TP1 – Mandelbrot

**Juan Facundo Tkaczyszyn , Padrón Nro. 87.931**

*facu.tk@gmail.com*

**Santiago Weber, Padrón Nro. 93.789**

*santiago.weber91@gmail.com*

2do. Cuatrimestre de 2014

66.20 Organización de Computadoras − Práctica Martes

Facultad de Ingeniería, Universidad de Buenos Aires

## 

## Tabla de Contenidos

[Tabla de Contenidos 1](#_Toc403436514)

[Resumen 2](#_Toc403436515)

[Mandelbrot 3](#_Toc403436516)

[Interfáz 3](#_Toc403436517)

[Salida: 3](#_Toc403436518)

[Archivo/Salida Standard 3](#_Toc403436519)

[Formato PGM 3](#_Toc403436520)

[Desarrollo 4](#_Toc403436521)

[Pruebas 6](#_Toc403436522)

[Código 7](#_Toc403436523)

[mips32\_plot.S 7](#_Toc403436524)

[Conclusiones 25](#_Toc403436525)

# Resumen

El set de Mandelbrot es un fractal. A lo largo de este trabajo práctico lo analizamos, y construimos un programa que permite dibujarlo centrado y acercado a donde se le indique. Este informe refleja las consideraciones que tomamos al encarar el trabajo práctico, las pruebas y el código fuente entregable.

# Mandelbrot

Utilizando el algoritmo proveído por la cátedra, calculamos la intensidad de cada pixel de una manera un tanto distinta a la del trabajo práctico anterior. En este trabajo, la fórmula utilizada es la de Mandelbrot de orden 3.Empieza con la ecuación:



Donde c y z son números complejos y n es cero o un número entero positivo. Empezando en z0 = 0, c esta en el set de Mandelbrot si el valor absoluto de Zn nunca excede cierto número.

# Interfáz

El programa tiene que ser capaz de leer argumentos pasados por línea de comandos. Para parámetros como la resolución ( ej: 640x480 ), o el centro ( ej: 1-4.5i), debe validar que se cumpla con el formato correcto y se traiga el tipo de dato correcto.

# Salida:

## Archivo/Salida Standard

El programa toma el parámetro de entrada y debe decidir si tiene que salir a un archivo, o salir por salida standard[5]. En caso que salga por un archivo debe validar que sea posible la escritura al mismo.

## Formato PGM

El formato PGM[6] es una formato para almacenar información gráfica en un texto plano. Se detalla abajo un ejemplo de un cuadrado negro sobre un fondo blanco.

P2 # Header

4 # Cantidad de filas

4 # Cantidad de columnas

255 # Maximo valor que puede tener un punto

255 225 255 255 # Matriz de puntos

255 0 0 255

255 0 0 255

255 225 255 255

# Desarrollo

El objetivo de este trabajo práctico es familiarizarse con el conjunto de instrucciones MIPS, y para ello tradujimos la siguiente función (ver sección Código para la traducción):

|  |
| --- |
| **#include <debug.h>** |
| **#include <stdio.h>** |
| **#include <defs.h>** |
| **#include <param.h>** |
|  |
| **void** |
| **mips32\_plot**(**param\_t** **\***parms) |
| { |
| **float** cr, ci; |
| **float** zr, zi; |
| **float** sr, si; |
| **float** absz; |
| **int** x, y; |
| **int** c; |
|  |
| */\* Header PGM. \*/* |
| fprintf(parms**->**fp, "P2\n"); |
| fprintf(parms**->**fp, "%u\n", (**unsigned**)parms**->**x\_res); |
| fprintf(parms**->**fp, "%u\n", (**unsigned**)parms**->**y\_res); |
| fprintf(parms**->**fp, "%u\n", (**unsigned**)parms**->**shades); |
|  |
| */\** |
| *\* Barremos la región rectangular del plano complejo comprendida* |
| *\* entre (parms->UL\_re, parms->UL\_im) y (parms->LR\_re, parms->LR\_im).* |
| *\* El parámetro de iteración es el punto (cr, ci).* |
| *\*/* |
| **for** (y **=** 0, ci **=** parms**->**UL\_im; |
| y **<** parms**->**y\_res; |
| **++**y, ci **-=** parms**->**d\_im) { |
| **for** (x **=** 0, cr **=** parms**->**UL\_re; |
| x **<** parms**->**x\_res; |
| **++**x, cr **+=** parms**->**d\_re) { |
| zr **=** cr; |
| zi **=** ci; |
|  |
| */\** |
| *\* Determinamos el nivel de brillo asociado al punto* |
| *\* (cr, ci), usando la fórmula compleja recurrente* |
| *\* f = f^3 + c.* |
| *\*/* |
| **for** (c **=** 0; c **<** parms**->**shades; **++**c) { |
| **if** ((absz **=** zr**\***zr **+** zi**\***zi) **>=** 4.0f) |
| **break**; |
| sr **=** zr **\*** zr **\*** zr |
| **-** 3 **\*** zi **\*** zi **\*** zr |
| **+** cr; |
| si **=** 3 **\*** zr **\*** zr **\*** zi |
| **-** zi **\*** zi **\*** zi |
| **+** ci; |
| zr **=** sr; |
| zi **=** si; |
| } |
|  |
| **if** (fprintf(parms**->**fp, "%u\n", (**unsigned**)c) **<** 0) { |
| fprintf(stderr, "i/o error.\n"); |
| exit(1); |
| } |
| } |
| } |
|  |
| */\* Flush any buffered information before quit. \*/* |
| **if** (fflush(parms**->**fp) **!=** 0) { |
| fprintf(stderr, "cannot flush output file.\n"); |
| exit(1); |
| } |
| } |

# Pruebas

Corrimos el programa original en C con los parámetros por default, y obtuvimos la siguiente salida:

P2

3

3

255

5

40

255

5

8

255

5

33

26

Luego corrimos el programa traducido y obtuvimos el mismo resultado.

# Código

## mips32\_plot.S

#include <mips/regdef.h>

#include <sys/syscall.h>

# typedef struct \_\_sFILE {

# 0 unsigned char \*\_p;

# 4 int \_r;

# 8 int \_w;

# 12 short \_flags;

# 14 short \_file; // fileno, if Unix descriptor, else -1

#define FILE\_FD 14

# typedef struct param\_t {

# 0 float UL\_re;

# 4 float UL\_im;

# 8 float LR\_re;

# 12 float LR\_im;

# 16 float d\_re;

# 20 float d\_im;

# 24 size\_t x\_res;

# 28 size\_t y\_res;

# 32 size\_t shades;

# 36 FILE \*fp;

#define PARAM\_T\_FP 36

#define PARAM\_T\_SHADES 32

#define PARAM\_T\_Y\_RES 28

#define PARAM\_T\_X\_RES 24

#define PARAM\_T\_D\_IM 20

#define PARAM\_T\_D\_RE 16

#define PARAM\_T\_LR\_IM 12

#define PARAM\_T\_LR\_RE 8

#define PARAM\_T\_UL\_IM 4

#define PARAM\_T\_UL\_RE 0

# void mips32\_plot( param\_t \* parms )

#

# Imprimo numero segun a0

#

# Layout del stack

# ABA 92 -- padding

# 88 a0 \*param

# -- -- -- -------

# SRA 84 ra

# 80 $fp

# 76 gp

# 72 -- padding

# FRA 68 -- UL\_re

# 64 -- UL\_im

# 60 -- d\_re

# 56 -- d\_im

# 52 -- cr

# 48 -- ci

# LTA 44 -- x\_res

# 40 -- y\_res

# 36 -- shades

# 32 -- fd

# 28 -- x

# 24 -- y

# 20 -- brightness

# 16 -- padding

# ABA 12 a3

# 8 a2

# 4 a1

# 0 a0

#define MP\_STACK\_SIZE 88

#define MP\_PARAM 88

#define MP\_RA 84

#define MP\_FP 80

#define MP\_GP 76

#define MP\_UL\_RE 68

#define MP\_UL\_IM 64

#define MP\_D\_RE 60

#define MP\_D\_IM 56

#define MP\_CR 52

#define MP\_CI 48

#define MP\_X\_RES 44

#define MP\_Y\_RES 40

#define MP\_SHADES 36

#define MP\_FILE\_D 32

#define MP\_X 28

#define MP\_Y 24

#define MP\_BRIGHTNESS 20

.text

.align 2

.globl mips32\_plot

.ent mips32\_plot

mips32\_plot:

.frame $fp, MP\_STACK\_SIZE, ra

.set noreorder

.cpload t9

.set reorder

subu sp, sp, MP\_STACK\_SIZE

.cprestore MP\_GP

sw gp, MP\_GP(sp)

sw $fp, MP\_FP(sp)

move $fp, sp

sw ra, MP\_RA($fp)

sw a0, MP\_PARAM($fp) # Guardo PARAM en el caller ABA

lw t0, MP\_PARAM($fp) # x\_res = param->x\_res

lw t0, PARAM\_T\_X\_RES(t0)

sw t0, MP\_X\_RES($fp)

lw t0, MP\_PARAM($fp) # y\_res = param->y\_res

lw t0, PARAM\_T\_Y\_RES(t0)

sw t0, MP\_Y\_RES($fp)

lw t0, MP\_PARAM($fp) # shades = param->shades

lw t0, PARAM\_T\_SHADES(t0)

sw t0, MP\_SHADES($fp)

lw t0, MP\_PARAM($fp) # fd = param->fp->fd

lw t0, PARAM\_T\_FP(t0)

lh t0, FILE\_FD(t0)

sw t0, MP\_FILE\_D($fp)

lw t0, MP\_PARAM($fp) # d\_re = param->d\_re

l.s $f0, PARAM\_T\_D\_RE(t0)

s.s $f0, MP\_D\_RE($fp)

lw t0, MP\_PARAM($fp) # d\_im = param->d\_im

l.s $f0, PARAM\_T\_D\_IM(t0)

s.s $f0, MP\_D\_IM($fp)

lw t0, MP\_PARAM($fp) # ul\_re = param->ul\_re

l.s $f0, PARAM\_T\_UL\_RE(t0)

s.s $f0, MP\_UL\_RE($fp)

lw t0, MP\_PARAM($fp) # ul\_im = param->ul\_im

l.s $f0, PARAM\_T\_UL\_IM(t0)

s.s $f0, MP\_UL\_IM($fp)

li t0, 0 # Inicializo brightness = 0

sw t0, MP\_BRIGHTNESS($fp)

li t0, 0 # Inicializo Y = 0

sw t0, MP\_Y($fp)

li t0, 0 # Inicializo X = 0

sw t0, MP\_X($fp)

lw a0, MP\_FILE\_D($fp) # Escribo el HEADER

lw a1, MP\_X\_RES($fp) # a0: file descriptor

lw a2, MP\_Y\_RES($fp) # a1: x\_res

lw a3, MP\_SHADES($fp) # a1: y\_res

la t9, header\_pgm # a3: shades

jalr t9 # header\_pgm( fd, x\_res, y\_res, shades )

l.s $f0, MP\_UL\_IM($fp) # ci = UL\_im

s.s $f0, MP\_CI($fp)

MP\_FOR\_Y\_START:

lw t0, MP\_Y($fp) # t0 = Y

lw t1, MP\_Y\_RES($fp) # t1 = Y\_RES

bge t0, t1, MP\_FOR\_Y\_END # si (y >= y\_res) ir MP\_FOR\_Y\_END

li t0, 0 # Inicializo X = 0

sw t0, MP\_X($fp)

l.s $f0, MP\_UL\_RE($fp) # cr = UL\_re

s.s $f0, MP\_CR($fp)

MP\_FOR\_X\_START:

lw t0, MP\_X($fp) # t0: X

lw t1, MP\_X\_RES($fp) # t1: X\_RES

bge t0, t1, MP\_FOR\_X\_END # si (x >= x\_res) ir MP\_FOR\_X\_END

lw a0, MP\_SHADES($fp) # a0: shades

lw a1, MP\_CR($fp) # a1: cr

lw a2, MP\_CI($fp) # a2: ci

la t9, calc\_brightness

jalr t9

sw v0, MP\_BRIGHTNESS($fp)

lw a0, MP\_FILE\_D($fp) # a0: file descriptor

lw a1, MP\_BRIGHTNESS($fp) # a1: brightess

la t9, write\_sint # write\_sint( fd, brightness )

jalr t9

lw t0, MP\_X($fp) # x = x + 1

addi t0, t0, 1

sw t0, MP\_X($fp)

l.s $f2, MP\_CR($fp) # cr = cr + d\_re

l.s $f0, MP\_D\_RE($fp)

add.s $f0, $f2, $f0

s.s $f0, MP\_CR($fp)

b MP\_FOR\_X\_START

MP\_FOR\_X\_END:

lw t0, MP\_Y($fp) # y = y + 1

addi t0, t0, 1

sw t0, MP\_Y($fp)

l.s $f2, MP\_CI($fp) # ci = ci - d\_im

l.s $f0, MP\_D\_IM($fp)

sub.s $f0, $f2, $f0

s.s $f0, MP\_CI($fp)

b MP\_FOR\_Y\_START

MP\_FOR\_Y\_END:

li v0, SYS\_sync # No encuentro implementacion en syscalls

syscall # de fflush.

# Lo mas cercano es SYS\_sync.

# no lleva parametros y no devuelve nada

MP\_END:

lw ra, MP\_RA(sp) # Destruimos el frame.

move sp, $fp

lw $fp, MP\_FP(sp)

lw gp, MP\_GP(sp)

addu sp, sp, MP\_STACK\_SIZE

jr ra

.end mips32\_plot # Retorno

# void header\_pgm( int fd, int x\_res, int y\_res, int shades )

#

# Imprimo numero segun a0

#

# Layout del stack

# ABA 44 a3 shades

# 40 a2 y\_res

# 36 a1 x\_res

# 32 a0 fd

# -- -- -- -------

# SRA 28 ra

# 24 $fp

# 20 gp

# 16 -- padding

# ABA 12 -- padding

# 8 a2

# 4 a1

# 0 a0

#define HP\_STACK\_SIZE 32

#define HP\_A3 44

#define HP\_A2 40

#define HP\_A1 36

#define HP\_A0 32

#define HP\_RA 28

#define HP\_FP 24

#define HP\_GP 20

.text

.align 2

.globl header\_pgm

.ent header\_pgm

header\_pgm:

.frame $fp, HP\_STACK\_SIZE, ra

.set noreorder

.cpload t9

.set reorder

subu sp, sp, HP\_STACK\_SIZE

.cprestore HP\_GP

sw gp, HP\_GP(sp)

sw $fp, HP\_FP(sp)

move $fp, sp

sw ra, HP\_RA($fp)

sw a0, HP\_A0($fp)

sw a1, HP\_A1($fp)

sw a2, HP\_A2($fp)

sw a3, HP\_A3($fp)

lw a0, HP\_A0($fp) # a0: standard output file descriptor.

la a1, HP\_P2 # a1: data pointer.

li a2, 3 # a2: data length.

la t9, sys\_write # encapsulo SYS\_write para manejar errores

jalr t9

lw a0, HP\_A0($fp) # a0: fd

lw a1, HP\_A1($fp) # a1: x\_res

la t9, write\_sint # write\_sint( fd, x\_res )

jalr t9

lw a0, HP\_A0($fp) # a0: fd

lw a1, HP\_A2($fp) # a1: y\_res

la t9, write\_sint # write\_sint( fd, y\_res )

jalr t9

lw a0, HP\_A0($fp) # a0: fd

lw a1, HP\_A3($fp) # a1: shades

la t9, write\_sint # write\_sint( fd, shades )

jalr t9

lw ra, HP\_RA(sp) # Destruimos el frame.

move sp, $fp

lw $fp, HP\_FP(sp)

lw gp, HP\_GP(sp)

addu sp, sp, HP\_STACK\_SIZE

jr ra

.end header\_pgm # Retorno

.rdata

HP\_P2:

.asciiz "P2\n"

# int calc\_brightness( int shades, float cr, float ci )

#

# Layout del stack

# ABA 92 padding

# 88 a2

# 84 a1

# 80 a0

# -- -- -- -------

# SRA 76 ra

# 72 $fp

# 68 gp

# 64 -- padding

# FRA 60 -- CB\_ZR

# 56 -- CB\_ZI

# 52 -- CB\_ZRXZR

# 48 -- CB\_ZIXZI

# 44 -- CB\_ABSZ

# 40 -- CB\_SR

# 36 -- CB\_SI

# 32 -- CB\_ZK\_CUBE

# 28 -- CB\_ZK\_SQUARE

# 24 -- CB\_ACCUMULATOR

# 20 -- CB\_CR

# 16 -- CB\_CI

# LTA 12 -- CB\_SHADES

# 8 -- CB\_BRIGHTNESS

# 4 -- -------

# 0 -- -------

#define CB\_STACK\_SIZE 80

#define CB\_A2 88

#define CB\_A1 84

#define CB\_A0 80

#define CB\_RA 76

#define CB\_FP 72

#define CB\_GP 68

#define CB\_ZR 60

#define CB\_ZI 56

#define CB\_ZRXZR 52

#define CB\_ZIXZI 48

#define CB\_ABSZ 44

#define CB\_SR 40

#define CB\_SI 36

#define CB\_ZK\_CUBE 32

#define CB\_ZK\_SQUARE 28

#define CB\_ACCUMULATOR 24

#define CB\_CR 20

#define CB\_CI 16

#define CB\_SHADES 12

#define CB\_BRIGHTNESS 8

.text

.align 2

.globl calc\_brightness

.ent calc\_brightness

calc\_brightness:

.frame $fp, CB\_STACK\_SIZE, ra

.set noreorder

.cpload t9

.set reorder

subu sp, sp, CB\_STACK\_SIZE

.cprestore CB\_GP

sw gp, CB\_GP(sp)

sw $fp, CB\_FP(sp)

move $fp, sp

sw ra, CB\_RA($fp)

sw a0, CB\_A0($fp) # Guardo en caller ABA

sw a1, CB\_A1($fp)

sw a2, CB\_A2($fp)

l.s $f0, CB\_A1($fp) # cr = a1

s.s $f0, CB\_CR($fp)

l.s $f0, CB\_A2($fp) # ci = a2

s.s $f0, CB\_CI($fp)

lw t0, CB\_A0($fp) # shades = a0

sw t0, CB\_SHADES($fp)

l.s $f0, CB\_CR($fp) # zr = cr

s.s $f0, CB\_ZR($fp)

l.s $f0, CB\_CI($fp) # zi = ci

s.s $f0, CB\_ZI($fp)

li t0, 0 # brightness = 0

sw t0, CB\_BRIGHTNESS($fp)

CB\_FOR:

lw t0, CB\_BRIGHTNESS($fp) # t0 = brightness

lw t1, CB\_SHADES($fp) # t1 = shades

bge t0, t1, CB\_RETURN # si (brightness >= shades) ir CB\_RETURN

l.s $f2, CB\_ZR($fp) # ZRXZR = ZR\*ZR

l.s $f0, CB\_ZR($fp)

mul.s $f0, $f2, $f0

s.s $f0, CB\_ZRXZR($fp)

l.s $f2, CB\_ZI($fp) # ZIXZI = ZI\*ZI

l.s $f0, CB\_ZI($fp)

mul.s $f0, $f2, $f0

s.s $f0, CB\_ZIXZI($fp)

l.s $f2, CB\_ZIXZI($fp) # ABSZ = ZRXZR + ZIXZI

l.s $f0, CB\_ZRXZR($fp)

add.s $f0, $f2, $f0

s.s $f0, CB\_ABSZ($fp)

l.s $f0, CB\_FLOAT\_FOUR # si ( 4 < absz ) ir fin

l.s $f2, CB\_ABSZ($fp)

c.lt.s $f0, $f2

bc1t CB\_RETURN

l.s $f2, CB\_ZI($fp) # ZK\_SQUARE = ZI \* ZI

l.s $f0, CB\_ZI($fp)

mul.s $f0, $f2, $f0

s.s $f0, CB\_ZK\_SQUARE($fp)

l.s $f2, CB\_ZR($fp) # ZK\_CUBE = ZR \* ZR \* ZR

l.s $f0, CB\_ZR($fp)

mul.s $f0, $f2, $f0

mul.s $f0, $f2, $f0

s.s $f0, CB\_ZK\_CUBE($fp)

l.s $f2, CB\_ZK\_SQUARE($fp)

l.s $f0, CB\_ZR($fp) # accumulator = zk\_square \* zr

mul.s $f0, $f2, $f0 #

l.s $f2, CB\_FLOAT\_MINUS\_THREE # accumulator = accumulator \* (-3)

mul.s $f0, $f2, $f0 #

l.s $f2, CB\_CR($fp) # accumulator = accumulator + cr

add.s $f0, $f2, $f0 #

l.s $f2, CB\_ZK\_CUBE($fp) # accumulator = accumulator + zk\_cube

add.s $f0, $f2, $f0 #

s.s $f0, CB\_SR($fp) # sr = accumulator;

l.s $f2, CB\_ZR($fp) # ZK\_SQUARE = ZR \* ZR

l.s $f0, CB\_ZR($fp)

mul.s $f0, $f2, $f0

s.s $f0, CB\_ZK\_SQUARE($fp)

l.s $f2, CB\_ZI($fp) # ZK\_CUBE = ZI \* ZI \* ZI

l.s $f0, CB\_ZI($fp)

mul.s $f0, $f2, $f0

mul.s $f0, $f2, $f0

s.s $f0, CB\_ZK\_CUBE($fp)

l.s $f2, CB\_ZK\_SQUARE($fp)

l.s $f0, CB\_ZI($fp) # ACCUMULATOR = ZK\_SQUARE \* ZI

mul.s $f0, $f2, $f0 #

l.s $f2, CB\_FLOAT\_THREE # ACCUMULATOR = ACCUMULATOR \* 3

mul.s $f0, $f2, $f0 #

l.s $f2, CB\_ZK\_CUBE($fp) # ACCUMULATOR = ACCUMULATOR - ZK\_CUBE

sub.s $f0, $f0, $f2 #

l.s $f2, CB\_CI($fp) # ACCUMULATOR = ACCUMULATOR + CI

add.s $f0, $f2, $f0 #

s.s $f0, CB\_SI($fp) # SI = ACCUMULATOR

l.s $f0, CB\_SR($fp) # ZR = SR

s.s $f0, CB\_ZR($fp)

l.s $f0, CB\_SI($fp) # ZI = SI

s.s $f0, CB\_ZI($fp)

lw t0, CB\_BRIGHTNESS($fp) # brightness = brightness + 1

addi t0, t0, 1

sw t0, CB\_BRIGHTNESS($fp)

b CB\_FOR

CB\_RETURN:

lw v0, CB\_BRIGHTNESS($fp)

lw ra, CB\_RA(sp) # Destruimos el frame.

move sp, $fp

lw $fp, CB\_FP(sp)

lw gp, CB\_GP(sp)

addu sp, sp, CB\_STACK\_SIZE

jr ra

.end calc\_brightness # Retorno

.rdata

.align 2

CB\_FLOAT\_FOUR:

.word 1082130432

.align 2

CB\_FLOAT\_THREE:

.word 1077936128

.align 2

CB\_FLOAT\_MINUS\_THREE:

.word -1069547520

# void write\_sint( int fd, int number )

#

# Descompongo el int en digitos y los mando a imprimir

#

# Layout del stack

# ABA 60 -- padding

# 56 -- padding

# 52 a1 number

# 48 a0 fd

# -- -- -- -------

# SRA 44 ra

# 40 $fp

# 36 gp

# 32 -- padding

# LTA 28 t3 h\_digit

# 24 t2 m\_digit

# 20 t1 l\_digit

# 16 t0 rem

# ABA 12 -- padding

# 8 -- padding

# 4 -- padding

# 0 a0

#define WS\_STACK\_SIZE 48

#define WS\_A1 52

#define WS\_A0 48

#define WS\_RA 44

#define WS\_FP 40

#define WS\_GP 36

#define WS\_H\_DIGIT 28

#define WS\_M\_DIGIT 24

#define WS\_L\_DIGIT 20

#define WS\_REM 16

.text

.align 2

.globl write\_sint

.ent write\_sint

write\_sint:

.frame $fp, WS\_STACK\_SIZE, ra

.set noreorder

.cpload t9

.set reorder

subu sp, sp, WS\_STACK\_SIZE

.cprestore WS\_GP

sw ra, WS\_RA(sp)

sw $fp, WS\_FP(sp)

sw gp, WS\_GP(sp)

move $fp, sp

sw a0, WS\_A0($fp)

sw a1, WS\_A1($fp)

lw t0, WS\_A1($fp) # Me guardo el valor a0 en otra variable

sw t0, WS\_REM($fp) # temporal para trabajar

li t1, 0 # Inicializo las variables con 0

sw t1, WS\_L\_DIGIT($fp)

li t2, 0

sw t2, WS\_M\_DIGIT($fp)

li t3, 0

sw t3, WS\_H\_DIGIT($fp)

lw t0, WS\_REM($fp) # Busco el digito mas bajo y lo guardo

li t4, 10 # Haciendo rem entre el numero

rem t1, t0, t4 # y 10.

sw t1, WS\_L\_DIGIT($fp) # Lo guardo en su lugar

divu t0, t0, t4 # REM/=10

sw t0, WS\_REM($fp)

lw t0, WS\_REM($fp) # Busco el digito del medio y lo guardo

li t4, 10 # Haciendo rem entre el numero

rem t2, t0, t4 # y 10.

sw t2, WS\_M\_DIGIT($fp) # Lo guardo en su lugar

divu t0, t0, t4 # REM/=10

sw t0, WS\_REM($fp)

lw t0, WS\_REM($fp) # Busco el digito mas grande y lo guardo

li t4, 10 # Haciendo rem entre el numero

rem t3, t0, t4 # y 10.

sw t3, WS\_H\_DIGIT($fp) # Lo guardo en su lugar

lw t3, WS\_H\_DIGIT($fp) # Si el numero mas alto es un 0

beqz t3, WS\_WRITE\_M\_DIGIT # Lo salteo, no lo imprimo

lw a0, WS\_A0($fp) # a0: standard output file descriptor.

lw a1, WS\_H\_DIGIT($fp) # a1: numero a imprimir

la t9, write\_digit # write\_digit: imprimo numero segun a0

jalr t9

WS\_WRITE\_M\_DIGIT:

lw t3, WS\_H\_DIGIT($fp) # El digito del medio lo imprimo solo

li t4, 0 # si el digito mas alto es mayor a 0

bgt t3, t4, WS\_WRITE\_M\_D\_OUT # o el digito del medio es mayor a 0

lw t2, WS\_M\_DIGIT($fp)

li t4, 0

bgt t2, t4, WS\_WRITE\_M\_D\_OUT

b WS\_WRITE\_L\_DIGIT # sino lo salteo

WS\_WRITE\_M\_D\_OUT:

lw a0, WS\_A0($fp) # a0: standard output file descriptor.

lw a1, WS\_M\_DIGIT($fp) # a1: numero a imprimir

la t9, write\_digit # write\_digit: imprimo numero segun a0

jalr t9

WS\_WRITE\_L\_DIGIT:

lw a0, WS\_A0($fp) # a0: standard output file descriptor.

lw a1, WS\_L\_DIGIT($fp) # a1: numero a imprimir

la t9, write\_digit # write\_digit: imprimo numero segun a0

jalr t9

lw a0, WS\_A0($fp) # a0: standard output file descriptor.

la a1, WS\_NL # a1: salto de linea

li a2, 1 # a2: data length.

la t9, sys\_write # encapsulo SYS\_write para manejar errores

jalr t9

lw ra, WS\_RA(sp) # Destruimos el frame.

move sp, $fp

lw $fp, WS\_FP(sp)

lw gp, WS\_GP(sp)

addu sp, sp, WS\_STACK\_SIZE

jr ra

.end write\_sint # Retorno

.rdata

WS\_NL:

.asciiz "\n"

# void write\_digit( int fd, int digit )

#

# Imprimo numero segun a0

#

# Layout del stack

# ABA 44 -- padding

# 40 -- padding

# 36 a1

# 32 a0

# -- -- -- -------

# SRA 28 ra

# 24 $fp

# 20 gp

# 16 -- padding

# ABA 12 -- padding

# 8 a2

# 4 a1

# 0 a0

#define WD\_STACK\_SIZE 32

#define WD\_A1 36

#define WD\_A0 32

#define WD\_RA 28

#define WD\_FP 24

#define WD\_GP 20

.text

.align 2

.globl write\_digit

.ent write\_digit

write\_digit:

.frame $fp, WD\_STACK\_SIZE, ra

.set noreorder

.cpload t9

.set reorder

subu sp, sp, WD\_STACK\_SIZE

.cprestore WD\_GP

sw gp, WD\_GP(sp)

sw $fp, WD\_FP(sp)

move $fp, sp

sw ra, WD\_RA($fp)

sw a0, WD\_A0($fp)

sw a1, WD\_A1($fp)

lw a0, WD\_A0($fp) # a0: standard output file descriptor.

la a1, WD\_NUMBERS # a1: data pointer.

lw t0, WD\_A1($fp)

add a1, a1, t0

li a2, 1 # a2: data length.

la t9, sys\_write # encapsulo SYS\_write para manejar errores

jalr t9

lw ra, WD\_RA(sp) # Destruimos el frame.

move sp, $fp

lw $fp, WD\_FP(sp)

lw gp, WD\_GP(sp)

addu sp, sp, WD\_STACK\_SIZE

jr ra

.end write\_digit # Retorno

.rdata

WD\_NUMBERS:

.asciiz "0123456789"

# Layout del stack

# 24 a2 count

# 20 a1 buf

# 16 a0 fd

# --

# 12 - padding

# 8 ra

# 4 $fp

# 0 gp

#define SW\_STACK\_SIZE 16

#define SW\_A2 24

#define SW\_A1 20

#define SW\_A0 16

#define SW\_RA 8

#define SW\_FP 4

#define SW\_GP 0

.text

.align 2

.globl sys\_write

.ent sys\_write

sys\_write:

.frame $fp, SW\_STACK\_SIZE, ra

.set noreorder

.cpload t9

.set reorder

subu sp, sp, SW\_STACK\_SIZE

.cprestore SW\_GP

sw gp, SW\_GP(sp)

sw $fp, SW\_FP(sp)

move $fp, sp

sw ra, SW\_RA($fp)

sw a0, SW\_A0($fp)

sw a1, SW\_A1($fp)

sw a2, SW\_A2($fp)

li v0, SYS\_write # ver dentro de <sys/syscall.h>.

lw a0, SW\_A0($fp) # a0: standard output file descriptor.

lw a1, SW\_A1($fp) # a1: data pointer.

lw a2, SW\_A2($fp) # a2: data length.

syscall

bne a3, zero, SW\_ERROR # Si a3 != 0 hay un error

bgez v0, SW\_RETURN # Si vuelve del syscall sin errores

# salgo normalmente

SW\_ERROR:

li v0, SYS\_write # Si no salio normalmente

li a0, 2 # Imprimo mensaje de error por stderr

la a1, ioerror

li a2, 11

syscall

li v0, SYS\_exit # exit(1)

li a0, 1

syscall

SW\_RETURN:

lw ra, SW\_RA(sp) # Destruimos el frame.

move sp, $fp

lw $fp, SW\_FP(sp)

lw gp, SW\_GP(sp)

addu sp, sp, SW\_STACK\_SIZE

jr ra

.end sys\_write # Retorno

.rdata

ioerror:

.asciiz "i/o error.\n"

# Conclusiones

A lo largo de este trabajo práctico logramos familiarizarnos con el uso de la terminal a fin de crear túneles entre sistemas operativos, uno siendo el host y el otro el guest. Además obtuvimos un mejor manejo de de cómo funciona la arquitectura MIPS, y logramos, al igual que en el trabajo práctico anterior, generar una imagen que representa al conjunto de Mandelbrot, donde los puntos más claros son los que pertenecen al conjunto, y los más oscuros son los que no.A lo largo de este trabajo práctico logramos familiarizarnos con el uso de la terminal a fin de crear túneles entre sistemas operativos, uno siendo el host y el otro el guest. Además obtuvimos un mejor manejo de de cómo funciona la arquitectura MIPS, y logramos, al igual que en el trabajo práctico anterior, generar una imagen que representa al conjunto de Mandelbrot, donde los puntos más claros son los que pertenecen al conjunto, y los más oscuros son los que no.