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

5 de Octubre de 2009

Índice

1. Archivos adjuntos

Implementación

- src:
  - *bordes.c*
  - *asmSobel.asm*
  - *asmPrewitt.asm*
  - *asmRoberts.asm*
- img:
  - *lena.bmp*

Includes

- *offset.inc*
- *macros.mac*

Enunciado

- *EnunciadoTP1A.pdf*

Informe

- *tp1-a.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

Este trabajo consistió en escribir en lenguaje ensamblador distintas funciones para buscar bordes de imagen, Tanto en función de x, como en y, o ambas. La razón de escribir código en lenguaje de bajo nivel radica en que es imperativo conocer el manejo interno más básico de las instrucciones de la arquitectura del computador.

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

4. Desarrollo

Antes de mostrar los códigos de cada función explicamos algunas cuestiones generales a todos los algoritmos:

- Nos fue pedido que en todos los algoritmos sea respetada la convención C, de forma que los mismos pasos (guardado de los registros edi, esi y ebx, ajuste de la pila, etc) se encuentran al principio de cada función.
- En todas las funciones, consideramos que los parámetros (excepto los que se pedían) no debían ser modificados, y por lo tanto, los mismos los guardamos en variables locales o bien registros. En algunos casos esto es muy importante, por ejemplo, cuando nos pasan un puntero a una lista, si modificamos 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 las imágenes se guardan en memoria alineadas
- Por cuestiones de simpleza decidimos en los algoritmos también procesar la basura de la imagen ya que la libreria opencv cuando hace el resize de la imagen se limita al ancho y alto dado como parametro, y ademas esta decición no cambia en absoluto la visualización de la imagen resultante.

A continuación se exponen las funciones implementadas, junto con sus respectivos códigos en assembler y se explica a su vez los registros especiales utilizados y su funcionamiento en líneas generales.

4.1. macros.mac

Estos son las macros utilizadas en cada funcion.

```
1  ;========================
2  ; MACRO doEnter
3  ;========================
4  ; Escribe el encabezado que arma el stack frame
5  ; Entrada:
6  ;       tamanio          tamanio de memoria a reservar al entrar al proc
7  ;========================
8  %macro doEnter 0-1 0
9          push ebp
10         mov ebp, esp
11 %if %1 <> 0
12         sub esp, %1
```

```
13  %endif
14          push edi
15          push esi
16          push ebx
17  %endmacro
18  ;=============================
19  ; MACRO doLeave
20  ;=============================
21  ; Escribe la salida que restaura el stack frame previo
22  ; Entrada:
23  ;         tamanio           tamanio de memoria reservada al entrar al proc
24  ;         doRet             se marca en 1 si debe llamar a ret
25  ;=============================
26  %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
```

4.2. Recorrido de la matriz de imagen

1. **Cosas que tuvimos en cuenta**

   La matriz la recorrimos procesando la basura

   Cada pixel ocupa exactamente un byte, ya que la imagen que recive la función es en escala de grises

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

4.3. Funciones implementadas

- Intentamos reducir el acceso a memoria utilizando registros para utilizar en aquellas operaciones que se encontraban dentro de los ciclos, para evitar multiples accesos a memoria. Si bien esto llevó a tener un codigo menos entendible a simple vista, lo explicaremos en detalle.

4.3.1. Sobel

1. **Cosas que tuvimos en cuenta**

   Lo que tuvimos en cuenta si tuvimos algo.

2. **Registros y variables locales utilizadas**

   - Variables locales

     variable 1 bla.

     variable 2 bla.

   - Registros de propósito general

     eax

     ebx

     ecx es el puntero a la imagen resultado.

     edx

     esi es el puntero a la imagen de entrada.

     edi

3. **Funcionamiento**

   ACA VA EL FUNCIONAMIENTO

4. **Código**

```
1 ACA VA EL CODIGO
```

4.3.2. Prewitt

1. **Cosas que tuvimos en cuenta**

   Lo que tuvimos en cuenta si tuvimos algo.

2. **Registros y variables locales utilizadas**

   - Variables locales

     variable 1 bla.

     variable 2 bla.

   - Registros de propósito general

     eax

     ebx

     ecx es el puntero a la imagen resultado.

     edx

     esi es el puntero a la imagen de entrada.

     edi

3. **Funcionamiento**

   ACA VA EL FUNCIONAMIENTO

4. **Código**

```
1 ACA VA EL CODIGO
```

4.3.3. Roberts

1. **Cosas que tuvimos en cuenta**

   Lo que tuvimos en cuenta si tuvimos algo.

2. **Registros y variables locales utilizadas**

   - Variables locales

     variable 1 bla.

     variable 2 bla.

   - Registros de propósito general

     eax

     ebx

     ecx es el puntero a la imagen resultado.

     edx

     esi es el puntero a la imagen de entrada.

     edi

3. **Funcionamiento**

   ACA VA EL FUNCIONAMIENTO

4. **Código**

```
1 ACA VA EL CODIGO
```

5. Resultados

COMPLETAR CON RESULTADOS

6. Conclusiones

Luego de implementar este trabajo en assembler, podemos concluir que si bien nos da una gran libertad a la hora de programar (modificar directamente la memoria, trabajar con los registros directamente, etc.), la claridad del código de estos programas no es tan claro como los de lenguaje de alto nivel que manejamos *C*, *C++*, etc. Vemos esto como una desventaja ya que teniendo en cuenta que el trabajo es grupal, en ocasiones tuvimos que repartir las tareas, es decir, algunas funciones, y luego, al juntarnos y ver las implementaciones que había hecho cada uno se dificultaba entenderlas.
De todas formas, consideramos que la realización de este trabajo nos sirvió para obtener un importante conocimiento sobre el lenguaje de bajo nivel, el cual complementa entender cosas de lenguajes de más alto nivel, y consideramos además que es una muy buena práctica para ayudarnos a comprender el funcionamiento interno de un computador.