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

Organización del Computador 2
Primer parcial – 11/10/2011

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)

Se desea implementar la función `primer_y_ultimo` que se encarga de buscar el elemento más chico y más gande de una lista y colocarlo en la primera y útlima posición respectivamente. La lista esta compuesta por elementos de la forma:

```
typedef struct t_nodo {
      unsigned int id;
      struct nodo *siguiente;
      struct nodo *anterior;
} __attribute__ ((__packed__)) nodo;
```

Donde `siguiente` y `anterior` son punteros a elementos de la lista doblemente encadenada. `id` funciona como un identificador único que sirve para determinar el mayor y menor elemento de la lista. La aridad de la función principal a implementar es: `nodo* primer_y_ultimo(node* nodo)`. La misma toma un puntero al primer elemento de la lista y retorna el nuevo puntero al primer elemento de la lista.

1. (10 puntos) Construir una función en ASM que encuentre el mayor elemento de la lista. Definir convenientemente su aridad.
2. (10 puntos) Construir una función en ASM que dada una lista y un puntero a un nodo de la lista, quita al mismo de la lista. La función **NO** debe borrar el nodo. Definir convenientemente su aridad.
3. (20 puntos) Construir la función `primer_y_ultimo` en ASM, utilizando algunas de las funciones descriptas anteriormente.

Ej. 2. (40 puntos)

Se desea implementar la función `combinar` que dadas 2 imágenes de igual tamaño y en escala de grises retorna una tercera formada a partir de estas 2. Cada pixel de la imagen generada se forma de la siguiente manera:

$$dst(i,j) = (alpha \cdot (src\_1(i,j) - src\_2(i,j)))/255 + src\_2(i,j)$$

La aridad de la función es:

`void combinar (unsigned char* src_1, unsigned char* src_2, unsigned char* dst, int ancho, int alto, float alpha)`

Aclaraciones:

- La función se debe implementar utilizando instrucciones SIMD y se deben procesar por lo menos 4 elementos simultáneamente.
- No se debe perder precisión en los cálculos (salvo por las conversiones de punto flotante a entero).
- La fórmula se debe aplicar según el enunciado, no se pueden reorganizar los términos.
- El *ancho* y *alto* de las imagenes puede tener cualquier valor mayor que 16, las mismas no tienen padding.
- En cada instrucción SSE se **debe** mostrar el contenido del registro destino.

Ej. 3. (20 puntos)

Sean `f` y `g` dos funciones de enteros en enteros con la siguiente aridad:

- `int f(int x)`
- `int g(int x)`

Se pide implementar la función `componer` que se encarga de tomar las funciones `f` y `g` como parámetros y retornar una nueva función. La función `componer` debe solicitar memoria donde construir la nueva función. Esta nueva función (`gof(x)`) es el resultado de componer las dos funciones tomadas como parámetro con el siguiente comportamiento:

- `gof(x)=g(f(x))`

La aridad de la función `componer` es: `void* componer(void* f, void* g)`

1. (5 puntos) Escribir el código `ASM` de la función `gof(x)` utilizando solamente las instrucciones:
   • `push X` • `call X` • `add esp,X` • `mov [esp],X` • `ret`
   El valor `X` representa cualquier valor válido para la instrucción en cuestión.
2. (15 puntos) Escribir el código `ASM` de la función `componer`.

Suponer que las siguientes constantes corresponden a los *opcode* de las instrucciones mencionadas:

```
opcodepushmem     = PUSH [ESP+<imm>]   (4bytes, 2bytes para el inmediato)
opcodecall        = CALL <addr>        (2bytes, 4bytes para la dirección)
opcodeaddesp      = ADD ESP,<imm>      (2bytes, 2bytes para el inmediato)
opcodemovespeax   = MOV [ESP],EAX      (4bytes)
opcoderet         = RET                (1byte)
```