

## Administración y Organización de Computadores

Curso 2021-2022

### **Práctica: Procesamiento de imágenes en ensamblador**

El principal objetivo de esta práctica es completar una aplicación de procesamiento de imágenes, escrita en C++ sobre Linux, a través de la implementación en ensamblador de varias de las operaciones que debe proporcionar el programa. La integración de los dos lenguajes se llevará a cabo mediante las facilidades de ensamblado en línea proporcionadas por el compilador *gcc*. La programación en ensamblador se realizará considerando la arquitectura de un procesador Intel o compatible de 64 bits.

A continuación, se detallan diferentes aspectos a tener en cuenta para el desarrollo de esta práctica.

#### **Descripción de la aplicación**

El código que deberá ser completado estará incluido en una aplicación C++, disponible como proyecto de Qt. La siguiente figura muestra una captura de pantalla en un instante de ejecución de la aplicación:

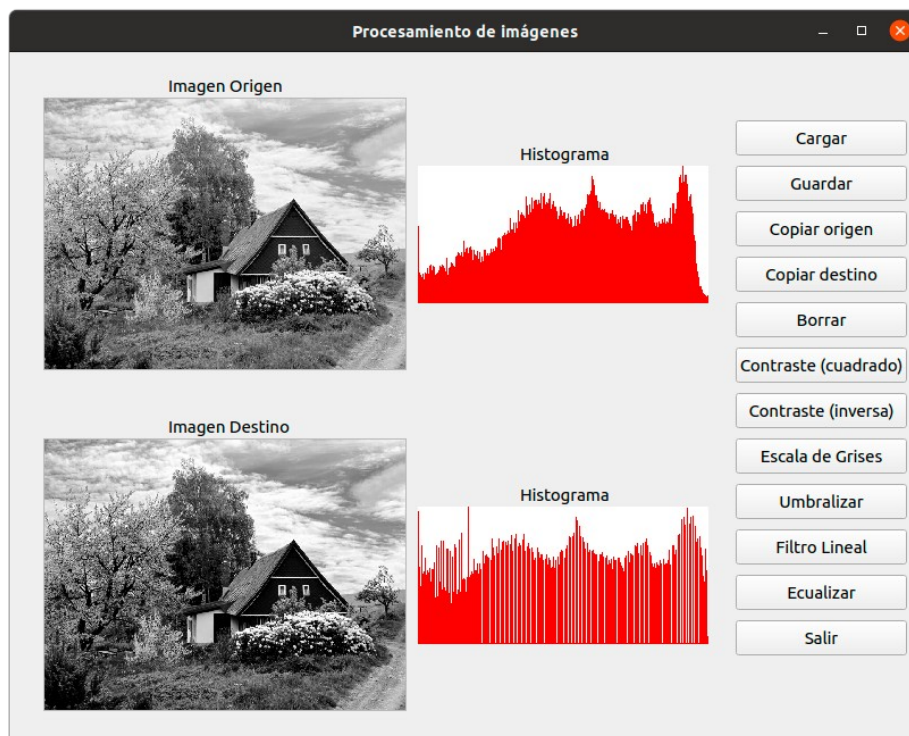


Figura 1: interfaz principal de la aplicación

## ¿Qué es Qt?

Qt es un marco de desarrollo de aplicaciones que, entre otras aportaciones, proporciona herramientas y librerías de clases para la creación de interfaces de usuario en entornos de escritorio.

## Puesta en marcha del proyecto

El fichero que contiene la descripción del proyecto (*pracaoc.pro*) se encuentra disponible en la carpeta principal de la aplicación. Dicho fichero puede ser utilizado para importar el proyecto desde diferentes entornos de desarrollo. No obstante, se recomienda trabajar con Qt Creator (paquete *qtcreator*). Es necesario instalar además los paquetes *qt5-default* y *qttools5-dev-tools*.

## Funcionamiento de la aplicación

La interfaz principal (figura 1) está compuesta por dos ventanas de imagen, dos histogramas y una serie de botones a través de los cuales es posible ejecutar las diferentes operaciones que proporciona la aplicación. Las dos ventanas de imagen se corresponden con la imagen origen (ventana superior) y la imagen destino (ventana inferior). La imagen origen es la imagen sobre la que se realizan la mayoría de las operaciones de procesamiento. El resultado de cada una de estas operaciones se almacena en la imagen destino. Las imágenes procesadas por el programa tienen una resolución de 320x240 (320 columnas y 240 filas) y cada píxel está representado por un nivel de gris (0-negro, 255-blanco). A la izquierda de cada ventana se muestra el histograma de la imagen correspondiente. El histograma de una imagen representa la frecuencia de aparición de cada nivel de gris en la imagen. Se trata de un vector de 256 entradas, una por cada nivel de gris. Cada elemento  $h[i]$  contiene el número de píxeles de imagen con nivel de gris  $i$ . Se describen a continuación las opciones incluidas en la aplicación:

- ✓ **Cargar:** carga, en la imagen origen, la imagen del fichero indicado. Si la imagen del fichero tiene una resolución superior a la soportada por el programa, ésta es recortada. Además, es posible cargar imágenes en color. En tal caso, esta opción se encarga de transformar la imagen a escala de grises.
- ✓ **Guardar:** guarda, en el fichero indicado, la imagen destino.
- ✓ **Copiar origen:** copia la imagen origen en la imagen destino.
- ✓ **Copiar destino:** copia la imagen destino en la imagen origen.
- ✓ **Borrar:** borra el contenido de la imagen destino poniendo a 0 cada uno de sus píxeles.
- ✓ **Contraste (cuadrado):** oscurece la imagen aplicando la función cuadrada. Si suponemos que *imgO* contiene la imagen origen e *imgD* la imagen destino, para un determinado píxel situado en la posición ( $f$ ,  $c$ ), el procedimiento realizaría el siguiente cálculo:  $imgD[f,c] = (imgO[f,c])^2/255$ .

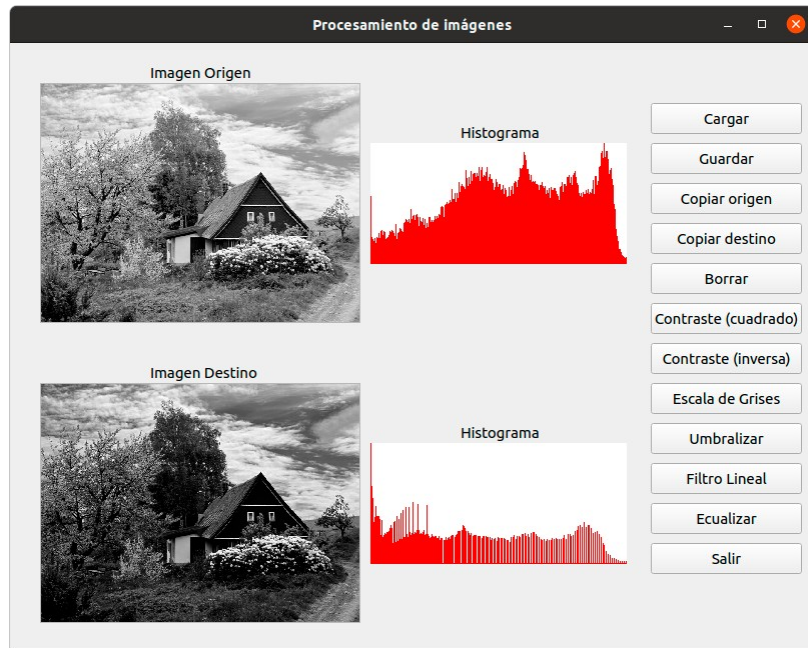


Figura 2: ejecución de la opción *Contraste (cuadrado)*.

- **Contraste (inversa):** el nivel de gris de cada píxel de la imagen origen es invertido ( $255 - \text{nivel de gris}$ ) y almacenado en la imagen destino:  $\text{imgD}[f,c] = 255 - \text{imgO}[f,c]$ .

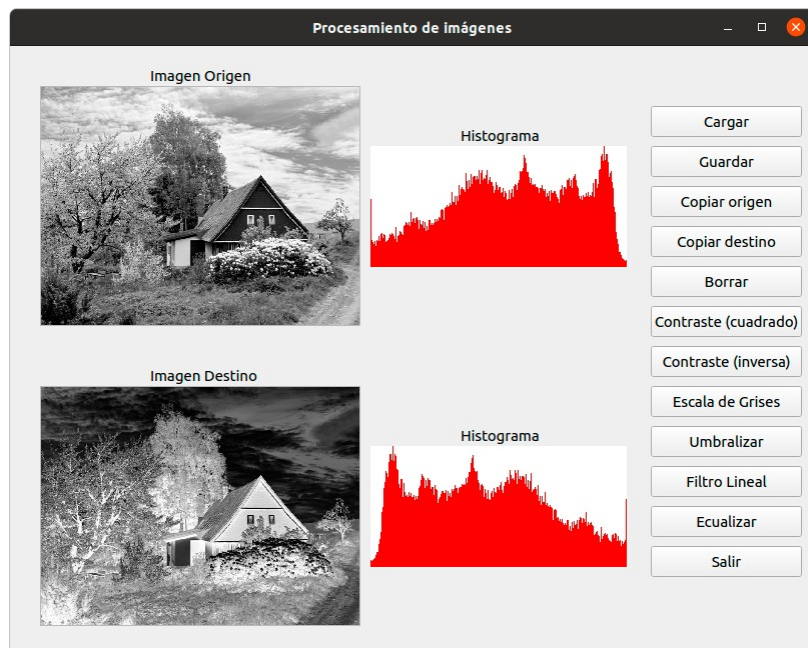


Figura 3: ejecución de la opción *Contraste (inversa)*.

- **Escala de grises:** modifica el mínimo y máximo nivel de gris de la imagen según los valores seleccionados a través del diálogo de la figura 4. Siendo  $mO$  y  $MO$  el mínimo y máximo nivel de gris de la imagen origen y  $mD$  y  $MD$  el mínimo y máximo nivel de gris seleccionado, cada píxel de la imagen origen sufriría la siguiente transformación:

$$\text{imgD}[f,c] = ((\text{imgO}[f,c] - mO) * (MD - mD)) / (MO - mO) + mD$$



Figura 4: diálogo de selección de escala de la opción *Escala de grises*.

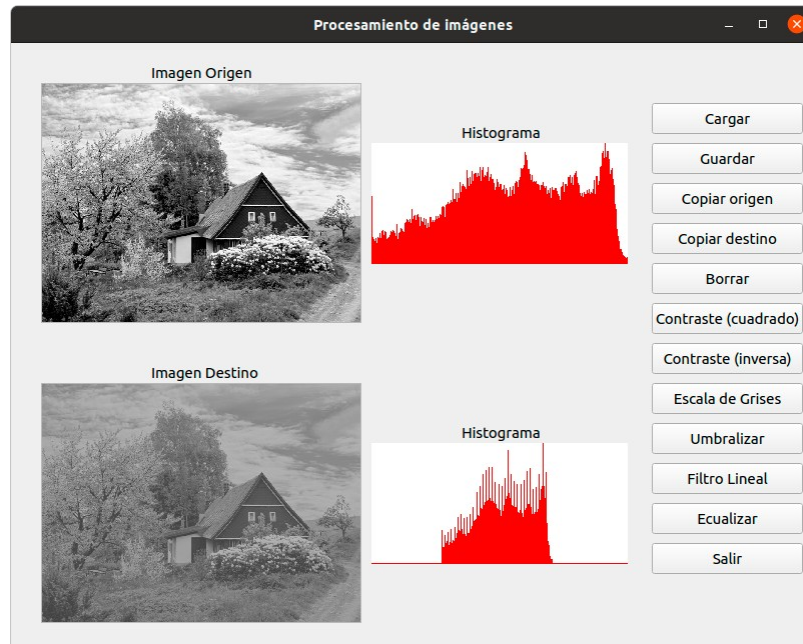


Figura 5: ejecución de la opción *Escala de grises*.

- **Umbralizar:** permite clasificar los píxeles de la imagen en 3 tipos distintos definidos por los rangos de niveles de gris establecidos por un umbral mínimo y un umbral máximo. Los píxeles cuyo nivel de gris superen el umbral máximo, tomarán el valor 255. Asimismo, aquellos píxeles con nivel de gris inferior al umbral mínimo, tomarán el valor 0. Los píxeles cuyo nivel de gris se encuentre entre los dos umbrales, cambiarán su valor a 127.

Cuando se activa esta opción, aparece un diálogo como el de la figura 6 a través del cual el usuario deberá seleccionar el valor de los dos umbrales. Una vez que pulse la opción “aceptar”, se llevará a cabo el proceso descrito sobre la imagen origen y se almacenará el resultado en la imagen destino.

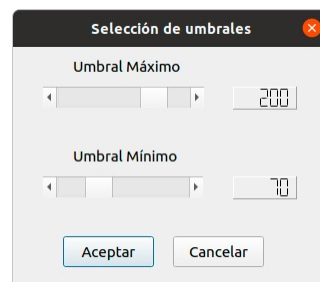


Figura 6: diálogo de selección de umbrales de la opción *Umbralizar*.

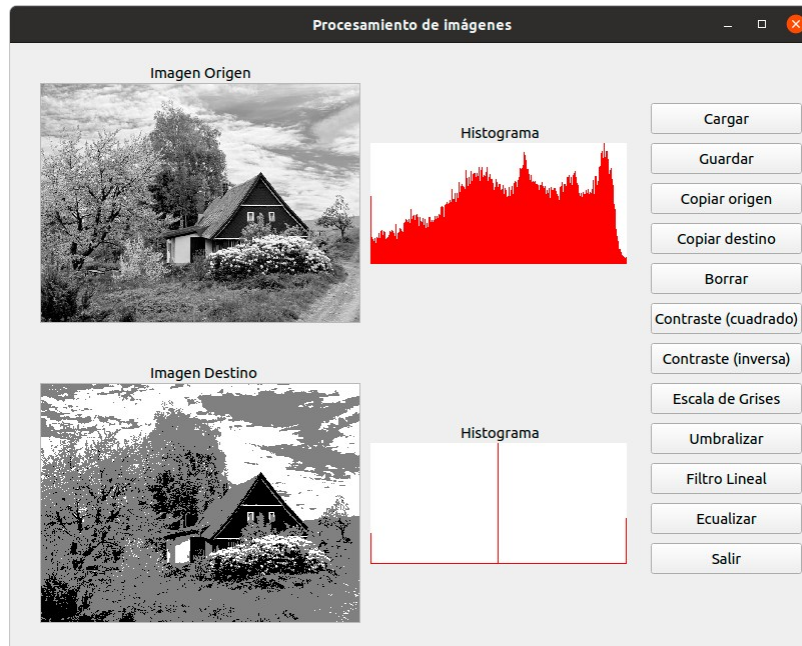


Figura 7: ejecución de la opción *Umbralizar*.

- **Filtro lineal:** el filtrado lineal de una imagen se realiza aplicando una operación de convolución entre la imagen y una matriz constante denominada máscara (*kernel*). Esta operación se utiliza, por ejemplo, como método de suavizado de imágenes y de detección de bordes. Para el caso de una máscara  $K$  de tamaño  $3 \times 3$ , cada píxel situado en la posición  $(f, c)$  de la imagen origen se obtendría como:

$$imgD[f, c] = \sum_{i=-1}^1 \sum_{j=-1}^1 imgO[f+i, c+j] * K[i+1, j+1]$$

Para mantener los valores resultantes de la operación anterior dentro de los rangos adecuados, es posible aplicar un factor de normalización ( $n$ ). Incluyendo esta normalización, la operación a realizar sería:

$$imgD[f, c] = \frac{\sum_{i=-1}^1 \sum_{j=-1}^1 imgO[f+i, c+j] * K[i+1, j+1]}{n}$$

En la aplicación que deberá ser completada en esta práctica, el usuario podrá indicar la máscara a utilizar así como el factor de normalización a través del diálogo de la figura 8. Como resultado, el programa deberá construir la imagen destino siguiendo la operación anterior.



Figura 8: diálogo de selección de máscara asociado a la opción *Filtro Lineal*.



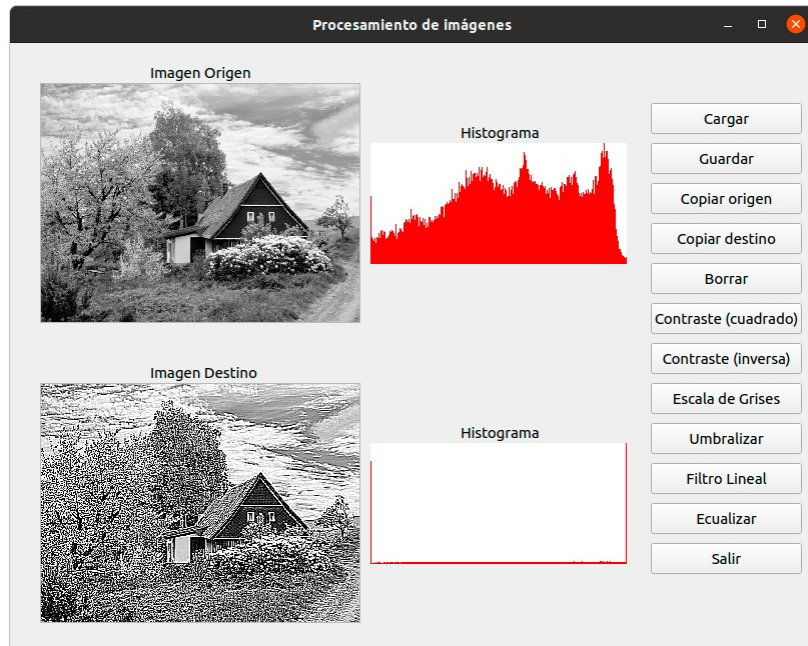


Figura 9: ejecución de la opción *Filtro lineal*.

- **Ecualizar:** la ecualización del histograma es una operación que trata de aproximar el histograma de una imagen a un histograma plano en el que cada nivel de gris tenga aproximadamente el mismo número de píxeles. Para llevar a cabo este proceso, se utiliza el histograma acumulado:

$$H[i] = \sum_{k=0}^i h[k]$$

A partir de  $H$  se calcula, para cada nivel de gris  $i$  de la imagen origen, qué nivel de gris  $i'$  le correspondería en la imagen destino:

$$i' = \frac{256}{W * H} H[i] - 1$$

siendo  $W$  y  $H$  el ancho y alto de la imagen.

Para no tener que realizar la operación anterior por cada píxel de la imagen, el proceso de ecualización se realiza en dos fases. En una primera fase, se genera una tabla conocida como tabla LUT (tabla de consulta - *Look Up Table*). La tabla LUT asocia un nivel de gris de la imagen origen con el correspondiente en la imagen destino:

$$LUT[i] = i' = \frac{256}{W * H} H[i] - 1$$

En la segunda fase, se genera la imagen destino utilizando la tabla anterior. Así, por cada píxel  $imgO[f,c]$  de la imagen origen, se obtiene el píxel correspondiente  $imgD[f,c]$  de la imagen destino accediendo a la entrada de la tabla LUT que corresponda:

$$\begin{aligned} i &= imgO[f,c] \\ imgD[f,c] &= LUT[i] \end{aligned}$$

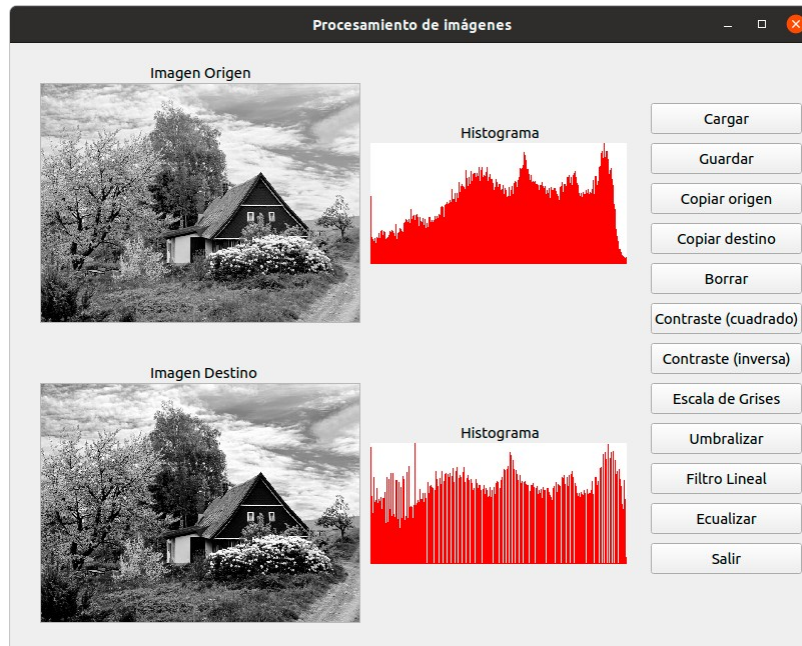


Figura 10: ejecución de la opción *Ecualizar*.

### Extensiones principales en x86-64

- En relación a la representación de datos en memoria, en 64 bits, los punteros y los datos de tipo *long* ocupan 64 bits. El resto de tipos mantiene el mismo tamaño que en la línea de procesadores de 32 bits (int: 4 bytes, short: 2 bytes, ...).
- Todas las instrucciones de 32 bits pueden utilizarse ahora con operandos de 64 bits. En ensamblador, el sufijo de instrucción para operandos de 64 bits es “q”.
- Los registros de 32 bits se extienden a 64. Para hacer referencia a ellos desde un programa en lenguaje ensamblador, hay que sustituir la letra inicial “e” por “r” (%rax, %rbx, ...). Los registros de 32 bits de la IA32 se corresponden con los 32 bits de menor peso de estos nuevos registros.
- El byte bajo de los registros %rsi, %rdi, %rsp y %rbp es accesible. Desde el lenguaje ensamblador, el acceso a estos registros se realiza a través del nombre del registro de 16 bits finalizado con la letra “l” (%sil, %dil, ...).
- Aparecen 8 nuevos registros de propósito general de 64 bits (%r8, %r9, ..., %r15). Es posible acceder a los 4, 2 o al último byte de estos registros incluyendo en su nombre el sufijo d, w o b (%r8d – 4 bytes, %r8w – 2 bytes, %r8b – 1 byte).

### Componentes de la aplicación

El código fuente de la aplicación está formado por 5 ficheros: *main.cpp*, *pracoc.cpp*, *pracoc.h*, *imageprocess.cpp* e *imageprocess.h*. Además, se incluyen cuatro formularios de QT (*mainForm.ui*, *escalaForm.ui*, *filtroForm.ui* y *umbralForm.ui*) y el fichero de descripción del proyecto (*pracaoc.pro*). El contenido de cada fichero fuente es el siguiente:

- ✓ *main.cpp*: contiene el procedimiento principal que permite lanzar la aplicación, así como crear y mostrar la ventana principal que actúa como interfaz entre el usuario y la aplicación.
- ✓ *pracaoc.h*: fichero que contiene la definición de la clase principal de la aplicación (*pracAOC*). Esta clase contiene los elementos principales de gestión de la aplicación. Entre los atributos, se encuentran las definiciones de las interfaces de usuario incluidas en el programa, así como de las variables que permiten almacenar la información de las imágenes origen y destino utilizadas en las opciones de procesamiento de la aplicación. Estas variables son *imgO* e *imgD*. Ambas están definidas como un array de tipo uchar (unsigned char) de 320x240 elementos. La imagen dentro de cada uno de estos arrays se

encuentra almacenada por filas. Esto implica que el acceso a un determinado píxel situado en una fila  $f$  y una columna  $c$  de imagen se realiza a través de la posición del array  $f*320+c$ .

- ✓ *pracaoc.cpp*: Incluye la implementación de los métodos de la clase *pracaoc*. En su mayoría, estos métodos se encargan de responder a los distintos eventos de las interfaces de usuario incluidas en el programa y de llamar a las funciones de procesamiento de imagen que correspondan en cada caso (disponibles en *imageprocess.cpp* e *imageprocess.h*).
- ✓ *imageprocess.h*: contiene la definición de las funciones implementadas en el fichero *imageprocess.cpp*.
- ✓ *imageprocess.cpp*: implementación de las funciones de procesamiento de imagen que se ejecutan a través de las distintas opciones de la aplicación. La mayoría de estas funciones contienen una implementación vacía. El objetivo de esta práctica es completarlas para que el funcionamiento de la aplicación sea el descrito anteriormente.

Dentro del fichero *imageprocess.cpp*, se ha incluido la implementación de la primera función (*copiar*) a modo de ejemplo. Como ya se ha comentado anteriormente, la implementación de las restantes es objeto de esta práctica. En algunos de los procedimientos, aparecen 2 bloques *asm* para poder incluir 2 implementaciones. La compilación de uno u otro bloque se realiza comentando o descomentando la sentencia “`#define WITHOUT_SSE`” que hay al principio del fichero. En el primer bloque se incluirá la implementación obligatoria que utilizará instrucciones de CPU realizando un procesamiento secuencial. En el segundo bloque, se incluirá una implementación opcional utilizando un procesamiento vectorizado mediante instrucción SSE. Para compilar el código con esta segunda implementación deberá comentarse la línea “`#define WITHOUT_SSE`”.

```
void imageprocess::copiar(uchar * imgO, uchar * imgD)
{
#ifdef WITHOUT_SSE
    asm volatile(
        "mov %0, %%rsi ;"
        "mov %1, %%rdi ;"
        "mov $320*240, %%rcx ;"
        "bcopia: ;"
        "mov (%%rsi), %%al ;"
        "mov %%al, (%%rdi) ;"
        "inc %%rsi ;"
        "inc %%rdi ;"
        "loop bcopia ;"

        :
        : "m" (imgO), "m" (imgD)
        : "%rax", "%rcx", "%rsi", "%rdi", "memory"

    );
#else
    asm volatile(
        "mov %0, %%rsi ;"
        "mov %1, %%rdi ;"
        "mov $320*240/16, %%rcx ;"
        "bcopia: ;"
        "movdqa (%%rsi), %%xmm0 ;"
        "movdqa %%xmm0, (%%rdi) ;"
        "add $16, %%rsi ;"
        "add $16, %%rdi ;"
        "loop bcopia ;"
    );
#endif
}
```



```

:
: "m" (imgO), "m" (imgD)
: "%rax", "%rcx", "%rsi", "%rdi", "%xmm0", "memory"
);

#endif

}

```

La función *copiar* es invocada en la ejecución de las opciones “Copiar Origen” y “Copiar Destino” para copiar la imagen origen en la destino y viceversa. Esta función incluye como parámetros un array que contiene la imagen a copiar (*imgO*) y un segundo array que especifica la imagen donde se debe realizar la copia (*imgD*). Ambos parámetros son tratados como operandos de entrada del bloque de código en ensamblador (la lista de operandos de salida está vacía). El motivo para que esto sea así es que en ningún caso se van a modificar estos parámetros, puesto que su contenido es la dirección de comienzo de los bloques de memoria donde se encuentran las dos imágenes. Lo que sí se va a modificar es el contenido del bloque de memoria apuntado por *imgD*, pero esto no afecta a la dirección almacenada en dicho parámetro.

Tras la definición de los operandos, se incluye la lista de registros utilizados dentro del código. Además de los registros indicados, dado que la memoria es modificada, la lista incluye también la palabra “memory”.

Una vez aclaradas estas definiciones, analicemos a continuación paso a paso el código ensamblador incluido en el primer bloque *asm* del procedimiento:

- ✓ Las dos primeras instrucciones se encargan de copiar las direcciones iniciales de memoria de las dos imágenes, indicadas por los dos operandos (%0=*imgO*, %1=*imgD*), en dos registros, *%rsi* y *%rdi*, para su posterior direccionamiento. Así, a través del registro *%rsi* podremos acceder a cada uno de los píxeles de *imgO* y, mediante el registro *%rdi*, tendremos acceso a los píxeles de *imgD*.
- ✓ La copia de cada píxel de *imgO* en la imagen *imgD* se realiza mediante un bucle con tantas iteraciones como indica el tamaño de las imágenes (320x240). Este bucle se lleva a cabo mediante la instrucción *loop*, por lo que, lo siguiente que hace el procedimiento es inicializar el registro *%rcx* con el total de iteraciones.
- ✓ Una vez inicializado *%rcx*, comienza el bucle de copia. Cada iteración consiste en la copia del píxel actual de *imgO* – (*%rsi*) – en el píxel correspondiente de *imgD* – (*%rdi*) –. Dicha copia se lleva a cabo a través del registro *al*, puesto que no es posible indicar los dos elementos de memoria como operandos de la instrucción *mov*. Tras esta operación y antes de pasar a la siguiente iteración del bucle, los índices *%rsi* y *%rdi* son incrementados para que apunten a los siguientes elementos de *imgO* e *imgD*.

El segundo bloque *asm* incluye una implementación equivalente a la anterior pero procesando simultáneamente 16 píxeles (16 bytes). Así, el número de iteraciones del bucle vectorizado es 320\*240/16 (valor inicial de *%rcx*) y los incrementos de *%rsi* y *%rdi* se realizan de 16 en 16. La transferencia entre grupos de píxeles de la imagen origen a la imagen destino se hace a través de la instrucción “*movdqa*” utilizando como registro intermedio *%xmm0*.

### **Especificación de los objetivos de la práctica**

El principal objetivo de esta práctica es completar el código de la aplicación descrita para que su funcionamiento sea el que se detalla en la sección “*Funcionamiento de la aplicación*”, incluida en esta documentación. Para ello, se deberán implementar, en lenguaje ensamblador, los procedimientos “vacíos” del módulo “*imageprocess.cpp*”. Estos procedimientos están asociados

con las opciones del programa. Para cada uno de ellos, se proporciona la estructura inicial del bloque ensamblador, en la que se ha incluido la definición de operandos que afectan a la implementación. La lista de registros utilizados incluye únicamente la palabra “memory”, ya que en todos los casos la memoria es modificada. La inclusión de registros dentro de esta lista dependerá de la implementación que se desarrolle en cada caso, por lo que, será necesario completarla para cada uno de los procedimientos.

Se describe a continuación la funcionalidad de cada uno de los procedimientos a completar. Para todos ellos, los parámetros *imgO* e *imgD* contienen, respectivamente, la dirección de los bloques de memoria que almacenan el contenido de las imágenes origen y destino. Para una descripción más detallada de estas funciones, se recomienda revisar la sección “*Funcionamiento de la aplicación*”.

- ✓ *void imageprocess::borrar(uchar \* imgD)*: borra todos los píxeles de *imgD*, asignándoles un valor 0 (negro) a cada uno de ellos. Es invocado cuando el usuario selecciona la opción “Borrar”.
- ✓ *void imageprocess::cambiarContrasteFCuadrada(uchar \* imgO, uchar \* imgD)*: modifica el contraste de cada píxel de *imgO* aplicando la función cuadrada. Devuelve el resultado en *imgD*. Es invocado cuando el usuario selecciona la opción “Contraste (cuadrado)”.
- ✓ *void imageprocess::cambiarContrasteFInversa(uchar \* imgO, uchar \* imgD)*: invierte el nivel de gris de cada píxel de *imgO* y devuelve el resultado en *imgD*. Es invocado cuando el usuario selecciona la opción “Contraste (inversa)”.
- ✓ *void imageprocess::cambiarEscalaGris(uchar \* imgO, uchar \* imgD, uchar minO, uchar maxO, uchar minD, uchar maxD)*: modifica la escala de grises de *imgO* a partir de los valores indicados en *minO*, *maxO*, *minD* y *maxD*. Devuelve el resultado en *imgD*. Es invocado cuando el usuario selecciona la opción “Escala de Grises”.
- ✓ *void imageprocess::umbralizar(uchar \* imgO, uchar \* imgD, uchar uMax, uchar uMin)*: aplica los umbrales máximo y mínimo, indicados en *uMax* y *uMin*, a la imagen almacenada en *imgO* y devuelve el resultado en *imgD*. Este procedimiento es invocado al seleccionar la opción “Umbralizar”.
- ✓ *void imageprocess::filtroLineal(uchar \* imgO, int \* kernel, int norm, uchar \* imgD)*: aplica un filtro lineal a partir de la máscara indicada en el parámetro *kernel* a la imagen almacenada en *imgO*. Devuelve el resultado de la operación en *imgD*. Este procedimiento es invocado cuando el usuario ejecuta la opción “Filtro Lineal”.
- ✓ *void imageprocess::ecualizarHistograma(int \* histoOrig, uchar \* tablaLUT)*: genera una tabla LUT ecualizando el histograma almacenado en *histoOrig*. Devuelve el resultado en *tablaLUT*. Junto con el siguiente procedimiento, este procedimiento es invocado cuando el usuario ejecuta la opción “Ecualizar”.
- ✓ *void imageprocess::aplicarTablaLUT(uchar \* imgO, uchar \* tablaLUT, uchar \* imgD)*: genera una imagen, que devuelve en *imgD*, aplicando la tabla LUT almacenada en el parámetro *tablaLUT* a la imagen indicada en *imgO*. Es invocado cuando el usuario ejecuta la opción “Ecualizar” para completar el ecualizado de la imagen tras haber obtenido la tabla LUT correspondiente por el procedimiento anterior.

Es obligatorio implementar el primer bloque *asm* de todos los procedimientos. La nota máxima de la parte obligatoria será de Notable(8). La implementación mediante SSE del segundo bloque *asm* de los procedimientos *borrar*, *cambiarContrasteFCuadrada*, *cambiarContrasteFInversa* y *cambiarEscalaGris* será opcional. Esta implementación adicional podrá suponer hasta 2 puntos más en la nota de la práctica.

Nota: la práctica se realizará de manera individual.

Fecha de entrega de la práctica: 11 de enero de 2022.

Entrega: la entrega se realizará a través de la subida al aula virtual de la asignatura de un archivo .zip que contenga el proyecto completo.