**Trabajo Práctico N°2**

*Procesamiento de Imágenes:*

*Etapa 2: Superponer*

**Materia:**

* Organización del computador II (OC2)

**Docente:**

* Carlos Holtman (choltman@ungs.edu.ar)

**Alumnos:**

* Gonzalo Pesado (gonzapesa@gmail.com)
  + LEG: 38990987 / 2014
* Lucas Vargas ([lucasjv92@gmail.com](mailto:lucasjv92@gmail.com))
  + LEG: 37369589 / 2014

**Año 2017**

**Desarrollo del proyecto**

Como segunda etapa del Trabajo Práctico, nos focalizamos en la creación de un programa capaz de Superponer 2 imágenes indicando qué valor RGB usaremos como color transparente sobre la primer imagen, es decir nuestra **figura**, para luego superponerla sobre la segunda imagen, es decir nuestro **fondo**, e indicando un valor de tolerancia de transparencia para finalmente guardarlo en un archivo nuevo.

El software está desarrollado en 2 lenguajes de programación: Assembler de Intel IA32 (Lenguaje de código máquina) y C. El entorno de desarrollo y ejecución se realizó bajo sistema operativo Windows de 32 bits.

A su vez, el software debe ejecutarse colocando en orden los siguientes argumentos: el nombre de archivo de la figura, el nombre de archivo del fondo, el alto de la imagen, el ancho, el valor de tolerancia, el color de transparencia (3 valores RGB de 0 a 255) y el nombre del archivo donde se guardará la nueva imagen.

El programa a su vez también cuenta con un .bat el cual compila el código fuente (“Superponer.c”) y sus respectivos headers, junto con el OBJ que crea NASM del archivo que posee todas las funciones de Assembler “funcionesASM\_TP\_E2.asm”). Este .bat mostrará al usuario las rutas de NASM y GCC que utilizará para compilar. En caso de que no sean las rutas correctas, se deberá cambiar la ruta directamente del .bat a la ruta correcta.

Por último, el .bat hace uso del programa **GraphicsMagick** para que una vez creado el archivo .rgb nuevo, el mismo se visualice por pantalla.

También contamos con batchs para:

* ● **convertirImagenes.bat:** Convertir 2 imágenes que queramos nosotros a formato .rgb.
* ● **visualizar.bat:** Visualizar un archivo con extensión .rgb que indiquemos nosotros
* **Superponer.bat:** Llamar a nuestro programa realizado para superponer 2 imágenes, indicando todos los parámetros necesarios para ejecutar dicha superposición.

Como herramientas para desarrollar el mismo se usaron los siguientes:

* Editores: Sublime, Notepad++, Notepad, Wordpad.
* IDE: SASM para testear código de assembler, y Codeblocks para probar código de C.
* Compiladores: NASM para compilar los códigos .asm, y GCC para compilar el código de C y llamar a los objetos de código creados anteriormente por NASM para vincular las funciones de Assembler llamadas desde C.
* GraphicsMagick para exportar las imágenes a un archivo .rgb (vector de bytes) y visualizar las imágenes con ese formato.

Con respecto al corazón de las funciones de Assembler de Intel IA32, el programa se basa en los siguientes pasos pasos los cuales son:

**Gen\_mascara:**

1) **cargarPatrones**: En él se cargaran todas las máscaras a los registros para que las operaciones entre vectores sean mucho más rápidas, se asume que llegan a entrar vectores de 4 colores RGB, por lo que estas máscaras están acordes a los mismos. Dentro de esta también se realiza el carga p y cargar1menosP para tener dichos valores en un vector de 4 posiciones, para ya directamente realizar los cálculos en el mismo registro.

2) **ciclar**:

2.A) **cargarCanalR/G/B:** Carga las 4 primeras posiciones de los punteros de los canales de la primera imagen a un registro XMM (con respecto al puntero ecx utilizado como contador).

2.B) **convertirINT\_PF**: Realiza la conversión a PF de los vectores de los 3 canales de la imagen 1 para poder operar con mayor precisión y realizar los cálculos de la superposición que fueron explicados en las consignas del TP.

2.C) **operarVectores**: Mediante operaciones entre los registros cargados previamente, se calcula *d* = √(*r*1 − *r* )2 + (*g*1 − *g* )2 + (*b*1 − *b* )2 . para determinar si un pixel es transparente.

2.D) **recuperarResultadoComparación**: Realiza la conversión del vector en PF al vector de formato entero.

2.E) **copiarResultado**: Escribe en el puntero de la máscara el vector calculado previamente.

2.F) **operarConLosRestantes**: Fija el puntero a los próximos 4 valores de los vectores de los canales RGB, si todavía no se llegó al tamaño total pasado por parámetro, seguirá realizando el ciclo con las próximas 4 posiciones de los vectores de los canales RGB.

**Aplica\_mascara:**

1) **ciclo**:

1.A) **cargarCanal1/2/Mascara:** Carga las 4 primeras posiciones de los punteros de un canal de la primer y segunda imagen a un registro XMM (con respecto al puntero ecx utilizado como contador). Lo mismo sucede con el puntero de la máscara.

1.B) **determinarTranspolacion**: Usando funciones lógicas entre los registros de los canales y la máscara saca el fondo de la figura, la figura del fondo y luego junta a ambos.

1.C) **copiarResultadoTranspolacion**: Copia el resultado de la operación anterior a 4 posiciones del vector resultado (con respecto al puntero ecx como contador).

1.D) **evaluarRestantes**: Fija el puntero a los próximos 4 valores de los vectores de los canales RGB, si todavía no se llegó al tamaño total pasado por parámetro, seguirá realizando el ciclo con las próximas 4 posiciones de los vectores de los canales RGB.

**Código Fuente: C**

**Superponer.c**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include "Funciones.h"
5. **void** gen\_mascara(unsigned **char** \*r, unsigned **char** \*g, unsigned **char** \*b, **int** cantidad, **int** rt, **int** gt, **int** bt, **float** t, unsigned **char** \*mascara);
6. **void** aplica\_mascara(unsigned **char** \*mascara, unsigned **char** \*canal1, unsigned **char** \*canal2, **int** cantidad, unsigned **char** \*resultado);
8. **int** main(**int** argc, **char** \* argv[])
9. {
10. **if**(argc != 10)
11. {
12. printf("Los argumentos del programa deben ser: \n");
13. printf("imagen1.rgb imagen2.rgb filas(entero) columnas(entero) rojo(0-255) verde(0-255) azul(0-255) distancia(Decimal) resultado.rgb");
14. **return** 0;
15. }
16. ///argv[0] Ejecutable (Superponer)
17. ///argv[1] Imagen1 rgb Figura
18. ///argv[2] Imagen2 rgb Fondo
19. ///argv[3] filas
20. ///argv[4] columnas
21. ///argv[5] Valor Rojo Transparente
22. ///argv[6] Valor Verde Transparente
23. ///argv[7] Valor Azul Transparente
24. ///argv[8] Distancia de color del pixel para ser considerado transparente
25. ///argv[9] ImagenResultado rgb
27. **char** \*nombreImagen1 = argv[1];
28. **char** \*nombreImagen2 = argv[2];
29. **int** filas = atoi(argv[3]);
30. **int** columnas = atoi(argv[4]);
31. **char** \*rt = argv[5];
32. **char** \*gt = argv[6];
33. **char** \*bt = argv[7];
34. **int** valorR = atoi(rt);
35. **int** valorG = atoi(gt);
36. **int** valorB = atoi(bt);
37. **float** t = atof(argv[8]);
38. **char** \*nombreImagenResultado = argv[9];
40. unsigned **char** \*img1;
41. unsigned **char** \*img2;
42. unsigned **char** \*resultado;
44. unsigned **char** \*r1;
45. unsigned **char** \*g1;
46. unsigned **char** \*b1;
48. unsigned **char** \*r2;
49. unsigned **char** \*g2;
50. unsigned **char** \*b2;
52. unsigned **char** \*resultadoRojo;
53. unsigned **char** \*resultadoVerde;
54. unsigned **char** \*resultadoAzul;
56. unsigned **char** \*mascara;
58. **int** cantidadPixelPorPunto = 3;
59. **int** longitud = filas \* columnas \* cantidadPixelPorPunto;
60. **int** cantPixeles = filas \* columnas;
61. **int** cantidadBytesMascara = cantBytesMascara(cantPixeles);
63. img1 = reservarMemoria(longitud);
64. img2 = reservarMemoria(longitud);
65. resultado = reservarMemoria(longitud);
67. r1 = reservarMemoria(cantPixeles);
68. g1 = reservarMemoria(cantPixeles);
69. b1 = reservarMemoria(cantPixeles);
71. r2 = reservarMemoria(cantPixeles);
72. g2 = reservarMemoria(cantPixeles);
73. b2 = reservarMemoria(cantPixeles);
75. resultadoRojo = reservarMemoria(cantPixeles);
76. resultadoVerde = reservarMemoria(cantPixeles);
77. resultadoAzul = reservarMemoria(cantPixeles);
79. mascara = reservarMemoria(cantidadBytesMascara);
81. leer\_rgb(nombreImagen1,img1,filas,columnas);
82. leer\_rgb(nombreImagen2,img2,filas,columnas);
84. separar\_rgb(img1,longitud,r1,g1,b1);
85. separar\_rgb(img2,longitud,r2,g2,b2);
87. gen\_mascara(r1,g1,b1,cantPixeles,valorR,valorG,valorB,t,mascara);
89. aplica\_mascara(mascara, r1, r2, cantidadBytesMascara, resultadoRojo);
90. aplica\_mascara(mascara, g1, g2, cantidadBytesMascara, resultadoVerde);
91. aplica\_mascara(mascara, b1, b2, cantidadBytesMascara, resultadoAzul);
93. combinar\_rgb(resultadoRojo,resultadoVerde,resultadoAzul,cantPixeles,resultado);
95. escribir\_rgb(nombreImagenResultado,resultado,filas,columnas);
97. ///Liberamos toda la memoria
98. free(img1);
99. free(img2);
100. free(resultado);
102. free(r1);
103. free(g1);
104. free(b1);
106. free(r2);
107. free(g2);
108. free(b2);
110. free(resultadoRojo);
111. free(resultadoVerde);
112. free(resultadoAzul);
114. free(mascara);
116. **return** 0;
117. }

**Funciones.h**

1. #ifndef FUNCIONES\_H\_INCLUDED
2. #define FUNCIONES\_H\_INCLUDED
3. unsigned **char** \*reservarMemoria(**int** cantidadBytes)
4. {
5. unsigned **char** \*memoria = malloc(cantidadBytes);
6. **if**(memoria == NULL)
7. {
8. printf("Error al reservarMemoria");
9. exit(0);
10. **return** NULL;
11. }
12. **return** memoria;
13. }
15. **void** leer\_rgb(**char** \*archivo, unsigned **char** \*buffer, **int** filas, **int** columnas)
16. {
17. **FILE** \*fp;
18. fp = fopen(archivo, "rb");
19. fread (buffer, filas \* columnas \* 3, 1, fp);
20. fclose (fp);
21. }
23. **void** escribir\_rgb(**char** \*archivo, unsigned **char** \*buffer, **int** filas, **int** columnas)
24. {
25. **FILE** \*fpNuevo;
26. fpNuevo = fopen(archivo,"wb");
27. fwrite(buffer,filas\*columnas\*3,1,fpNuevo);
28. fclose(fpNuevo);
29. }
30. **void** escribir\_mascara(**char** \*archivo, unsigned **char** \*buffer, **int** filas, **int** columnas)
31. {
32. **FILE** \*fpNuevo;
33. fpNuevo = fopen(archivo,"wb");
34. fwrite(buffer,filas\*columnas,1,fpNuevo);
35. fclose(fpNuevo);
36. }
37. **void** separar\_rgb(unsigned **char** \*rgb, **int** cantidad, unsigned **char** \*r, unsigned **char** \*g, unsigned **char** \*b)
38. {
39. **int** i;
40. **int** cont = 0;
41. **for**(i = 0; i < cantidad; i+=3)
42. {
43. r[cont] = rgb[i];
44. g[cont] = rgb[i+1];
45. b[cont] = rgb[i+2];
46. cont++;
47. }
48. }
49. **int** cantBytesMascara(**int** cantidadPixeles)
50. {
51. **if**(cantidadPixeles % 16 != 0)
52. {
53. **int** cantidadNueva = 1;
54. **while**(cantidadNueva >= cantidadPixeles)
55. {
56. cantidadNueva \*= 16;
57. }
58. cantidadPixeles = cantidadNueva;
59. }
60. **return** cantidadPixeles;
61. }
62. **void** combinar\_rgb(unsigned **char** \*r, unsigned **char** \*g, unsigned **char** \*b, **int** cantidad, unsigned **char** \*rgb)
63. {
64. **int** i;
65. **int** cont = 0;
66. **for**(i = 0; i < cantidad; i++)
67. {
68. rgb[cont] = r[i];
69. rgb[cont+1] = g[i];
70. rgb[cont+2] = b[i];
71. cont+=3;
72. }
73. }
75. **void** mostrarRGB(unsigned **char** \*vector, **int** cantidad)
76. {
77. printf("Mostrando vector\n");
78. **int** i;
79. **for**(i = 0; i < cantidad; i++)
80. {
81. printf("%d",vector[i]);
82. }
83. printf("\n");
84. }
85. #endif // FUNCIONES\_H\_INCLUDED

**Código Fuente: Assembler**

global \_gen\_mascara  
global \_aplica\_mascara  
  
section .data   
align 16 ;PARA ALINEAR LAS MASCARAS  
PATRON\_INT\_PF db 3,255,255,255,2,255,255,255,1,255,255,255,0,255,255,255  
PATRON\_PF\_INT db 12,8,4,0,255,255,255,255,255,255,255,255,255,255,255,255  
PATRON\_CARGA\_TRANSP db 0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3  
  
section .text  
  
\_gen\_mascara:  
**push** ebp  
**mov** ebp,esp  
;EBP + 0 EBP ACTUAL  
;EBP + 4 DIR RETORNO  
;EBP + 8 PUNTERO R  
;EBP + 12 PUNTERO G  
;EBP + 16 PUNTERO B  
;EBP + 20 INT CANTIDAD  
;EBP + 24 INT RT  
;EBP + 28 INT GT  
;EBP + 32 INT BT  
;EBP + 36 FLOAT T  
;EBP + 40 PUNTERO MASCARA  
  
;XMM0 PUNTERO R  
;XMM1 PUNTERO G  
;XMM2 PUNTERO B  
;XMM3 R TRANSP  
;XMM4 G TRANSP  
;XMM5 B TRANSP  
;XMM6 MASCARA PASA PUNTEROS  
;XMM7 MASCARA CONVIERTE TRANSPARENTES -> MASCARA PF - VECTOR  
  
cargarT:  
**mov** ebx, [EBP + 36] ;EBX = VALOR DE T  
  
cargarTransparente:  
movdqu xmm7,[PATRON\_CARGA\_TRANSP] ;MASCARA CARGA TRANSPARENTES  
cargaR:  
**movd** xmm3, [EBP + 24] ;PASO RT A REGISTRO  
pshufb xmm3,xmm7 ;ACOMODO RT  
cargaG:  
**movd** xmm4, [EBP + 28] ;PASO BT A REGISTRO  
pshufb xmm4,xmm7 ;ACOMODO BT  
cargaB:  
**movd** xmm5, [EBP + 32] ;PASO GT A REGISTRO  
pshufb xmm5,xmm7 ;ACOMODO GT

cargarPatrones:  
movdqu xmm6,[PATRON\_INT\_PF] ;MASCARA DE VECTOR - PF  
movdqu xmm7,[PATRON\_PF\_INT] ;MASCARA DE PF - VECTOR  
  
convertirPF\_TRANSP\_RGB:  
;CONVIERTO VECTOR DE Rt EN PF  
cvtdq2ps xmm3,xmm3   
;CONVIERTO VECTOR DE Gt EN PF  
cvtdq2ps xmm4,xmm4   
;CONVIERTO VECTOR DE Bt EN PF  
cvtdq2ps xmm5,xmm5   
  
cargarContadores:  
**mov** ecx, 0 ;ECX = CONTADOR, LO PONGO EN 0  
**mov** edx, [EBP + 20] ;EDX = CANTIDAD TOTAL  
  
ciclar:  
  
cargarCanalR:  
**mov** eax,[EBP + 8] ;MUEVO PUNTERO DE R  
**movd** xmm0,**dword** [eax + ecx] ;COPIO VECTOR DE R  
pshufb xmm0, xmm6 ;ACOMODO VECTOR R   
cargarCanalG:  
**mov** eax,[EBP + 12] ;MUEVO PUNTERO DE G  
**movd** xmm1,**dword** [eax + ecx] ;COPIO VECTOR DE G  
pshufb xmm1, xmm6 ;ACOMODO VECTOR G   
cargarCanalB:  
**mov** eax,[EBP + 16] ;MUEVO PUNTERO DE B  
**movd** xmm2,**dword** [eax + ecx] ;COPIO VECTOR DE B  
pshufb xmm2, xmm6 ;ACOMODO VECTOR B  
  
convertirINT\_PF:  
;CONVIERTO VECTOR DE R EN PF  
cvtdq2ps xmm0,xmm0   
;CONVIERTO VECTOR DE G EN PF  
cvtdq2ps xmm1,xmm1   
;CONVIERTO VECTOR DE B EN PF  
cvtdq2ps xmm2,xmm2   
  
operarVectores:  
**subps** xmm0 , xmm3 ;(r1 - r)   
**subps** xmm1 , xmm4 ;(g1 - g)   
**subps** xmm2 , xmm5 ;(b1 - b)  
**mulps** xmm0 , xmm0 ;(r1 - r) \* (r1 - r)  
**mulps** xmm1 , xmm1 ;(g1 - g) \* (g1 - g)  
**mulps** xmm2 , xmm2 ;(b1 - b) \* (b1 - b)  
**addps** xmm0 , xmm1 ;(r1 - r)2 + (g1 - g)2  
**addps** xmm0 , xmm3 ;(r1 - r)2 + (g1 - g)2 + (b1 + b)2  
**sqrtps** xmm0 , xmm0 ;v[ (r1 - r)2 + (g1 - g )2 + (b1 -b)2 ]

cargarT\_Vector:  
movdqu xmm2,[PATRON\_CARGA\_TRANSP] ;MASCARA CARGA TRANSPARENTES  
**movd** xmm1, EBX ;PASO T A REGISTRO  
pshufb xmm1, xmm2 ;ACOMODO T  
  
determinarTransparencia:  
**CMPPS** xmm0, xmm1, 2 ;comparo por <= con el valor de t,   
 ;tengo 0's en las que no son trans  
recuperarResultadoComparacion:  
pshufb xmm0, xmm7 ;PF A INT  
  
copiarResultado:  
**mov** eax, [EBP + 40] ;PUNTERO DE MASCARA  
**movd** [eax + ecx],xmm0 ;COPIO EL RESULTADO AL VECTOR  
 ;EN LA POSICION INDICADA  
operarConLosRestantes:  
**add** ecx,4 ;ACCEDO A LOS PROXIMOS 4 VECTORES  
**cmp** ecx, edx  
**jl** ciclar  
  
**mov** esp,ebp  
**pop** ebp  
**ret**

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\_aplica\_mascara:  
**push** ebp  
**mov** ebp,esp  
  
;EBP + 0 EBP ACTUAL  
;EBP + 4 DIR RETORNO  
;EBP + 8 PUNTERO MASCARA  
;EBP + 12 PUNTERO CANAL 1  
;EBP + 16 PUNTERO CANAL 2  
;EBP + 20 INT CANTIDAD  
;EBP + 24 PUNTERO RESULTADO  
  
;ASUMO EL CANAL 1 COMO FIGURA, UTILIZO DE A 16 BITS A LA VEZ  
  
;XMM0 = CANAL 1  
;XMM1 = CANAL 2  
;XMM2 = MASCARA / RESULTADO  
  
cargar\_Contadores:  
**mov** ecx, 0 ;ECX = CONTADOR, LO PONGO EN 0  
**mov** edx, [EBP + 20] ;EDX = CANTIDAD TOTAL  
  
ciclo:  
  
cargarCanal1:  
**mov** eax,[EBP + 12] ;MUEVO PUNTERO DE CANAL1  
movdqu xmm0,[eax + ecx] ;COPIO VECTOR DE CANAL1  
cargarCanal2:  
**mov** eax,[EBP + 16] ;MUEVO PUNTERO DE CANAL2  
movdqu xmm1,[eax + ecx] ;COPIO VECTOR DE CANAL2  
cargarMASCARA:  
**mov** eax,[EBP + 8] ;MUEVO PUNTERO DE MASCARA  
movdqu xmm2,[eax + ecx] ;COPIO VECTOR DE MASCARA  
  
determinarTraspolacion:  
**PAND** xmm1,xmm2 ;SACO LA FIGURA DEL FONDO  
**PANDN** xmm2,xmm0 ;SACO EL FONDO DE LA FIGURA  
**POR** xmm2,xmm1 ;JUNTO EL FONDO Y LA FIGURA  
  
copiarResultadoTraspolacion:  
**mov** eax, [EBP + 24] ;PUNTERO DE RESULTADO  
movdqu [eax + ecx],xmm2 ;COPIO EL RESULTADO AL VECTOR  
 ;EN LA POSICION INDICADA  
evaluarRestantes:  
**add** ecx,16 ;ACCEDO A LOS PROXIMOS 4 VECTORES  
**cmp** ecx, edx  
**jl** ciclo  
  
**mov** esp,ebp  
**pop** ebp  
**ret**

**Imágenes de Prueba**

**Imagen 1 (Figura) Imagen 2 (Fondo)**

** **

**Resultados de Superposición con diferentes valores de T y RGB:**

**RGB (0,0,0) T = 100**

****

**RGB (255,0,0) T=50**

****

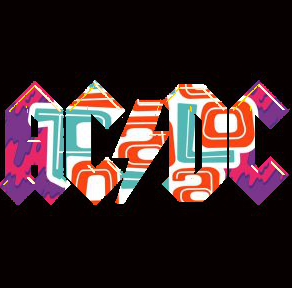
**RGB (0,0,0) T=5.5**

****

**RGB(0,255,0) T=300**



**RGB(255,0,0) T=200**



**RGB(255,0,0) T=30**

****

**Conclusión**

Durante el desarrollo de este trabajo práctico, pudimos ver como Interpolar y Superponer 2 imágenes usando operaciones, registros e instrucciones **XMM**. Estas operaciones sobre imágenes requieren el uso de valores de punto flotante debido a que el trabajo de las mismas se hace entre canales **RGB**, alto y ancho de las mismas y al operar entre ellas podemos llegar a obtener esos valores. Además vimos que al usar registros de XMM podemos operar vectores de datos mucho más rápido que los que usábamos en el 8086.

El uso de punto flotante sobre valores como la proporción de una imagen con otra, o el umbral de transparencia logra tener resultados muy diferentes en las imágenes de resultado.

Del lado del código de C, pudimos apreciar cómo podemos leer y escribir archivos binarios, y como llamar a un programa realizado por nosotros mediante parámetros, permitiendo de esta forma obtener resultados diferentes de ambas operaciones sobre imágenes mucho más fácil y rápido.

Para lograr todo esto, fue requerido usar un programa externo llamado **GraphicsMagick**, el cual nos permitió manipular las imágenes descargadas, convertirlas al formato RGB, visualizar esas imágenes, y las que obtuvimos como resultado realizando la interpolación y superposición.

Cabe destacar también, que pudimos apreciar y aprender sobre conceptos de imágenes y que la manipulación de las mismas en sí requiere de un conocimiento extra el cual obtuvimos a lo largo del desarrollo del trabajo práctico y las clases.

**Problemas Encontrados**

En esta parte vamos a destacar los problemas más importantes que encontramos y sus soluciones, cabe destacar que ya nos esperábamos este tipo de problemas debido a la complejidad del lenguaje y el problema. Por suerte se puso mucho empeño para la resolución de los mismos y el sistema final quedó con la estabilidad y robustez que buscábamos. Muchos de los mismos no volvieron a aparecer debido a la similaridad de las operaciones con respecto a la primera parte, nos fue muy útil el hecho de ya conocer el manejo de las instrucciones XMM de la primera parte para lograr resolver esta 2da parte. Los problemas fueron:

* **Función gen\_Mascara - ASM:** El hecho de estar limitados a tener solo 8 registros XMM nos jugo en contra, debido a que si tuvieramos 1 registro más (para dejar ahí un vector de T) no nos veríamos en la necesidad de cargar la máscara acomoda T y el valor de T para realizar la ultima comparacion de la funcion por cada ciclo que realice el programa**.** Con solo 1 registro más hubiéramos podido dejar T cargado en un registro con la forma de un vector de 4 posiciones para realizar la operación.
* **Función gen\_Mascara - ASM:** Por una idea equivocada del método CMPPS, realizabamos una conversión a entero del vector con la distancia al color elegido como transparente, pero CMPPS compara asumiendo que ambos son PS por lo cual el resultado no era el deseado, luego de darnos cuenta de este error procedimos a eliminar dicha conversión quedando la función operativa.
* **Función aplica\_mascara - ASM:** Debido a un mal manejo de los operadores lógicos involucrados, se obtenía el efecto contrario al buscado en el superponer, ya que en vez de aplicar el fondo en los pixeles de la figura que son transparentes realizaba lo contrario, una vez que advertimos el mismo, pasamos a cambiar de sentido las operaciones lógicas quedando luego el resultado deseado. El error del mismo se vio durante el debug realizado en el ASM utilizando vectores de prueba.
* **Función aplica\_mascara - C:** Debido a una idea equivocada, la aplicación de la máscara se hacía **3 veces utilizando el mismo vector**, quedando una imagen que no era acorde a los resultados de la Superposición. Luego de darnos cuenta del error, pasamos a crear 3 vectores, 1 por cada color, para así luego combinarlos y obtener la imagen resultado.