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

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:

$$z_{n+1} = Z_n^3 + c$$

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.