Trabajo Práctico 1

Agustin Alejandro Linari, *Padrón Nro. 81.783*
agustinlinari@gmail.com

Juan Ignacio López Pecora, *Padrón Nro. 84.700*
jlopezpecora@gmail.com

Pablo Daniel Sívori, *Padrón: 84.026*
sivoridaniel@gmail.com

2° Cuatrimestre de 2016
66.20 Organizacion de Computadoras
Facultad de Ingenieria, Universidad de Buenos Aires

25 de octubre de 2016

Resumen

En el presente trabajo utilizamos el conjunto de instrucciones MIPS y el concepto de ABI para resolver parte de la lógica del programa realizado en el trabajo práctico 0.

Índice

Parte I

Desarrollo

1. Introduccion

El objetivo del presente trabajo práctico es familiarizarse con el código de instrucciones MIPS 32. Para ello implementaremos la lógica de cómputo del fractal con dicho código de instrucciones. También durante el desarrollo del mismo haremos uso de la convención de llamada de funciones definida en la ABI de MIPS 32. Se hará uso de la syscall SYSwrite para realizar la impresión de los caracteres ascii sobre el archivo pgm. Para utilizar la syscall de manera eficiente se utilizará un buffer de tamaño parametrizable con el fin de reducir la cantidad de llamdas a la misma. Finalmente compilaremos el programa en el emulador GXemul para poder ejectuar el mismo en un sistema operativo NetBSD.

2. Build

El correspondiente informe se puede construir utilizando el make con la etiqueta doc la cual borra y genera el informe en formato pdf.

Para la compilación del trabajo práctico, copiar el directorio tp1 al emulador y luego ejecutar dentro del directorio tp1/src los siguientes comandos.

```
$ make clean
$ make makefiles
$ make
```

Para ejecutar las pruebas, en primer lugar realizar el build y luego desde el directorio tp1/test correr el script run-tests.sh.

3. Diseño e Implementación del Programa

El código fuente del programa se puede encontrar en el anexo A.

Para la implementación de la solución se codificaron cuatro funciones.

3.1. Funciones de escritura

Las funciones buff_write y buff_flush son las encargadas de administrar la impresión de los caracteres sobre el buffer y por lo tanto se encargan también de efectuar la syscall SYS_write. Más precisamente, buff_write utiliza un buffer

de chars definido en una región .data de memoria y accedido mediante un indice también estático. Recibe como parámetros el file descriptor que indica a que archivo hay que imprimir, un char* con la secuencia de caracteres a escribir en el buffer y otro int para indicar el tamaño en bytes de esta secuencia. La función buff_flush es la encargada de efectuar la llamada a la syscall, mandando todo el buffer a imprimir y luego reiniciar su índice para poder seguir imprimiendo sobre el mismo. Recibe como parámetro el file descriptor.

A continuación mostramos el diagrama de stack frame de las mismas.

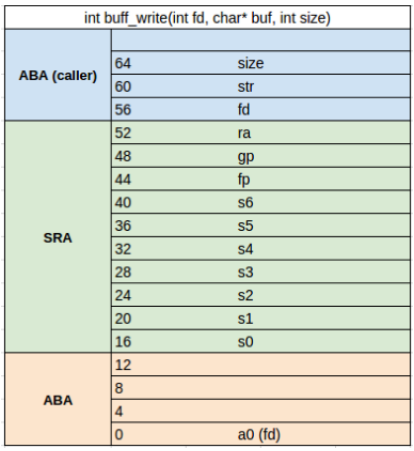


Figura 1: Stack1

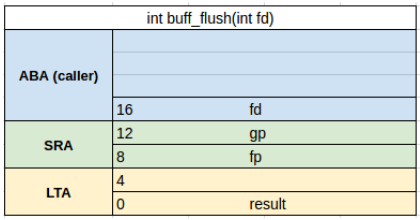


Figura 2: Stack2

Como se puede observar en este caso optamos por hacer uso de los registros S, sin embargo, también podriamos haber utilizado variables locales y alojarlas en la LTA. Los registros S son almacenados en el SRA al iniciar la función y luego son restaurados a su valor original, de esta forma respetamos la ABI.

3.2. Función para conversión de enteros a caracteres ascii

Se necesito crear una función que realice la conversión de enteros a caracteres ascii y los imprima. En este caso, aprovechamos uno de los ejemplos brindados por el curso: la función print_dnames.S. Tomamos como base la misma y la adaptamos para imprimir cada dígito como un caracter ascii. La función la renombramos como:

int print_int(unsigned int n, unsigned int fd)

Recibe como primer argumento el número a convertir y como segundo el file descriptor. La misma, a diferencia de la original, hace uso de las funciones buff_write y buff_flush para imprimir el número entero en el buffer.

A continuación mostramos el diagrama de stack frame de la misma.

int print_int(int n, int fd)		
ABA (caller)	44	fd
	40	n
SRA	36	
	32	ra
	28	fp
	24	sp
LTA	20	
	16	r
ABA	12	a3
	8	a2
	4	a1
	0	a0

Figura 3: Stack3

3.3. Función generadora de fractal

Por último, mostramos la función que contiene la lógica para generar el fractal e imprimirlo en un archivo con formato pgm:

int mips32_plot(param_t* params)

Esta función contiene la lógica principal para generar el fractal de Julia y hace uso de todas las funciones presentadas previamente.

A continuación mostramos el diagrama de stack frame de la misma.

int mips32_plot(param_t* params)		
ABA (caller)	88	params (param_t*)
SRA	84	
	80	ra
	76	gp
	72	fp
LTA	68	
	64	cpi
	60	cpr
	56	res
	52	c
	48	y
	44	x
	40	absz
	36	si
	32	sr
	28	zi
	24	zr
	20	ci
	16	cr
ABA	12	a3
	8	a2
	4	a1
	0	a0

Figura 4: Stack4

3.4. Pruebas

Se creó un script para la ejecución de diversos tests. El mismo se puede ubicar dentro de la carpeta test con el nombre run.test.sh. Junto a él, se pueden encontrar archivos pgm que fueron generados a partir de la ejecución de la versión C de mips32_plot. Para poder correr el script es necesario que se haya realizado el build previamente. El mismo correrá un set de pruebas, comparando las salidas para los siguientes casos:

Caso default

```
$ ./tp1 -o out.pgm;
```

Caso punto perteneciente a conjunto de Julia

```
$ ./tp1 -c 0.01+0i -r 1x1 -o out.pgm;
```

Caso punto no perteneciente a conjunto de Julia

```
$ ./tp1 -c 10+0i -r 1x1 -o out.pgm;
```

Caso de prueba 1 variando constante C

```
$ ./tp1 –C 0.285+0i -o out.pgm;
```

Caso de prueba 2 variando constante C

```
$ ./tp1 –C -0.8+0.156i -o out.pgm;
```

Caso de prueba 3 variando constance C

```
$ ./tp1 –C 0+0.8i -o out.pgm;
```

Caso de prueba idem 2 pero con zoom sobre una región

```
$ ./tp1 –C -0.8+0.156i –w 0.5 –H 0.5 -o out.pgm;
```

4. Corridas de Programa

Se corre el programa obteniendose los siguientes tiempos de ejecución que se detallan en los siguientes cuadros.

Código C	real	usr	sys
1	1m19.363s	1m19.143s	0m0.141s
2	1m23.195s	1m23.014s	0m0.129s
3	1m21.480s	1m19.335s	0m0.133s
promedio	1m 21.346s	1m 20.497s	0.134s

Cuadro 1: Tiempos promedios de ejecución en código C.

Código C	real	usr	sys
1	1m0.449s	0m52.722s	0m7.680s
2	1m0.879s	0m53.042s	0m7.808s
3	1m1.246s	0m53.261s	0m7.937s
promedio	1m0.858s	53.008s	7.808s

Cuadro 2: Tiempos promedios de ejecución en código Mips buffer size 64 bytes.

Código C	real	usr	sys
1	1m4.262s	1m3.237s	0m0.984s
2	1m1.246s	1m0.319s	0m0.883s
3	0m58.547s	0m57.573s	0m0.937s
promedio	1m1.352s	1m0.376s	0.935s

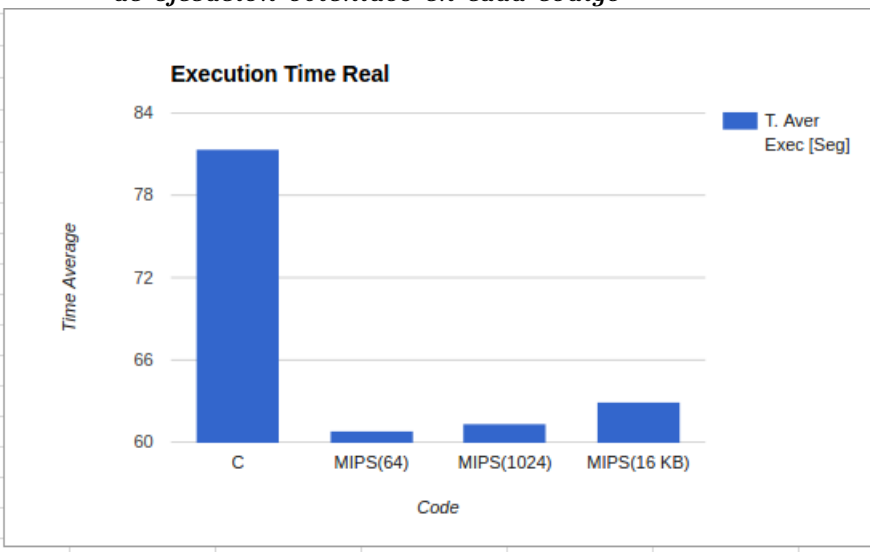
Cuadro 3: Tiempos promedios de ejecución en código Mips buffer size 1 KB.

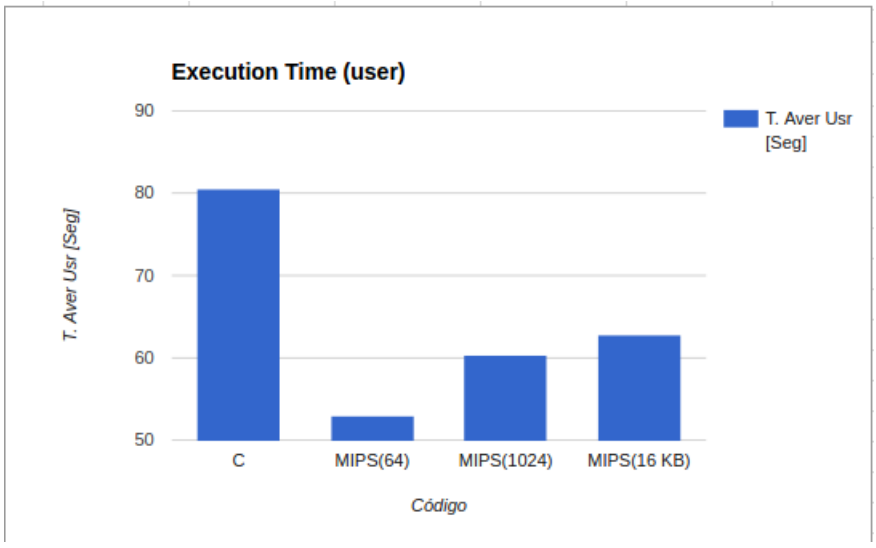
Código C	real	usr	sys
1	1m3.797s	1m3.589s	0m0.156s
2	1m0.617s	1m0.480s	0m0.133s
3	1m4.500s	1m4.249s	0m0.215s
promedio	1m2.971s	1m2.772s	0.168s

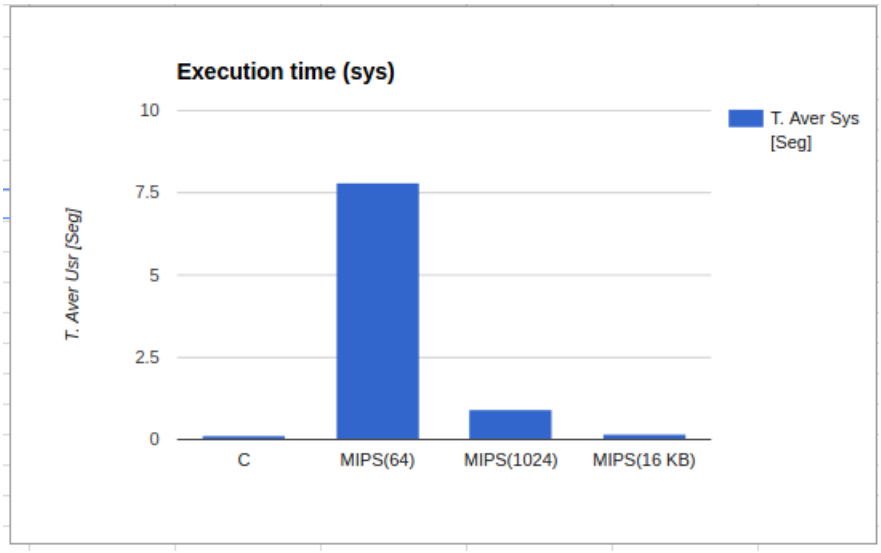
Cuadro 4: Tiempos promedios de ejecución en código Mips buffer size 16 KB.

- Todos los programas fueron compilados con el parámetro -O0 (sin optimizaciones).
- Todas las mediciones corresponden a la ejecución de los programas utilizando el comando time.

Representamos gráficamente los valores obtenidos en los cuadros anteriores, y hacemos una comparación entre los tiempos promedios de ejecución obtenidos en cada código




Execution Time (user)
90
80
70
60
50
T. Aver Usr [Seg]
C
MIPS(64)
MIPS(1024)
MIPS(16 KB)
Código
T. Aver Usr [Seg]


Execution time (sys)
10
7.5
5
2.5
0
T. Aver Usr [Seg]
C
MIPS(64)
MIPS(1024)
MIPS(16 KB)
T. Aver Sys [Seg]

Speed Up	Value
tc / tmips 64B	1.34
tc / tmips 1KB	1.33
tc / tmips 16KB	1.29

Cuadro 5: Speed UP

4.1. Observaciones

- La implementación de mips en cualquiera de sus variantes se ejecuta en menos tiempo que la implementación C pura, con un speed up aproximado de 1/3
- A medida que el buffer se agranda, el tiempo de sistema se reduce. Esto se debe a que se producen menos syscalls a write.
- Podemos plantear la hipótesis razonable de que printf está implementada con un buffer (debido al bajo sys time). Para nuestra implementación MIPS, el tamaño del buffer que obtuvo un tiempo sys del mismo orden que la implementación C fue de 16 KB.
- Contrario a lo que nuestra intuición indicaba, aumentar el buffer para valores mayores a 64 bytes no necesariamente significó (en promedio) en un aumento de performance. Esto puede estar relacionado con la arquitectura del cache emulado, el tamaño de bloque y su política de reemplazo.

4.2. Conclusión

Con la realización de este trabajo hemos podido apreciar la diferencia de performance entre dos implementaciones de distinta naturaleza de un mismo algoritmo, implementado en C y en assembly MIPS32.

A la hora de programar, es común que se codifique utilizando lenguajes de alto nivel. El lenguaje de programación C es un lenguaje de propósito general clásico cuyo diseño provee construcciones que mapean de manera eficiente instrucciones de máquina típicas. Las ventajas de utilizar un lenguaje de alto nivel como C son portabilidad (a nivel código fuente) entre diferentes arquitecturas donde se haya implementado el compilador, aumento de productividad - dado que se abstrae de cuestiones de bajo nivel íntimamente ligadas con la arquitectura de la máquina - y reducción en el costo de mantenimiento. Sin embargo, estas ventajas traen aparejado un costo en la performance del programa.

En algunos casos, los requerimientos funcionales de un programa requieren de una performance que puede ser difícil de alcanzar para una implementación en un lenguaje de alto nivel. Mediante un análisis cuantitativo, se determina qué segmentos de código consumen la mayor cantidad de recursos de una computadora -ciclos de CPU, memoria, etc-. Para el caso particular de este trabajo, la función de cómputo del fractal es central en el desempeño de la aplicación.

Parte II

Apendice

A. Codigo fuente

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


#define STACK_SIZE        88
#define BUFFER_SIZE       64
#define RA_POS            80
#define GP_POS            76
#define FP_POS            72

.data
.align 2
endl: .asciiz "\n"
p2: .asciiz "P2"

.text
.abicalls
.align 2
.global mips32_plot
.ent mips32_plot

mips32_plot:
    .frame        $fp, STACK_SIZE, ra
    .set          noreorder
    .cpload       t9
    .set          reorder
      subu          sp, sp, STACK_SIZE
      sw            ra, RA_POS(sp)              #save ra
      # directiva para codigo PIC
      .cprestore GP_POS     # inserta aqui "sw gp, GP_POS(sp)",
                # mas "lw gp, GP_POS(sp)" luego de cada jal.
      sw            $fp,FP_POS(sp)              #save fp
      move          $fp, sp                     #fp->sp


    ####
    #aca empieza el codigo de la function
      ####
      sw            a0, STACK_SIZE($fp)
      l.s           $f2, 24(a0)             #cpr = parms->cp_re
      l.s           $f4, 28(a0)             #cpi = parms-> cp_im
      s.s           $f2, 60($fp)
      s.s           $f4, 64($fp)

      ##impresion del header

      # imprimir P2
      li            a2, 2
```

```
49          la              a1, p2
50          lw              a0, STACK_SIZE($fp)  #a0 <- params
51          lw              a0, 44(a0)           #a0 <- params->fd
52          la              t9, buff_write
53          jal             ra, t9
54          bltz            v0, end               #si hay error retornamos
55
56          # imprimir fin de linea
57          li              a2, 1
58          la              a1, endl
59          lw              a0, STACK_SIZE($fp)  #a0 <- params
60          lw              a0, 44(a0)           #a0 <- params->fd
61          la              t9, buff_write
62          jal             ra, t9
63          bltz            v0, end               #si hay error retornamos
64
65          # imprimir(x_res)
66          lw              t0, STACK_SIZE($fp) #a1 <- params
67          lw              a1, 44(t0)          #a1 <- params->fd
68          lw              a0, 32(t0)          #a0 <- params->x_res
69          la              t9, print_int
70          jal             ra, t9
71          bltz            v0, end               #si hay error retornamos
72
73          # imprimir fin de linea
74          li              a2, 1
75          la              a1, endl
76          lw              a0, STACK_SIZE($fp)  #a0 <- params
77          lw              a0, 44(a0)           #a0 <- params->fd
78          la              t9, buff_write
79          jal             ra, t9
80          bltz            v0, end               #si hay error retornamos
81
82          # imprimir(y_res)
83          lw              t0, STACK_SIZE($fp) #a1 <- params
84          lw              a1, 44(t0)          #a1 <- params->fd
85          lw              a0, 36(t0)          #a0 <- params->y_res
86          la              t9, print_int
87          jal             ra, t9
88          bltz            v0, end               #si hay error retornamos
89
90          # imprimir fin de linea
91          li              a2, 1
92          la              a1, endl
93          lw              a0, STACK_SIZE($fp)  #a0 <- params
94          lw              a0, 44(a0)           #a0 <- params->fd
95          la              t9, buff_write
96          jal             ra, t9
97          bltz            v0, end               #si hay error retornamos
98
99          # imprimir(shades)
100         lw              t0, STACK_SIZE($fp) #a1 <- params
101         lw              a1, 44(t0)          #a1 <- params->fd
102         lw              a0, 40(t0)          #a0 <- params->shades
103         la              t9, print_int
104         jal             ra, t9
105         bltz            v0, end               #si hay error retornamos
```

```
106
107         # imprimir fin de linea
108         li              a2, 1
109         la              a1, endl
110         lw              a0, STACK_SIZE($fp)  #a0 <- params
111         lw              a0, 44(a0)           #a0 <- params->fd
112         la              t9, buff_write
113         jal             ra, t9
114         bltz            v0, end               #si hay error retornamos
115
116 # CICLO FOR Y
117
118         li              t0,0
119         sw              t0, 48($fp)          #y=0
120         lw              t0, STACK_SIZE($fp) #params
121         l.s             $f2, 4(t0)            #params->UL_im
122         s.s             $f2, 20($fp)          #ci = params->UL_im
123 loop_y:
124         lw              t0, 48($fp)          #t0=y
125         l.s             $f2, 20($fp)         #f2=ci
126         lw              t2, STACK_SIZE($fp) #t2=params
127         lw              t3, 36(t2)           #t3=params->y_res
128         bge             t0, t3, end_y        #y<params->y_res
129
130 #CICLO FOR X
131         li              t0, 0                 #t0=0
132         sw              t0, 44($fp)           #x=0
133         lw              t0, STACK_SIZE($fp)  #params
134         l.s             $f2, 0(t0)            #params->UL_re
135         s.s             $f2, 16($fp)          #cr = params->UL_re
136 loop_x:
137         lw              t0, 44($fp)          #t0=x
138         l.s             $f2, 16($fp)         #f2=cr
139         lw              t2, STACK_SIZE($fp)  #t2=params
140         lw              t3, 32(t2)           #t3=params->x_res
141         bge             t0, t3, end_x        #x<params->y_res
142
143         l.s             $f4, 20($fp)         #ci
144         s.s             $f2, 24($fp)         #zr=cr
145         s.s             $f4, 28($fp)         #zi=ci
146
147 #CICLO FOR C
148
149         li              t0, 0                 #t0=0
150         sw              t0, 52($fp)           #c=0
151 loop_c:
152         lw              t0, 52($fp)          #t0=c
153         lw              t1, STACK_SIZE($fp)  #params
154         lw              t1, 40(t1)           #params->shades
155         bge             t0, t1, end_c        #c<params->shades
156
157 #IF
158         l.s             $f2, 24($fp)         #f2=zr
159         l.s             $f4, 28($fp)         #f4=zi
160         mul.s           $f6, $f2, $f2        #f6=zr*zr
161         mul.s           $f8, $f4, $f4        #f8=zi*zi
162         add.s           $f10, $f6,$f8         #f10=zr*zr+zi*zi
```

```
163         s.s             $f10, 40($fp)          #absz=zr*zr+zi*zi
164         li.s            $f12, 4.0
165         c.lt.s          $f12, $f10             # (absz < 4.0)
166         bc1t            end_c                  #if (absz>4.0f) break
167         sub.s           $f6, $f6, $f8          #f6=zr*zr-zi*zi
168         l.s             $f8 , 60($fp)          #f8 =cpr
169         add.s           $f6, $f6, $f8          #f6=zr*zr-zi*zi+cpr
170         s.s             $f6, 32($fp)           #sr=zr*zr-zi*zi+cpr
171         li.s            $f10, 2.0
172         mul.s           $f8, $f2, $f10            #f8=2*zr
173         mul.s           $f8, $f8, $f4          #f8=2*zr*zi
174         l.s             $f4, 64($fp)           #f4 = cpi
175         add.s           $f8, $f8, $f4          #f8=2*zr*zi + cpi
176         s.s             $f8, 36($fp)           #si=2*zr*zi + cpi
177         s.s             $f6, 24($fp)           #zr=sr
178         s.s             $f8, 28($fp)           #zi=si
179
180         lw              t0, 52($fp)            #t0=c
181         add             t0, t0, 1              #c+1
182         sw              t0, 52($fp)            #c=c+1
183         j loop_c
184 end_c:
185
186         # imprimir(c)
187         lw              a1, STACK_SIZE($fp) #a1 <- params
188         lw              a1, 44(a1)          #a1 <- params->fd
189         lw              a0, 52($fp)          #a0 <- c
190         la              t9, print_int
191         jal             ra, t9
192         bltz            v0, end                #si hay error retornamos
193
194         # imprimir fin de linea
195         li              a2, 1
196         la              a1, endl
197         lw              a0, STACK_SIZE($fp)  #a0 <- params
198         lw              a0, 44(a0)          #a0 <- params->fd
199         la              t9, buff_write
200         jal             ra, t9
201         bltz            v0, end                #si hay error retornamos
202
203         lw              t0, 44($fp)            #t0=x
204         l.s             $f2, 16($fp)           #f2=cr
205         lw              t2, STACK_SIZE($fp)  #t2=params
206         add             t0, t0,1               #x+1
207         sw              t0, 44($fp)            #x=x+1
208         l.s             $f4, 16(t2)            #f4=params->d_re
209         add.s           $f2, $f2, $f4             #cr=params->d_re
210         s.s             $f2, 16($fp)           #cr=cr-params->d_re
211         j loop_x
212
213 end_x:
214         lw              t0, 48($fp)            #t0=y
215         l.s             $f2, 20($fp)           #f2=ci
216         lw              t2, STACK_SIZE($fp)  #t2=params
217         add             t0, t0,1               #y+1
218         sw              t0, 48($fp)            #y=y+1
219         l.s             $f4, 20(t2)            #f4=params->d_im
```

```
220         sub.s           $f2, $f2, $f4           #ci-params->d_im
221         s.s             $f2, 20($fp)          #ci = ci-params->d_im
222         j loop_y
223
224 end_y:
225      #FLUSH
226         lw              a0, STACK_SIZE($fp)  #a0 <- params
227         lw              a0, 44(a0)          #a0 <- params->fd
228         la              t9, buff_flush
229         jal             ra, t9
230
231 end:  #finalizacion
232         lw               ra, RA_POS(sp)              #restore ra
233         lw               gp, GP_POS(sp)              #restore gp
234         lw               $fp, FP_POS(sp)             #restore fp
235         addu             sp, sp, STACK_SIZE          #restore sp
236         jr               ra
237         .end             mips32_plot
```

../src/mips32_plot.S

```
1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4      .text             # segmento de texto del programa
5
6      .abicalls
7      .align 2         # alineacion 2^2
8
9      .globl     print_int
10     .ent       print_int
11 print_int:
12     # debugging info: descripcion del stack frame
13     .frame     $fp, 40, ra              # $fp: registro usado como
           frame pointer
14                                        # 40: size del stack frame
15                                        # ra: registro que almacena
                                              el return address
16     # bloque para codigo PIC
17     .set     noreorder                 # apaga reordenamiento de
           instrucciones
18     .cpload t9                         # directiva usada para
           codigo PIC
19     .set     reorder                   # enciende reordenamiento de
            instrucciones
20
21     # creo stack frame
22     subu          sp, sp, 40           # 4 (SRA) + 2 (LTA) + 4 (ABA
           )
23
24     # directiva para codigo PIC
25     .cprestore 24                      # inserta aqui "sw gp, 24(sp
           )",
26                                        # mas "lw gp, 24(sp)" luego
                                              de cada jal.
27     # salvado de callee-saved regs en SRA
28     sw            $fp, 28(sp)
```

```
29      sw              ra, 32(sp)
30
31      # de aqui al fin de la funcion uso $fp en lugar de sp.
32      move            $fp, sp
33
34      # salvo 1er arg (siempre)
35      sw              a0, 40($fp)             # n: a0, sp+40
36
37      # salveo 2do arg
38      sw              a1, 44($fp)             # fd: a1, sp+44
39
40      # r = n % 10;
41      remu            t0, a0, 10              # r: t0, sp+16    ## remU
42      sw              t0, 16($fp)
43
44      # n /= 10;
45      lw              a0, 40($fp)             ## redundante
46      divu            a0, a0, 10              ## divU!
47      sw              a0, 40($fp)
48
49      ## if (n > 0)
50      lw              a0, 40($fp)             ## redundante
51      beq             a0, zero, _write_digit # n>0 equivale a n!=0 (n
            unsigned)
52
53      #    print_int(n);
54      lw              a0, 40($fp)             ## redundante
55      la              t9, print_int
56      jal             ra, t9
57
58  _write_digit:
59      # write(1, digit[r], 1);
60      # calculo auxiliar: obtengo digit[r]
61      lw              a0, 16($fp)             # r: a0, sp+16
62      sll             a0, a0, 2               # escalo r con tam de ptr (
            r*=4)
63      lw              a0, digit(a0)           # digit[r]: a0
64      sw              a0, 20($fp)             # almaceno digit[r] en LTA
65
66      # cargo argumentos y llamo a buff_write
67      li              a2, 1
68      lw              a1, 20($fp)             # digit[r] ## (por que no
            hago move a1, a0?)
69      lw              a0, 44($fp)             # fd
70      la              t9, buff_write
71      jal             ra, t9
72
73      # return;
74      # restauro callee-saved regs
75      lw              gp, 24(sp)
76      lw              $fp, 28(sp)
77      lw              ra, 32(sp)
78      # destruyo stack frame
79      addu            sp, sp, 40
80      # vuelvo a funcion llamante
81      jr              ra
82
```

```
83      .end      print_int
84      .size     print_int,.-print_int
85
86
87
88      .rdata              # segmento read-only data
89
90      .align 2
91  digit:    .word digit_0, digit_1, digit_2, digit_3, digit_4, \
92            digit_5, digit_6, digit_7, digit_8, digit_9
93      .size digit, 40
94
95      .align 0        # alineacion 2^0
96
97  digit_0: .asciiz "0"
98  digit_1: .asciiz "1"
99  digit_2: .asciiz "2"
100 digit_3: .asciiz "3"
101 digit_4: .asciiz "4"
102 digit_5: .asciiz "5"
103 digit_6: .asciiz "6"
104 digit_7: .asciiz "7"
105 digit_8: .asciiz "8"
106 digit_9: .asciiz "9"
```

../src/print_int.S

```
1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3
4  #define WRITE_STACK_SIZE        56
5  #define WRITE_RA_POS            52
6  #define WRITE_GP_POS            48
7  #define WRITE_FP_POS            44
8
9  #define FLUSH_STACK_SIZE        16
10 #define FLUSH_GP_POS            12
11 #define FLUSH_FP_POS            8
12
13 #define BUFFER_SIZE             5
14
15 .data
16 .align 2
17 index:  .word -1
18 buffer: .space BUFFER_SIZE
19
20
21 .text
22 .abicalls
23 .align 2
24 .global buff_write
25 .ent buff_write
26
27 buff_write:
28     .frame      $fp, WRITE_STACK_SIZE, ra
29     .set        noreorder
30     .cpload     t9
```

```
31      .set            reorder
32      subu            sp, sp, WRITE_STACK_SIZE
33      sw              ra, WRITE_RA_POS(sp)        #save ra
34      sw              gp, WRITE_GP_POS(sp)        #save gp
35      sw              $fp, WRITE_FP_POS(sp)       #save fp
36      sw              s6, 40(sp)                  #save s6
37      sw              s5, 36(sp)                  #save s5
38      sw              s4, 32(sp)                  #save s4
39      sw              s3, 28(sp)                  #save s3
40      sw              s2, 24(sp)                  #save s2
41      sw              s1, 20(sp)                  #save s1
42      sw              s0, 16(sp)                  #save s0
43      move            $fp, sp                     #fp->sp
44
45      sw              a0, 56($fp)                 #save arg(fd)
46      sw              a1, 60($fp)                 #save arg(str)
47      sw              a2, 64($fp)                 #save arg(size)
48
49      move            s0, a0                      #s0 == fd
50      move            s1, a1                      #s1 == str
51      move            s2, a2                      #s2 == size
52
53      li              s3, 0                       #s3 == result = 0
54      li              s4, 0                       #s4 == i = 0
55      la              s5, buffer                  #s5 == *buf
56      la              t0, index
57      lw              s6, 0(t0)                   #s6 == index
58
59  loop:
60      bge             s4, s2, end                 # if (i >= size)
61      la              t0, index
62      lw              s6, 0(t0)                   #s6 == index
63      add             s6, s6, 1                   #++index
64      la              t0, index
65      sw              s6, 0(t0)                   #save index
66      addu            t0, s1, s4                  #t0 <- *str + i      (
            str[i])
67
68      lb              t0, 0(t0)                   #t0 <- str[i]
69      sb              t0, buffer(s6)              #t0 -> buffer[index]
70
71      li              t0, BUFFER_SIZE
72      subu            t0, t0, 1                   #buffer_size-1
73      bne             s6, t0, continue
74      move            a0, s0                      #a0 <- fd
75      la              t9, buff_flush
76      jalr            t9
77      move            s3, v0                      #s3 <- result
78      bltz            s3, end
79
80  continue:
81
82      addu            s4, s4, 1                   #++i
83      j               loop
84
85  end:
86      #finalizacion
```

```
87      move            v0, s3                          #v0 <- result
88      lw              ra, WRITE_RA_POS(sp)            #restore ra
89      lw              gp, WRITE_GP_POS(sp)            #restore gp
90      lw              $fp, WRITE_FP_POS(sp)           #restore fp
91      lw              s6, 40(sp)                      #restore s6
92      lw              s5, 36(sp)                      #restore s5
93      lw              s4, 32(sp)                      #restore s4
94      lw              s3, 28(sp)                      #restore s3
95      lw              s2, 24(sp)                      #restore s2
96      lw              s1, 20(sp)                      #restore s1
97      lw              s0, 16(sp)                      #restore s0
98      addu            sp, sp, WRITE_STACK_SIZE        #restore sp
99      jr              ra
100     .end            buff_write
101
102
103
104
105 .global buff_flush
106 .ent buff_flush
107
108
109
110
111 buff_flush:
112     .frame          $fp, FLUSH_STACK_SIZE, ra
113
114     subu            sp, sp, FLUSH_STACK_SIZE
115     sw              gp, FLUSH_GP_POS(sp)            #save gp
116     sw              $fp, FLUSH_FP_POS(sp)           #save fp
117     move            $fp, sp                         #fp->sp
118
119     sw              a0, 16($fp)                     #save fd
120     la              a1, buffer                      #a1 <- buffer
121     la              t0, index
122     lw              a2, 0(t0)                       #a2 <- index
123     add             a2, a2, 1                       #a2 <- index + 1
124     li              v0, SYS_write
125     syscall                                         #llamo en sys_write
126                                                     #en v0 me deja el
                                                            resultado
127
128     bnez            a3, quit                        #chequeo a3 para ver
            si
129                                                     #el syscall se
                                                            ejecuto bien
130
131     li              t0, -1                          #index = -1
132     la              t1, index
133     sw              t0, 0(t1)
134     #finalizacion
135     lw              gp, FLUSH_GP_POS(sp)                #restore gp
136     lw              $fp, FLUSH_FP_POS(sp)               #restore fp
137     addu            sp, sp, FLUSH_STACK_SIZE            #restore sp
138     jr              ra
139
140 quit:
```

```
141     li              a0, -1
142     li              v0, SYS_exit
143     syscall
144     .end            buff_flush
```

../src/buff_write.S

B. Enunciado original

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 1: conjunto de instrucciones MIPS
2^o cuatrimestre de 2016

`$Date: 2016/10/02 21:23:43 $`

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 de Julia y sus vecindades introducido en el *TP0* [1], en el cual la lógica de cómputo del fractal deberá tener soporte nativo para MIPS32 sobre NetBSD/pmax.

El código fuente con la versión inicial del programa, se encuentra disponible en [2]. El mismo deberá ser considerado como punto de partida de todas las implementaciones.

4.1. Soporte para MIPS

El entregable producido en este trabajo deberá implementar la lógica de cómputo del fractal en assembly MIPS32, con soporte nativo para NetBSD/pmax.

Para ello, cada grupo deberá tomar el código fuente de base para este TP, [2], y reescribir la función `mips32_plot()` sin cambiar su API. Esta función está ubicada en el archivo `mips32_plot.c`.

4.2. 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.

4.3. Compilación

El código fuente provisto por la cátedra provee los makefiles necesarios para compilar el ejecutable a partir de la versión en C con el archivo `mips32_plot.c`. Para poder compilar el código desarrollado deberán cambiar la definición en el archivo `Makefile.in` la línea número 6:

```
SRCS = mips32_plot.c main.c mygetopt_long.c
```

por

```
SRCS = mips32_plot.S main.c mygetopt_long.c
```

Luego deberán invocar la siguiente secuencia de comandos para limpiar los archivos temporales y generar los nuevos Makefiles:

```
$ make clean
$ make makefiles
$ make
```

4.4. Detalles de la implementación

Para optimizar los accesos a las llamadas a servicio del sistema (`syscalls`), deben utilizar un buffer de n bytes para escribir los datos de salida para luego ser enviados al archivo de salida. El tamaño n debe ser parametrizable mediante un `#define`.

5. Informe

El informe, a entregar en formarto impreso y digital[1] deberá incluir:

- Documentación relevante al diseño e implementación del código esarrollado para adaptar el programa. Incluír el diagrama de *stack frame* de las funciones implementadas en MIPS32.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente. Especificar modificaciones realizadas a los archivos provistos por la cátedra si es que los hubo.
- Las corridas de prueba, con los comentarios pertinentes.[2]
- El código fuente, en lenguaje C (y MIPS32 donde corresponda)
- Este enunciado.

[1] En CD, DVD o memoria flash.

[2] Las pruebas provistas deben ejecutarse correctamente en NetBSD sobre MIPS32 sin modificación alguna.

6. Fecha de entrega

La fecha de vencimiento será el Martes 01/11.

Referencias

[1] Trabajo Práctico 0, 2do cuatrimestre de 2016.
`https://groups.yahoo.com/neo/groups/orga-comp/files/TPs/tp0-2016-2q.pdf`.

[2] Código fuente con el esqueleto del trabajo práctico.
`https://drive.google.com/open?id=0B93s6e6NY_j1TFV2TFBqbUNKZ3M`.