U.B.A. FACULTAD DE INGENIERÍA

Departamento de Electrónica
Organización de Computadoras 66-20
Ing. Informática

Trabajo práctico Nº01
Julia Set - MIPS

Apellido y Nombre:	*Padrón:*
Cabrera, Jorge	93310
Capolupo, Mauro	90283

Fecha de Entrega : 08/11/2016
Fecha de Reentrega : 29/11/2016
Fecha de Aprobación :
Calificación :
Firma de Aprobación :

1 Diseño e implementación del programa

En éste trabajo se implementa en assembly MIPS un algoritmo sencillo, que permite dibujar el conjunto de Julia y sus vecindades correspondiente a un polinomio cuadrático, según un determinado conjunto de parámetros que serán ingresados por línea de comando.

Su implementación puede dividirse en dos etapas:

- Etapa 1: se lee los parámetros recibidos por parámetros y se los parsea.
- Etapa 2: se procede a ejecutar el algoritmo que calcula los puntos que, dado un C, pertenecen al conjunto de julia.

1.1 Parámetros

- -r: setea la resolución de la imágen generada
- -c: especifica el centro de la imágen
- -C: indica el complejo que se le va a sumar en cada iteración al número complejo asociado a cada pixel,

$$z^2 + C$$

- -w: ancho del ractángulo de la región del plano complejo a dibujar
- -H: alto del ractángulo de la región del plano complejo a dibujar
- -o: permite colocar la imagen de salida en el archivo pasado como parámetro o stdout en caso de que el argumento sea '-'

2 Comando(s) para compilar el programa en NetBSD

gcc -Wall -g -O -I. -DLINUX3 -o tp1 main.c mygetopt_long.c mips32_plot.S

3 Comando(s) para ejecutar el programa en NetBSD

./tp1 -o Julia.pgm

4 Pruebas

Prueba 1

Generamos el conjunto Julia con parámetros válidos

./tp1 -r 640x480 -c 1-1i -w 4 -H 2 -o Julia_1.pgm

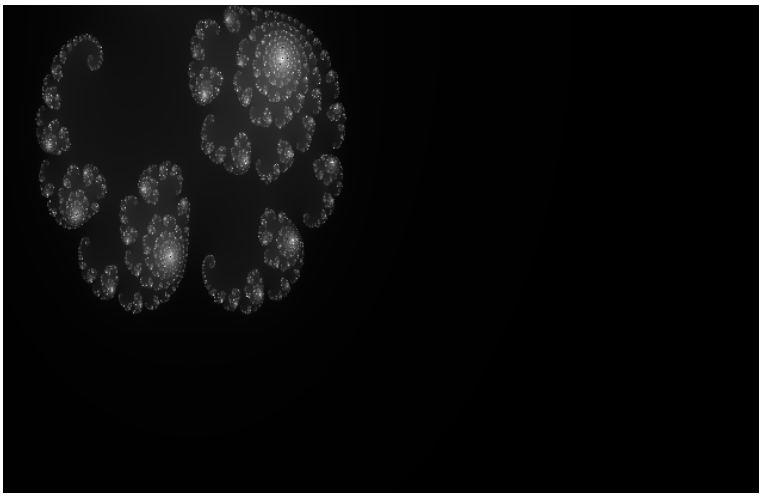

Figure 1: Prueba 1

Prueba 2

Generamos el conjunto Julia con parámetros válidos

./tp1 -C 0.376-0.1566i -c 0+0i -w 4 -H 4 -o Julia_2.pgm

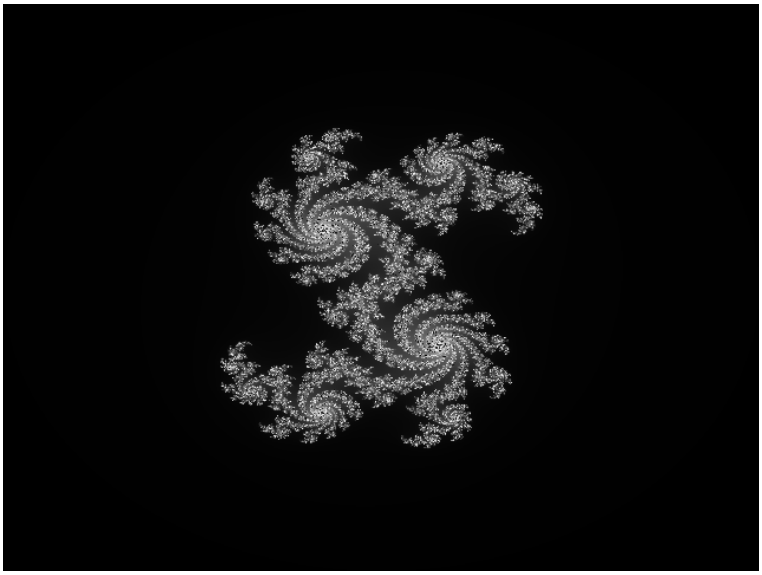

Figure 2: Prueba 2

Prueba 3

Generamos el conjunto Julia con parámetros válidos

./tp1 -r 480x240 -c -0.1+0.1i -w 4 -H 2 -o Julia_3.pgm

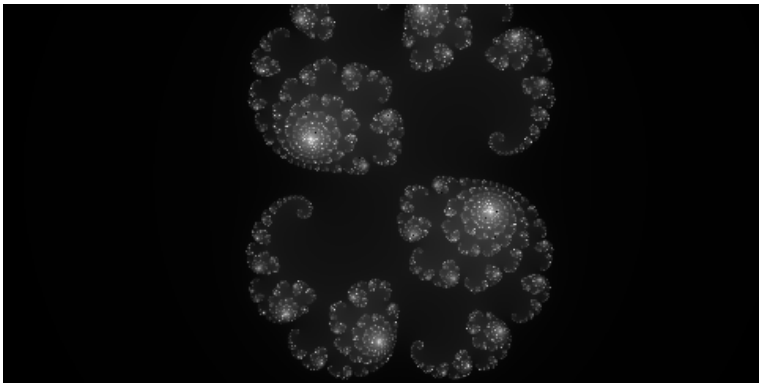

Figure 3: Prueba 3

Prueba 4

Generamos el conjunto Julia con parámetros válidos sin especificar c

./tp1 -r 640x480 -w 4 -H 2 -o Julia_4.pgm

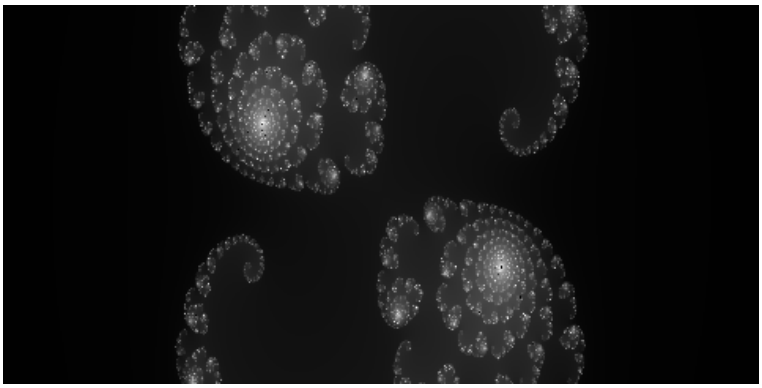

Figure 4: Prueba 4

5 Pruebas casos de errores

Prueba 1

Parámetro r inválido

./tp1 -r 640-480 -c 1-1i -w 4 -H 2 -o Julia.pgm

Se esperaba que el caracter delimitador sea 'x'. El programa nos mostrará las opciones válidas.

Prueba 2

Parámetro w inválido

./tp1 -r 640x480 -c 1-1i -w A -H 2 -o Julia.pgm

Los valores tanto de 'w' como de 'H' deben ser valores enteros. El programa nos mostrará el siguiente mensaje

invalid width specification.

Prueba 3

Parámetro c inválido

./tp1 -r 640x480 -c a-bi -w 2 -H 2 -o Julia.pgm

Los valores correspondientes a la parte real e imaginaria del parámetro complejo c deben ser reales. El programa nos mostrará el siguiente mensaje

invalid center specification.

Prueba 4

Parámetro r inválido

./tp1 -r 0x480 -c 2-3i -w 2 -H 2 -o Julia.pgm

Tanto el ancho como el alto de la imágen deben ser mayores a 0. El programa nos mostrará las opciones válidas.

Prueba 5

Parámetro c inválido

./tp1 -r 240x480 -c 2-3 -w 2 -H 2 -o Julia.pgm

El valor del parámetro c debe ser un número complejo. El programa nos mostrará el siguiente mensaje

invalid center specification.

6 Codigo

6.1 Código C

```
#include <debug.h>
#include <stdio.h>
#include <defs.h>
#include <param.h>

int
mips32_plot(param_t *parms)
{
        float cr, ci;
        float zr, zi;
        float sr, si;
        float absz;
        int x, y;
        int c;
        int res;

        float cpr = parms->cp_re;
        float cpi = parms-> cp_im;

        // Notar que esta implementaci n hace uso de un file pointer. Es no es
        // lo que debe hacerse en MIPS32. Debe hacerse uso del syscall write
        // con el file descriptor provisto en la estructura param_t
        FILE *fp = fdopen(parms->fd, "w");

        /* Header PGM. */
        res = fprintf(fp, "P2\n%u\n%u\n%u\n",
                (unsigned)parms->x_res,
                (unsigned)parms->y_res,
                (unsigned)parms->shades);
        if (res < 0) {
                fprintf(stderr, "io error.\n");
                return -1;
        }

        /*
         * Barremos la regi n rectangular del plano complejo comprendida
         * entre (parms->UL_re, parms->UL_im) y (parms->LR_re, parms->LR_im).
         * El par metro de iteraci n 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 f rmula 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 - zi * zi + cpr;
                                si = 2 * zr * zi + cpi;
                                zr = sr;
                                zi = si;
                        }

                        if (fprintf(fp, "%u\n", (unsigned)c) < 0) {
                                fprintf(stderr, "i/o error.\n");
                                return -1;
                        }
                }
        }

        /* Flush any buffered information before quit. */
        if (fflush(fp) != 0) {
                fprintf(stderr, "cannot flush output file.\n");
                return -1;
        }
        return 0;
}
```

6.2 Implementación MIPS

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

BUFFSIZE = 4096

#Offsets
A0_OFF = 0
A1_OFF = 4
A2_OFF = 8
A3_OFF = 12

CR_OFF = 16
CI_OFF = 20
ZR_OFF = 24
ZI_OFF = 28
SR_OFF = 32
SI_OFF = 36
ABSZ_OFF = 40

X_OFF = 44
Y_OFF = 48
C_OFF = 52

RES_OFF = 56
CPR_OFF = 60
CPI_OFF = 64
FP_OFF = 68

GP_OFF = 72
FP_OFF = 76
GP_OFF = 80

.text
.ent mips32_plot
.globl mips32_plot

mips32_plot:
        subu    sp,sp,88
        sw      gp, GP_OFF(sp)
        sw      $fp, FP_OFF(sp)
        sw      ra, GP_OFF(sp)
        move    $fp,sp
        sw      a0,88($fp)

        #float cpr = parms->cp_re;
        l.s     $f0,24(a0)
        s.s     $f0, CPR_OFF($fp)

        #float cpi = parms-> cp_im;
        l.s     $f0, 28(a0)
        s.s     $f0, CPI_OFF($fp)

        #contador de bytes a escribir.
        move    s1, zero

        #hago append del header en el buffer.
        la      t0, header
        la      t1, buffer

        lb      t2,0(t0)
        sb      t2,0(t1)
        lb      t2,1(t0)
        sb      t2,1(t1)
        lb      t2,2(t0)
        sb      t2,2(t1)

        #s1 va a tener el numero de bytes que agregue al buffer.
        addiu   s1, s1, 3

        #levanto params->x_res y le hago append al buffer.
append_x_res:
        lw      t0, 88($fp)
        lw      s2, 32(t0) # x
        la      t9, append_to_buffer
        jal     ra, t9

        la      t9, append_y_res
        b       append_new_line

append_y_res:
        lw      t0, 88($fp)
        lw      s2, 36(t0) #y_res
        la      t9, append_to_buffer
        jal     ra, t9

        la      t9, append_shades
        b       append_new_line
```

```
append_shades:
        lw      t0, 88($fp)
        lw      s2, 40(t0)              #shades
        la      t9, append_to_buffer
        jal     ra, t9

        la      t9, calculate_fractal
        b       append_new_line

append_new_line:
        la      t1, buffer
        la      t3, endline
        lb      t3, 0(t3)
        addu    t2,t1,s1
        sb      t3,0(t2)
        addiu   s1,s1,1
        jr t9

calculate_fractal:
        sw      v0, RES_OFF($fp)
        bgez    v0, pre_for

pre_for:
        lw      t0,88($fp)
        sw      zero, Y_OFF($fp)        # y = 0
        sw      zero, X_OFF($fp)        # x = 0
        sw      zero, C_OFF($fp)        # c = 0
        lw      t0, 88($fp)             # t0 = &parms_t
        l.s     $f2, 4(t0)
        s.s     $f2, CI_OFF($fp)        # ci = UL_im
        l.s     $f4, 0(t0)
        s.s     $f4, CR_OFF($fp)        # cr = UL_re

first_for:
        lw      t0, 88($fp)
        lw      t1, Y_OFF($fp)          # t1 = y
        lw      t2,     36(t0)          # t2 = y_res
        bge     t1, t2, end_first_for

second_for:
        lw      t0, 88($fp)
        lw      t3, X_OFF($fp)          # t3 = x
        lw      t4, 32(t0)              # t4 = x_res
        bge     t3, t4, end_second_for

        l.s     $f6,CR_OFF($fp)         # $f6 = cr
        s.s     $f6,ZR_OFF($fp)         # zr = cr
        l.s     $f8,CI_OFF($fp)         # $f8 = ci
        s.s     $f8,ZI_OFF($fp)         # zi = ci

        #for (c = 0; c < parms->shades; ++c) {
third_for:
        lw      t0,88($fp)
        lw      t7,C_OFF($fp)           # t7 = c
        lw      t8,40(t0)               # t8 = shades
        bge     t7,t8,end_third_for

        l.s     $f2,ZR_OFF($fp)         # $f2 = zr
        l.s     $f4,ZI_OFF($fp)         # $f4 = zi
        l.s     $f6,60($fp)             # $f6 = cpr
        l.s     $f8,64($fp)             # $f8 = cpi

        # ( (absz = zr*zr + zi*zi) > 4.0f) ? break:continue
        mul.s   $f10,$f2,$f2
        mul.s   $f12,$f4,$f4
        add.s   $f10,$f10,$f12
        s.s     $f10,ABSZ_OFF($fp)      # absz = zr * zr + zi * zi
        li.s    $f12,4.0
        c.le.s  $f12,$f10
        bc1t    end_third_for

        #sr = zr * zr - zi * zi + cpr;
        mul.s   $f10,$f2,$f2
        mul.s   $f12,$f4,$f4
        sub.s   $f12,$f10,$f12
        add.s   $f12,$f12,$f6           # $f12 = sr
        s.s     $f12,SR_OFF($fp)

        #si = 2 * zr * zi + cpi;
        mul.s   $f10,$f2,$f4
        li.s    $f14,2.0
        mul.s   $f10,$f10,$f14
        add.s   $f10,$f10,$f8           # $f10 = si
        s.s     $f10,SI_OFF($fp)

        #zr = sr;
        s.s     $f12,ZR_OFF($fp)

        #zi = si;
        s.s     $f10,ZI_OFF($fp)
```

```
        #++c
        lw      t7,C_OFF($fp)
        addi    t7,t7,1
        sw      t7,C_OFF($fp)
        b       third_for

end_first_for:
        sw      zero,Y_OFF($fp)         # y = 0
        lw      t0,88($fp)
        l.s     $f2,4(t0)
        s.s     $f2,CI_OFF($fp)         # ci = UL_im

        #Si no tengo bytes acumulados -> finish_ok
        beqz    s1,finish_ok

        #Flush de los bytes acumulados
        li      v0,SYS_write
        lw      t0,88($fp)
        lw      a0,44(t0)               #params->fd

        la      a1,buffer
        move    a2,s1
        syscall

        blt     v0,zero,showError
        li      v0,0
        b       finish_ok

        #++y, ci -= parms->d_im
end_second_for:
        sw      zero,X_OFF($fp)         # x = 0
        lw      t0,88($fp)
        l.s     $f4,0(t0)
        s.s     $f4,CR_OFF($fp)         # cr = UL_re
        lw      t1,Y_OFF($fp)           # t1 = y
        addi    t1,t1,1
        sw      t1,48($fp)
        l.s     $f2,CI_OFF($fp)         # $f2 = ci
        l.s     $f4,20(t0)              # $f4 = d_im
        sub.s   $f4,$f2,$f4
        s.s     $f4,CI_OFF($fp)
        b       first_for

end_third_for:

        li      t0,BUFFSIZE

        #Si la cantidad de bytes acumulados supero el buffer, hago un flush del mismo.
        bge     s1,t0,writeToFile

        #cargo en s2 el valor de C
        lw      t1,C_OFF($fp)
        move    s2,t1
        la      t9, append_to_buffer
        jal     ra, t9
        b writeEndline

writeToFile:
        li      v0,SYS_write            #write syscal value
        lw      t0,88($fp)
        lw      a0,44(t0)               #params->fd
        la      a1,buffer
        move    a2,s1
        syscall

        blt     v0,zero,showError
        move    s1,zero
        la      t1,buffer
        b       end_third_for

writeEndline:
        li      t0,BUFFSIZE
        bge     s1,t0,writeToFileAndFallbackWriteEndline

        la      t9, increment_second_for
        b       append_new_line

writeToFileAndFallbackWriteEndline:
        li      v0,SYS_write            #write syscal value
        lw      t0,88($fp)
        lw      a0,44(t0)               #params->fd
        la      a1,buffer
        move    a2,s1
        syscall

        blt     v0,zero,showError
        la      t1,buffer
        move    s1,zero

        b       writeEndline
```

```
        #++x, cr += parms->d_re
increment_second_for:
        sw      zero,C_OFF($fp)         # c = 0
        lw      t0,88($fp)
        lw      t1,X_OFF($fp)           # t1 = x
        addi    t1,t1,1
        sw      t1,X_OFF($fp)
        l.s     $f2,CR_OFF($fp)         # $f2 = cr
        l.s     $f4,16(t0)              # $f4 = d_re
        add.s   $f4,$f2,$f4
        s.s     $f4,CR_OFF($fp)
        b       second_for

showError:
        li      v0,SYS_write
        li      a0,2
        la      a1,error
        li      a2,10
        syscall
        b       finish_error

finish_error:
        li      v0,-1
        b       finish

finish_ok:
        li      v0,0

finish:
        move    sp,$fp
        lw      ra,80(sp)
        lw      $fp,76(sp)
        lw      gp, GP_OFF(sp)
        addu    sp,sp,88

        jr      ra

        .end    mips32_plot


# Function append_to_buffer

.ent append_to_buffer
append_to_buffer:

        .set noreorder
        .cpload t9
        .set reorder

        subu    sp, sp, 8
        sw      gp, 0(sp)
        sw      $fp, 4(sp)

#Bloque de c digo para convertir un entero a char.
#Dado un entero, por ej 640, va diviendo por 10 para reducir a orden 1 (0 a 9) y asi utilizarlo como
#offset del arreglo digits

        li      t0, 10
        li      t7, 10
        li      t6, 1

getOrder:
        divu    t1, s2, t7
        blt     t1, t6, endGetOrder
        mul     t7, t7, 10
        b       getOrder

endGetOrder:
        divu    t7,t7,10

digitToChar:
        blt     t7, t6, end
        divu    t1, s2, t7
        la      t2, digits
        addu    t2, t2, t1              #offset del array digit
        lb      t1, 0(t2)               #digit as char

    la          t3, buffer
        addu    t3, t3, s1
        sb      t1, 0(t3)
        addiu   s1, s1, 1

        remu    s2, s2, t7
        divu    t7, t7, 10

        b       digitToChar

end:
        move v0, s1
        lw      gp, 0(sp)
        lw      $fp, 4(sp)
```

```
        addu sp, sp, 8
        jr ra

.end    append_to_buffer

.data

digits: .ascii "0123456789"
buffer: .space BUFFSIZE
header: .ascii "P2\n"
endline: .ascii "\n"
error: .asciiz "io error.\n"
```

7 Conclusiones

Se han puesto en práctica conocimientos sobre la arquitectura MIPS, herramientas para su desarrollo - gdb -,como así también el manejo de registros y stack de memoria.

En este caso no hemos observado una mejora al traducir las funciones mencionadas a MIPS, por el contrario los tiempos se han visto incrementados. Posiblemente a la ineficiente forma en la que los datos se piden en el programa traducido. Lo que se hizo fue comparar el tiempo de ejecución de los casos de pruebas para la implementación en MIPS vs la implementación en C:

Caso de prueba	MIPS	C
Prueba 1	0m18.582s	0m18.949s
Prueba 2	0m34.301s	0m31.148s
Prueba 3	0m12.051s	0m11.148s
Prueba 4	0m33.680s	0m30.348s

8 Enunciado

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`