrebrescribe con el bit 4 del caracter.
- Los bits 0-2 del tercer píxel se sobrebrescriben con los bits 1-3 del caracter.
- El bit 0 del cuarto píxel se sobrebrescribe con el bit 0 del caracter.

(30p) Implementar en SIMD la función `esconder_texto(char* imagen, char* texto)` que realiza el proceso de estenografía del texto en la imagen siguiendo el patrón detallado.

Ej. 3. (30 puntos)

Considerar una convención especial donde al principio y al final de cada función se ejecutan las siguientes instrucciones:

```
push rbp
mov rbp, rsp
push rdi
push rsi
push rdx
push rcx
push r8
push r9
sub rsp, 128
movdqa [rbp-56], xmm0
movdqa [rbp-72], xmm1
movdqa [rbp-88], xmm2
movdqa [rbp-104], xmm3
movdqa [rbp-120], xmm4
movdqa [rbp-136], xmm5
movdqa [rbp-152], xmm6
movdqa [rbp-168], xmm7
...
add rsp, 176
pop rbp
ret
```

(10p) a. Graficar el contenido de la pila luego de dos llamados a funciones anidados.

(20p) b. Implementar la función `printStackTrace(FILE* file)` que imprime sobre el archivo `file` en líneas independientes y utilizando `fprintf`, los datos siguientes en orden: registros `rdi`, `rsi`, `xmm0`, `xmm1` y los dos primeros valores de 8 bytes pasados por pila. Esta acción se realiza para cada uno de los llamados a función anidados dentro de la pila. Considerar que inicialmente el valor de `rbp` es cero para el primer llamado a función.

Nota: El formato para imprimir 8 bytes en hexadecimal es `%08X`. Los registros `xmm` deben ser impresos como dos registros de 8 bytes consecutivos.