66.20 Organización de Computadoras
Trabajo Práctico 1:
MIPS

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

11 de noviembre 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

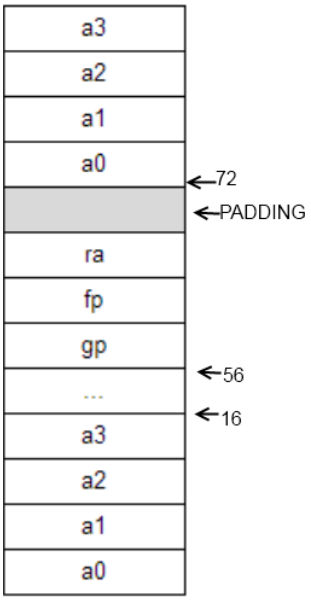

Figura 1: Stack frame generado

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

Se realizaron algunas corridas de prueba para confirmar la funcionalidad de la función mips32_plot.S. Se compararon los gráficos obtenidos con los que resultaban de ejecutar la misma función compilando con mips32_plot.c .

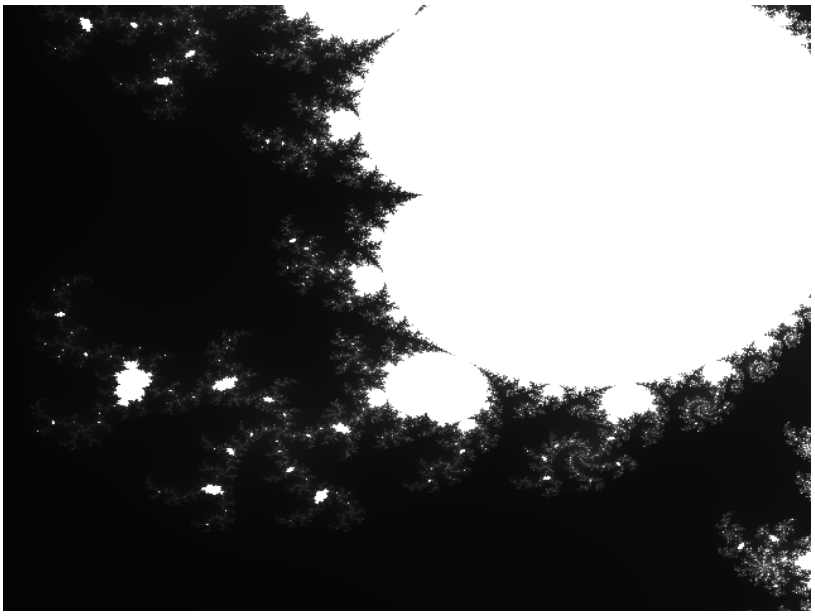

Figura 2: Ejecución con parámetros default en C

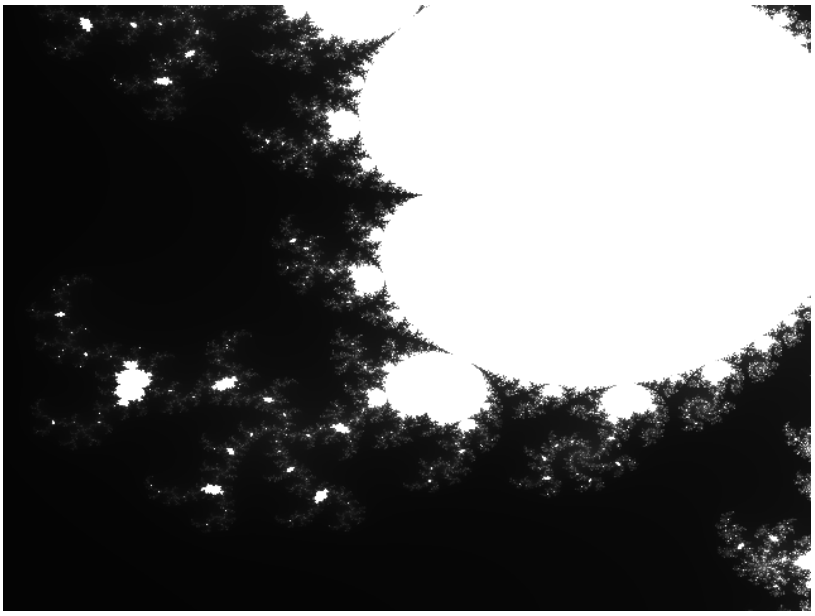

Figura 3: Ejecución con parámetros default en Assembly

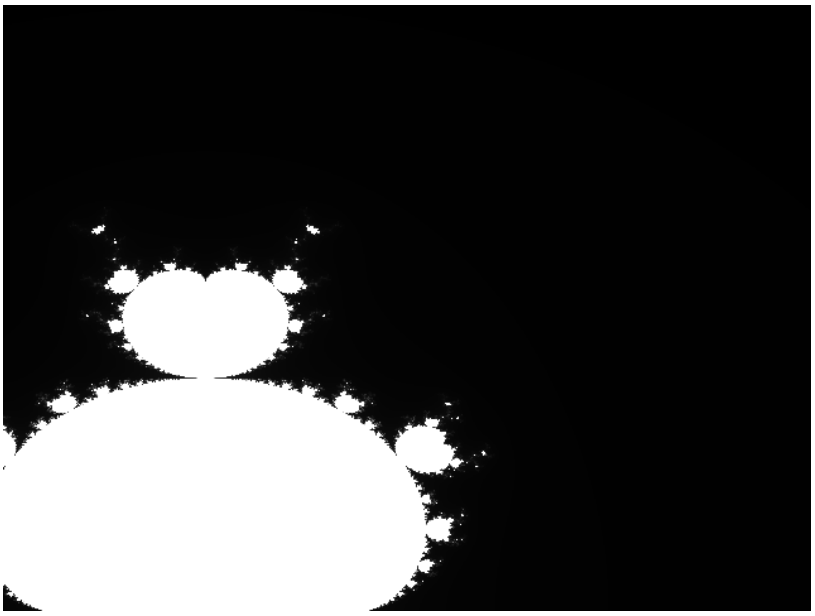

Figura 4: Ejecución con centro en 0,5;1i, resolución 640x480, ancho 2 y alto 2 en C

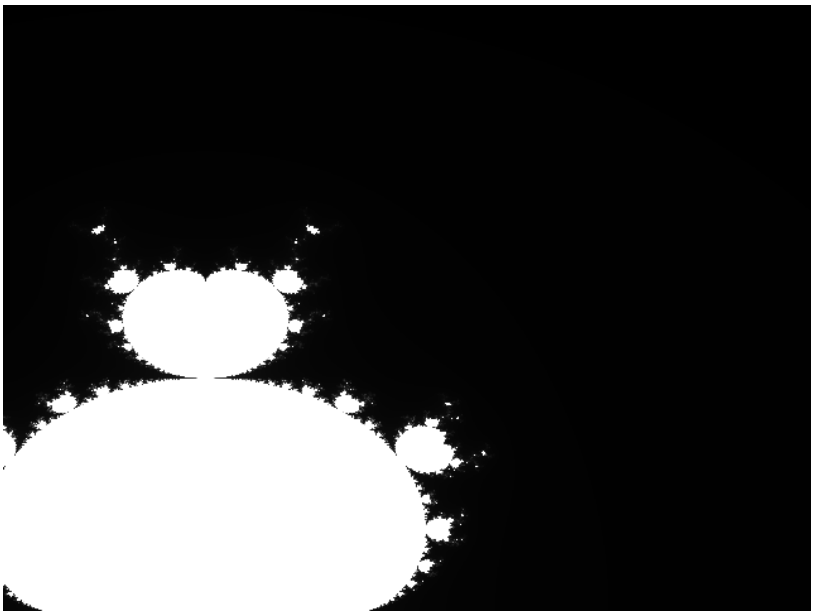

Figura 5: Ejecución con centro en 0,5;1i, resolución 640x480, ancho 2 y alto 2 en asm

7. Mediciones

Realizamos algunas mediciones para comparar el tiempo de ejecución del programa según se hubiera compilado con la función mips32_plot en Assembler, en C o en C con optimizaciones.

Se obtuvieron los siguientes resultados:

Resolución	Assembler (s)	C(s)	C optimizado (s)
10x10	0,117	0,117	0,102
100x100	10,187	9,859	8,184
200x200	45,051	39,531	31,949
640x480	199,762	164,559	131,164

8. Conclusiones

A diferencia de lo que esperábamos a priori, los resultados para la corrida de Assembler fueron más lentos que los obtenidos para C, incluso con las optimizaciones apagadas.

Esto se debe probablemente a que al tratarse de la primera incursión de programación en Assembler, el código producido es de menor calidad (en términos de operaciones realizadas) que el resultante de compilar C.

Adicionalmente, la programación en Assembler resulta en más esfuerzo por parte del progamador ya que generalmente resulta en programas más largos desde el punto de vista de las líneas de código.

En nuestro caso la función en Assembler tiene 255 líneas de código (sin contar comentarios o espacios en blanco) mientras que en C tiene 48.

Adicionalmente, para Assembler tuvimos que escribir una función auxiliar para convertir de enteros a caracteres, aumentando 70 líneas para totalizar 325. Esto representa 6,78 veces más código que en C.

9. Código

9.1. mips32_plot.c

```c
#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 region rectangular del plano complejo comprendida
   * entre (parms->UL_re, parms->UL_im) y (parms->LR_re, parms->LR_im).
   * El parametro de iteracion 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 formula 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);
  }
}
```

9.2. 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 ERROR_MSG_LENGTH 19
#define STDERR 2

#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
bne a3, zero, write_error

#imprimo xres
lw  t0,FRAME_SIZE($fp)
lw  t1,FD(t0)
#t1 = FD
lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
lw  t0,x_res(t0)
move  a0,t0
move  a1,t1
jal write_int


#Imprimir fin de linea
lw  t0,FRAME_SIZE($fp)
lw  t1,FD(t0)
lh  t1,descriptor(t1) #me muevo al campo _file mediante un short

#t1 = FD
li  v0,SYS_write
move  a0,t1
la a1,endln
li a2,1
syscall
bne a3, zero, write_error


#imprimo yres
lw  t0,FRAME_SIZE($fp)
lw  t1,FD(t0)
#t1 = FD
lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
lw  t0,y_res(t0)
move  a0,t0
move  a1,t1
jal write_int


#Imprimir fin de linea
lw  t0,FRAME_SIZE($fp)
lw  t1,FD(t0)
lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
#t1 = FD
li  v0,SYS_write
move  a0,t1
la a1,endln
li a2,1
syscall
bne a3, zero, write_error


#imprimo shades
lw  t0,FRAME_SIZE($fp)
lw  t1,FD(t0)
#t1 = FD
lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
lw  t0,shades(t0)
move  a0,t0
move  a1,t1
jal write_int

#Imprimir fin de linea
lw  t0,FRAME_SIZE($fp)
lw  t1,FD(t0)
lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
```

```
  #t1 = FD
  li  v0,SYS_write
  move  a0,t1
  la a1,endln
  li a2,1
  syscall
  bne a3, zero, write_error

  #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     #(x<x_rs){inc y}

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

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

loop_c:
  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 #f4 = zr*zr
  mul.s $f6, $f6, $f6 #f6 = zi*zi
  add.s $f4, $f4, $f6
  li.s $f6, 4.0
  c.le.s $f6, $f4 # if 4<=absz then write_out
  bc1t write_out

  # calculo sr
  l.s $f4, LTA_POS+20($fp)
  mul.s $f6, $f4, $f4
  mul.s $f16, $f6, $f4 # f16 = zr*zr*zr
  l.s $f6, LTA_POS+16($fp)
  mul.s $f6, $f6, $f6
  mul.s $f6, $f6, $f4
  li.s $f4, -3.0
  mul.s $f6, $f4
  l.s $f4, LTA_POS+12($fp)
```

```
  add.s $f6, $f6, $f16
  add.s $f6, $f6, $f4
  s.s $f6, LTA_POS+28($fp) # sr

  # calculo si
  l.s $f4, LTA_POS+20($fp)
  mul.s $f16, $f4, $f4
  l.s $f4, LTA_POS+16($fp) # f4 = zi
  li.s $f6, 3.0
  mul.s $f16, $f16, $f4
  mul.s $f16, $f16, $f6 #f16 = 3*zi*zr*zr
  mul.s $f6, $f4, $f4
  mul.s $f6, $f6, $f4
  li.s $f4, -1.0
  mul.s $f6, $f6, $f4
  l.s $f4, LTA_POS+4($fp)
  add.s $f6, $f6, $f16
  add.s $f6, $f6, $f4
  s.s $f6, LTA_POS+32($fp) # si

  # zr = sr y zi = si
  l.s $f4, LTA_POS+28($fp)
  l.s $f6, LTA_POS+32($fp)
  s.s $f4, LTA_POS+20($fp)
  s.s $f6, LTA_POS+16($fp)


  # ++c para terminar el ciclo
  lw t0, LTA_POS+24($fp)
  addiu t0, 1
  sw t0, LTA_POS+24($fp)
  j loop_c

write_out:
  lw t0, LTA_POS+24($fp)
  lw t1, FRAME_SIZE($fp)
  lw t1, FD(t1)
  lh t1, descriptor(t1)
  move a0, t0
  move a1, t1
  jal write_int
  #Imprimir fin de linea
  lw  t0,FRAME_SIZE($fp)
  lw  t1,FD(t0)
  lh  t1,descriptor(t1) #me muevo al campo _file mediante un short
  #t1 = FD
  li  v0,SYS_write
  move  a0,t1
  la a1,endln
  li a2,1
  syscall
  bne a3, zero, write_error

inc_x:
  lw  t0,LTA_POS+8($fp) #x
  addi  t0,1
  sw  t0,LTA_POS+8($fp)
  lw  t1,FRAME_SIZE($fp)
  l.s $f4,d_re(t1)  #f4 = parms->d_re
  l.s $f6,LTA_POS+12($fp) #cr
  add.s $f6,$f6,$f4
  s.s $f6,LTA_POS+12($fp)
  j loop_x

inc_y:
  lw  t0,LTA_POS($fp) #y
  addi  t0,1
  sw  t0,LTA_POS($fp)
  lw  t1,FRAME_SIZE($fp)
```

```
  l.s $f4,d_im(t1)  #f4 = parms->d_im
  l.s $f6,LTA_POS+4($fp)  #ci
  sub.s $f6,$f6,$f4
  s.s $f6,LTA_POS+4($fp)
  j loop_y

write_error:
  li a0, STDERR
  la a1, error_msg
  li a2, ERROR_MSG_LENGTH
  li v0, SYS_write
  syscall
  j end

  #PRINTF FUNCIONANDO, imprime t0!!
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
error_msg:
  .asciiz "Error de escritura\n"
endln:
  .asciiz "\n"
msgP2:
  .asciiz "P2\n"
space:
  .asciiz " "


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

9.3. write_int.S

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

#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 beq  a3, zero,  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"
```