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

Organización del Computador 2
Primer parcial – 03/10/17

1 (35)	2 (35)	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)

En los partidos de basket, aunque es muy raro, un equipo puede hacer más del doble de puntos que el contrario. En particular, nos va a interesar estudiar los partidos en los cuales el equipo visitante haya hecho igual o más del doble de los puntos del equipo local. Tenemos la historia de una temporada de una liga de basket almacenada en una lista simplemente enlazada, en la que cada nodo almacenamos la cantidad de puntos de cada equipo y un texto que describe el partido.

```
typedef struct {
  unsigned char puntos_equipo_local;
  unsigned char puntos_equipo_visitante;
  char* descripcion;
  partido* siguiente;
} partido;
```

Se pide construir una función que tome una lista y separe los nodos de la misma en dos listas, de forma que una de estas contenga los partidos que nos interesan estudiar y otra los partidos que no nos interesan.

a- (6p) Indicar los desplazamientos dentro de la estructura, notar que no es `__packed__`.

b- (4p) Plantear la aridad de la función a realizar. Justificar el porqué de los parámetros.

c- (12p) Escribir en C la función pedida.

d- (28p) Ahora deseamos tener una función como la anterior pero que, en vez de separar en dos listas, debe borrar los nodos que no nos interesan. Dar la aridad de esa función y escribirla en ASM.

Ej. 2. (40 puntos)

Supongamos que tenemos una estructura como la que sigue:

```
typedef struct {
  int9_t a;
  uint10_t b;
  uint13_t c;
} misterio __attribute__((packed));
```

Donde el dato `a` mide 9 bits, el dato `b` mide 10 bits y el dato `c` mide 13 bits. Notar que `a` es con signo y `b` y `c` son sin signo.

Observar que un dato de tipo `misterio` mide 32 bits.

El objetivo de este ejercicio será procesar arreglos que contienen `misterio`s usando SIMD aplicando la siguiente función:

$$f(a,b,c) = \begin{cases} a+b+c & \text{si } a > 0 \\ b+c & \text{si } a = 0 \\ -a+b & \text{si } a < 0 \end{cases}$$

El largo de los arreglos que procesaremos tendrán largo múltiplo de 4.

a- (20p) Escribir en ASM una función `void suma_misterio(misterio* misterios, int tamanio, uint32_t* resultado)` que aplique la función f a cada estructura `misterio` y coloque el resultado en `resultado`. [1]

b- (20p) Escribir en ASM una función `void promedio_misterio(misterio* misterios, int tamanio, float* resultado)` que calcule el promedio de los campos de cada estructura `misterio` y coloque el resultado en `resultado`. [2]

Ej. 3. (20 puntos)

La conjetura de Collatz dice que si a un número natural se le aplica sucesivamente la función:

$$c(n) = \begin{cases} n/2 & \text{si } n \text{ es par} \\ 3n+1 & \text{si } n \text{ es impar} \end{cases}$$

Entonces la sucesión eventualmente termina en 1. Por ejemplo, $10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$.

Un programador hizo una función para chequear que la propiedad valga para un n dado.

```
1  int collatz(int n) {
2    int m;
3    int resultado;
4
5    if (n %2 == 0) m = n / 2;
6    else m = 3*n+1;
7
8    if (esUno(n)) {
9      resultado = 1;            // Llegamos a 1, cumple la propiedad.
10   } else {
11     resultado = collatz(m);   // Todavia no llegamos, seguimos.
12   }
13   return resultado;
14 }
```

Tal que el código compilado de `collatz` corriendo siempre estará en el rango de direcciones [0xB000, 0xB138], de la siguiente manera:

```
0x0000B000 | collatz: push   rbp
0x0000B001 |          mov    rbp, rsp
0x........ |          (...)
0x0000B138 |          ret
```

Notar que el programador cometió un error, y en vez de llamar a la función `esUno` con parámetro `m`, la llamó con parámetro `n`. Nuestro objetivo es implementar `esUno` de tal manera que arregle el problema.

a- (5p) Dibujar la pila justo antes de hacer el llamado a la función `esUno` de la linea 11. Puede asumir que la función guarda en la pila los registros que necesite.

b- (15p) Implementar la función `int esUno(int x)` en ASM. `esUno` tiene que devolver verdadero (1) si x es igual a 1 y falso (0) si x es distinto de 1. Pero en caso que sea llamada por `collatz`, debe ignorar el parámetro, y chequear `m` en su lugar.

[1] Poscondición: `resultado[i] = f(misterios[i].a, misterios[i].b, misterios[i].c)` para todo $0 \leq i <$ `tamanio`

[2] Poscondición: `resultado[i] = (misterios[i].a + misterios[i].b + misterios[i].c) / 3` para todo $0 \leq i <$ `tamanio`