Diseño Basado en Microprocesadores
Examen parcial diciembre 2017

Tiempo: 2 horas.

1. Describe las características generales de la memoria de los microprocesadores x86. (1 punto)

2. Describe el formato de los direccionamientos de memoria indirectos por registro de 32 bits de los microprocesadores x86 y pon un ejemplo de cada una de las combinaciones posibles. (1 punto).

3. Realiza un diagrama donde se muestren los registros de 64 bits de los microprocesadores x86-64 indicando los nombres de los registros de 64 bits así como el de los registros de 32, 16 y 8 bits a los que se puede acceder dentro de éstos. (1 punto).

4. Describe los tipos de datos escalares y empaquetados y tanto enteros como de punto flotante que pueden manejarse en los registros XMM. Realiza un diagrama de cada uno de ellos. (1 punto).

5. Escribe un fichero fuente en ensamblador para un microprocesador x86 en el que se defina una función que devuelva el resultado de evaluar la función $f(x) = ax + b$, siendo x, a y b datos de tipo `int`. El prototipo de la función es:

```
1   int fun(int a, int x, int b);
```

   La función podrá enlazarse en el código generado por un compilador de C de **32 bits.** (2 puntos).

6. Escribe un fichero fuente en ensamblador para un microprocesador x68-64 en el que se defina una función que busque en un array de datos de tipo `unsigned int` aquellos cuyos bits de 0 a 15 sean iguales a sus bits de 16 a 31 y los copie en otro array. Por ejemplo, un dato con el valor `0x1A3F1A3F` cumple la condición para ser copiado. El prototipo de la función es:

```
1   int copiar_0a15_igual_16a31(const unsigned int * ptr_origen,
2                               unsigned int num_datos,
3                               unsigned int * ptr_destino,
4                               unsigned int * num_copiados);
```

   donde:

- **ptr_origen** apunta al array con los datos entre los que buscar.
- **num_datos** indica el número de datos contenidos en el array apuntado por `ptr_origen`.
- **ptr_destino** apunta al array donde hay que copiar aquellos datos que tengan los bits 0 a 15 iguales a los bits 16 a 31.
- **num_copiados** apunta a una variable de tipo `unsigned int` donde la función debe dejar almacenado el número de datos del array origen que ha copiado en el array destino.

La función retorna 1 si realizó su trabajo correctamente y 0 si alguno de los puntero s es nulo.
La función podrá enlazarse con el código generado por un compilador de C para Linux de 64 bits. (2 puntos).

7. Escribe un fichero fuente con una función en ensamblador para un microprocesador x86-64 que use instrucciones SSE para sumar dos vectores cuyas componentes son números en punto flotante de precisión simple. El prototipo de la función es:

```
1  int sumar_vectores(const float * vector_1,
2                     const float * vecto_2,
3                     unsigned int dimension,
4                     float * vector_suma);
```

donde:

- **vector_1** es un puntero a la primera componente del primer vector a sumar. Las componentes del vector están almacenadas una a continuación de otra.
- **vector_2** es un puntero a la primera componente del segundo vector a sumar. Las componentes del vector están almacenadas una ca continuación de la otra.
- **dimension** es la dimensión (número de elementos) de los vectores a sumar. **La dimensión de los vectores no es necesariamente múltiplo de 4.**
- **vector_suma** es un puntero a la posición de memoria a partir de la cual debe quedar almacenado el vector resultado de la suma.

Los punteros deben estar alineados en direcciones múltiplos de 16.
La función retorna 1 si puede realizar correctamente su trabajo y 0 si alguno de los puntero es nulo o no está alienado.
Aprovecha la capacidad de las instrucciones SSE de operar en paralelo sobre varios datos aunque, si la dimensión de los vectores no es un múltiplo de 4, algunos elementos tendrán que ser sumados mediante instrucciones SSE escalares.
La función podrá enlazarse con el código generado por un compilador de C para Linux de **64 bits.** (2 puntos).