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

Organización del Computador 2
Recuperatorio del Primer Parcial – 03/07/2014

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.

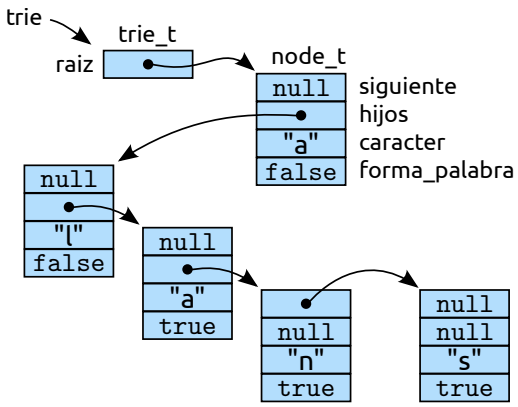
Ej. 1. (40 puntos)

Considerando una estructura de trie como la siguiente:

```
typedef struct trie_t {
        nodo *raiz;
} __attribute__((packed)) trie;

typedef struct nodo_t {
        struct nodo_t *sig;
        struct nodo_t *hijos;
        char c;
        bool fin;
} __attribute__((packed)) nodo;
```

donde `bool` equivale a 1 `byte`.



La estructura almacena letras, las mismas están ordenadas lexicográficamente por *nivel* del trie. Un *nivel* está compuesto por los nodos que se encuentran enlazados a través del campo `sig`.

Se pide implementar una extraña funcionalidad que consiste en cortar todas las palabras del trie a partir de que contengan una determinada letra pasada por parámetro. Es decir, buscar entre todos los niveles del trie al caracter pasado por parámetro, marcarlo como fin de palabra y eliminar todo el resto de las ramas del trie. En caso de que se realice al menos una poda, la función devolverá `0`, en caso contrario `1`. La aridad de la función será: `int podar_trie(trie* t, char c)`

(a) **(10p)** Implementar en `lenguaje C` la función `podar_trie`.

(b) **(30p)** Implementar en `ASM` la función `podar_trie` utilizando como pseudo-código la implementación en `lenguaje C` del punto anterior.

Notas:

- La función no debe perder memoria y debe dejar la estructura del trie en un estado consistente.
- Se cuenta con la función `trie_borrar_nodos` que a partir de un doble puntero a un nodo, borra todos los nodos internos que le siguen incluyéndolo, es decir, borrar el *nodoInterno*, todos los *siguientes* y todos los *hijos*.
  La aridad de la función es `void trie_borrar_nodos(nodo **nodoInterno)`

Ej. 2. (40 puntos)

En Orga2 llamamos `miraQueFiltro` al filtro que dadas dos imágenes en escala de grises nos devuelve simultáneamente otras dos también en escala de grises.
La primera imagen devuelta, que llamaremos `igualdadRara` es una imagen donde sólo se encuentran aquellos pixeles que coinciden en ambas imágenes, los que no coinciden son puestos en blanco. Para ser más concretos la podemos definir de la siguiente forma:

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

La segunda imagen devuelta que llamaremos `sumaRara` es una imagen donde los pixeles de ambas imágenes que no coinciden entre sí son sumados con saturación. Aquellos que coinciden entre sí son definidos como el mínimo valor representable. En terminos matemáticos:

$$sumaRara[ij] = \begin{cases} saturar(A[ij] + B[ij]) & \text{si } A[ij] \neq B[ij] \\ 0 & \text{si no} \end{cases}$$

(a) **(10p)** Implementar la porción de código ASM necesario dentro de un ciclo para el cálculo simultáneo de pixeles de `igualdadRara` con SIMD para dos imágenes suponiendo que `RAX` y `RBX` contienen los punteros a las mismas.

(b) **(10p)** Implementar la porción de código ASM necesario dentro de un ciclo para el cálculo simultáneo solamante de la suma saturada de pixeles utilizada en la definición de `sumaRara` con SIMD para dos imágenes suponiendo que `RAX` y `RBX` contienen los punteros a las mismas.

(c) **(20p)** Utilizando los dos ejercicios anteriores, escribir el código ASM de `miraQueFiltro` con SIMD. La aridad de la función es: `int miraQueFiltro(unsigned char *A, unsigned char *B, unsigned int filas, unsigned int columnas, unsigned char *igualdadRara, unsigned char *sumaRara)`

Notas:

- Para cada operación que use registros XMM debe indicar el estado del registro destino mediante un dibujo de su contenido.
- Puede asumir que las filas y columnas de las imágenes tienen una multiplicidad de 16.
- Se debe recorrer una sola vez cada imagen y procesar la máxima cantidad de pixeles posible.

Ej. 3. (20 puntos)

Considerando que al llamar a una función se almacena en la pila la dirección de retorno de la misma y en esta función siempre se almacena en `RBP` la base de la pila, se pide armar una función denominada `superRET` la cual toma un solo parámetro entero. Este parámetro indica la cantidad de llamados a función sobre los que se debe retornar.

Por ejemplo, si se llama a `funcionA`, luego a `funcionB`, luego a `funcionC`, y desde esta última se ejecuta `superRET(2)`, la próxima instrucción a ejecutar será el código perteneciente a `funcionA` que continuaba en su ejecución.

(a) **(5p)** Dibuje un ejemplo de la pila y cómo se almacenan las direcciones de retorno.

(b) **(15p)** Escriba el código ASM de la función `superRET`.
La aridad de la función será: `void superRET(unsigned int n)`