ORGANIZACIÓN DE COMPUTADORAS

INFORME
TRABAJO PRÁCTICO 0

Alumnos

93198 - Peña, Maximiliano
maxipenia@gmail.com

93665 - Poggio, Demian
demianpoggio@gmail.com

95505 - Iogha, Octavio
octaviomdq93@gmail.com

Fecha de Vencimiento

Martes 1 de Noviembre

Objetivos

El objetivo de este trabajo práctico es adaptar una función implementada en lenguaje C, al lenguaje Assembly MIPS32. Para ello, se volvieron a utilizar las herramientas ya vistas en el TP0.

Introducción Teórica

A partir de una imágen patrón del programa encargado de dibujar las regiones de Julia, se debe analizar el funcionamiento del algoritmo, e implementarlo correctamente en el lenguaje Assembly MIPS.
En la implementación del mismo, además del uso de las funciones provistas por el lenguaje assembler, también se han tenido que hacer uso de las syscalls, específicamente la llamada al sistema `"SYS_WRITE"`.

Compilación

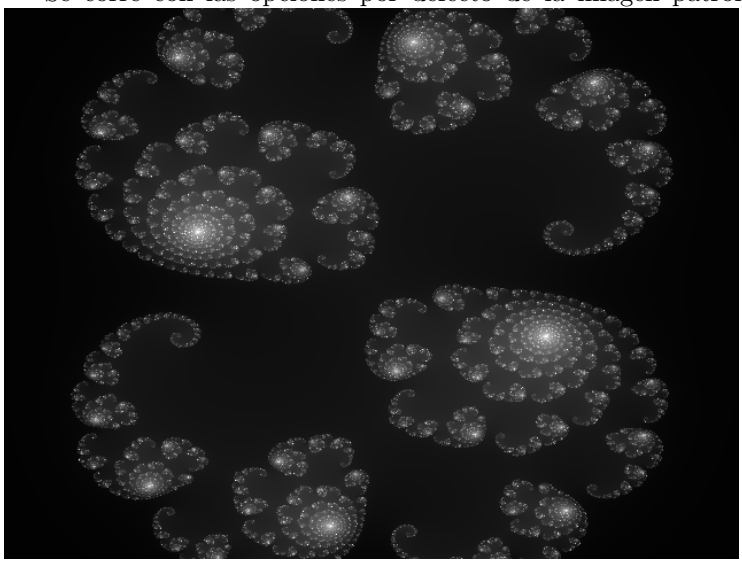
NOTA: Dependiendo de qué archivo se quiera utilizar para compilar la imágen, el Makefile.in puede contener modificaciones en la línea 6.
Suponiendo que se quiere utilizar la versión implementada en MIPS assembly, entonces la línea 6 debe ser:
`SRCS = mips32_plot.S main.c mygetopt_long.c`.
Para compilar el programa,basta con ejecutar

```
make clean
make makefiles
make
```

Corridas de prueba

Pruebas

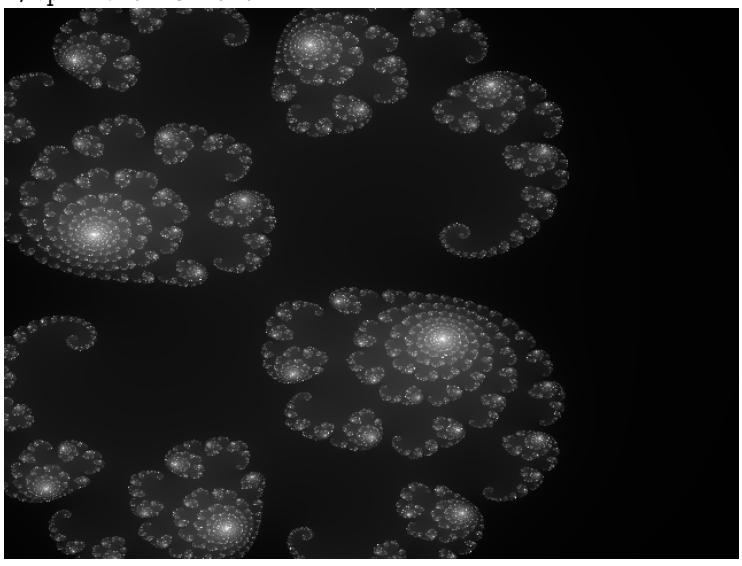
Se corre con las opciones por defecto de la imágen patrón, y se obtiene:



Lo cual coincide con la imágen esperada.

Ahora se corre con el centro corrido, precisamente:

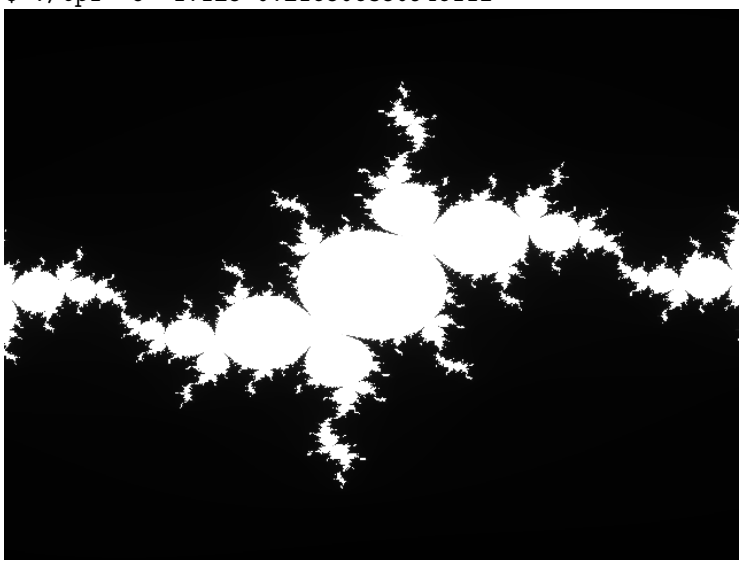
```
./tp1 -c 0.282+0.01i
```



Lo cual coincide con la imágen esperada.

Por último, se corre la siguiente línea:

```
$ ./tp1 -C -1.125-0.216506350946111i
```



Nuevamente se obtiene el resultado esperado.

Código MIPS

```
1 #include <mips/regdef.h>
2 #include <sys/syscall.h>
3 #ifndef BUF_SZ 8192
4 #define BUF_SZ 8192
5 #endif
6 #define STACK_SIZE 112
7 #define A1_STACK 108
8 #define T0_STACK 104
9 #define T1_STACK 100
10 #define T2_STACK 96
11 #define T3_STACK 92
12 #define T4_STACK 88
13 #define T5_STACK 84
14 #define T6_STACK 80
15 #define T7_STACK 76
16 #define F30_STACK 72
17 #define F28_STACK 64
18 #define F26_STACK 56
19 #define F24_STACK 48
20 #define F22_STACK 40
21 #define F20_STACK 32
22 #define RA_STACK 24
23 #define FP_STACK 20
```

```
24 #define GP_STACK 16
25 #define A0_STACK (STACK_SIZE)
26
27
28   .data
29   .align 2
30 enter1:    .asciiz "\n"
31 header:    .asciiz "P2\n"
32   .align 2
33 buffer: .space BUF_SZ                # Espacio para
     buffer.
34   .align 2
35 buffer2: .space 32
36   .text
37   .abicalls
38   .align 2
39   .global mips32_plot
40   .ent mips32_plot
41 mips32_plot:
42   .frame $fp, STACK_SIZE, ra
43   .set noreorder
44   .cpload t9
45   .set reorder
46
47   subu sp, sp, STACK_SIZE
48
49   .cprestore GP_STACK
50
51   sw ra, RA_STACK(sp)
52   sw $fp, FP_STACK(sp)
53   sw gp, GP_STACK(sp)
54   s.d $f20, F20_STACK(sp) # Como voy a usar los
       registros f20-f30
55   s.d $f22, F22_STACK(sp) # los guardo, ya que son de
       tipo save
56   s.d $f24, F24_STACK(sp) # que se espera que
       conservemos su valor.
57   s.d $f26, F26_STACK(sp) # Al finalizar la ejecucion,
       traeremos
58   s.d $f28, F28_STACK(sp) # los valores originales a los
       registros
59   s.d $f30, F30_STACK(sp) # nuevamente.
60   move $fp, sp
61
62   move t0, a0        # t0 puntero a parms, recibido por
       parametro
63   sw t0, A0_STACK($fp)  # Salvamos los punteros
64
65   lw t5, 32(t0)      # parms->x_res
66   lw t6, 36(t0)      # parms->y_res
```

```
67   lw t4, 40(t0)      # parms->shades
68   lw t7, 44(t0)      # parms->fd
69
70   l.s $f10, 24(t0)      # cpr = parms->cp_re
71   l.s $f26, 28(t0)      # cpi = parms->cp_im
72   l.s $f12, 0(t0)     # parms->UL_re
73   l.s $f28, 4(t0)     # parms->UL_im
74   l.s $f14, 16(t0)      # parms->d_re
75   l.s $f30, 20(t0)      # parms->d_im
76
77
78
79
80   ### Aca podria guardar el header PGM en el buffer...
81   ### Para el buffer necesitaria un contador
82   ### que me indique cuan lleno esta.
83   ### Para el contador se puede usar t0 que ya no se
84   ### usa mas.
85
86 #Imprimir Header
87   la a0,header
88   la a1,buffer
89   lb t0,(a0)
90   sb t0,(a1)     # guardado P
91   lb t0,1(a0)
92   sb t0,1(a1)    # guardado 2
93   lb t0,2(a0)
94   sb t0,2(a1)    # guardado \n
95   addu a1,a1,3
96
97   move a0,t5
98     jal itoa     # guardado x_res
99
100  move a1,v0
101  move a0,t6
102  jal itoa     # guardado y_res
103
104  move a1,v0
105  move a0,t4
106  jal itoa     # guardado shades
107
108  move t0,v0     # se mantiene la direccion del puntero a
         buffer
109
110 #Comienzo del Programa
111
112  li t1, 0         # y = 0
113  mov.s $f4, $f28      # ci = parms->UL_im
114 forUno:
115  li t2, 0         # x = 0
```

```
116   mov.s $f20, $f12      # cr = parms->UL_re
117 forDos:
118   mov.s $f6, $f20     # zr = cr
119   mov.s $f22, $f4     # zi = ci
120
121   li t3, 0        # c = 0
122 forTres:
123   mul.s $f8, $f6, $f6   # sr = zr * zr
124   mul.s $f16, $f22, $f22    # aux = zi * zi
125   add.s $f18, $f8, $f16   # absz = zr*zr + zi*zi
126   li.s $f0,4
127   sub.s $f18, $f18, $f0   # absz -= 4
128   li.s $f0,0
129   c.le.s $f0,$f18
130   bc1t finForTres # break
131
132   sub.s $f8, $f8, $f16    # sr -= aux
133   add.s $f8, $f8, $f10    # sr += cpr
134
135   mul.s $f24, $f6, $f22   # si = zr * zi
136   li.s $f0,2
137   mul.s $f24, $f24, $f0   # si *= 2
138   add.s $f24, $f24, $f26    # si += cpi
139
140   mov.s $f6, $f8      # zr = sr
141   mov.s $f22, $f24      # zi = si
142
143   addu t3, t3, 1      # c++
144   subu t8, t4, t3     # Condicion forTres
145   bgtz t8, forTres    #
146 finForTres:
147
148   ### Se debe guardar el valor de c (t3) en el buffer.
149
150   move a0,t3
151   move a1,t0
152   jal save_reg    # se guardan registros temporales
153   jal itoa      # se imprime en el buffer
154   jal load_reg    # se cargan registros guardados
155   move t0,v0
156
157   ### Checkeo error de escritura tambien..
158   ### Nota parms->fp esta en t7
159
160   addu t2, t2, 1      # x++
161   add.s $f20, $f20, $f14    # cr += parms->d_re
162   subu t8, t5, t2     # Condicion forDos
163   bgtz t8, forDos     #
164
165   add t1, t1, 1     # y++
```

```
166   sub.s $f4, $f4, $f30    # ci -= parms->d_im
167   subu t8, t6, t1      # Condicion forUno
168   bgtz t8, forUno      #
169
170   ### Se recorrio todo, se terminaron los fors.
171   ### Aca hay que revisar si hay algo en el buffer y
          escribirlo.
172
173   lw t3,buffer
174   beq t3,t0,bufferVacio # se chequea si el buffer esta
          vacio. de no ser asi, se imprime
175   li  v0, SYS_write
176   li  a2,BUF_SZ   # El tamano del buffer
177   la  a1,buffer   # Direccion del buffer
178   move  a0,t7     # File descriptor guardado en t7
179   syscall
180
181 bufferVacio:
182
183   lw ra, RA_STACK(sp)
184   lw $fp, FP_STACK(sp)
185   lw gp, GP_STACK(sp)
186   l.d $f20, F20_STACK(sp) # Restauro valores originales
          de
187   l.d $f22, F22_STACK(sp) # los f20-f30
188   l.d $f24, F24_STACK(sp)
189   l.d $f26, F26_STACK(sp)
190   l.d $f28, F28_STACK(sp)
191   l.d $f30, F30_STACK(sp)
192
193   addu sp, sp, STACK_SIZE
194   j ra
195
196
197 save_reg:
198   sw t0,T0_STACK($fp)
199   sw t1,T1_STACK($fp)
200   sw t2,T2_STACK($fp)
201   sw t3,T3_STACK($fp)
202   sw t4,T4_STACK($fp)
203   sw t5,T5_STACK($fp)
204   sw t6,T6_STACK($fp)
205   sw t7,T7_STACK($fp)
206   j ra
207
208 load_reg:
209   lw t0,T0_STACK($fp)
210   lw t1,T1_STACK($fp)
211   lw t2,T2_STACK($fp)
212   lw t3,T3_STACK($fp)
```

```
213   lw t4,T4_STACK($fp)
214   lw t5,T5_STACK($fp)
215   lw t6,T6_STACK($fp)
216   lw t7,T7_STACK($fp)
217   j ra
218
219
220 # Funcion integer to ascii
221 # Transfiere un integer guardado en a0 a una
222 # direcion en el buffer guardada en a1
223
224 itoa:         # a0: numero a guardar, a1: direccion de
      guardado
225     #addu  v0, a1, 8    # retorna la siguiente direccion
226     #addu  a1, a1, 6    # posiciono sobre el final
227
228     move v0,a1
229     la a1,buffer2
230     addu a1,a1,30
231     li   t0, '\n'
232     sb   t0, 1(a1)      # se coloca \n al final del
          string
233     li   t0,'0'
234     li   t1,0
235     sb   t0, (a1)     # inicia el string en 0
236     slt  t1, a0, zero   # setea t1 si a0 es negativo
237     li   t2, 10
238     beq  a0, zero, iend # termina si a0=0
239 loop:
240     divu  a0, t2        # a /= 10
241     mflo a0
242     mfhi t3             # se obtiene el resto
243     addu  t3, t3, t0  # se convierte a ascii
244     sb   t3, (a1)     # se almacena en a1
245     subu  a1, a1, 1    # dec. buf ptr
246     bne  a0, zero, loop  # if not zero, loop
247     addu a1, a1, 1    # adjust buf ptr
248 iend:
249     beq  t1, 0, nolz  # was < 0?
250     subu a1, a1, 1
251     li   t0, '-'
252     sb   t0, (a1)
253 nolz:
254   la t0,buffer2
255   addu t0,t0,32
256   la t2,buffer
257   addu t2,t2,BUF_SZ
258 loop2:
259   lb t1,(a1)
260   sb t1,(v0)
```

```
261   addu a1,a1,1
262   addu v0,v0,1
263   beq v0,t2,bufferLleno
264 retorno:
265   bne a1,t0,loop2
266     j   ra            # of the string
267
268 bufferLleno:
269   sw a1,A1_STACK($fp)
270   li  v0, SYS_write
271   li  a2,BUF_SZ   # El tamano del buffer
272   la  a1,buffer   # Direccion del buffer
273   move  a0,t7     # File descriptor guardado en t7
274   syscall
275   la v0,buffer
276   lw a1,A1_STACK($fp)
277   la t0,buffer2
278   addu t0,t0,32
279   la t2,buffer
280   addu t2,t2,BUF_SZ
281   j retorno
282
283   .end mips32_plot
```

mips32_plot.S

Conclusión

En este trabajo práctico, se han podido utilizar todas las herramientas inicialmente presentadas en el TP0, aplicandolas a la implementación de una función hecha en C, en lenguaje assembler MIPS32.
Por otro lado, se ha podido practicar el uso de la ABI dada por la cátedra, para el desarrollo de las diferentes funciones implementadas en el lenguaje assembly.
Por último, se ha podido interactuar con el sistema operativo netBSD, al hacer uso de las llamadas al sistema, o "syscalls", para escribir un buffer de tamaño predeterminado al archivo resultante.

Enunciado

Adjundto a partir de la siguiente hoja.

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 1: conjunto de instrucciones MIPS
2^o cuatrimestre de 2016

`$Date: 2016/10/02 22:23:34 $`

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto en la sección 4.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 5, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Descripción

Se trata de un modificar un programa que dibuje el conjunto de Julia y sus vecindades introducido en el *TP0* [1], en el cual la lógica de cómputo del fractal deberá tener soporte nativo para MIPS32 sobre NetBSD/pmax.

El código fuente con la versión inicial del programa, se encuentra disponible en [2]. El mismo deberá ser considerado como punto de partida de todas las implementaciones.

4.1. Soporte para MIPS

El entregable producido en este trabajo deberá implementar la lógica de cómputo del fractal en assembly MIPS32, con soporte nativo para NetBSD/pmax.

Para ello, cada grupo deberá tomar el código fuente de base para este TP, [2], y reescribir la función `mips32_plot()` sin cambiar su API. Esta función está ubicada en el archivo `mips32_plot.c`.

4.2. Casos de prueba

El informe trabajo práctico deberá incluir una sección dedicada a verificar el funcionamiento del código implementado. Para ello, será necesario escribir pruebas orientadas a probar el programa completo, ejercitando los casos más comunes de funcionamiento, los casos de borde, y también casos de error.

4.3. Compilación

El código fuente provisto por la cátedra provee los makefiles necesarios para compilar el ejecutable a partir de la versión en C con el archivo `mips32_plot.c`. Para poder compilar el código desarrollado deberán cambiar la definición en el archivo `Makefile.in` la línea número 6:

```
SRCS = mips32_plot.c main.c mygetopt_long.c
```

por

```
SRCS = mips32_plot.S main.c mygetopt_long.c
```

Luego deberán invocar la siguiente secuencia de comandos para limpiar los archivos temporales y generar los nuevos Makefiles:

```
$ make clean
$ make makefiles
$ make
```

4.4. Detalles de la implementación

Para optimizar los accesos a las llamadas a servicio del sistema (`syscalls`), deben utilizar un buffer de `BUF_SZ` bytes para escribir los datos de salida para luego ser enviados al archivo de salida. El tamaño `BUF_SZ` debe ser configurable, en tiempo de compilación, mediante un `#define`.

```
#ifndef BUF_SZ 8192
#define BUF_SZ 8192
#endif
```

Como podemos ver arriba, el valor por defecto de este parámetro es `8192` bytes.

5. Informe

El informe, a entregar en formarto impreso y digital[1] deberá incluir:

- Documentación relevante al diseño e implementación del código esarrollado para adaptar el programa. Incluír el diagrama de *stack frame* de las funciones implementadas en MIPS32.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente. Especificar modificaciones realizadas a los archivos provistos por la cátedra si es que los hubo.
- Las corridas de prueba, con los comentarios pertinentes.[2]
- El código fuente, en lenguaje C (y MIPS32 donde corresponda)
- Este enunciado.

[1] En CD, DVD o memoria flash.

[2] Las pruebas provistas deben ejecutarse correctamente en NetBSD sobre MIPS32 sin modificación alguna.

6. Fecha de entrega

La fecha de vencimiento será el Martes 01/11.

Referencias

[1] Trabajo Práctico 0, 2do cuatrimestre de 2016.
`https://groups.yahoo.com/neo/groups/orga-comp/files/TPs/tp0-2016-2q.pdf`

[2] Código fuente con el esqueleto del trabajo práctico.
`https://drive.google.com/open?id=0B93s6e6NY_j1TFV2TFBqbUNKZ3M`