Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Ciencias de la Computación

Organización del computador II
Segundo cuatrimestre 2009

Grupo: POPA

Apellido y nombre	**L.U.**	**mail**
Cerrutti, Mariano Javier	525/07	vscorza@gmail.com
Huel, Federico Ariel	329/07	federico.huel@gmail.com
Mita, Rogelio Iván	635/07	rogeliomita@gmail.com

26 de noviembre de 2009

Índice

1. Archivos adjuntos

Implementación

- src:
  - *bordes.c*
  - *sobel.asm*
  - *prewitt.asm*
  - *roberts.asm*
  - *freiChen.asm*
- img:
  - *lena.bmp*

Includes

- *offset.inc*
- *macros.mac*
- *tp1bSobelPrewitt.mac*
- *tp1bRoberts.mac*
- *tp1bFreiChen.mac*

Enunciado

- *EnunciadoTP1b.pdf*

Informe

- *TP1-b.pdf*

2. Instrucciones de uso

Decidimos escribir los códigos de las funciones de forma separada para mayor claridad. Utilizamos también un archivo (*macros.mac*) para definir nuestras macros y (*offset.inc*) para darle nombres declarativos a los datos de la estructura imagen de `openCV`. En el cd adjunto al trabajo práctico, se encuentran todos los archivos fuente clasificados según el tipo. Decidimos utilizar un `makefile` para compilar todos los archivos a la vez.

3. Introducción

El motivo de este trabajo es ecribir un algoritmo de detección de bordes en lenguaje ensamblador. Para esto utilizaremos el concepto de derivadas parciales en un espacio de dimensión dos dado por el arreglo de píxeles que conforman una imagen, de esta manera podemos interesarnos en la derivada parcial respecto de x: dx o la derivada parcial respecto de y: dy.
En la práctica el concepto de derivada parcial se aproxima aplicando operadores especiales (*Roberts*,*Prewitt* o *Sobel*) que aplican una matriz de convolución a cada pixel de la imagen.
Nuestra motivación a la hora de escribir el código en lenguaje ensamblador viene dada por la necesidad de conseguir una solución eficiente en términos de tiempo para resolver el problema de detectar bordes en una imagen de tamaño arbitrario.

> Las funciones en código ensamblador estan divididas según el operador que se desea aplicar a la imagen para buscar bordes.

En particular, para este trabajo se ha hecho uso de la tecnología `SSE` de `Intel`, cuyo diseño es motivado por la necesidad de procesar varios datos en paralelo para operar sobre arreglos de bytes que pueden representar información de imagen (nuestro caso), sonido o video, o cualquier otro uso que pueda darse al procesamiento paralelo de señales o datos.
Los registros tienen un tamaño de 128 bits, permitiendo en nuestro caso llegar a cargar hasta ocho pixeles en un mismo registro. La justificación del uso de registros de mayor tamaño viene dada por la reducción en la cantidad de accesos a memoria realizados para procesar una misma cantidad de pixeles.

4. Desarrollo

Antes de presentar los algoritmos pertinentes a cada función hacemos algunas anotaciones generales:

- Nos fue pedido que en todos los algoritmos sea respetada la convención `C`, de forma que la construcción del `stack frame` (guardado de los registros edi, esi y ebx, ajuste de la pila, etc) se encuentra al principio de cada función dentro del macro `doEnter` junto con su contrapartida que se encuentra definida en el macro `doLeave`.

- En todas las funciones, consideramos que los parámetros pasados por referencia (excepto aquellos que debían cambiarse explícitamente de acuerdo al enunciado) no debían ser modificados, y por lo tanto, los mismos han sido guardados en variables locales o bien en registros. En algunos casos esto es muy importante, por ejemplo, cuando nos pasan un puntero a una lista, si se modifica ese puntero, se pierde la dirección de esa lista luego de llamar a la función; y esto no puede ocurrir.

- También tuvimos en cuenta que en las imágenes las filas de píxeles se guardan en memoria de forma alineada.

- Por cuestiones de simplicidad decidimos que los algoritmos también procesarán la basura (restos de bytes en memoria) que era incluída al alinear la imagen, ya que al momento de la visualización serán considerados aquellos píxeles contenidos dentro del ancho y el alto definido para la imagen.

- Se han optimizado los algoritmos para permitir procesar la mayor cantidad de pixeles por carga de memoria, esto nos ha llevado a mantener buenas prácticas de diseño para nuestros algoritmos.

A continuación se exponen las funciones implementadas, junto con sus respectivos códigos en lenguaje ensamblador y se explica el uso que se da a los registros de propósito general y el funcionamiento de las mismas en líneas generales.

4.1. Enfoque general

Para el cálculo aproximado de las derivadas parciales aplicamos matrices de convolución, que nos permiten aplicar un valor a un pixel de destino a partir de operar suma de productos a sus píxeles circundantes en la imagen de origen, en el ejemplo de la derivada parcial x para el operador de `Sobel` la matriz tendría este aspecto:

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

Podemos explicar su aplicación con el siguiente pseudocódigo:

1: $(\forall p_{i,j} : i \in ancho(imagenOrigen), j \in alto(imagenOrigen))(p_{i,j} \in [0 - 255])$
2: **for** $i \in ancho(imagenOrigen)$ **do**
3: **for** $j \in alto(imagenOrigen)$ **do**
4: $p_{i,j} \leftarrow -p_{i-1,j-1} - 2 * p_{i-1,j} - p_{i-1,j+1} + p_{i+1,j-1} + 2 * p_{i+1,j} + p_{i+1,j+1}$
5: **end for**
6: **end for**

Esta es la manera en la que aplicaremos matrices de convolución a nuestras imágenes para conseguir nuevas imágenes que contienen la información de las derivadas parciales para intentar detectar bordes, i.e. cambios bruscos en la intensidad de la imagen respecto de la dirección de abscisas y ordenadas o en ambas.

4.2. Archivos incluídos

4.2.1. offset.inc

Aquí se encuentran los desplazamientos a dato para cada una de las estructuras que usamos para la imagen del `openCV` por motivos de claridad en el código

```
1  ; typedef struct _IplImage
2  ; {
3  ;       int nSize;
4  ;       int ID;
5  ;       int nChannels;
6  ;       int alphaChannel;
7  ;       int depth;
8  ;       char colorModel[4];
9  ;       char channelSeq[4];
10 ;       int dataOrder;
11 ;       int origin;
12 ;       int align;
```

```
;       int width;
;       int height;
;       struct _IplROI *roi;
;       struct _IplImage *maskROI;
;       void *imageId;
;       struct _IplTileInfo *tileInfo;
;       int imageSize;
;       char *imageData;
;       int widthStep;
;       int BorderMode[4];
;       int BorderConst[4];
;       char *imageDataOrigin;
; }
; IplImage;

%define DEPTH           16
%define WIDTH           40
%define HEIGHT          44
%define IMAGE_DATA      68
%define WIDTH_STEP      72
```

4.2.2. macros.mac

Estos son las macros utilizadas en cada función. Para este trabajo sólo utilizamos `doEnter` y `doLeave` pero se han mantenido las macros restantes por suponer su utilidad futura.

```
;=============================
; MACRO doEnter
;=============================
; Escribe el encabezado que arma el stack frame
; Entrada:
;       tamanio         tamanio de memoria a reservar al entrar al proc
;=============================
%macro doEnter 0-1 0
        push ebp
        mov ebp, esp
%if %1 <> 0
        sub esp, %1
%endif
        push edi
        push esi
        push ebx
%endmacro
;=============================
; MACRO doLeave
;=============================
; Escribe la salida que restaura el stack frame previo
; Entrada:
;       tamanio         tamanio de memoria reservada al entrar al proc
;       doRet           se marca en 1 si debe llamar a ret
;=============================
%macro doLeave 0-2 0,0
```

```
27          pop ebx
28          pop esi
29          pop edi
30  %if %1 <> 0
31          add esp, %1
32  %endif
33          pop ebp
34  %if %2 <> 0
35          ret
36  %endif
37  %endmacro
38
39  ;=========================
40  ; MACRO doWrite
41  ;=========================
42  ; Escribe una cadena a consola
43  ; Entrada:
44  ;       mensaje         direccion de comienzo de la cadena
45  ;       len             largo de la cadena a escribir
46  ;=========================
47  %macro doWrite 1
48          %%msg: db %1
49          %%len: equ $- %%msg
50          mov eax,4                       ;inicializa escritura a consola
51          mov ebx,1
52          mov ecx,%%msg
53          mov edx,%%len
54          int 80h
55  %endmacro
56  ;=========================
57  ; MACRO doEnd
58  ;=========================
59  ; Termina la ejecucion con el codigo deseado
60  ; Entrada:
61  ;       codigo          codigo de error deseado, cero en su defecto
62  ;=========================
63  %macro doEnd 0-1 0
64          mov eax,1
65          mov ebx,%1
66          int 80h
67  %endmacro
68  ;=========================
69  ; MACRO doMalloc
70  ;=========================
71  ; Pide la cantidad especificada de memoria
72  ; Entrada:
73  ;       cantidad        cantidad de memoria a reserver
74  ;=========================
75  %macro doMalloc 1
76          push %1
77          call malloc
78          add esp, 4
79  %endmacro
```

```
80  ;=============================
81  ; MACRO doRetc
82  ;=============================
83  ; Retorna si se cumple la condicion especificada
84  ; Entrada:
85  ;       condicion       condicion ante la cual retornar
86  ;=============================
87  %macro  doRetc 1
88          j%-1    %%skip
89          ret
90    %%skip:
91  %endmacro
92  ;=============================
93  ; MACRO doWriteFile
94  ;=============================
95  ; Escribe a archivo solo si esta definido el DEBUG
96  ; Entrada:
97  ;       arch            puntero al archivo
98  ;       msg             texto a escribir
99  ; Uso:
100 ;       doWriteFile [FHND], {"hola mundo",13,10}
101 ;=============================
102 %macro  doWriteFile 2+
103 %ifdef DEBUG
104         jmp     %%endstr
105   %%str:        db      %2
106   %%endstr:
107         mov     dx,%%str
108         mov     cx,%%endstr-%%str
109         mov     bx,%1
110         mov     ah,0x40
111         int     0x21
112 %endif
113 %endmacro
```

4.3. Recorrido de la matriz de imagen

1. **Cosas que tuvimos en cuenta**

   La matriz fue recorrida procesando la basura, i.e, en todo el ancho de la imagen alineada

   Cada pixel ocupa exactamente un byte, ya que la imagen que recibe la función es en escala de grises (0-255)

   En la arquitectura `IA-32` los datos son guardados en memoria de forma *little-endian*, por lo que el byte más significativo de un dato se encuentra en la posición de memoria más alta.

4.4. Funciones implementadas

- Intentamos reducir el acceso a memoria utilizando registros para aquellas operaciones que se encontraban dentro de los ciclos, evitando así múltiples accesos a memoria. Si bien esto llevó a tener un código menos legible, lo explicaremos en detalle.

- Variables locales

  SRC Contiene el puntero a la imagen de entrada.

  DST Contiene el puntero a la imagen resultado.

  WIDTH Contiene el tamaño de ancho de la imagen.

  HEIGHT Contiene la altura de la imagen.

  XORDER Variable que especifica si el cálculo debe hacerse sobre la derivada X.

  YORDER Variable que especifica si el cálculo debe hacerse sobre la derivada Y.

  REMAINDER Variable que se usa para mantener el resto del ancho de la imagen módulo el ancho del registro

  eax Se utiliza como acumulador, para realizar los cálculos de los píxeles.

  ebx Se utiliza como acumulador, para realizar los cálculos de los píxeles y para mantener el dato del alto de la imagen.

  esi Se utiliza para recorrer la imagen de salida.

  edi Se utiliza para recorrer la imagen de entrada.

- **Funcionamiento**

  Lo que intentamos hacer en todos los algoritmos es optimizar la cantidad de cálculos por cada carga que hacemos de la memoria principal a los registro de procesamiento de señales.

4.4.1. Sobel y Prewitt

Presentaremos la el código de la función completa de `Sobel` y `Prewitt`.

1. **Orden X**

Para aplicar el operador en el orden X se cargan seis registros `xmm` con 16 bytes de la imagen en cada uno, correspondientes a seis filas consecutivas, esto es, a seis tiras de 16 pixeles de la imagen. Luego utlizamos dos funciones auxiliares que desempaquetan la parte alta o baja del registro, convirtiendo los datos de byte a word, permitiendo operar sobre ellos sin perder precisión por desbordamiento. Luego son empaqutados y saturados como corresponde. De los 16×6 pixeles que se cargan en los registros de procesamiento de señales se consigue procesar 14×4 pixeles de la imagen final. Los últimos registros `xmm6` y `xmm7` son utilizados como registros temporales y de acumulación. Ya que sobre estos se desempaquetan los datos cargados en los anteriores y sobre uno de ellos se acumula el resultado de aplicar la matriz de convolución. Se avanza sobre una base de 14 pixeles horizontalmente, por lo expuesto anteriormente, y al completar una fila, para aprovechar que se han procesado varias filas a la vez, el índice salta cuatro filas. Para compensar los pixeles que pueden haber quedado sin procesar, se calcula, al entrar a la función se calcula el resto del ancho de imagen módulo el ancho de los registros.

```
1  ;========================
2  ; MACRO obtenerBajo
3  ;========================
4  ; Recupera la parte alta de un dato empaquetado a byte
5  ; Entrada:
6  ;       registro1       registros sobre los cuales operar
7  ;       registro2
8  ;       registro3
9  ;       offset          indica si debe desplazar el dato original
10 ;========================
11 %macro obtenerBajo 2-3 0
12         movdqu          %1, %2          ;copio el dato
13 %if %3 != 0
14         psrldq          %1, %3          ;desplazo dos columnas
15 %endif
16         punpcklbw       %1, %1          ;desempaqueto la parte baja
17         psllw           %1, 8           ;retiro el excedente alto
18         psrlw           %1, 8
19 %endmacro
20
21 %macro obtenerAlto 2-3 0
22         movdqu          %1, %2          ;copio el dato
23 %if %3 != 0
24         psrldq          %1, %3          ;desplazo dos columnas
25 %endif
26         punpckhbw       %1, %1          ;desempaqueto la parte baja
27         psllw           %1, 8           ;retiro el excedente alto
28         psrlw           %1, 8
29 %endmacro
30
31 ;========================
32 ; MACRO sobelPrewittX
33 ;========================
34 ; Cuerpo de codigo que aplica los operadores de Sobel
35 ; y Prewitt en el orden x
```

```
36 ; Entrada:
37 ;         registro1        registros sobre los cuales operar
38 ;         registro2
39 ;         registro3
40 ;         duplica?         debe duplicar el valor del primer registro?
41 ;         procesaAmbos     indica si debe procesar el dato bajo y el alto
42 ;=========================
43 %macro   sobelPrewittX 4-5 0
44          ;*************************
45          ; VOY A APLICAR EL OPERADOR
46          ; EN LA PARTE BAJA
47          ;*************************
48          obtenerBajo      xmm6, %2, 2      ;obtengo los bajos
49                                            ;de la segunda fila
50 %if %4 = 0
51          psllw            xmm6, 1          ;multiplico por dos
52 %endif
53          obtenerBajo      xmm7, %1, 2      ;obtengo los bajos
54                                            ;de la primera fila
55          paddusw          xmm6, xmm7       ;sumo saturado
56          obtenerBajo      xmm7, %3, 2      ;obtengo los bajos
57                                            ;de la tercera fila
58          paddusw          xmm6, xmm7       ;sumo saturado
59
60          obtenerBajo      xmm7, %2         ;obtengo los bajos
61                                            ;de la segunda fila
62 %if %4 = 0
63          psllw            xmm7, 1          ;multiplico por dos
64 %endif
65          psubusw          xmm6, xmm7
66          obtenerBajo      xmm7, %1         ;obtengo los bajos
67                                            ;de la primera fila
68          psubusw          xmm6, xmm7       ;sumo saturado
69          obtenerBajo      xmm7, %3         ;obtengo los bajos
70                                            ;de la tercera fila
71          psubusw          xmm6, xmm7       ;sumo saturado
72
73          ;************************
74          ; TERMINE COPIO A DESTINO
75          ;************************
76          packuswb         xmm6, xmm6
77
78          movq             [eax], xmm6      ;copio los 8 bytes
79                                            ;al destino
80
81 %if %5 = 0
82          add              eax, 8           ;salto la linea siguiente
83          ;*************************
84          ; VOY A APLICAR EL OPERADOR
85          ; EN LA PARTE ALTA
86          ;*************************
87          obtenerAlto      xmm6, %2, 2      ;obtengo los bajos
88                                            ;de la segunda fila
```

```
89  %if %4 = 0
90          psllw           xmm6, 1         ;multiplico por dos
91  %endif
92          obtenerAlto     xmm7, %1, 2     ;obtengo los bajos
93                                          ;de la primera fila
94          paddusw         xmm6, xmm7      ;sumo saturado
95          obtenerAlto     xmm7, %3, 2     ;obtengo los bajos
96                                          ;de la tercera fila
97          paddusw         xmm6, xmm7      ;sumo saturado
98
99          obtenerAlto     xmm7, %2        ;obtengo los bajos
100                                         ;de la segunda fila
101 %if %4 = 0
102         psllw           xmm7, 1         ;multiplico por dos
103 %endif
104         psubusw         xmm6, xmm7
105         obtenerAlto     xmm7, %1        ;obtengo los bajos
106                                         ;de la primera fila
107         psubusw         xmm6, xmm7      ;sumo saturado
108         obtenerAlto     xmm7, %3        ;obtengo los bajos
109                                         ;de la tercera fila
110         psubusw         xmm6, xmm7      ;sumo saturado
111
112         ;************************
113         ; TERMINE COPIO A DESTINO
114         ;************************
115         packuswb        xmm6, xmm6
116         movq            [eax], xmm6     ;copio los 4 bytes al destino
117                                         ;de los cuales 2 son validos
118         sub             eax, 8
119 %endif
120         add             eax, edx        ;salto la linea siguiente
121 %endmacro
```

2. **Orden Y**

En el caso del orden Y, para aprovechar la carga de los pixeles en los registros `xmm`, y sin hacer uso de las operaciones propias de `sse3` generamos una máscara en uno de los registros que se mantiene en memoria y se utiliza para filtrar los pixeles intercalados de a byte, para poder realizar esta secuencia de operaciones: *filtrar* → *desplazar* → *sumar/restar*.
Lo que vamos a hacer es operar sobre los pixeles de forma intercalada, primero sobre los pares, filtramos, acomodamos, operamos, y luego saturamos. Esto nos dejará procesados los pixeles pares primero, luego los impares, y es por esto que debemos aplicar una operación `por` en lugar de, por ej. `movdqu` para conservar los pixeles ya calculados.

```
1  ;=========================
2  ; MACRO sobelPrewittY
3  ;=========================
4  ; Cuerpo de codigo que aplica los operadores de Sobel
5  ; y Prewitt en el orden y
6  ; Entrada:
```

```
7  ;       registro1       registros sobre los cuales operar
8  ;       registro2
9  ;       duplica?        debe duplicar el valor del primer registro?
10 ;       procesaAmbos    indica si debe procesar el dato bajo y el alto
11 ;=========================
12 %macro  sobelPrewittY 2-3 0
13         ;**********************
14         ; PROCESO PIXEL 2,4,6,8
15         ;**********************
16         movdqu  xmm5, %1        ;cargo el primer registro
17                                 ;y enmascaro para pasar a word
18         pand    xmm5, xmm7
19         movdqu  xmm6, %1        ;cargo y me quedo con el pixel
20                                 ;una a derecha
21         pslldq  xmm6, 1
22         pand    xmm6, xmm7
23 %if %3 = 0
24         psllw   xmm6, 1         ;multiplico por dos
25 %endif
26         paddusw xmm5, xmm6      ;sumo el segundo pixel
27         movdqu  xmm6, %1
28         pslldq  xmm6, 2         ;cargo y me quedo con el
29                                 ;pixel dos a derecha
30         pand    xmm6, xmm7
31         paddusw xmm5, xmm6      ; ya tengo acumulado en xmm5 la
32                                 ;parte positiva de los pixeles
33                                 ;2, 4, 6, 8
34
35         movdqu  xmm6, %2
36         pand    xmm6, xmm7
37         psubusw xmm5, xmm6
38         movdqu  xmm6, %2
39         pslldq  xmm6, 1
40         pand    xmm6, xmm7
41 %if %3 = 0
42         psllw   xmm6, 1         ;multiplico por dos
43 %endif
44         psubusw xmm5, xmm6
45         movdqu  xmm6, %2
46         pslldq  xmm6, 2         ;cargo y me quedo con el
47                                 ;pixel dos a derecha
48         pand    xmm6, xmm7
49         psubusw xmm5, xmm6      ; ya el operador aplicado
50                                 ;en xmm5 a los pixeles, 2, 4, 6, 8
51         pxor    xmm6, xmm6
52         packuswb        xmm5, xmm6
53         punpcklbw       xmm5, xmm6      ;saturo los pixeles
54         movdqu  xmm6, [eax]
55         por     xmm5, xmm6
56         movdqu  [eax], xmm5     ;copio los pixeles
57
58         ;**********************
59         ; PROCESO PIXEL 1,3,5
```

```
60          ;**********************
61          movdqu   xmm5, %1          ;cargo el primer registro y
62                                     ;enmascaro para pasar a word
63          psrldq   xmm5, 1
64          pand     xmm5, xmm7
65          movdqu   xmm6, %1          ;cargo y me quedo con el pixel
66                                     ;una a derecha
67          pand     xmm6, xmm7
68  %if %3 = 0
69          psllw    xmm6, 1           ;multiplico por dos
70  %endif
71          paddusw  xmm5, xmm6        ;sumo el segundo pixel
72          movdqu   xmm6, %1
73          pslldq   xmm6, 1           ;cargo y me quedo con el pixel
74                                     ;dos a derecha
75          pand     xmm6, xmm7
76          paddusw  xmm5, xmm6        ; ya tengo acumulado en xmm5 la
77                                     ;parte positiva de los pixeles
78                                     ;1, 3, 5
79
80          movdqu   xmm6, %2
81          psrldq   xmm6, 1
82          pand     xmm6, xmm7
83          psubusw  xmm5, xmm6
84          movdqu   xmm6, %2
85          pand     xmm6, xmm7
86  %if %3 = 0
87          psllw    xmm6, 1           ;multiplico por dos
88  %endif
89          psubusw  xmm5, xmm6
90          movdqu   xmm6, %2
91          pslldq   xmm6, 1           ;cargo y me quedo con el pixel
92                                     ;dos a derecha
93          pand     xmm6, xmm7
94          psubusw  xmm5, xmm6        ;ya el operador aplicado en xmm5
95                                     ;a los pixeles, 2, 4, 6, 8
96
97          pxor     xmm6, xmm6
98          packuswb         xmm5, xmm6
99          punpcklbw        xmm5, xmm6       ;saturo los pixeles
100         pslldq   xmm5, 1
101         movdqu   xmm6, [eax]
102         por      xmm5, xmm6
103         movdqu   [eax], xmm5               ;copio los pixeles
104
105         add              eax, edx          ;salto la linea siguiente
106 %endmacro
```

4.4.2. Roberts

1. **Ambos órdenes**

   En el caso de `Roberts` pueden cargarse seis filas de 16 pixeles y procesarse efecti-

vamente 15×5 pixeles de la imagen de destino. Se utilizan cinco registros de `xmm` para cargar los pixeles de origen y dos registros para uso temporal y acumulador. Simplemente se cargan, se mueven a los registros de uso temporal, se desplazan en el sentido correspondiente, se restan y se mueven (empaquetando y desempaquetando para saturar el dato) al destino. :

```
1  %define REMAINDER          [ebp - 4]
2  %define STEP_X             15
3  %define RESERVED_BYTES     0
4
5
6  ;=========================
7  ; MACRO robertsXY
8  ;=========================
9  ; Cuerpo de codigo que aplica el operador de
10 ;Roberts en el orden x o y
11 ; Entrada:
12 ;        registro1        registros sobre los cuales operar
13 ;        registro2
14 ;        esX?
15 ;=========================
16 %macro   robertsXY 3
17         ;procesamos la parte baja
18         movdqu           xmm7, %2
19         movdqu           xmm6, %1
20 %if  %3 = 0
21         psrldq           xmm6, 1          ;desplazo para hacer
22                                           ;la aritmetica
23 %else
24         psrldq           xmm7, 1
25 %endif
26         punpcklbw        xmm7, xmm7
27         psllw            xmm7, 8          ;limpio la parte alta
28                                           ;del word
29         psrlw            xmm7, 8
30         punpcklbw        xmm6, xmm6
31         psllw            xmm6, 8          ;limpio la parte alta
32                                           ;del word
33         psrlw            xmm6, 8
34         psubusw          xmm6, xmm7
35
36         packuswb         xmm6, xmm6
37
38         movdqu           xmm7, [eax]
39         por              xmm6, xmm7
40         movq             [eax], xmm6      ;copio los 4 bytes
41                                           ;al destino
42
43         add              eax, 8           ;paso a los proximos
44                                           ;cuatro bytes
45
46         ;procesamos la parte alta
47         movdqu           xmm7, %2
```

```
48          movdqu          xmm6, %1
49  %if %3 = 0
50          psrldq          xmm6, 1
51  %else
52          psrldq          xmm7, 1
53  %endif
54          punpckhbw       xmm7, xmm7
55          psllw           xmm7, 8
56          psrlw           xmm7, 8
57          punpckhbw       xmm6, xmm6
58          psllw           xmm6, 8
59          psrlw           xmm6, 8
60          psubusw         xmm6, xmm7
61
62          packuswb        xmm6, xmm6
63          psllq           xmm6, 8         ;limpio el byte m s alto
64                                          ;pues no tiene dato valido
65          psrlq           xmm6, 8
66
67          movdqu          xmm7, [eax]
68          por             xmm6, xmm7
69          movq            [eax], xmm6     ;copio los 4 bytes al destino
70          sub             eax, 8
71          add             eax, edx        ;salto la linea siguiente
72  %endmacro
```

4.4.3. Frei-Chen

1. **Orden X**

   El funcionamiento es similar al de `Sobel` y `Prewitt`, con la diferencia que en lugar de procesar en dos partes se procesa en cuatro, esto se debe a que las operaciones en precisión simple se realizan sobre flotantes de 32 bits, en lugar de hacerlo sobre valores enteros representados con 16 bits. Para conseguirlo se mantienen dos registros para uso temporal y de acumulador, y sobre ellos se desempaquetan y transforman a valores de precisión simple las cuatro partes correspondientes del registro de 128 bits. De esta forma, una vez que se procesaron los pixeles convertidos a precisión simple se escribe su resultado en la imagen de destino de a cuatro pixeles ($4 \times 8bits$). Antes de volcar el resultado en el destino se empaquetan y desempaquetan los datos para saturarlos.
   Para poder multiplicar los valores por la constante $\sqrt{2}$ lo calculamos en el registro `mm7` de la `FPU` y luego lo movemos a un registro de 128 bits como escalar y lo copiamos a través de un `shuffle`.

```
1   %define REMAINDER       [ebp - 4]
2   %define STEP_X          14
3   %define STEP_Y          12
4   %define RESERVED_BYTES  4
5
6   ;========================
7   ; MACRO obtener
8   ;========================
```

```
9  ; Recupera la parte correspondiente de un
10 ; dato empaquetado a byte
11 ; Entrada:
12 ;        registro1       registros sobre los cuales operar
13 ;        registro2
14 ;        registro3
15 ;        offset          indica si debe desplazar el dato original
16 ;=========================
17 %macro obtener 3-4 0
18         movdqu          %2, %3          ;copio el dato
19 %if %4 != 0
20         psrldq          %2, %4          ;desplazo dos columnas
21 %endif
22 %if %1 = 2
23         punpckhbw       %2, %2          ;desempaqueto la parte alta
24 %elif %1 = 3
25         punpckhbw       %2, %2          ;desempaqueto la parte alta
26 %else
27         punpcklbw       %2, %2          ;desempaqueto la parte baja
28 %endif
29         psllw           %2, 8           ;retiro el excedente alto
30         psrlw           %2, 8
31 %if %1 = 1
32         punpckhbw       %2, %2          ;desempaqueto la parte alta
33                                         ;de la parte baja
34 %elif %1 = 3
35         punpckhbw       %2, %2          ;desempaqueto la parte alta
36                                         ;de de la parte alta
37 %else
38         punpcklbw       %2, %2          ;desempaqueto la parte baja
39                                         ;correspondiente
40 %endif
41         pslld           %2, 24
42         psrld           %2, 24
43         cvtdq2ps        %2, %2          ;convierto a precision simple
44 %endmacro
45
46 ;=========================
47 ; MACRO cargarRaiz2
48 ;=========================
49 ; Carga raiz de dos en el registro xmm7
50 ; Entrada:
51 ;        registro1       registros sobre los cuales operar
52 ;=========================
53 %macro cargarRaiz2 1
54         movq2dq         %1, mm7         ;copio la raiz en precision
55                                         ;doble
56         pshufd          %1, %1, 0x00    ;copio el mas bajo a todos
57                                         ;los dword ps
58 %endmacro
59
60 ;=========================
61 ; MACRO aplicarOperadorX
```

```
62  ;=========================
63  ; Aplicar operador a cuatro bytes
64  ; Entrada:
65  ;       registro1       registros sobre los cuales operar
66  ;       registro2       registros sobre los cuales operar
67  ;       registro3       registros sobre los cuales operar
68  ;       word            word a obtener
69  ;       salta a la linea?       salta a la linea siguiente?
70  ;=========================
71  %macro aplicarOperadorX 4-5 0
72          ;*************************
73          ; VOY A APLICAR EL OPERADOR
74          ; A LA PARTE CORRESPONDIENTE
75          ;*************************
76          cargarRaiz2     xmm7
77          obtener         %4, xmm6, %2, 2 ;obtengo la parte
78                                          ;correspondiente
79                                          ;de la segunda fila
80          mulps           xmm6, xmm7
81
82          obtener         %4, xmm7, %1, 2 ;obtengo la parte
83                                          ;correspondiente
84                                          ;de la primera fila
85          addps           xmm6, xmm7      ;sumo
86          obtener         %4, xmm7, %3, 2 ;obtengo la parte
87                                          ;correspondiente
88                                          ;de la tercera fila
89          addps           xmm6, xmm7      ;sumo
90
91          obtener         %4, xmm7, %1    ;obtengo la parte
92                                          ;correspondiente
93                                          ;de la primera fila
94          subps           xmm6, xmm7      ;resto
95          obtener         %4, xmm7, %3    ;obtengo la parte
96                                          ;correspondiente
97                                          ;de la tercera fila
98          subps           xmm6, xmm7      ;resto
99                                          ;aca estoy usando un
100                                         ;truco que es dividir
101                                         ;todo menos un item por
102                                         ;raiz de dos
103                                         ;sumar el item y
104                                         ;multiplicar todo
105                                         ;por raiz de dos, i.e:
106                                         ;((a+b+c-d-f)/2^(.5))-e*2^(.5)
107         cargarRaiz2     xmm7
108         divps           xmm6, xmm7
109         obtener         %4, xmm7, %2    ;obtengo la parte
110                                         ;correspondiente
111                                         ;de la segunda fila
112         subps           xmm6, xmm7
113         cargarRaiz2     xmm7
114         mulps           xmm6, xmm7
```

```
115         ;************************
116         ; TERMINE COPIO A DESTINO
117         ;************************
118         cvttps2dq       xmm6, xmm6
119         packusdw        xmm6, xmm6
120         packuswb        xmm6, xmm6
121         movd            [eax], xmm6     ;copio los 4 bytes al destino
122 %if %5 = 0
123         add             eax, 4          ;salto la linea siguiente
124 %endif
125 %endmacro
126 ;=========================
127 ; MACRO freichenX
128 ;=========================
129 ; Cuerpo de codigo que aplica el operador de freichen en el orden x
130 ; Entrada:
131 ;       registro1       registros sobre los cuales operar
132 ;       registro2
133 ;       registro3
134 ;       procesaAmbos    indica si debe procesar los datos mas altos
135 ;=========================
136 %macro  freiChenX 3-4 0
137 %if %4 = 0
138         aplicarOperadorX %1, %2, %3, 0
139         aplicarOperadorX %1, %2, %3, 1
140         aplicarOperadorX %1, %2, %3, 2
141         aplicarOperadorX %1, %2, %3, 3, 1
142         sub             eax, 12
143 %else
144         aplicarOperadorX %1, %2, %3, 0, 1
145 %endif
146         add             eax, edx        ;salto la linea siguiente
147 %endmacro
```

2. **Orden Y**

Otra vez, el algoritmo para la aplicación del operador en el orden Y guarda muchas similitudes con su contraparte de `Sobel` o `Prewitt` en el sentido que ha de ser cargado de a cuatro líneas (cuatro registros) reservando tres registros para la aplicación de máscaras, carga de datos temporales y acumulación. Y también serán ubicados en su lugar a partir de un filtro aplicado con una máscara que selecciona los pixeles correspondientes, procesando en paralelo cuatro de los bytes, operando sobre éstos y avanzando el índice de la imagen destino, rotando los registros cargados, y así procesar todos los pixeles posibles dentro de las líneas almacenadas en registros de 128 bits. En cada proceso y al igual que en el algoritmo de orden x se convierten los datos a precisión simple y luego se transforman a enteros de 32 bits empaquetados, para poder empaquetar y desempaquetar con facilidad se descartan los valores negativos. Otra vez en lugar de mover los datos a destino con una operación de copia directa estamos usando una operación de `por` para poder escribir los pixeles procesados en paralelo, conservando los que han sido escritos anteriormente.

```
;=========================
; MACRO aplicarOperadorY
;=========================
; Aplica el operador freiChen en y a cuatro pixeles
; Entrada:
;       registro1       registros sobre los cuales operar
;       registro2
;       esGrupoMasBajo? indica si es el grupo mas bajo
;=========================
%macro aplicarOperadorY 2-3 0
        movdqu          xmm5, %1        ;cargo el primer registro
                                        ;y enmascaro para pasar a word
        armarMascara    xmm7
        pand            xmm5, xmm7
        cvtdq2ps        xmm5, xmm5      ;convierto a precision simple

        movdqu          xmm6, %1
        pslldq          xmm6, 2         ;cargo y me quedo con el
                                        ;pixel dos a derecha
        pand            xmm6, xmm7
        cvtdq2ps        xmm6, xmm6
        addps           xmm5, xmm6

        movdqu          xmm6, %2
        pand            xmm6, xmm7
        cvtdq2ps        xmm6, xmm6
        subps           xmm5, xmm6

        movdqu          xmm6, %2
        pslldq          xmm6, 2         ;cargo y me quedo con el
                                        ;pixel dos a derecha
        pand            xmm6, xmm7
        cvtdq2ps        xmm6, xmm6
        subps           xmm5, xmm6      ;ya el operador aplicado en
                                        ;xmm5 a los pixeles
                                        ;2, 4, 6, 8


        movdqu          xmm6, %1        ;cargo y me quedo con el pixel
                                        ;una a derecha
        pslldq          xmm6, 1
        pand            xmm6, xmm7
        cvtdq2ps        xmm6, xmm6      ;lo mismo enmascaro y convierto
                                        ;a precision simple luego
                                        ;de desplazar
        cargarRaiz2     xmm7
        mulps           xmm6, xmm7
        addps           xmm5, xmm6      ;multiplico por raiz de
                                        ;dos y acumulo


        movdqu          xmm6, %2
        pslldq          xmm6, 1
```

```
54          armarMascara    xmm7
55          pand            xmm6, xmm7
56          cvtdq2ps        xmm6, xmm6
57          cargarRaiz2     xmm7
58          mulps           xmm6, xmm7
59          subps           xmm5, xmm6
60
61          cvttps2dq       xmm5, xmm5
62
63          pxor            xmm6, xmm6
64
65  ;Descartamos los valores negativos
66          movdqu          xmm7, xmm5
67          pcmpgtd         xmm7, xmm6
68          pand            xmm5, xmm7
69
70          packssdw        xmm5, xmm6
71          packsswb        xmm5, xmm6
72          punpcklbw       xmm5, xmm6      ;saturo los pixeles
73          punpcklwd       xmm5, xmm6      ;saturo los pixeles
74          movdqu          xmm6, [eax]
75          por             xmm5, xmm6
76          movdqu          [eax], xmm5     ;copio los pixeles
77  %if %3 = 0
78          inc             eax
79          movdqu          xmm7, %1        ;simulando rotates
80          pslldq          xmm7, 15
81          psrldq          %1, 1
82          por             %1, xmm7
83          movdqu          xmm7, %2
84          pslldq          xmm7, 15
85          psrldq          %2, 1
86          por             %2, xmm7
87  %else
88          movdqu          xmm7, %1        ;simulando rotates para
89                                          ;dejar como estaba
90          psrldq          xmm7, 11
91          pslldq          %1, 3
92          por             %1, xmm7
93          movdqu          xmm7, %2
94          psrldq          xmm7, 11
95          pslldq          %2, 3
96          por             %2, xmm7
97
98          sub             eax, 3
99          add             eax, edx        ;salto la linea siguiente
100         ;add            eax, edx        ;salto la linea siguiente
101 %endif
102 %endmacro
103
104 ;=========================
105 ; MACRO armarMascara
106 ;=========================
```

```
107 ; Arma la mascara para seleccionar pixeles pares o impares
108 ; Entrada:
109 ;         registro1        registros sobre los cuales operar
110 ;=============================
111
112 ;000000ff000000ff 000000ff00000000
113 %macro armarMascara 1
114         pcmpeqb  %1, %1                    ;armo la mascara
115                                            ;paso todo a uno
116         pslld    %1, 24
117         psrld    %1, 24                    ;descarto los tres primeros
118                                            ;bytes en cada dword
119         psrldq   %1, 1                     ;descarto el ultimo byte
120         pslldq   %1, 1
121 %endmacro
122 ;=============================
123 ; MACRO freichenY
124 ;=============================
125 ; Cuerpo de codigo que aplica el operador de freichen en el orden y
126 ; Entrada:
127 ;         registro1        registros sobre los cuales operar
128 ;         registro2
129 ;=============================
130 %macro   freiChenY 2
131         ; PROCESO PIXEL 3,7,11,15
132         aplicarOperadorY %1, %2
133         ;PROCESO PIXEL 2,6,10,14
134         aplicarOperadorY %1, %2
135         ; PROCESO PIXEL 1,5,9,13
136         aplicarOperadorY %1, %2
137         ; PROCESO PIXEL 0,4,8,12
138         aplicarOperadorY %1, %2, 1
139 %endmacro
```

5. Resultados

```
imagen 1024 x 768:

cvSobel01 tarda en un promedio de 100 corridas: 21496417
cvSobel10 tarda en un promedio de 100 corridas: 21240004
cvSobel11 tarda en un promedio de 100 corridas: 20093462
asmSobel01 tarda en un promedio de 100 corridas: 2186426
asmSobel10 tarda en un promedio de 100 corridas: 3156124
asmSobel11 tarda en un promedio de 100 corridas: 5329166
cvSobel10 demoro: 21334379
cvSobel01 demoro: 21298612
cvSobel11 demoro: 20207645
asmSobel10 demoro: 2548121
asmSobel01 demoro: 3217386
asmSobel11 demoro: 5721240
asmPrewitt10 demoro: 2496433
asmPrewitt01 demoro: 3588309
asmPrewitt11 demoro: 5683244
asmRoberts10 demoro: 2055716
asmRoberts01 demoro: 2052588
asmRoberts11 demoro: 4284935
asmFreiChen10 demoro: 8669932
asmFreiChen01 demoro: 11275539
asmFreiChen11 demoro: 19450508
-----------------------------------------------------
Proposito general:
asmSobel01 tarda en un promedio de 100 corridas: 21722034
asmSobel10 tarda en un promedio de 100 corridas: 20952528
asmSobel11 tarda en un promedio de 100 corridas: 40506787
asmSobel10 demoro: 21720212
asmSobel01 demoro: 21030930
asmSobel11 demoro: 40562000
asmPrewitt10 demoro: 20541695
asmPrewitt01 demoro: 21027343
asmPrewitt11 demoro: 39380101
asmRoberts10 demoro: 21572550
asmRoberts01 demoro: 9251153
asmRoberts11 demoro: 28648697



Imagen 1600 x 1200:

cvSobel01 tarda en un promedio de 100 corridas: 64629757
cvSobel10 tarda en un promedio de 100 corridas: 58466592
cvSobel11 tarda en un promedio de 100 corridas: 60267866
asmSobel01 tarda en un promedio de 100 corridas: 5675814
asmSobel10 tarda en un promedio de 100 corridas: 8027895
asmSobel11 tarda en un promedio de 100 corridas: 13336611
cvSobel10 demoro: 64738711
cvSobel01 demoro: 58334633
```

```
cvSobel11 demoro: 59916075
asmSobel10 demoro: 6058315
asmSobel01 demoro: 9326455
asmSobel11 demoro: 13821527
asmPrewitt10 demoro: 6240113
asmPrewitt01 demoro: 8580648
asmPrewitt11 demoro: 14136860
asmRoberts10 demoro: 6364043
asmRoberts01 demoro: 5500435
asmRoberts11 demoro: 10229359
asmFreiChen10 demoro: 20931258
asmFreiChen01 demoro: 27253389
asmFreiChen11 demoro: 47268219
-------------------------------------------------
Proposito general:
asmSobel01 tarda en un promedio de 100 corridas: 62167096
asmSobel10 tarda en un promedio de 100 corridas: 57031508
asmSobel11 tarda en un promedio de 100 corridas: 113866178
asmSobel10 demoro: 62156131
asmSobel01 demoro: 57158743
asmSobel11 demoro: 113898691
asmPrewitt10 demoro: 59797730
asmPrewitt01 demoro: 58153855
asmPrewitt11 demoro: 112996535
asmRoberts10 demoro: 61708300
asmRoberts01 demoro: 44887055
asmRoberts11 demoro: 104265539



Imagen 3296 x 2472:

cvSobel01 tarda en un promedio de 100 corridas: 264058666
cvSobel10 tarda en un promedio de 100 corridas: 242124053
cvSobel11 tarda en un promedio de 100 corridas: 246805669
asmSobel01 tarda en un promedio de 100 corridas: 24710288
asmSobel10 tarda en un promedio de 100 corridas: 34586703
asmSobel11 tarda en un promedio de 100 corridas: 59155348
cvSobel10 demoro: 263067537
cvSobel01 demoro: 241957668
cvSobel11 demoro: 246281108
asmSobel10 demoro: 25204770
asmSobel01 demoro: 34600100
asmSobel11 demoro: 59122311
asmPrewitt10 demoro: 25534680
asmPrewitt01 demoro: 34851411
asmPrewitt11 demoro: 60214807
asmRoberts10 demoro: 23821250
asmRoberts01 demoro: 23942656
asmRoberts11 demoro: 48725179
asmFreiChen10 demoro: 87244739
asmFreiChen01 demoro: 121413685
asmFreiChen11 demoro: 201409744
```

```
-----------------------------------------------------------------
proposito general:
asmSobel01 tarda en un promedio de 100 corridas: 254878321
asmSobel10 tarda en un promedio de 100 corridas: 238509534
asmSobel11 tarda en un promedio de 100 corridas: 470568287
asmSobel10 demoro: 260648870
asmSobel01 demoro: 238654780
asmSobel11 demoro: 473785606
asmPrewitt10 demoro: 245286260
asmPrewitt01 demoro: 242413650
asmPrewitt11 demoro: 465092188
asmRoberts10 demoro: 259926030
asmRoberts01 demoro: 188326068
asmRoberts11 demoro: 425289272



Imagen 9466 x 7781 (50 corridas):

cvSobel01 tarda en un promedio de 50 corridas: 1887282788
cvSobel10 tarda en un promedio de 50 corridas: 1879896597
cvSobel11 tarda en un promedio de 50 corridas: 1762629576
asmSobel01 tarda en un promedio de 50 corridas: 210250426
asmSobel10 tarda en un promedio de 50 corridas: 301825348
asmSobel11 tarda en un promedio de 50 corridas: 511810459
cvSobel10 demoro: 1880594264
cvSobel01 demoro: 1877404826
cvSobel11 demoro: 1759663583
asmSobel10 demoro: 209304910
asmSobel01 demoro: 301189136
asmSobel11 demoro: 510795948
asmPrewitt10 demoro: 216600808
asmPrewitt01 demoro: 308188096
asmPrewitt11 demoro: 523424262
asmRoberts10 demoro: 187772463
asmRoberts01 demoro: 188300933
asmRoberts11 demoro: 375835966
asmFreiChen10 demoro: 774139846
asmFreiChen01 demoro: 1004473840
asmFreiChen11 demoro: 1780969062
-----------------------------------------------------------------
proposito general (25 corridas):
asmSobel01 tarda en un promedio de 25 corridas: 1951103136
asmSobel10 tarda en un promedio de 25 corridas: 1860946539
asmSobel11 tarda en un promedio de 25 corridas: -688392667
asmSobel10 demoro: 1950112160
asmSobel01 demoro: 1859535182
asmSobel11 demoro: -687698119
asmPrewitt10 demoro: 1836947486
asmPrewitt01 demoro: 1865921386
asmPrewitt11 demoro: -796757633
asmRoberts10 demoro: 1961670537
asmRoberts01 demoro: 775878759
```

```
157 asmRoberts11 demoro: -1761325656
```

6. Conclusiones

Hemos podido observar realmente la mejora en tiempo de ejecución de los algoritmos que sacan provecho del procesamiento de valores en paralelo a través de la tecnología de procesamiento de señales, en particular en aquellos casos en los que no solo optimizamos la carga de valores a los registros de 128 bits, sino en los que pudimos escribir a su vez varios bytes a memoria en una sola operación. En cuanto al uso concreto, el pasaje a datos de precisión simple no parece justificarse frente a la penalización de tiempo en la que incurre. Por lo que la aplicación de un operador de `Frei-Chen` en aplicaciones críticas respecto del tiempo no parece justificarse, al menos en nuestra escasa experiencia.
La optimización de estas operaciones, sin embargo, demostró ser más bien costosa y difícil de rastrear, presentando un dilema a considerar en lo que respecta a la decisión tiempo de ejecución versus tiempo de desarrollo.