Diseño Basado en Microprocesadores
Práctica 3
Programación en ensamblador x86 de 64 bits

Índice

---

1. Objetivos

En esta práctica se crearán funciones en ensamblador de 64 bits.

2. Flags del compilador, enlazador y ensamblador para generar código de 64 bits

Para que el compilador y el enlazador generen código nativo de 64 bits, suprimiremos los flags `-m32` que hemos venido usando en ambos hasta ahora.

Para que el ensamblador NASM genere un módulo objeto de 64 bits e información de depuración en formato *dwarf* usaremos los siguientes flags:

```
-f elf64 -g -F dwarf
```

3. Ejercicios

3.1. Ejercicio 1

Escribe una función en ensamblador de 64 bits que permita contar el número de veces que un determinado valor entero aparece en un array. El prototipo de la función es

```
int contar_valor_en_array_64(int valor,
                             int * array,
                             int longitud);
```

Por ejemplo

```
int v[10] = {1, 2, 3, 3, 3, 4, 5, 6, 7, 3};
ptintf("%d", contar_valor_en_array_64(3, v, 10));
```

imprimirá 4.

3.2. Ejercicio 2

Escribe un función en ensamblador de 64 bits que calcule la suma de los elementos de la diagonal de una matriz cuadrada de elementos de tipo `long`. Recuerda que en Linux el tipo `long` tiene 64 bits. El prototipo de la función es

```
int sumar_diagonal_64(long * matriz,
                      int num_filas_columnas,
                      long * resultado);
```

La función retorna 0 si hay algún error en los argumentos y 1 si todos son válidos. El resultado se devuelve mediante el puntero `resultado`.

3.3. Ejercicio 3

Escribe una función en ensamblador de 64 bits que sirva para ordenar de menor a mayor un array de datos de tipo `int`. El prototipo de la función es

```
int ordenar_64(int * array, int longitud);
```

La función retorna 0 si hay algún error en los argumentos y 1 si todos son válidos.

Puedes usar el algoritmo de ordenación que prefieras.