TP 1: Conjunto de instrucciones MIPS

Martín Queija, *Padrón Nro. 96.455*
tinqueija@gmail.com

Estanislao Ledesma, *Padrón Nro. 96.622*
estanislaomledesma@gmail.com

Agustin Luques, *Padrón Nro. 96.803*
agus.luques@hotmail.com

2do. Cuatrimestre de 2016
66.20 Organización de Computadoras – Práctica Martes
Facultad de Ingeniería, Universidad de Buenos Aires

November 1, 2016

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 22:23:34 $`

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 `BUF_SZ` bytes para escribir los datos de salida para luego ser enviados al archivo de salida. El tamaño `BUF_SZ` debe ser configurable, en tiempo de compilación, mediante un `#define`.

```
#ifndef BUF_SZ 8192
#define BUF_SZ 8192
#endif
```

Como podemos ver arriba, el valor por defecto de este parámetro es `8192` bytes.

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`

1 Diseño e implementación

La función mips32_plot implementada con código mips utiliza un stack frame de tamaño variable, ya que esta hace uso de un buffer de almacenamiento en LTA para los syscall de impresión en el archivo, cuyo tamaño se setea previo a la compilación (mediante define). El mismo está compuesto por las cuatro áreas características: SRA (saved register area), la cual almacena los datos de los registros s, fp, sp, ra y gp; FRA (float register area), la cual tiene espacio de almacenamiento para los registros de punto flotante utilizados en el programa (principalmente para los ciclos); LTA (local and temporary area), con espacio para todos los componentes del parámetro de la función (parms, cuyo puntero se haya en la ABA de la caller), para BUF_PTR (tamaño del buffer para flushear), RA SAVE buffer flush (para guardar ra antes de flushear), a0 previo al flush y espacios para algunos registros temporales (t) y s, cuando se llama a alguna función auxiliar. También posee el buffer, de tamaño BUF_SZ, con lo cual, a partir de esta posición, todas las posiciones del stack se indican con constantes definidas. Por último, también posee el ABA (argument building area) con el espacio típico: cuatro palabras (una por cada registro a). El diagrama del mismo se encuentra en la tabla 1.

También se implementaron cuatro pseudo-funciones, print-headers (para imprimir los encabezados), shade_a_buffer (para escribir el shade en el buffer), flush_buffer (para flushear todo lo escrito en el buffer) e inttostring (para obtener el string de un entero a partir de este último). Se trata de pseudo-funciones ya que no se reservó espacio para stack frames de cada una, sino que se trabajó con registros directamente que escriben en el buffer alocado de la función principal mips32_p?ot. Para acceder a estas, siempre primero se salva los registros en la LTA de la mips32_plot, ra, a0, y todos aquellos que sean sobreescritos en la función llamada, que luego deban ser utilizados.

2 Compilación

Para compilar la función implementada en Mips, se utilizó Gxemul para emular un procesador MIPS R3000 con una imagen de sistema operativo NETBSD 3.0 con plataforma pmax. Este se ejecutó con soporte X11, mediante la generación de un túnel entre el HostOS y el GuestOS con SSH/SSHD. La transferencia de archivos (los necesarios para compilar el tp) se hizo con la utilidad SCP. Para compilar mips32_plot.S junto con el main.c y demás archivos necesarios se utilizó el makefile provisto por la cátedra. Para esto se emiten los comandos “make clean”, “make makefiles” y “make” en ese orden, obteniendo así el ejecutable. El archivo de salida se puede enviar al sistema operativo HostOS mediante SCP nuevamente.

3 Código MIPS

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

#ifndef BUF_SZ
```

```
#define BUF_SZ 8192
#endif

#define BUF_OFFSET 100 + BUF_SZ
#define BUF_LOW_OFFSET 104


#define FP0_OFFSET BUF_OFFSET + 4
#define FP1_OFFSET FP0_OFFSET + 4
#define FP2_OFFSET FP1_OFFSET + 4
#define FP3_OFFSET FP2_OFFSET + 4
#define FP4_OFFSET FP3_OFFSET + 4
#define FP5_OFFSET FP4_OFFSET + 4
#define FP6_OFFSET FP5_OFFSET + 4
#define FP7_OFFSET FP6_OFFSET + 4
#define FP8_OFFSET FP7_OFFSET + 4
#define FP9_OFFSET FP8_OFFSET + 4
#define FP10_OFFSET FP9_OFFSET + 4
#define FP11_OFFSET FP10_OFFSET + 4


#define S0_OFFSET FP11_OFFSET + 4
#define S1_OFFSET S0_OFFSET + 4
#define S2_OFFSET S1_OFFSET + 4
#define S3_OFFSET S2_OFFSET + 4
#define S4_OFFSET S3_OFFSET + 4
#define S5_OFFSET S4_OFFSET + 4
#define S6_OFFSET S5_OFFSET + 4
#define S7_OFFSET S6_OFFSET + 4



#define SP_OFFSET S7_OFFSET + 4
#define GP_OFFSET SP_OFFSET + 4
#define FP_OFFSET GP_OFFSET + 4
#define RA_OFFSET FP_OFFSET + 4
#define STACK_SZ RA_OFFSET + 4

        .text
        .abicalls
        .align 2
        .globl  mips32_plot
        .ent    mips32_plot

mips32_plot:
        .frame  $fp, STACK_SZ, ra

        .set    noreorder
        .cpload t9
        .set    reorder
```

```
        subu sp, sp, STACK_SZ                   # 4(SRA) + 15 + BUF_SZ (LTA) +

        .cprestore GP_OFFSET                    #guardo gp en 80 + sp
        sw $fp, FP_OFFSET(sp)                   #guardo $fp en 84 + sp
        sw ra, RA_OFFSET(sp)                    #guardo ra en 88 + sp

        move $fp, sp                            #$fp = sp (desde aca se usa $fp


        sw a0, STACK_SZ($fp)                    #guardo el parametro(*parms) en

#
#       Guardo la data de *params en el stack local.
#

        lw t0, 0(a0)
        sw t0, 16($fp)                                  #guardo UL_Re en 16 + $

        lw t0, 4(a0)
        sw t0, 20($fp)                                  #guardo UL_Im en 20 + $

        lw t0, 8(a0)
        sw t0, 24($fp)                                  #guardo LR_Re en 24 + $

        lw t0, 12(a0)
        sw t0, 28($fp)                                  #guardo LR_Im en 28 + $

        lw t0, 16(a0)
        sw t0, 32($fp)                                  #guardo D_Re en 32 + $f

        lw t0, 20(a0)
        sw t0, 36($fp)                                  #guardo D_Im en 36 + $f

        lw t0, 24(a0)
        sw t0, 40($fp)                                  #guardo CP_Re en 40 + $

        lw t0, 28(a0)
        sw t0, 44($fp)                                  #guardo CP_Im en 44 + $

        lw t0, 32(a0)
        sw t0, 48($fp)                                  #guardo X_RES en 48 + $

        lw t0, 36(a0)
        sw t0, 52($fp)                                  #guardo Y_RES en 52 + $

        lw t0, 40(a0)
        sw t0, 56($fp)                                  #guardo SHADES en 56 +

        lw t0, 44(a0)
```

```
        sw t0, 60($fp)                                  #guardo FILE POINTER en

#
#       Guardo los registros flotantes y caller saved registers que voy a usar
#
error:                                          #Si vuelvo de error restauro todo, en v
        s.s $f0, FP0_OFFSET($fp)
        s.s $f1, FP1_OFFSET($fp)
        s.s $f2, FP2_OFFSET($fp)
        s.s $f3, FP3_OFFSET($fp)
        s.s $f4, FP4_OFFSET($fp)
        s.s $f5, FP5_OFFSET($fp)
        s.s $f6, FP6_OFFSET($fp)
        s.s $f7, FP7_OFFSET($fp)
        s.s $f8, FP9_OFFSET($fp)
        s.s $f9, FP9_OFFSET($fp)
        s.s $f10, FP10_OFFSET($fp)
        s.s $f11, FP11_OFFSET($fp)

        sw s0, S0_OFFSET($fp)
        sw s1, S1_OFFSET($fp)
        sw s2, S2_OFFSET($fp)
        sw s3, S3_OFFSET($fp)
        sw s4, S4_OFFSET($fp)
        sw s5, S5_OFFSET($fp)
        sw s6, S6_OFFSET($fp)
        sw s7, S7_OFFSET($fp)

        l.s $f6, 40($fp) #f6: cpr = parms->cp_re
        l.s $f7, 44($fp) #f7: cpi = parms->cp_im

        sw zero, 64($fp)                                #inicializo el BUF_PTR

################################################
################################################


        jal print_header


                                                        #Inicializaciones para

        addu s1, zero, zero #s1: y = 0
        l.s $f0, 20($fp) #f0: ci = parms->UL_Im
        lw t1, 52($fp) #t1: parms-> y_res


for1:
        bge s1, t1, fin #si y >= y_res => fin
```

```
                                                        #Inicializaciones para

        addu s2, zero, zero                             #s2: x = 0
        l.s $f1, 16($fp)                                #f1: cr = parms->UL_Re
        lw t2, 48($fp)                                  #t2: parms-> x_res

for2:
        bge s2, t2, fin_for1                            #si x >= x_res => fin_f
        mov.s $f2, $f1                                  #f2: zr = cr
        mov.s $f3, $f0                                  #f3: zi = ci

                                                        #Inicializaciones para
        addu s3, zero, zero                             #s3: c = 0
        lw t3, 56($fp)                                  #t3: parms->shades

for3:
        bge s3, t3, fin_for2                            #si c >= shades => fin_
        mul.s $f4, $f2, $f2                             #f4: zr * zr
        mul.s $f5, $f3, $f3                             #f5: zi * zi
        add.s $f4, $f4, $f5                             #f4: absz = zr * zr + z
        l.s $f5, CTE_FOR3                                       #f5 = CTE_FOR3
        c.lt.s $f5, $f4
#si CTE_FOR3 < absz => code = 1
        bc1t fin_for2                                           #si code = 1 =>
        mul.s $f4, $f2, $f2                             #f4: zr * zr
        mul.s $f5, $f3, $f3                             #f5: zi * zi
        sub.s $f4, $f4, $f5                             #f4: zr * zr - zi * zi
        add.s $f4, $f4, $f6                             #f4: sr = zr * zr - zi
        mul.s $f5, $f2, $f3                             #f5: zr * zi
        l.s $f8, CTE_MULT                               #f8: CTE_MULT
        mul.s $f5, $f5, $f8                             #f5: 2 * zr * zi
        add.s $f5, $f5, $f7                             #f5: si = 2 * zr * zi +
        mov.s $f2, $f4                                  #f2: zr = sr
        mov.s $f3, $f5                                  #f3: zi = si
        addi s3, s3, 1                                  #s3: ++c
        b for3
fin_for2:

        sw s1, 80($fp)
        sw s2, 84($fp)
        sw s3, 88($fp)  ### ESTOS SON LOS REGISTROS QUE USA EL CICLO
        sw t1, 92($fp) ###tienen lugares especiales en el stack si necesitas gu
        sw t2, 96($fp)
        sw t3, 100($fp)


        move a0, s3
        jal shade_a_buffer

        lw s1, 80($fp)
```

```
        lw s2, 84($fp)
        lw s3, 88($fp)
        lw t1, 92($fp)  ### REPONGO LOS REGISTROS QUE USA EL CICLO
        lw t2, 96($fp)
        lw t3, 100($fp)


        addi s2, s2, 1                                  #s2: ++x
        l.s $f8, 32($fp)                                #f8: parms->d_re
        add.s $f1, $f1, $f8                             #f1: cr += parms->d_re
        b for2                                          #vuelve a for2
fin_for1:
        addi s1, s1, 1                                  #s1: ++y
        l.s $f8, 36($fp)                                #f8: parms->d_im
        sub.s $f0, $f0, $f8                     #f0: ci -= parms->d_im
        b for1                                          #vuelve a for1




fin:
#Antes de terminar, flusheo lo que queda en el buffer
        sw ra, 68($fp)
        sw a0, 72($fp)
        lw a2, 64($fp)                                  #Establezco en a2 la ca
        jal flush_buffer
        lw ra, 68($fp)
        lw a0, 72($fp)
        #Reestablezco los registros flotantes y caller saved registers a sus va

        l.s $f0, FP0_OFFSET($fp)
        l.s $f1, FP1_OFFSET($fp)
        l.s $f2, FP2_OFFSET($fp)
        l.s $f3, FP3_OFFSET($fp)
        l.s $f4, FP4_OFFSET($fp)
        l.s $f5, FP5_OFFSET($fp)
        l.s $f6, FP6_OFFSET($fp)
        l.s $f7, FP7_OFFSET($fp)
        l.s $f8, FP9_OFFSET($fp)
        l.s $f9, FP9_OFFSET($fp)
        l.s $f10, FP10_OFFSET($fp)
        l.s $f11, FP11_OFFSET($fp)

        lw s0, S0_OFFSET($fp)
        lw s1, S1_OFFSET($fp)
        lw s2, S2_OFFSET($fp)
        lw s3, S3_OFFSET($fp)
        lw s4, S4_OFFSET($fp)
```

```
        lw s5, S5_OFFSET($fp)
        lw s6, S6_OFFSET($fp)
        lw s7, S7_OFFSET($fp)

        move sp, $fp                            #restauro valor de sp
        lw ra, RA_OFFSET(sp)                    #restauro valor de ra
        lw $fp, FP_OFFSET(sp)                   #restauro valor de $fp

        addu sp, sp, STACK_SZ                   #subo stack pointer

        jr ra                                   #vuelvo a la funcion caller


##################################################
##################################################

#Int to String: el int va en a0, cantidad de chars guardados en v0

inttostr:
        addi t0, zero, 10               #establezco divisor base 10
        move t1, zero                   #t1=0
        la t2, stringbuffer             #t2=stringbufferadress
        add t3, a0, zero                #t3=a0 (int a imprimir)
loop1:  div t3, t0                      #t3/10 division:t4 resto:t5
        mflo t4
        mfhi t5
        addi t1, t1, 1                  #aumento contador de caracteres
        bne  t3, 0, loop1               #cuando la division sea 0 ya termino el
        add t3, a0, zero                #t3=a0 (int a imprimir)
        move v0, t1                     #v0=cant chars guardados
loop2:  div t3, t0                      #vuelvo a divir esta vez guardando en e
        mflo t4
        mfhi t5
        addi t1, t1, -1                 #decremento el contador de chars
        add t6, t2, t1                  #t6=offset en string buffer = stringbuf
        addi t5, t5, 48                 #le sumo 48 al valor del resto para que
        sb t5, 0(t6)                    #lo guardo en t6
        bne t3, 0, loop2                #cuando la division sea 0 ya temrino el
        jr ra

##################################################

#Shade a buffer: Escribe el int en a0 al buffer, si el buffer se llena se flush

shade_a_buffer:
        addi s0, $fp, BUF_LOW_OFFSET    #s0 = BUFFER_LOW
        lw s1, 64($fp)                  #Cargo a s1 el BUF_PTR
        add s5, s0, s1                  #s5 = BUFFER_LOW + BUF_PTR

        sw ra, 68($fp)
```

```
        jal inttostr
        lw ra, 68($fp)

        move s2, v0

        move s6, zero                   #s6 = 0
        la s7, stringbuffer             #s7 = stringbufferadress
loop:   add s4, s7, s6                  #s4 = stringbufferadress + s6
        addi s6, s6, 1                  #s6 = s6 + 1

        lb t2, 0(s4)                    #guardo el primer byte del int de dir(s
        sb t2, 0(s5)                    #lo guardo en s5

        addi s1, s1, 1                  #incremento BUF_PTR
        addi s5, s5, 1                  #incremento el puntero al buffer
        bne s1, BUF_SZ, next            #si BUF_PTR no es igual a BUF_SZ NO flu


        sw ra, 68($fp)
        sw a0, 72($fp)
        li a2, BUF_SZ                   #Establezco la cantidad a flushear (tod
        jal flush_buffer
        lw ra, 68($fp)
        lw a0, 72($fp)
        addi s1, zero, 0                #flushee, entonces reseteo el buffptr
        add s5, s0, zero                #reseteo el puntero al buffer
next:
        addi s2, s2, -1                 #decremento la cantidad de chars guarda
        bne  s2, 0, loop                #si la cantidad de chars es != 0 loopeo

        addi t4, zero, 10               #asci code new line
        sb t4, 0(s5)

        addi s1, s1, 1                  #incremento BUF_PTR
        addi s5, s5, 1                  #incremento el puntero al buffer
        bne s1, BUF_SZ, next2           #si BUF_PTR no es igual a BUF_SZ NO flu


        sw ra, 68($fp)
        sw a0, 72($fp)
        li a2, BUF_SZ                   #Establezco la cantidad a flushear (tod
        jal flush_buffer
        lw ra, 68($fp)
        lw a0, 72($fp)
        addi s1, zero, 0                #flushee, entonces reseteo el buffptr
        add s5, s0, zero                #reseteo el puntero al buffer
        jr ra

next2:  sw s1, 64($fp) # guardo el buff ptr en el stack
        jr ra
```

```
################################################

#Flush Buffer: flushea el buffer en su totalidad al File Pointer en 60(sp), req

flush_buffer:
        li v0, SYS_write                #cargo la instrucción para escribir en
        lw a0, 60(sp)                   #cargo el file pointer en a0
        addi a1, $fp, BUF_LOW_OFFSET #cargo el offset del buffer en a1
        syscall
        blt v0, zero, error             #si v0 < 0 go to error
        sw zero, 64($fp)                #Reseteo el buff ptr
        jr ra


################################################

#Print header: Imprime los datos del encabezado de la imagen

print_header:
        move s6, zero                   #s6 = 0 (buffer pointer)
        addi s0, $fp, BUF_LOW_OFFSET    #s0 = BUFFER_LOW
        add s6, s6, s0                  #s6 = BUFFER_LOW + s6 (buffer pointer)
        addi t0, zero, 80               #80 = 'P'
        sb t0, 0(s6)                    #'P' en buffer
        addi s6, s6, 1                  #s6++
        addi t0, zero, 50               #50 = '2'
        sb t0, 0(s6)                    #'2' en buffer
        addi s6, s6, 1                  #s6++
        addi t0, zero, 10               #10 = '\n'
        sb t0, 0(s6)                    #'\n' en buffer
        addi s6, s6, 1                  #s6++
        la s7, stringbuffer             #s7 = stringbufferadress
        lw a0, 48($fp)                  #Cargo en a0 el valor de x_res
        sw ra, 68($fp)                  #Guardo la dirección de retorno en 68($
        jal inttostr                    #llamo a inttostr
        lw ra, 68($fp)                  #Recupero la dirección de retorno

        add t2, zero, zero              #t2 = 0
looph1: add t1, t2, s7          #t1 = stringbufferaddress + t2(contador de cara
        lb t0, 0(t1)                    #primer elemento del stringbuffer en t0
        sb t0, 0(s6)                    #almaceno primer elemento del stringbuf
        addi t2, t2, 1                  #t2++
        addi s6, s6, 1                  #s6++
        bne v0, t2, looph1              #cantidad de caracteres a imprimir != c
        addi t0, zero, 10               #10 = '\n'
        sb t0, 0(s6)                    #'\n' en buffer
        addi s6, s6, 1                  #s6++

        lw a0, 52($fp)                  #Cargo en a0 el valor de y_res
```

```
        sw ra, 68($fp)                  #Guardo la dirección de retorno en 68($
        jal inttostr                    #llamo a inttostr
        lw ra, 68($fp)                  #Recupero la direccion de retorno
        la s7, stringbuffer             #s7 = stringbufferadress
        add t2, zero, zero              #t2 = 0 (contador de caracteres)
looph2: add t1, t2, s7          #t1 = stringbufferaddress + t2(contador de cara
        lb t0, 0(t1)                    #primer elemento del stringbuffer en t0
        sb t0, 0(s6)                    #almaceno primer elemento del stringbuf
        addi t2, t2, 1                  #t2++
        addi s6, s6, 1                  #s6++
        bne v0, t2, looph2              #cantidad de caracteres a imprimir != c
        addi t0, zero, 10               #10 = '\n'
        sb t0, 0(s6)                    #'\n' en buffer
        addi s6, s6, 1                  #s6++

#       Repito pasos anteriores con shades
        lw a0, 56($fp)                  #Cargo en a0 el valor de shades
        sw ra, 68($fp)
        jal inttostr                    #llamo a inttostr
        lw ra, 68($fp)
        la s7, stringbuffer             #s7 = stringbufferadress
        add t2, zero, zero
looph3: add t1, t2, s7
        lb t0, 0(t1)
        sb t0, 0(s6)
        addi t2, t2, 1
        addi s6, s6, 1
        bne v0, t2, looph3
        addi t0, zero, 10               #10 = '\n'
        sb t0, 0(s6)
        addi s6, s6, 1

        addi s0, $fp, BUF_LOW_OFFSET    #s0 = BUFFER_LOW
        sub a2, s6, s0                  #Establezco la cantidad a flushear
        sw ra, 68($fp)                  #Guardo la dirección de retorno en 68($
        sw a0, 72($fp)                  #Guardo a0 en 72($fp)
        jal flush_buffer                #llamo a flush_buffer
        lw ra, 68($fp)                  #Recupero la dirección de retorno
        lw a0, 72($fp)                  #Recupero a0
        jr ra



        .end mips32_plot


.data
stringbuffer:   .space 32
CTE_FOR3: .float 4.0
CTE_MULT: .float 2
```

References

[1] Conjunto de Julia, WikiPedia, https://es.wikipedia.org/wiki/Conjunto_de_Julia.

[2] Arquitectura de Computadores-John L. Hennessy - David A. Patterson - Un Enfoque Cuantitativo