Se incluye a continuación un pseudocódigo por cada uno de los procedimientos que se deben implementar en esta práctica. Para aquellos datos usados que no están definidos como operandos se deja libertad de utilizar registros o variables locales, aunque en algunos casos sea obligatorio el uso de registros (cuando los datos se usen para direccionar la memoria). En cualquier caso, se recuerda que en la arquitectura de 64 bits hay 8 nuevos registros de propósito general que pueden utilizarse como almacenes de datos auxiliares, evitando el uso de variables locales.

borrar(imgD)

```
{
    dirDest = imgD;
    Para(p=0; p<320*240; p++){
        [dirDest] = 0;
        dirDest++;
    }
}
```

cambiarContrasteFCuadrada(imgO, imgD)

```
{
    dirOrig = imgO;
    dirDest = imgD;
    Para(p=0; p<320*240; p++){
        g = [dirOrig];
        [dirDest] = (g*g)/255;
        dirOrig++;
        dirDest++;
    }
}
```

cambiarContrasteFInversa(imgO, imgD)

```
{
    dirOrig = imgO;
    dirDest = imgD;
    Para(p=0; p<320*240; p++){
        g = [dirOrig];
        [dirDest] = 255 - g;
        dirOrig++;
        dirDest++;
    }
}
```

cambiarEscalaGrises(imgO, imgD, minO, maxO, minD, maxD)

```
{
    dirOrig = imgO;
    dirDest = imgD;
    rangoO = maxO - minO;
    rangoD = maxD - minD;
    Para(p=0; p<320*240; p++){
        g = [dirOrig];
        [dirDest] = ((g - minO)*rangoD)/rangoO + minD;
        dirOrig++;
        dirDest++;
    }
}
```

```
umbralizar(imgO, imgD, uMax, uMin)
{
    dirOrig = imgO;
    dirDest = imgD;
    Para(p=0; p<320*240; p++)
    {
        Si([dirOrig]>uMax)
            [dirDest]=255;
        Sino
        {
            Si([dirOrig]<uMin)
                [dirDest]=0;
            Sino
                [dirDest]=127;
        }
        dirDest++;
        dirOrig++;
    }
}
```

```
filtroLineal(imgO, kernel, norm, imgD)
{
   dirOrig = imgO;
   dirDest = imgD;

   Para(f=0; f<240; f++)
   {
      Para(c=0; c<320; c++)
      {
         dirKernel = kernel;
         acum = 0;
         Para(kf=-1; kf<=1; kf++)
            Para(kc=-1; kc<=1; kc++)
            {
               fp = f+kf;
               cp = c+kc;
               if(cp>=0 y cp<320 y fp>=0 y fp<240)
               {
                  offPixel = fp*320 + cp;
                  acum = acum + [dirOrig + offPixel]*[dirKernel]
               }
               dirKernel++;
            }

         acum = acum / norm;
         Si(acum<0)
            acum = 0;
         Si(acum>255)
            acum = 255;
         [dirDest] = acum;
         dirOrig++;
         dirDest++;

      }
   }
}
```

```
ecualizarHistograma(histoOrig, tablaLUT)
{
   dirHOrig = histoOrg;
   dirHAcum = acumHisto;

   acum = 0;
   Para(n=0; n<256; n++)
   {
      acum = acum + [dirHOrig + n*4];
      [dirHAcum + n*4] = acum
   }

   dirLUT = tablaLUT;
   Para(n=0; n<256; n++)
   {
      g = (256*[dirHAcum + n*4])/(320*240);
      Si(g>0)
         g=g-1;
      [dirLUT + n] = g;
   }

}


aplicarTablaLUT(imgO, tablaLUT, imgD)
{
   dirOrig = imgO;
   dirLUT = tablaLUT;
   dirDest = imgD;
   Para(p=0; p<320*240; p++)
   {
      gOrig = [dirOrig];
      gDest = [dirLUT + gOrig];
      [dirDest] = gDest;
      dirOrig = dirOrig++;
      dirDest = dirDest++;
   }
}
```

Implementación con SSE

- **Función *borrar***: utilizando SSE, es posible hacer la puesta a 0 de 16 píxels en cada una de las iteraciones. El número de iteraciones del bucle vectorial se reduce por lo tanto a 320*240/16.
- **Función *cambiarContrasteFCuadrada***: la aplicación de esta transformación de píxel requiere hacer una operación de división que solo está disponible en el subconjunto de instrucciones de SSE dedicado a datos empaquetados reales. Esto implica que cada píxel tendrá que convertirse a tipo *float* y que, por lo tanto, solo podrán procesarse 4 píxels simultáneamente. La conversión a *float* de cada píxel supone en primer lugar la conversión de byte a entero de 32 bits. Las operaciones de desempaquetado (*punpck*...) se pueden usar para este fin. Tras procesar los píxels, será necesario realizar la conversión inversa (de float a entero de 32 bits y de entero de 32 bits a byte) antes de almacenar el resultado en la imagen destino. Además de estas consideraciones, para realizar la división entre 255, será necesario replicar dicho valor en formato *float* en todos los elementos de un registro SSE.
- **Función *cambiarContrasteFInversa***: la operación puede aplicarse simultáneamente a 16 píxels en cada iteración. Es necesario replicar para ello el valor 255 en todos los bytes de un registro SSE antes de comenzar el bucle vectorial.
- **Función *cambiarEscalaGrises***: al igual que ocurre en la función *cambiarContrasteFCuadrada*, las operaciones de división obligan a utilizar formato de datos de tipo *float*, por lo que solo podrán procesarse 4 píxels en cada iteración. Además de realizar las conversiones correspondientes, los valores asociados con los parámetros *minO*, *maxO*, *minD* y *maxD* deberán replicarse en registros SSE.