66.20 Organización de Computadoras
Trabajo Práctico 0:
Infraestructura básica

Burdet Rodrigo, *Padrón Nro. 93440*
rodrigoburdet@gmail.com

Colangelo Federico, *Padrón Nro. 89869*
federico.colangelo@semperti.com

Manzano Matias, *Padrón Nro. 83425*
matsebman@gmail.com

2do. Cuatrimestre de 2014
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

21 de octubre de 2014

1. Objetivos

Familiarizarse con la programación en assembly y el concepto de ABI , implementando un programa (y su correspondiente documentación) que resuelva el problema piloto que presentaremos más abajo.

2. Resumen

En el presente trabajo, se implementó un algoritmo que permite graficar los conjuntos de Mandelbrot dados ciertos parámetros para permitir centrarnos en una región en particular de dicho conjunto. El programa fue realizado en 2 etapas. La primera en c, donde se crea el marco para el dibujo. La segunda parte en assembly MIPS, encargada de dibujar fractales a un archivo. El programa en c linkea contra el código assembly MIPS para lograr así más performance.

3. Desarrollo

3.1. Paso 1: Configuración de Entorno de Desarrollo

El primer paso fue configurar el entorno de desarrollo, de acuerdo a la guía facilitada por la cátedra. Trabajamos con distribuciones Linux y con el GxEmul proporcionado por la cátedra, emulando un sistema NetBSD.

3.2. Paso 2: Implementación del programa

El programa debe ejecutarse por línea de comando y la salida del mismo dependerá del valor de los argumentos con los que se lo haya invocado.

3.2.1. Ingreso de parámetros

El formato para invocar al programa es el siguiente:

```
./tp1-2014-2-bin [OPTIONS]
```

Los parámetros válidos que puede recibir el programa son los siguientes:

-o,	**–output**	`(Parámetro obligatorio. Especifica archivo de salida, - para stdout).`
-r,	**–resolution**	`(Resolución de la imagen de salida).`
-c,	**–center**	`(Centro de la imagen).`
-w,	**–width**	`(Ancho del rectángulo a dibujar).`
-H,	**–height**	`(Alto del rectángulo a dibujar).`
-v,	**–version**	`(Muestra la versión).`
-h,	**–help**	`(Muestra la ayuda).`

3.2.2. Interpretación de parámetros

Para parsear los parámetros se usó la librería de GNU getopt, en particular se usó `getopt_long` para permitir el pasaje de parámetros largos.

4. Compilación del programa

El proyecto puede ser corrido en cualquier ambiente pero se debe tener en consideración qué algoritmo se va a ejecutar ya que el código assembly está fuertemente ligado a un tipo de arquitectura. Es por esta razón que si queremos estamos sobre MIPS podremos linkear contra `mips32plot.S` pero eso no va a ser posible en x86 donde sólo podremos usar código c. Para compilar en MIPS se puede usar el comando `make` ya que el `Makefile` fue modificado para linkear el .S correspondiente. Para compilar en otro ambiente es necesario modificar el `Makefile` para que compile el código en c en vez del assembly.

5. Programa MIPS

El programa fue realizado siguiendo la ABI provista por la cátedra y generando un stack como se muestra a continuación

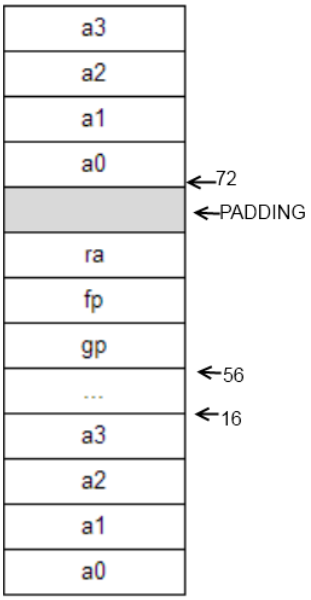

Figura 1: Stack frame generado

Se ve en la figura que LTA mide 40 bytes. El tamaño no fue arbitrario si no que se tuvieron en cuenta las variables locales que se tendrían que almacenar. Para imprimir numeros a archivos se uso un procedimiento tambien hecho en assembly llamado `write_int.S` Para debuggear se usaron llamadas de MIPS a C de la función printf en versiones para enteros y floats, las mismas se pueden ver al final del código adjunto como `miprintf` y `printff` respectivamente.

Nota: El programa en la versión actual contiene un bug que no permite la correcta impresión del brillo de los pixeles.

6. Corridas de prueba y Mediciones

7. Conclusiones

Como se enuncia en el objetivo de este trabajo práctico, aprendimos a instalar y manejar el GxEmul, a realizar transferencias de archivos en Linux, así como también compilar y ejecutar programas en el NetBSD. Por otro lado, aprendimos a manejar y escribir informes en LaTeX. De este modo, estamos preparados para que en los próximos trabajos prácticos, nos aboquemos directamente al desarrollo de los mismos.

8. Código

8.1. mips32_plot.S

```
#include <mips/regdef.h>
#include <sys/syscall.h>

#defino las posiciones de los argumentos dentro del struct paramt_t
#define UL_re 0
#define UL_im 4
#define LR_re 8
#define LR_im 12
#define d_re 16
#define d_im 20
#define x_res 24
#define y_res 28
#define shades 32
#define FD 36

#define descriptor 14



#define FRAME_SIZE  72
#define GP_POS    FRAME_SIZE-16
#define FP_POS    FRAME_SIZE-12
#define RA_POS    FRAME_SIZE-8
#define LTA_POS   16


#mips32_plot(param_t *);
  .text
  .align  2
  .extern write_int
  .globl  mips32_plot
  .ent  mips32_plot
mips32_plot:
  .frame  $fp, FRAME_SIZE, ra
  .set  noreorder
  .cpload t9
  .set  reorder
  subu  sp, sp, FRAME_SIZE
  .cprestore GP_POS
  sw  $fp, FP_POS(sp)
  sw  ra,  RA_POS(sp)
  move  $fp, sp
  sw  a0, FRAME_SIZE($fp)
  sw  a1, FRAME_SIZE+4($fp)
  sw  a2, FRAME_SIZE+8($fp)
  sw  a3, FRAME_SIZE+12($fp)

  #t0 = &parms
  lw  t0,FRAME_SIZE($fp)
  lw  t1,FD(t0)
  #t1 = FD
  lh  t1,descriptor(t1) #me muevo al campo _file mediante un short

  #imprimir cabecera
  #imprimo P2
  li  v0,SYS_write
  move  a0,t1
  la a1,msgP2
  li a2,3
  syscall
  #TODO: revisar error
```

```
62
63  #imprimo xres
64  lw  t0,FRAME_SIZE($fp)
65  lw  t1,FD(t0)
66  #t1 = FD
67  lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
68  lw  t0,x_res(t0)
69  move  a0,t0
70  move  a1,t1
71  jal write_int
72
73
74  #Imprimir fin de linea
75  lw  t0,FRAME_SIZE($fp)
76  lw  t1,FD(t0)
77  lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
78
79  #t1 = FD
80  li  v0,SYS_write
81  move  a0,t1
82  la a1,endln
83  li a2,1
84  syscall
85
86
87  #imprimo yres
88  lw  t0,FRAME_SIZE($fp)
89  lw  t1,FD(t0)
90  #t1 = FD
91  lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
92  lw  t0,y_res(t0)
93  move  a0,t0
94  move  a1,t1
95  jal write_int
96
97
98  #Imprimir fin de linea
99  lw  t0,FRAME_SIZE($fp)
100 lw  t1,FD(t0)
101 lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
102 #t1 = FD
103 li  v0,SYS_write
104 move  a0,t1
105 la a1,endln
106 li a2,1
107 syscall
108
109
110 #imprimo shades
111 lw  t0,FRAME_SIZE($fp)
112 lw  t1,FD(t0)
113 #t1 = FD
114 lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
115 lw  t0,shades(t0)
116 move  a0,t0
117 move  a1,t1
118 jal write_int
119
120 #Imprimir fin de linea
121 lw  t0,FRAME_SIZE($fp)
122 lw  t1,FD(t0)
123 lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
124 #t1 = FD
125 li  v0,SYS_write
126 move  a0,t1
127 la a1,endln
128 li a2,1
129 syscall
130
```

```
  #inicializo loop_y
  li  t0,0
  sw  t0,LTA_POS($fp)   #guardo y=0 -> 0
  lw  t0,FRAME_SIZE($fp)  #traigo parms
  l.s $f4,UL_im(t0)   #f4=UL_im
  s.s $f4,LTA_POS+4($fp)  #ci=UL_im -> 4
loop_y:

  #cargo y_res
  lw  t0,FRAME_SIZE($fp)  #traigo parms
  lw  t0,y_res(t0)  #t0 = parms->y_res
  lw  t1,LTA_POS($fp) #t1 = y
    bge t1, t0, end # if (y > y_res) {end}

  #inicializo loop_x
  li  t0,0
  sw  t0,LTA_POS+8($fp) #guardo x=0 -> 8
  lw  t0,FRAME_SIZE($fp)  #traigo parms
  l.s $f4,UL_re(t0)   #f4 = UL_re
  s.s $f4,LTA_POS+12($fp) #cr=UL_re -> 12

loop_x:
  #cargo x_res
  lw  t0,FRAME_SIZE($fp)  #traigo parm
  lw  t0,x_res(t0)    #t0 = parms->x_res
  lw  t1,LTA_POS+8($fp) #t1 = x
  bge t1,t0,inc_y     #if (x>x_rs){inc y}

  l.s $f4,LTA_POS+4($fp)  #f4 = ci
  l.s $f6,LTA_POS+12($fp)
  s.s $f4,LTA_POS+16($fp) # zi=ci -> 16
  s.s $f6,LTA_POS+20($fp) # zr=cr -> 20

  #inicializo loop_c
  li  t0,0
  sw  t0,LTA_POS+24($fp)  # c -> 24

loop_c:

  ###### ES UN ++c
    lw  t0,LTA_POS+24($fp)
  addi  t0,1
  sw  t0,LTA_POS+24($fp)

  lw  t1,FRAME_SIZE($fp)  #traigo parms
  lw  t1,shades(t1) #t1 = parms->shades
  bge t0,t1,write_out # if (c>parms->shades) {write_out}

  l.s $f4, LTA_POS+20($fp)  #f4 = zr
  l.s $f6, LTA_POS+16($fp)  #f6 = zi
  mul.s $f4, $f4, $f4
  mul.s $f6, $f6, $f6
  add.s $f4, $f4, $f6
  li.s $f6, 4.0
  c.le.s $f6, $f4 # if 4<absz then write_out
  bc1t write_out

  l.s $f4, LTA_POS+20($fp) # f4 = zr
  mul.s $f6, $f4, $f4
  mul.s $f6, $f6, $f4 # f6 = zr ^ 3
  l.s $f8, LTA_POS+16($fp)
  mul.s $f8, $f8, $f8
  mul.s $f8, $f4, $f8 # f8 = zi ^ 2 * zr
  li.s $f10, 3.0
  mul.s $f8, $f8, $f10
  sub.s $f6, $f6, $f8
  l.s $f8, LTA_POS+12($fp)
  add.s $f6, $f6, $f8
  s.s $f6, LTA_POS+28($fp)
```

```
200
201   l.s $f4,LTA_POS+20($fp) # f4 = zr
202   l.s $f6,LTA_POS+LTA_POS+16($fp) #f6 = zi
203   li.s $f8,3.0
204   mul.s $f4,$f4,$f4 #zr = zr^2
205   mul.s $f4,$f4,$f6 #f4 = zr^2*zi
206   mul.s $f4,$f4,$f8 #f4 = zr^2*zi*3
207
208   mul.s $f8,$f6,$f6 #f8 = zi*zi
209   mul.s $f6,$f8,$f6 #f6 = zi*zi*zi
210   sub.s $f4,$f4,$f6 #f4 = 3*zr^2 *zi - zi^3
211
212   l.s $f6,LTA_POS+4($fp)
213   add.s $f4, $f4,$f6
214   s.s $f4,LTA_POS+32($fp)
215
216   l.s $f4, LTA_POS+28($fp)
217   l.s $f6, LTA_POS+32($fp)
218   s.s $f4, LTA_POS+20($fp)
219   s.s $f6, LTA_POS+16($fp)
220
221   j loop_c
222
223
224 write_out:
225   lw t0, LTA_POS+24($fp)
226   lw t1, FRAME_SIZE($fp)
227   lw t1, FD(t1)
228   lh t1, descriptor(t1)
229   move a0, t0
230   move a1, t1
231   jal write_int
232
233   #Imprimir espacio
234   lw  t0,FRAME_SIZE($fp)
235   lw  t1,FD(t0)
236   lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
237   li  v0,SYS_write
238   move  a0,t1
239   la a1,space
240   li a2,1
241   syscall
242   j inc_x
243
244 inc_x:
245   lw  t0,LTA_POS+8($fp) #x
246   addi  t0,1
247   sw  t0,LTA_POS+8($fp)
248   lw  t1,FRAME_SIZE($fp)
249   l.s $f4,d_re(t1)  #f4 = parms->d_re
250   l.s $f6,LTA_POS+12($fp) #cr
251   add.s $f6,$f6,$f4
252   s.s $f6,LTA_POS+12($fp)
253   j loop_x
254
255 inc_y:
256   lw  t0,LTA_POS($fp) #y
257   addi  t0,1
258   sw  t0,LTA_POS($fp)
259   lw  t1,FRAME_SIZE($fp)
260   l.s $f4,d_im(t1)  #f4 = parms->d_im
261   l.s $f6,LTA_POS+4($fp)  #ci
262   sub.s $f6,$f6,$f4
263   s.s $f6,LTA_POS+4($fp)
264   j loop_y
265
266
267   #PRINTF FUNCIONANDO, imprime t0!!
268 miprintf:
```

```
  la  a0,print
  move  a1,t0
  lw  t9,%call16(printf)(gp)
  jalr  t9

  #PRINTF DE FLOTS FUNCIONANDO, CARGAR FLOAT CON l.s en $f0
printff:
  cvt.d.s $f0,$f0
  la  a0,print_float
  mfc1  a2,$f0
  mfc1  a3,$f1
  lw  t9,%call16(printf)(gp)
  jalr  t9

end:
  lw  ra,RA_POS($fp)
  lw  gp,GP_POS($fp)
  lw  $fp,FP_POS($fp)
  addu  sp,sp,FRAME_SIZE
  jr  ra

  .end  mips32_plot


  .rdata
endln:
  .asciiz "\n"
msgP2:
  .asciiz "P2\n"
space:
  .asciiz " "


print:
  .asciiz "valor : %d \n"
print_float:
  .asciiz "valor float  : %f \n"
```

8.2. write_int.S

```
/* Funcion para escribir un entero a un archivo
// PARAMETROS
// a0: entero a escribir
// a1: file descriptor
// Valor de retorno: numeros de bytes escritos
*/

#include <mips/regdef.h>
#include <sys/syscall.h>
#define STDERR 2
#define FRAME_SIZE 40
#define GP_POS 24
#define TEMP_CHAR 16
#define ASCII_OFFSET 48
#define ERROR_MSG_LENGTH 19
.text
.align 2
.ent write_int
.global write_int

write_int:
### Inicializo el frame y guardo los registros correspondientes ###
.frame $fp, FRAME_SIZE, ra
.set noreorder
.cpload t9
.set reorder
subu sp, sp, FRAME_SIZE
.cprestore GP_POS
sw $fp, 28(sp)
sw ra, 32(sp)
move $fp, sp
sw a0, FRAME_SIZE($fp)
sw a1, FRAME_SIZE+4($fp)

### Comienzo de la funcion ###

### Analizo si el entero a escribir es zero ###
lw t0, FRAME_SIZE($fp)
bnez t0, no_zero
sw t0, FRAME_SIZE($fp)
addu t0, t0, zero
sw t0, TEMP_CHAR($fp)
j write

/* Hago a0 / 10 para obtener el ultimo digito del numero a escribir
// Este digito va a estar en el resto de la division
// Si el cociente es distinto de cero, debo llamar recursivamente a la funcion
// ya que el primer numero a escribir es el de mas significativo
*/
no_zero:
lw t0, FRAME_SIZE($fp)
li t1, 10
div t0, t1
sw t0, FRAME_SIZE($fp)
mflo t0 # cociente
mfhi t1 # resto
sw t1, TEMP_CHAR($fp)
beqz t0, write

### Preparo llamada recursiva ###
move a0, t0
lw a1, FRAME_SIZE+4($fp)
la t9, write_int
jal write_int
```

```
65
66 ### Escribo el caracter al archivo indicado ###
67 write:
68 lw t0, TEMP_CHAR($fp)
69 addu t0, t0, ASCII_OFFSET
70 sw t0, TEMP_CHAR($fp)
71 lw a0, FRAME_SIZE+4($fp)
72 la a1, TEMP_CHAR($fp)
73 li a2, 1
74 li v0, SYS_write
75 syscall
76
77 ### Chequeo error ###
78 bgez v0, end
79 li a0, STDERR
80 la a1, error_msg
81 li a2, ERROR_MSG_LENGTH
82 li v0, SYS_write
83 syscall
84
85 ### Rearmo el stack ###
86 end:
87 lw ra, 32($fp)
88 lw gp, 24($fp)
89 lw $fp, 28($fp)
90 addu sp, sp, FRAME_SIZE
91 jr ra
92
93 .end write_int
94
95 .data
96 error_msg: .asciiz "Error de escritura\n"
```