Organización de Computadoras

Trabajo Práctico 1
Implementación en MIPS Assembly

Nicolás Ledesma, *Padrón Nro. 93.118*
nicolas.angel.ledesma@gmail.com

Jonathan Moguilevsky, *Padrón Nro. 95.516*
jmoguilevsky@gmail.com

Leonardo Riego, *Padrón Nro. 94.104*
riegoleonardo@hotmail.com

2do. Cuatrimestre de 2019
66.20 Org. de Computadoras − Práctica: Santi, Pérez Masci, Natale
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

Este trabajo práctico grupal tiene como objetivo poner en práctica los conocimientos adquiridos sobre la arquitectura MIPS y la programación en assmebly en este entorno. Asimismo, se ponen en práctica la convenciones de código dispuestas por la MIPS ABI.

1. Enunciado

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 1: Programación MIPS
2^{do} cuatrimestre de 2019

`$Date: 2019/10/01 23:05:25 $`

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto a continuación.

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 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Descripción

El programa, deberá multiplicar matrices cuadradas de números reales, representados en punto flotante de doble precisión.

Las matrices a multiplicar ingresarán como texto por entrada estándar (`stdin`), donde cáda línea describe completamente cada par de matrices a multiplicar, según el siguiente formato:

$N\ a_{1,1}\ a_{1,2}\ ...\ a_{N,N}\ b_{1,1}\ b_{1,2}\ ...\ b_{N,N}$

La línea anterior representa a las matrices A y B, de NxN. Los elementos de la matriz A son los $a_{x,y}$, siendo x e y los indices de fila y columna respectivamente[1]. Los elementos de la matriz B se representan por los $b_{x,y}$ de la misma forma que los de A.

[1] Notar que es una representación del tipo *row major order*, siguiendo el orden en que C dispone las matrices en memoria.

El fin de línea es el caracter \n (*newline*). Los componentes de la línea están separados entre sí por uno o más espacios. El formato de los números en punto flotante son los que corresponden al especificador de conversión 'g' de `printf`[2].

Por ejemplo, dado el siguiente producto de matrices cuadradas:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

Su representación sería:

```
2 1 2 3 4 5 6 7 8
```

Por cada par de matrices que se presenten por cada línea de entrada, el programa deberá multiplicarlas y presentar el resultado por su salida estándar (`stdout`) en el siguiente formato, hasta que llegue al final del archivo de entrada (`EOF`):

$N\ c_{1,1}\ c_{1,2}\ ...\ c_{N,N}$

Ante un error, el progama deberá informar la situación inmediatamente (por `stderr`) y detener su ejecución.

[2] Ver man 3 printf, "Conversion specifiers".

4.1. Ejemplos

Primero, usamos la opción `-h` para ver el mensaje de ayuda:

```
$ tp1 -h
Usage:
  tp1 -h
  tp1 -V
  tp1 < in_file > out_file
Options:
  -V, --version     Print version and quit.
  -h, --help        Print this information and quit.
Examples:
   tp1 < in.txt > out.txt
   cat in.txt | tp1 > out.txt
```

A continuación, ejecutamos algunas pruebas:

```
$ cat example.txt
2 1 2 3 4 1 2 3 4
3 1 2 3 4 5 6.1 3 2 1 1 0 0 0 1 0 0 0 1

$ cat example.txt | ./tp1
2 7 10 15 22
3 1 2 3 4 5 6.1 3 2 1
```

En este ejemplo, realizamos las siguientes multiplicaciones, siendo los miembros izquierdos de la ecuación las matrices de entrada (`stdin`), y los miembros derechos las matrices de salida (`stdout`):

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6.1 \\ 3 & 2 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6.1 \\ 3 & 2 & 1 \end{pmatrix}$$

4.2. Interfaz

Las matrices deberán ser representadas por el tipo de datos `matrix_t`, definido a continuación:

```
typedef struct matrix {
        size_t rows;
        size_t cols;
        double* array;
} matrix_t;
```

Notar que los atributos `rows` y `cols` representan respectivamente la cantidad filas y columnas de la matriz. El atributo `array` contendrá los elementos de la matriz dispuestos en row-major order [3].

Los métodos a implementar, que aplican sobre el tipo de datos `matrix_t` son:

```
// Constructor de matrix_t
matrix_t* create_matrix(size_t rows, size_t cols);

// Destructor de matrix_t
void destroy_matrix(matrix_t* m);

// Imprime matrix_t sobre el file pointer fp en el formato solicitado
// por el enunciado
int print_matrix(FILE* fp, matrix_t* m);

// Multiplica las matrices en m1 y m2
matrix_t* matrix_multiply(matrix_t* m1, matrix_t* m2);
```

5. Implementación

A diferencia del TP anterior, en este caso deberá suministrarse una implementación de `matrix_multiply()` en código assembly MIPS. Así, el resto del programa deberá estar escrito en C, e interactuar con esta función para realiar la multiplicación de matrices.

6. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa;
- Comando(s) para compilar el programa;
- Las corridas de prueba, con los comentarios pertinentes;
- El código fuente, en lenguaje C y MIPS;
- El código MIPS32 generado por el compilador[3];

[3] Por motivos prácticos, en la copia impresa sólo es necesario incluir la primera página del código assembly MIPS32 generado por el compilador.

- Este enunciado.

7. Fechas

Fecha de vencimiento: martes 15/10.

Referencias

[1] GXemul, `http://gavare.se/gxemul/`.

[2] The NetBSD project, `http://www.netbsd.org/`.

[3] Row-major order (Wikipedia), `https://en.wikipedia.org/wiki/Row-major_order`.

2. Solución

2.1. Diseño

Algunos fragmentos del código entregado en el trabajo práctico anterior se mantienen sin modificaciones:

- Se conserva el código escrito para las funciones de lectura y procesamiento del input para generar las matrices de entrada, y para imprimir los resultados de la multiplicación.
- También se mantiene sin cambios la presentación de menus y mensajes de error.

A fines de reservar y liberar memoria alocada para matrices en forma compatible con el código assembly, las funciones **create_matrix** y **destroy_matrix** de C ahora utilizan las rutinas **my_malloc** y **my_free** definidas en el archivo **my_malloc.S**.

Por otra parte, se implementaron en lenguaje assembly de MIPS las funciones **matrix_multiply** y **create_matrix_S**

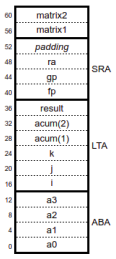
Si bien se mantiene una versión del código C de **create_matrix**, la nueva versión en assembly es empleada internamente dentro de la función **matrix_multiply** para generar la matriz resultado.

2.1.1. matrix_multiply

Esta función se encarga del cálculo del producto de las matrices ingresadas. Fue implementada siguiendo el mismo procedimiento que la versión previa escrita en lenguaje C.

La principal diferencia con el código C anterior es que internamente esta función llama a la implementación de **create_matrix_S** escrita en assembly.

`Stack frame`



- `SRA` Se guardaron los registros gp, fp y ra en este área, con 4 bytes de padding para que quede alineada a 8 bytes como dicta la ABI
- `LTA` Se reservaron espacios de memoria para las variables i, j y k que son los contadores utilizados en los distintos loops, 8 bytes para la variable acum utilizada para guardar los resultados parciales, y result que es utilizado como puntero a la estructura que guarda el resultado final.

Además, se definieron las constantes STACK_SIZE(tamaño del stack), OFFSET_RA, OFFSET_FP y OFFSET_GP para los offsets de las posiciones de memoria donde se guardaran los registros ra, fp y gp respectivamente

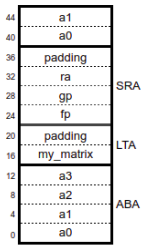
2.1.2. create_matrix_S

Función encargada de reservar memoria para una matriz con las dimensiones recibidas por parametro.

Esta función se encarga de alocar la memoria para la estructura que almacena la matriz resultado. Por ello internamente utiliza una rutina que reserva memoria en heap, llamada **my_malloc**, en el archivo **my_malloc.S**.

Cabe remarcar que la memoria alocada dentro de este código es luego liberada en el código C.

`Stack frame`



- `SRA` Se guardaron los registros gp, fp y ra en este área, con 4 bytes de padding para que quede alineada a 8 bytes como dicta la ABI
- `LTA` Se reserva una posicion en el stack donde se guardará la dirección de memoria de la matriz creada

Además, se definieron las siguientes constantes:

- `STACK_SIZE_CREATE_MATRIX:` tamaño stack
- `OFFSET_RA_CREATE_MATRIX:` offset ra
- `OFFSET_GP_CREATE_MATRIX:` offset gp
- `OFFSET_FP_CREATE_MATRIX:` offset fp
- `OFFSET_MY_MATRIX_CREATE_MATRIX:` offset variable my_matrix
- `A0_CALLEE_CREATE_MATRIX:` offset primer registro ABA
- `A0_CALLER_CREATE_MATRIX:` offset primer argumento de la función
- `A1_CALLER_CREATE_MATRIX:` offset segundo agurmento de la funcion

2.2. Código

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <getopt.h>
#include <unistd.h>
#include "mymalloc.h"

#define VERSION "1.0.0"
#define INIT_SIZE 2

/* ******* Buffer functions ******* */

char *init_buffer(size_t size)
{
    return calloc(size, size * sizeof(char));
}

char *push(char *buffer, char value, size_t *currLength, size_t *
    bufferSize)
{
    *currLength = *currLength + 1;

    if (*currLength >= *bufferSize)
    {
        char *newBuff;
        *bufferSize = *bufferSize * 2;
        newBuff = realloc(buffer, *bufferSize * sizeof(char));
        newBuff[*currLength - 1] = value;
        return newBuff;
    }
    buffer[*currLength - 1] = value;
    return buffer;
}

/* ******* Matrix implementation ******* */

typedef struct matrix
{
    size_t rows;
    size_t cols;
    double *array;
} matrix_t;

matrix_t *create_matrix(size_t rows, size_t cols)
{
    matrix_t *matrix = mymalloc(sizeof(matrix_t));
    double *array = mymalloc(sizeof(double) * cols * rows);
    matrix->array = array;
    matrix->cols = cols;
    matrix->rows = rows;
    return matrix;
```

```
}

extern matrix_t *matrix_multiply_2(matrix_t *matrix1, matrix_t *matrix2);

int print_matrix(FILE *fp, matrix_t *m)
{
    int elementCount = m->rows * m->cols;

    int i = 0;
    for (; i < elementCount; i++)
    {
        char *format = (i == elementCount - 1) ? "%.10g\n" : "%.10g␣";
        int res = fprintf(fp, format, m->array[i]);
        if (res < 0)
        {
            fprintf(stderr, "Error␣writing␣to␣file\n");
            return res;
        }
    }
    return elementCount;
}

void destroy_matrix(matrix_t *m)
{
    myfree(m->array);
    myfree(m);
}

/* ******* Value processing ******* */

int get_value(double *value_ptr, bool *eol, bool *eof)
{
    char c;
    bool stop = false;
    size_t length = 0;
    size_t bufferSize = INIT_SIZE;
    char *buffer = init_buffer(bufferSize);

    while (!stop)
    {
        c = (char)getchar();
        if (c == '␣' || c == '\n' || c == EOF)
        {
            *eol = (c == '\n');
            *eof = (c == EOF);
            stop = true;
        }
        else
        {
            char *new_buffer = push(buffer, c, &length, &bufferSize);
            buffer = new_buffer;
        }
    }
    if (length > 0)
```

```
    {
        buffer[length] = 0;
        int res = sscanf(buffer, "%lG", value_ptr);

        // Si no puede interpretar un double, da error.
        if (res == 0)
        {
            return -1;
        }
    }
    free(buffer);
    return length;
}

int read_matrix(matrix_t *matrix, int dim, bool *eol, bool *eof, bool
    eolExpected)
{
    int res;
    double value;

    int i = 0;
    for (; i < dim * dim; i++)
    {
        res = get_value(&value, eol, eof);

        if (res < 0 || (res == 0 && (*eol || *eof)))
        {
            fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
            return -1;
        }
        else
        {
            matrix->array[i] = value;
        }

        if (i == (dim * dim - 1) && *eol && !eolExpected) {
            fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
            return -1;
        }
    }

    return 0;
}

enum LineEnding
{
    EndOfLine,
    EndOfFile,
    Error
};

enum LineEnding process_line()
{
    double value;
```

```
int res;
size_t dim;

bool eol = false;
bool eof = false;

res = get_value(&value, &eol, &eof);

if (eof)
{
    return EndOfFile;
}

if (eol)
{
    fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
    return Error;
}

if (res <= 0)
{
    fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
    return Error;
}

dim = (size_t)value;
int response;

matrix_t *matrix1 = create_matrix(dim, dim);

response = read_matrix(matrix1, dim, &eol, &eof, false);

if (response == -1)
{
    destroy_matrix(matrix1);
    return Error;
}

matrix_t *matrix2 = create_matrix(dim, dim);
response = read_matrix(matrix2, dim, &eol, &eof, true);

if (response == -1)
{
    destroy_matrix(matrix1);
    destroy_matrix(matrix2);
    return Error;
}

if (!eol && !eof)
{
    destroy_matrix(matrix1);
    destroy_matrix(matrix2);
    fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
    return Error;
```

```
    }

    matrix_t *result = matrix_multiply_2(matrix1, matrix2);

    printf("%zu␣", dim);
    print_matrix(stdout, result);

    destroy_matrix(matrix1);
    destroy_matrix(matrix2);
    destroy_matrix(result);

    return EndOfLine;
}

static void print_usage(const char *src)
{
    printf("%s\n\t%s%s\n\t%s%s\n\t%s%s\n", "Usage:",
           src, "␣-h",
           src, "␣-V",
           src, "␣<␣in_file␣>␣out_file");
}

static void print_options(const char *src)
{
    printf("%s\n%s\n%s\n%s\n%s%s\t%s\n%s%s%s\n",
           "Options:",
           "\t-V,␣--version\tProgram␣version.",
           "\t-h,␣--help\tPrint␣help.",
           "Examples:",
           "\t", src, "tp0␣<␣in.txt␣>␣out.txt",
           "\tcat␣in.txt␣", src, "␣|␣tp0␣>␣out.txt");
}

static void print_help(const char *src)
{
    print_usage(src);
    print_options(src);
}

static void print_version()
{
    printf("Version␣%s\n", VERSION);
}

int argsHandler_parse_arguments(const int argc, char *const argv[])
{
    int arg;
    int option_index = 0;
    const char *short_opt = "hV";
    static struct option long_options[] = {
       {"help", no_argument, 0, 'h'},
       {"version", no_argument, 0, 'V'},
       {0, 0, 0, 0}};
```

```
    while ((arg = getopt_long(argc, argv, short_opt, long_options, &
        option_index)) != -1)
    {
        switch (arg)
        {
        case 'h':
            print_help(argv[0]);
            exit(EXIT_SUCCESS);
        case 'V':
            print_version();
            exit(EXIT_SUCCESS);
        case '?':
            exit(EXIT_FAILURE);
            break;
        default:
            break;
        }
    }

    // extern int optind
    if (optind < argc)
    {
        fprintf(stderr, "Unrecognized␣options:␣");
        while (optind < argc)
        {
            fprintf(stderr, "%s␣", argv[optind++]);
        }
        fprintf(stderr, "\n");
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}

int main(int argc, char *const argv[])
{
    argsHandler_parse_arguments(argc, argv);

    enum LineEnding result;

    do
    {
        result = process_line();
    } while (result == EndOfLine && result != Error);

    if (result == Error)
    {
        return -1;
    }

    return 0;
}
```

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

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

#define STACK_SIZE 56
#define OFFSET_RA 48
#define OFFSET_GP 44
#define OFFSET_FP 40

/*
   56+-------------+
     | padding |
   52+-------------*
     | ra |
   48+-------------+
     | gp |
   44+-------------+
     | fp |
   40+-------------+
     | result |
   36+-------------+
     | acum(2) |
   32+-------------+
     | acum |
   28+-------------+
     | k |
   24+-------------+
     | j |
   20+-------------+
     | i |
   16+-------------+
     | a3 |
   12+-------------+
     | a2 |
    8+-------------+
     | a1 |
    4+-------------+
     | a0 |
    0+-------------+
*/

    .text
    .abicalls
    .align 2

    .globl matrix_multiply_2
    .ent matrix_multiply_2

matrix_multiply_2:
    .frame $fp, STACK_SIZE, ra
    .set noreorder
    .cpload t9
    .set reorder

    subu sp, sp, STACK_SIZE #construccion stack
```

```
    .cprestore OFFSET_GP #guardo gp en el stack
    sw $fp, OFFSET_FP(sp) #guardo fp en el stack
    sw ra, OFFSET_RA(sp) #guardo ra en el stack, esto tal vez no sea
        necesario

    move $fp, sp

    sw a0, 56($fp)
    sw a1, 60($fp)

    lw t0, 56($fp) #matrix 1->t0
    lw t1, 60($fp) #matrix 2->t1

    lw t2, 0(t0) #matrix_1->rows -> t2
    lw t3, 4(t1) #matrix_2->cols -> t3

    move a0, t2
    move a1, t3
    jal create_matrix_S

    sw v0, 36($fp) #result

    li t0, 0
    sw t0, 16($fp) #inicializo i


first_for:
    lw t0, 16($fp) #t0 = i
    lw t1, 56($fp) #t1 = matrix1
    lw t2, 0(t1) #t2 = matrix1->rows

    #condicion del if, para saber si i < rows
    slt t3, t0, t2
    beq t3, zero, end

    li t1, 0 #inicializo j
    sw t1, 20($fp)

second_for:
    lw t0, 20($fp) #t0 = j
    lw t1, 60($fp) #matrix2
    lw t2, 4(t1) #t2 = matrix2->cols

    #pregunto si j < cols
    slt t3, t0, t2
    beq t3, zero, end_second_for

    l.d $f4, double_zero #f4=0.0
    s.d $f4, 28($fp) #acum=0.0

    li t1, 0
    sw t1, 24($fp) #inicializo k

third_for:
```

```
    lw t0, 16($fp) #t0 = i
    lw t1, 20($fp) #t1 = j
    lw t2, 24($fp) #t2 = k

    lw t3, 56($fp) #matrix1
    lw t4, 60($fp) #matrix2

    lw t5, 4(t4) #matrix2->cols

    slt t7, t2, t5 #k < cols??
    beq t7, zero, end_third_for

    lw t5, 4(t3) #matrix1->cols
    mul t5, t5, t0 #matrix1->cols * i
    addu t5, t5, t2 #matrix1->cols * i + k

    #matrix->array[cols * i + k]
    lw t6, 8(t3)
    add t7, t5, t5
    add t7, t7, t7
    add t7, t7, t7
    addu t7, t7, t6

    l.d $f4, 0(t7)

    lw t5, 4(t4) #matrix2->cols
    mul t5, t5, t2 #cols * k
    addu t5, t5, t1 #cols * k + j

    #matrix->array[cols * k + j]
    lw t6, 8(t4)
    sll t5, t5, 3
    addu t5, t5, t6

    l.d $f6, 0(t5)

    mul.d $f4, $f4, $f6

    l.d $f8, 28($fp) #load acum

    add.d $f4, $f8, $f4 #acum + matrix2->array[cols * k + j] * matrix1->
        array[cols*i + k]

    s.d $f4, 28($fp)

    #k++
    addi t2, t2, 1
    sw t2, 24($fp)

    b third_for

end_third_for:
    lw t0, 36($fp) #result
    lw t1, 4(t0) #result->cols
```

```
    lw t2, 16($fp) #i
    lw t3, 20($fp) #j

    mul t1, t1, t2
    addu t1, t1, t3 #t1 = cols * i + j

    l.d $f4, 28($fp) #$f4 = acum

    lw t0, 8(t0)
    add t1, t1, t1
    add t1, t1, t1
    add t1, t1, t1
    addu t1, t1, t0
    #lw t5, 0(t1) #result->array[cols * i + j] = acum

    #s.d $f4, 0(t5) #result->array[cols * i + j] = acum
    s.d $f4, 0(t1)

    #j++
    lw t0, 20($fp)
    addi t0, t0, 1
    sw t0, 20($fp)

    b second_for

end_second_for:
    #i++
    lw t0, 16($fp)
    addi t0, t0, 1
    sw t0, 16($fp)
    b first_for

end:
    lw v0, 36($fp)
    lw gp, OFFSET_GP(sp)
    lw $fp, OFFSET_FP(sp)
    lw ra, OFFSET_RA(sp)

    addu sp,sp, STACK_SIZE

    jr ra

    .end matrix_multiply_2

  .globl create_matrix_S
  .ent create_matrix_S


#define STACK_SIZE_CREATE_MATRIX 40
#define OFFSET_RA_CREATE_MATRIX 32
#define OFFSET_GP_CREATE_MATRIX 28
#define OFFSET_FP_CREATE_MATRIX 24
#define OFFSET_MY_MATRIX_CREATE_MATRIX 16
#define A0_CALLEE_CREATE_MATRIX 0
```

```
#define A0_CALLER_CREATE_MATRIX 40
#define A1_CALLER_CREATE_MATRIX 44

/*
   48+-------------+
     | a1 |
   44+-------------+
     | a0 |
   40+-------------+
     | padding |
   36+-------------+
     | ra |
   32+-------------+
     | gp |
   28+-------------*
     | fp |
   24+-------------+
     | padding |
   20+-------------+
     | my_matrix |
   16+-------------+
     | a3 |
   12+-------------+
     | a2 |
    8+-------------+
     | a1 |
    4+-------------+
     | a0 |
    0+-------------+
*/
create_matrix_S:
  .frame $fp, STACK_SIZE_CREATE_MATRIX, ra
  .set noreorder
  .cpload t9
  .set reorder

  subu sp, sp, STACK_SIZE_CREATE_MATRIX #construccion stack

  .cprestore OFFSET_GP_CREATE_MATRIX #guardo gp en el stack
  sw $fp, OFFSET_FP_CREATE_MATRIX(sp) #guardo fp en el stack
  sw ra, OFFSET_RA_CREATE_MATRIX(sp) #guardo ra en el stack, esto tal vez
       no sea necesario

  move $fp, sp

  sw a0, A0_CALLER_CREATE_MATRIX($fp)
  sw a1, A1_CALLER_CREATE_MATRIX($fp)


  li t0, 12 # sizeof(matrix_t)
  move a0, t0

  jal mymalloc #v0 = mymalloc(sizeof(matrix_t));
```

```
  sw v0, OFFSET_MY_MATRIX_CREATE_MATRIX($fp)


  lw t0, A0_CALLER_CREATE_MATRIX($fp) # t0 = rows
  lw t1, A1_CALLER_CREATE_MATRIX($fp) # t1 = cols
  mul t2, t0, t1 # t2 = rows * cols
  li t3, 8 # sizeof(double)
  mul t2, t2, t3 # t2 = sizeof(double) * cols * rows

  move a0, t2

  jal mymalloc #v0 = mymalloc(sizeof(double) * cols * rows); (el array)

  lw t0, OFFSET_MY_MATRIX_CREATE_MATRIX($fp) # t0 = puntero a la matriz
      creada
  sw v0, 8(t0)

  move v0, t0

  lw t0, A0_CALLER_CREATE_MATRIX($fp)
  lw t1, A1_CALLER_CREATE_MATRIX($fp)

  move a0, t0
  move a1, t1

  lw t0, A0_CALLER_CREATE_MATRIX($fp) # t0 = rows
  sw t0, 0(v0)

  lw t1, A1_CALLER_CREATE_MATRIX($fp) # t1 = cols
  sw t1, 4(v0)

  lw gp, OFFSET_GP_CREATE_MATRIX(sp)
  lw $fp, OFFSET_FP_CREATE_MATRIX(sp)
  lw ra, OFFSET_RA_CREATE_MATRIX(sp)

  addu sp,sp, STACK_SIZE_CREATE_MATRIX

  jr ra
  .end create_matrix_S

.data
   double_zero: .double 0.0
```

2.3. Compilación del programa

Para compilar el programa se utiliza el makefile desde el directorio donde se descomprimió el entregable:

```
$ make clean
$ make all
```

3. Pruebas

3.0.1. Ejecución

Para la ejecución de las pruebas generamos un script llamado `run_tests.sh` que se encarga de, para cada caso generado, ejecutar el programa y escribit el output en un archivo. Finalmente, se comparan los outputs contra archivos que contienen los resultados esperados, en la carpeta `results/`.

3.1. Casos de prueba

Para probar nuestro programa generamos una serie de casos de prueba:

- `big_matrix_ints.txt`: Matriz de 100x100 con valores enteros
- `big_matrix_doubles.txt`: Matriz de 100x100 con valores de punto flotante
- `exponential_doubles_matrix.txt`: Matriz pequeña con valores de punto flotante en notación exponencial.
- `invalid_input.txt`: Prueba con input invalido.
- `simple_multiline.txt`: Prueba de input con varias líneas.
- `single_line.txt`: Prueba simple de una sola línea.
- `small_double_matrix.txt`: Prueba simple de una matriz 3x3 de doubles.
- `small_int_matrix.txt`: Prueba simple de una matriz 3x3 de integers.

La ejecución de los casos se realiza de la siguiente manera:

```
$ ./run_tests.sh
```

Este script de ejecución valida también la impresión del menú y de la versión del programa.

4. Conclusiones

En este trabajo pusimos en práctica la definición de un programa en MIPS assembly, lo cual nos lleva a los siguientes aprendizajes y conclusiones:

- Al definir y crear el stack frame dentro de cada función que implementamos, se tiene pleno control (sin olvidar respetar la ABI) y conocimiento del uso de la memoria principal del programa.
- Se observa a su vez que las gestión de recursos de la máquina es absoluta, tanto en el uso de memoria como de los registros del procesador.
- Resulta tedioso al principio valerse de menos herramientas al encontrarnos desarrollando en tan bajo nivel y en un entorno de desarrollo tan cerrado como la máquina virtual con el sistema operativo NetBSD.
- La herramienta de debugging gdb resultó ser muy util a la hora de diagnosticar comportamientos indeseados en el programa en la medida que lo construimos. Permite visualizar de una forma muy práctica los registros y direcciones utilizados en tiempo real durante la ejecución del programa.