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

Organización del Computador 2
Recuperatorio del Primer Parcial – 11/07/2013

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.
- Existen tres notas posibles para este parcial: I (Insuficiente): 0 a 49 pts, RC (Rinde coloquio): 50 a 64 pts y A (Aprobado): 65 a 100 pts.

Ej. 1. (45 puntos)

Una matriz rala es aquella que contiene muchos elementos de valor 0. Para optimizar tanto en memoria como en cómputo, puede representarse a una matriz rala con listas enlazadas de elementos. En este caso, deseamos utilizar una matriz rala de *bytes con signo*. Cada elemento de la matriz tiene la siguiente estructura:

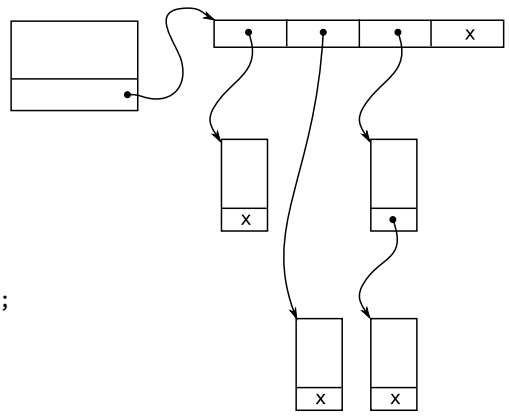

Figura 1: matriz rala de 4 x 2

```
typedef struct elemento_t {
    char valor;
    int fila, columna;
    struct elemento_t *siguiente_en_columna;
} elemento;
```

Donde `siguiente_en_columna` representa al siguiente elemento de la matriz en la misma columna con valor distinto de cero. Si no hay un siguiente elemento, el puntero es `NULL`. Con esta representación, los elementos de valor 0 no se guardan en memoria. Además, la matriz está representada por

```
typedef struct matriz_t {
    unsigned int ancho, alto;
    elemento *primeros[];
} matriz;
```

Donde `primeros` es un arreglo de punteros de tamaño `ancho`. Dada una columna j, `primeros[j]` apunta al primer elemento no nulo de esa columna, y en caso de no haberlo vale `null`.

Dadas dos matrices ralas A y B, se desea implementar un algoritmo que realice una operación entre ambas, $A = A$ op B específicamente, la operación es entre cada par de elementos en la misma fila columna de cada matriz, es decir

$$a_{i,j} = a_{i,j} \text{ op } b_{i,j}$$

Tener en cuenta que la matriz resultante debe continuar siendo rala, entonces:

a) si $a_{i,j}$ era 0, pero $a_{i,j}$ op $b_{i,j} \neq 0$, deberá crearse un nuevo elemento en A.

b) si $a_{i,j}$ no era 0, y $a_{i,j}$ op $b_{i,j} = 0$, deberá eliminarse el elemento de A en la posicion (i, j).

La aridad de la función `operar` es:

```
void operar(matriz *A, matriz *B, operacion *op)
```

y `op` es un puntero a una función que toma dos bytes a y b, y devuelve otro byte. Es decir:

```
typedef char (*operacion)(char a, char b);
```

(15p) (a) Escribir el pseudo-código de la función `operar`.

(30p) (b) Implementar en ASM de 64 bits la función `operar`.

Ej. 2. (43 puntos)

En Orga2 llamamos `SumatoriaMatricial` a la sumatoria de los elementos entre sí de una matriz. A algunos les gusta más hacer la `SumatoriaMatricialImpar`, sólo de aquellos elementos de la matriz cuyo valor es impar, a otros les gusta más la `SumatoriaMatricialPar`, sólo de aquellos con valor par.

(28p) 1. Escriba en lenguaje ensamblador utilizando SIMD, la función que dada un matriz `A` de elementos de tipo char sin signo devuelva, recorriendo sólo una vez la matriz, el máximo entre la `SumatoriaMatricialImpar` y `SumatoriaMatricialPar` de `A`. El prototipo de la función es el siguiente:

```
int MaxSumMatricialImparPar (unsigned char *A, unsigned int filas, unsigned int columnas)
```

Nota 1: Para cada operación que use registros `xmm` debe indicar el estado del registro destino mediante un dibujo de su contenido.
Nota 2: Puede asumir la multiplicidad de "filas" y "columnas" que considere conveniente.
Nota 3: Puede asumir que el resultado de la "SumatoriaMatricial" de A nunca excede el doble del tamaño de representación del tamaño original de los elementos de A.

(10p) 2. ¿Cuántos elementos de la matriz procesa simultáneamente? ¿Podría procesar más? Justifique.

(5p) 3. ¿Qué condiciones mínimas de multiplicidad para "filas" y "columnas" asume su algoritmo? ¿Por qué su algoritmo respeta estas condiciones?

Ej. 3. (12 puntos)

Suponga que Ud. mismo es un complilador de C corriendo en una máquina de 64 bits. Suponga también que un alumno de Orga2 (distinto de Ud.) le solicita que compile el código en C que se muestra a continuación, y que le muestre el código ensamblador interpretado (por ejemplo, en el caso de `gcc` esto sería `gcc -S -masm=intel codigo.c`).

El código C es el siguiente:

```
int n = 8;
int iesimoFibonacciIterativo(){
        int i = 2;
        int vector[n];
        vector[0] = 1;
        vector[1] = 1;

        for(i; i < n; i++)
                vector[i] = vector[i-1] + vector[i-2];

        return vector[n-1];
}
```

(8p) 1. Escriba en lenguaje ensamblador el segmento de código anterior según las especificaciones anteriores, respetando lo mas posible la ABI utilizada en la materia.

(4p) 2. Describa qué secciones utilizó en su código y justifique por qué puso cada fragmento de código en cada sección.

Nota: Ud. (como compilador) no tiene idea de cómo hacer optimizaciones al código.