6620. Organización de Computadoras
Trabajo Práctico 1:
Conjunto de instrucciones MIPS

Riesgo, Daniela, *Padrón Nro. 95557*
danielap.riesgo@gmail.com

Martin, Débora, *Padrón Nro. 90934*
debbie1mes.world@gmail.com

Constantino, Guillermo, *Padrón Nro. 89776*
guilleconstantino@gmail.com

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

Resumen

El presente trabajo tiene como objetivo familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa escrito en C que resuelva el problema del conjunto de Multibrot de orden 3, ocupándose de la función que calcula la velocidad de escape e imprime en el archivo de la salida lo correspondiente.

Índice

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 1: conjunto de instrucciones MIPS
2^o cuatrimestre de 2014

`$Date: 2014/10/05 18:12:50 $`

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 Multibrot de orden 3 [1] [2] [3] y sus vecindades, en el cual la lógica de de cómputo del fractal deberá tener soporte nativo para NetBSD/pmax.

El código fuente con la versión inicial del programa, se encuentra disponible en [4]. El mismo deberá ser considerado como punto de partida de todas las implementaciones.

4.1. Programa

El programa recibe, por línea de comando, la descripción de la región del plano complejo y las características del archivo imagen a generar. No interactúa con el usuario, ya que no se trata de un programa interactivo, sino más bien de una herramienta de procesamiento *batch*. Al

finalizar la ejecución, y volver al sistema operativo, el programa habrá dibujado el fractal en el archivo de salida.

El formato gráfico a usar es PGM o *portable gray map* [5], un formato simple para describir imágenes digitales monocromáticas.

4.2. Algoritmo

.

El algoritmo básico es simple: para algunos puntos c de la región del plano que estamos procesando haremos un cálculo repetitivo. Terminado el cálculo, asignamos el nivel de intensidad del pixel en base a la condición de corte de ese cálculo.

El color de cada punto representa la "velocidad de escape" asociada con ese número complejo: blanco para aquellos puntos que pertenecen al conjunto (y por ende la "cuenta" permanece acotada), y tonos gradualmente más oscuros para los puntos divergentes, que no pertenezcan al conjunto.

Más específicamente: para cada pixel de la pantalla, tomaremos su punto medio, expresado en coordenadas complejas, $c = c_{re} + c_{im}i$. A continuación, iteramos sobre $f_{i+1}(c) = f_i(c)^3 + c$, con $f_0(c) = c$. Cortamos la iteración cuando $|f_i(c)| > 2$, o después de `N` iteraciones.

En pseudo código:

```
para cada pixel $p {
        $f = $c = complejo asociado a $p;
        for ($i = 0; $i < $N - 1; ++$i) {
                if (abs($f) > 2)
                        break;
                $f = $f * $f * $f + $c;
        }
        dibujar el punto p con brillo $i;
}
```

Así tendremos, al finalizar, una representación visual de la cantidad de ciclos de cómputo realizados hasta alcanzar la condición de escape (ver figura 1).

4.3. Interfaz

A fin de facilitar el intercambio de código *ad-hoc*, normalizaremos algunas de las opciones que deberán ser provistas por el programa:

- `-r`, o `--resolution`, permite cambiar la resolución de la imagen generada. El valor por defecto será de 640x480 puntos.
- `-c`, o `--center`, para especificar el punto central de la porción del plano complejo dibujada, expresado en forma binómica (i.e. $a + bi$). Por defecto usaremos `-0.60 + 0.25i`.
- `-w`, o `--width`, especifica el ancho del rectángulo que contiene la región del plano complejo que estamos por dibujar. Valor por defecto: `0.1`.
- `-H`, o `--height`, sirve, en forma similar, para especificar el alto del rectángulo a dibujar. Valor por defecto: `0.1`.
- `-o`, o `--output`, permite colocar la imagen de salida, (en formato PGM [5]) en el archivo pasado como argumento; o por salida estándar `-stdout-` si el argumento es "-".

4.4. Soporte para MIPS

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

Para ello, cada grupo deberá tomar el código fuente de base para este TP, [4], y reescribir la función `mips32_plot()` sin cambiar su API. Esta función está ubicada en el archivo `mips32_plot.c`.

4.5. 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.

Incluimos en el apéndice **??** algunos ejemplos de casos de interés, orientados a ejercitar algunos errores y condiciones de borde.

4.6. Ejemplos

Generamos un dibujo usando los valores por defecto, barriendo la región rectangular del plano comprendida entre los vértices `-0.65 + 0.30i` y `-0.55 + 0.20i`.

```
$ tp1-2014-2-bin -o uno.pgm
```

La figura 1 muestra la imagen `uno.pgm`.

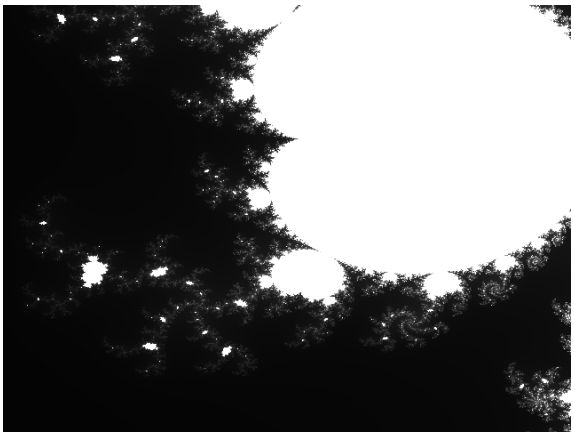


Figura 1: Región barrida por defecto.

A continuación, hacemos *zoom* sobre la región centrada en `-0.6 + 0.6i`, usando un rectángulo de `0.05` unidades de lado. El resultado podemos observarlo en la figura 2.

```
$ tp1-2014-2-bin --center -0.6+0.6i --width 0.05 --height 0.05 -o dos.pgm
```

5. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa.

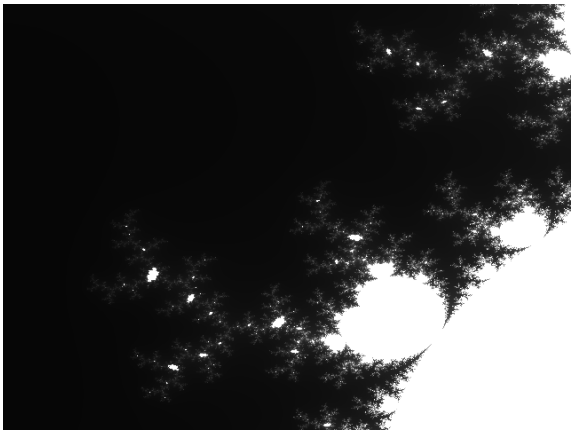


Figura 2: Región comprendida entre `-0.625 + 0.625i` y `-0.575 + 0.575i`.

- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C.
- Este enunciado.

6. Fecha de entrega

La última fecha de entrega y presentación sería el martes 21/10.

Referencias

[1] `http://en.wikipedia.org/wiki/Mandelbrot_set` (Wikipedia).

[2] Introduction to the Mandelbrot Set.
`http://www.olympus.net/personal/dewey/mandelbrot.html`.

[3] Smooth shading for the Mandelbrot exterior.
`http://linas.org/art-gallery/escape/smooth.html`. Linas Vepstas. October, 1997.

[4] Código fuente con el esqueleto del trabajo práctico.
`http://www.fiuba7504.com.ar/tp1-2014-2-src.tar.gz`.

[5] PGM format specification.
`http://netpbm.sourceforge.net/doc/pgm.html`.

2. Introducción

Al comenzar a utilizar nuevas herramientas, en cualquier ámbito, es necesaria una breve introducción al funcionamiento de las mismas: tener una noción de las prestaciones que ofrecen, asi también como de sus limitaciones.
En este trabajo práctico continuamos con las nuevas herramientas, esta vez con el objetivo de familiarizarse con el conjunto de instrucciones de MIPS32 a partir de una arguitectura NetBSD/pmax. De esta forma se pueden reforzar conocimientos como el alineamiento de datos y descubrir las instrucciones y pseudo-instrucciones que MIPS ofrece para trabajar. Otro aspecto importante del trabajo es la necesidad de trabajar con registros de punto flotante y entender cómo los registros están en procesadores separados y tienen otras instrucciones propias.

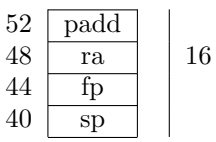
3. Programa a implementar

Se trata de, teniendo el programa en C como fuente, implementar una de las funciones, aquella que se encarga de dibujar el conjunto de Multibrot de orden 3 y sus vecindades, en lenguaje Assembly para MIPS32 en NetBSD/pmax. La misma recibirá por parámetro una estructura con la información definida en la línea de comando con la que se ejecuta el programa; información como el centro del plano complejo, la resolución deseada, entre otras. Al finalizar la ejecución y volver al programa, la función habrá escrito en el archivo de salida los parámetros necesarios y el brillo correspondiente a cada pixel de forma que se genere una imagen con el fractal buscado. El formato gráfico a usar es PGM o portable gray map, un formato simple para describir imágenes a digitales monocromáticas.

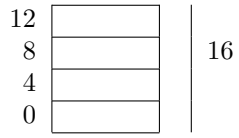
4. Explicación de la Implementación

Este trabajo presentó varios problemas a resolver. Por un lado, la escritura en un archivo. Se aprendió que la estructura FILE hace referencia a un descriptor de archivo único e intrínseco a cada archivo, necesario para escribir en el archivo. Este número ocupa menos de una palabra y entonces debió alinearse usando la instrucción proveída por MIPS para cargar media palabra. También relacionado a la escritura, un desafío fue convertir números contenidos en registros en cadenas de caracteres. Para esto se usó como guía un ejercicio similar hecho en clase. Por otro lado, se resolvieron las cuentas propiamente dicha manejando registros e instrucciones. Finalmente se tuvo que manejar tanto los registros y su contenido como el frame de la función, respetando las convenciones ABI y descifrando los tamaños de las distintas secciones del frame según cuánta información era necesaria descargar en el frame. Finalmente el frame quedó distribuido de la siguiente forma:

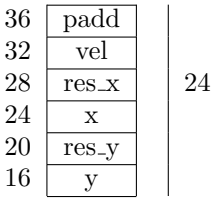
- **SRA**

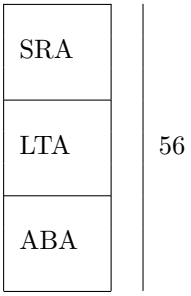


- **ABA**



- **LTA**





5. Código assembly

Para generar el ejecutable del programa se utilizó el makefile proveído de forma que se tome el código de $mips32_plot.S$ en vez de $mips32_plot.c$. Para tratar la impresión de strings, se decidió copiar el código necesario cada vez que se quería imprimir porque no ocupaba más de 8 líneas de código. A diferencia de la transformación necesaria de número a cadena que se requería en varias oportunidades que levaba bastantes líneas más. Se trató de usar etiquetas que identifiquen las partes de la función, constantes que expliciten la interpretación de un simple número y comentarios que expliquen el significado de las instrucciones usadas. También se trató de minimizar la transferencia de datos y por lo tanto se trató de manejar el procesamiento únicamente con los registros temporales, sin dar uso no necesario a la LTA.

6. Corridas de prueba

Como en este trabajo sólo se debía programar la parte de dibujo del fractal, los casos de prueba no testean la interfaz. Simplemente hicimos uso de las dos imágenes proveídas para saber cómo debía resultar la imagen y probar el programa.

1. Generamos la imagen por defecto, es decir, barriendo el rectángulo de plano complejo desde -0.65 + 0.30i hasta -0.55 + 0.20i:

```
> tp1-2014-2-bin -o uno.pgm
```

2. En este caso generamos una imagen que comprenda desde -0.625 + 0.625i hasta -0.575 + 0.575i:

```
> tp1-2014-2-bin --center -0.6+0.6i --width 0.05 --height 0.05 -o dos.pgm
```

7. Código fuente C

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

.text
.align 2

.global mips32_plot
.ent mips32_plot

mips32_plot:

FRAME_SPACE = 56
.frame $fp, FRAME_SPACE, ra
subu sp, sp, FRAME_SPACE # Pongo el stack pointer al final de mi frame.
.cprestore 40 # El gp en 40(sp)
sw $fp, 44(sp)
sw ra, 48(sp)

#Pongo el frame pointer al final de mi frame.
or $fp, sp, zero

# Información de datos
LARGO_NUM = 1
LARGO_P2 = 3
LARGO_ENTER = 1
LARGO_IO_ERROR = 11

#Offset dentro de la estructura param.
OFF_UPPER_LEFT_REAL = 0 #float: 1 registro f
OFF_UPPER_LEFT_IMAG = 4  #float: 1 registro f
```

```
OFF_LOWER_RIGHT_REAL = 8  #float: 1 registro f
OFF_LOWER_RIGHT_IMAG = 12  #float: 1 registro f
OFF_SCALE_REAL = 16  #float: 1 registro f
OFF_SCALE_IMAG = 20  #float: 1 registro f
OFF_RES_REAL = 24  #size_t = unsigned: 1 registros
OFF_RES_IMAG = 28  #size_t = unsigned: 1 registros
OFF_SHADES = 32  #size_t = unsigned: 1 registros
OFF_OUTPUT_FILE = 36  #puntero: 1 registro
#Offset del numero de archivo dentro de la estructura _sFile
OFF_FILE_NUMBER = 14  #short int: 1/2 registro


sw a0, FRAME_SPACE($fp)

l.s $f0, OFF_UPPER_LEFT_IMAG(a0) # F0 para upper_left_im
l.s $f2, OFF_SCALE_IMAG(a0) # F2 para d_im

l.s $f6, OFF_UPPER_LEFT_REAL(a0) # F6 para upper_left_re
l.s $f8, OFF_SCALE_REAL(a0) # F8 para d_re

lw t6, OFF_OUTPUT_FILE(a0)
lh t6, OFF_FILE_NUMBER(t6) # T6 para el file descriptor.

imprimir_p2:
li v0, SYS_write
move a0, t6
la a1, p2
li a2, LARGO_P2
syscall
# Si me devolvió un número no nulo en a3, hubo error.
bne a3, zero, imprimir_io_error
   blt v0, a2, imprimir_io_error
lw a0, FRAME_SPACE($fp) #recargo a0 con la estructura porque en el
# imprimir se pisa

imprimir_res_x:
lw t8, OFF_RES_REAL(a0)
la t9, imprimir_res_y
j imprimir_numero
lw a0, FRAME_SPACE($fp)

imprimir_res_y:
lw t8, OFF_RES_IMAG(a0)
la t9, imprimir_max_gray
j imprimir_numero
lw a0, FRAME_SPACE($fp)

imprimir_max_gray:
lw t8, OFF_SHADES(a0)
la t9, empezar
```

```
j imprimir_numero
lw a0, FRAME_SPACE($fp)

empezar:
li t0, 0  # T0 es y
lw t1, OFF_RES_IMAG(a0) # T1 res_y
mov.s $f4, $f0  # F4 copia del up_im para ir modificando (ci)
li t2, 0  # T2 para x
lw t3, OFF_RES_REAL(a0) # T3 para res_x
mov.s $f10, $f6  # F10 copia del up_re (cr)
li t4, 0  # T4 para la velocidad de escape.
lw t5, OFF_SHADES(a0) # T5 para el máximo de grises

loop_vertical:
beq t0, t1, terminar # Si y llegó a ser el último pixel, termino.
loop_horizontal:
beq t2, t3, aumento_loop_vert

mov.s $f12, $f4 # F12 guarda la copia del ci (zi)
mov.s $f14, $f10 # F14 guarda la copia del cr (zr)

loop_mandelbrot:
beq t4, t5, imprimir_brillo

# Calculo el módulo
mul.s $f16, $f12, $f12 # F16 guarda zi*zi
mul.s $f18, $f14, $f14 # F18 guarda zr*zr
add.s $f20, $f16, $f18 # F20 guarda zi*zi+zr*zr
li.s $f22, 4.0 # F22 guarda 4
c.lt.s $f22, $f20 # Si F22<F20 pone el flag en true.
bclt imprimir_brillo # Va a imprimir brillo si el flag es false, es decir,
# si el módulo es mayor a 4.

mul.s $f20, $f12, $f16 # F20 guarda zi*zi*zi
mul.s $f22, $f14, $f18 # F22 guarda zr*zr*zr
mul.s $f24, $f16, $f14 # F24 guarda zi*zi*zr
mul.s $f26, $f18, $f12 # F26 guarda zr*zr*zi
li.s $f28, 3.0 # F28 guarda 3
mul.s $f24, $f28, $f24 # F12 guarda 3*zi*zi*zr
mul.s $f26, $f28, $f26 # F26 guarda 3*zr*zr*zi
sub.s $f22, $f22, $f24 # F22 guarda zr*zr*zr - 3*zi*zi*zr
add.s $f22, $f22, $f4 # F22 guarda sr
sub.s $f26, $f26, $f20 # F26 guarda 3*zr*zr*zi - zi*zi*zi
add.s $f26, $f26, $f10 # F26 guarda si
mov.s $f12, $f26
mov.s $f14, $f22

add t4, t4, 1 # Aumento la velocidad de escape en 1
j loop_mandelbrot
```

```
imprimir_brillo:
OFF_FRAME_Y = 16
OFF_FRAME_RES_Y = 20
OFF_FRAME_X = 24
OFF_FRAME_RES_X = 28
OFF_FRAME_VEL = 32
guardar:
sw t0, OFF_FRAME_Y($fp)
sw t1, OFF_FRAME_RES_Y($fp)
sw t2, OFF_FRAME_X($fp)
sw t3, OFF_FRAME_RES_X($fp)
sw t4, OFF_FRAME_VEL($fp)

move t8, t4
la t9, recargar
j imprimir_numero

recargar:
lw t0, OFF_FRAME_Y($fp)
lw t1, OFF_FRAME_RES_Y($fp)
lw t2, OFF_FRAME_X($fp)
lw t3, OFF_FRAME_RES_X($fp)
lw t4, OFF_FRAME_VEL($fp)

aumento_loop_hor:
add t2, t2, 1  # Aumento x en 1.
add.s $f10, $f10, $f8  # Aumento cr según la escala.
j loop_horizontal
aumento_loop_vert:
add t0, t0, 1  # Aumento y en 1.
sub.s $f4, $f4, $f2  # A ci le resto según la escala.
j loop_vertical


terminar:
lw ra, 32($fp)
lw gp, 24($fp)
lw $fp, 28($fp)
addu sp, sp, FRAME_SPACE
jr ra

MAX_DIV = 100 # Es el máximo divisor de acuerdo a la cantidad máxima de
# cifras que puede tener el número a imprimir.

imprimir_numero: # Recibe registros T6, T8 (y lo cambia) y T9; usa
# registros T0, T1, T2, T3 y T4
# T6 es el archivo donde debo imprimir.
# T8 tiene el número de 3 cifras máximo.
# T9 tiene a donde debo volver para seguir ejecutando el programa.
li t0, MAX_DIV # T0 para el divisor.
```

```
imp_cifra:
beq t0, 0, volver # Si ya imprimió todo, vuelve al programa.
divu t1, t8, t0 # T1 para el cociente de num/div
rem t2, t8, t0 # T2 para el resto de num/div
escribir_num:
# El número a imprimir está en T1
la t3, numeros # T3 para la dirección del arreglo.
# La posición en el arreglo es número por 4 (ya que cada palabra son 4 bytes).
sll t4, t1, 2 # T4 tiene la poscición del número en el arreglo.
addu t4, t3, t4 # T4 tiene la dirección donde está la dirección de la
# etiqueta del número.
lw t4, 0(t4) # T4 tiene dirección de la etiqueta del número.

li v0, SYS_write
move a0, t6
lw a1, 0(t4)
li a2, LARGO_NUM
syscall
# Si me devolvió un número no nulo en a3, hubo error.
bne a3, zero, imprimir_io_error
blt v0, a2, imprimir_io_error

proximo:
move t8, t2
divu t0, t0, 10 # Para testear la siguiente cifra.
j imp_cifra

volver:
    # Imprimo un enter.
    li v0, SYS_write
 move a0, t6
la a1, enter
li a2, LARGO_ENTER
syscall
# Si me devolvió un número no nulo en a3, hubo error.
bne a3, zero, imprimir_io_error
    blt v0, a2, imprimir_io_error

    jr t9


STD_ERR = 2
imprimir_io_error:
li v0, SYS_write
li a0, STD_ERR
la a1, error_io
li a2, LARGO_IO_ERROR
syscall
j exit
```

```
exit:
li v0, SYS_exit
syscall


.end mips32_plot


.rdata

p2: .asciiz "P2\n"

.align 2

numeros: .word cero, uno, dos, tres, cuatro, cinco, seis, siete, ocho, nueve
cero: .ascii "0"
uno: .ascii "1"
dos: .ascii "2"
tres: .ascii "3"
cuatro: .ascii "4"
cinco: .ascii "5"
seis: .ascii "6"
siete: .ascii "7"
ocho: .ascii "8"
nueve: .ascii "9"


enter: .asciiz "\n"

error_io: .asciiz "i/o error.\n"
```

8. Conclusiones

El trabajo práctivo motivo de este informe ha presentado al equipo diversos desafíos. Estos desafíos llevaron a familiarizarse con las instrucciones que provee MIPS32, de forma de conocer los recursos que se tienen para resolver un problema y entender la arquitectura de los procesadores. Por otro lado, se reforzó el uso de las convenciones de ABI para manejar las llamadas a funciones; aunque la programada no llamaba a otra función, sí debía respetar a la función que la había llamado. Por las razones expuestas, consideramos muy útil— un trabajo práctico introductorio de esta naturaleza para empezar a conocer las utilidades del lenguaje y arquitectura estudiados.

Referencias

[1] GXemul, http://gxemul.sourceforge.net/

[2] The NetBSD Project, http://www.netbsd.org/

[3] PGM format specification. http://netpbm.sourceforge.net/doc/pgm.html

[4] Oetiker, Tobias, "The Not So Short Introduction To LaTeX2", http://www.physics.udel.edu/$\sim$dubois/lshort2e/