

Trabajo Práctico N⁰ 1 "Assembly MIPS"

Tomás Botalla, *Padrón 96356*

`tbotalla@gmail.com`

Samanta Loiza, *Padrón 91935*

`samiloiza@gmail.com`

Nahuel Sosa, *Padrón 91289*

`nahuelmartin.sosa@yahoo.com.ar`

1er. Cuatrimestre de 2016

66.20 Organización de Computadoras

Facultad de Ingeniería, Universidad de Buenos Aires

Índice

1. Introducción	3
2. Compilación y Ejecución del programa	3
3. Implementación	4
3.1. Desarrollo	4
3.2. Implementación en C	4
4. Pruebas	4
4.1. 1º Prueba	5
4.2. 2º Prueba	5
4.3. 3º Prueba	5
4.4. 4º Prueba	6
4.5. 5º Prueba	6
5. Conclusiones	7
6. Código C	8
6.1. Makefile	8
6.2. tp0.h	8
6.3. tp0.c	9
7. Código MIPS32	14
7.1. tp0.S	14
8. Stack Frame	18

Resumen

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva la multiplicación de matrices en lenguaje assembly, aplicando la convención de llamada a funciones estudiada en clase.

1. Introducción

2. Compilación y Ejecución del programa

Nota: Para poder compilar el programa debemos enviar los archivos fuente a NetBSD por medio del túnel creado con ssh.

Para compilar:

- Parados en la carpeta donde tenemos los fuente ejecutamos.

```
gcc -Wall -g -std=c99 tp0.S tp0.c -o tp0
```

o bien escribiendo el comando

```
make
```

Para ejecutar:

- Parados en la carpeta donde tenemos el ejecutable generado por la compilación, el programa se invoca a través de línea de comandos de la siguiente manera:

```
cat fileTest/test1.txt | ./tp0
cat fileTest/test1.txt | ./tp0 >out1.txt
./tp0 <test1.txt
./tp0 <test1.txt >out1.txt
```

El menú de ayuda especifica las opciones disponibles al momento de invocar el programa:

Usage:

```
./tp0 -h
```

```
./tp0 -V
```

OPTIONS:

```
-h, --help Print this information and quit.
```

```
-V, --version Print version and quit
```

Ejemplo:

```
./tp0 <in.txt >out.txt
cat in.txt | ./tp0 >out.txt
```

3. Implementación

3.1. Desarrollo

Para el desarrollo del trabajo se realizó un programa escrito en lenguaje C. El usuario tendrá la posibilidad de ver un mensaje de ayuda y la versión del mismo como el el tp0.

En base a un stream de entrada armamos matrices para ser multiplicadas de a pares para luego mostrar su resultado en un stream de salida.

Se genera la función encargada de multiplicar las matrices compuestas de números flotantes de doble precisión en lenguaje assembly MIPS32, también implementamos una función para imprimirla. Para el desarrollo del mismo analizamos los argumentos recibidos por la función y las variables declaradas localmente para definir el stack frame necesario para su ejecución. Una vez definido el stack frame comenzamos a definir los ciclos definidos para llevar a cabo las operaciones necesarias. Finalmente enviamos todos los archivos fuentes al emulador gxemul para poder linkear con gcc el código c con el assembly generado por nosotros.

3.2. Implementación en C

Implementamos diversas funciones para este trabajo, las cuales se encargan de:

- Checkear los argumentos recibidos:

int checkArguments(int,char[])*

4. Pruebas

Se utilizaron los siguientes archivos de prueba

- *badDim.txt*: a 1.0 2.0 3.0 4.0 1.0 2.0 2.2 4.0

- *badCant.txt*: 3 10 1.2 3.5

- *charTest.txt*

2 1.0 2.0 3.0 4.0 1.0 2.0 a 4.0

- *emptyfile.txt*

- *test1.txt*

2 1.0 2.0 3.0 4.0 1.0 2.0 3.0 4.0

3 1.0 2.0 3.0 4.0 5.0 6.1 3.0 2.0 1.0 1.0 0.0 0.0 0.0 1.0 0.0

0.0 0.0 1.0

- *test2.txt*
3 -1 3.5 2.3 6.9 -9.9 2.3 1 0 10.3 8.12 1 12 13 4.2 15.2 1.6 9.3
6.3

- *test3.txt*
2 -1.3 2.5 -9.3 12 3.2 7.4 -2 8
3 -6.2 2.0 3.0 4.0 2.3 6.1 3.0 -7.4 1.0 9.12 0.0 -1 0.0 1.0 0.0
0.0 0.0 1.0

- *test4.txt*
2 0.000000 -3.000000 -9.000000 11.000000 -1.000000 -4.000000 -1.000000
-4.000000

- *testHigh.txt*
4 5.200 13.900 4.200 1.500 1.500 6.200 14.700 0.800 17.200 6.300
1.800 0.900 0.600 3.400 5.300 4.700 6.300 1.400 17.300 12.400
11.300 0.800 1.400 3.700 5.900 3.400 16.300 9.400 6.500 12.300
11.700 18.200
3 12.300 18.100 -0.900 -6.500 12.700 11.300 6.400 3.200 7.400
4.700 3.400 11.300 16.400 -8.300 9.100 15.700 -17.100 4.600

4.1. 1º Prueba

En esta prueba probamos que la matriz sea válida.

```
cat fileTest/charTest.txt | ./tp0
Matriz 1 inválida
```

4.2. 2º Prueba

```
cat fileTest/emptyfile.txt | ./tp0
```

4.3. 3º Prueba

```
cat fileTest/badDim.txt | ./tp0
Error: Matriz inválida para multiplicación
```

4.4. 4º Prueba

```
./tp0 <fileTest/test2.txt
3 41.06000 35.09000 55.69000 -68.9920 -13.2900 -53.1900 24.60000
96.79000 76.89000
```

Con esta prueba multiplicamos dos pares de matrices y mostramos por pantalla el resultado obtenido. Si queremos redireccionar el resultado de salida `./tp0 <fileTest/test2.txt >fileTest/out2.txt`

■ *out2.txt*:

```
3 41.06000 35.09000 55.69000 -68.9920 -13.2900 -53.1900 24.60000
96.79000 76.890
```

4.5. 5º Prueba

```
./tp0 <fileTest/testHigh.txt

4 224.3600 51.13000 195.4300 182.6900 171.4400 66.88000 283.6000
194.2800 196.0200 46.31000 346.2500 269.8900 104.0200 79.39000 156.5200
155.3800
3 340.5200 -93.0200 299.5600 355.1400 -320.740 94.10000 198.7400
-131.340 135.4800
```

Con esta prueba multiplicamos dos pares de matrices y mostramos por pantalla el resultado obtenido. Si queremos redireccionar el resultado de salida `./tp0 <fileTest/testHigh.txt >fileTest/out3.txt`

■ *out3.txt*:

```
4 224.3600 51.13000 195.4300 182.6900 171.4400 66.88000
283.6000 194.2800 196.0200 46.31000 346.2500 269.8900 104.0200
79.39000 156.5200 155.3800
3 340.5200 -93.0200 299.5600 355.1400 -320.740 94.10000
198.7400 -131.340 135.4800
```

5. Conclusiones

Gracias al desarrollo del correspondiente trabajo pudimos conocer las instrucciones disponibles en MIPS32, pudimos observar claramente la intervención de los registros comunes y de punto flotante del procesador en la ejecución del código en C.

6. Código C

6.1. Makefile

```
CFLAGS= -Wall -g -std=c99
CC=gcc

all: tp0

tp0: tp0.c tp0.S
    $(CC) $(CFLAGS) tp0.S tp0.c -o tp0
clean:
    rm tp0
```

6.2. tp0.h

```
#ifndef TP0_H_
#define TP0_H_
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdbool.h>
#include <string.h>

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

void printHelp();

int checkArguments(int, char*[]);

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

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

/* Devuelve el primer elemento del archivo a leer, es
   decir la dimension*/
int readMatrixDimension();

/* Lee matriz de una linea del stdin*/
matrix_t* readMatrix(matrix_t* matrix);

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



```

// Multiplica las matrices en m1 y m2i
extern void matrix_multiply(matrix_t* m1, matrix_t* m2,
    matrix_t* out);

// Imprime en el archivo indicado por el file descriptor
    fd, el string C
// apuntador por str, sin incluir su byte nulo de
    finalizacion.
extern ssize_t print_string(int fd, char* str);

#endif /* TP0_H */

```

6.3. tp0.c

```

#include "tp0.h"

void printHelp() {
    char *help = "Usage:"
        "\t./tp0 -h\n"
        "\t./tp0 -V\n"
        "\t./tp0 <in_file> <out_file>\n"
        "Options:\n"
        "-V, --version\n\tPrint version and quit.\n"
        "-h, --help\n\tPrint this information and quit\n"
        ".\n"
        "Examples:\n"
        ".../tp0 <in.txt> <out.txt>\n"
        "...cat in.txt | ./tp0 >out.txt\n";

    printf("%s", help);
}

// Devuelve mensaje de ayuda si el segundo argumento es -h o
    --help y
// la version si es -V, -v o --version. En cualquier otro caso
    devuelve 1.
int checkArguments(int cantidadArgumentos, char* argumentos[])
{
    int retorno = 1;
    if ((cantidadArgumentos == 2)
        && ((strcmp(argumentos[1], "-h") == 0)
            || (strcmp(argumentos[1], "--help") == 0))
        ) {
        printHelp();
        retorno = 0;
    } else if ((cantidadArgumentos == 2)
        && (((strcmp(argumentos[1], "-V") == 0)
            || (strcmp(argumentos[1], "--version") ==
                0) || (strcmp(argumentos[1], "-v") ==
                0))) {
        printf("Version 1.0\n");
    }
}

```

```

        retorno = 0;
    }

    return retorno;
}

// Lee el primer elemento del stdin que corresponde a la
// dimension de la matriz cuadrada
int readMatrixDimension() {

    int dimension = 0;
    if(( fscanf(stdin, "%d", &dimension)==1) && !feof(stdin)){
        if (ferror (stdin)){
            printf ("Error_reading_stdin\n");
            exit(EXIT_FAILURE);
        }
    }
    return dimension;
}

// Constructor de matrix_t
matrix_t* create_matrix(size_t filas , size_t columnas){

    matrix_t* matriz;
    matriz = malloc(sizeof(matrix_t));
    if(matriz == NULL){
        fprintf(stderr, "Error_malloc_\n");
        return NULL;
    }
    (*matriz).array = malloc(sizeof(double)*filas*columnas);
    if((*matriz).array == NULL){
        fprintf(stderr, "Error_malloc_\n");
        return NULL;
    }
    (*matriz).rows = filas;
    (*matriz).cols = columnas;

    return matriz;
}

// Carga la matrix previamente creada con floats de la matrix
matrix_t* readMatrix(matrix_t* matrix){
    double value = 0.0;
    int cantNums, j;

    cantNums = (*matrix).rows * (*matrix).cols;

    for(j =0;j<cantNums; j++){
        if(fscanf(stdin, "%f", &value) ==1){
            if (ferror (stdin)){
                printf ("Error_reading_stdin\n");
                if(matrix!=NULL){
                    free((*matrix).array);

```

```

        free(matrix);
        matrix = NULL;
    }
    exit(EXIT_FAILURE);
}
(*matrix).array[j] = value;
} else{
    fprintf(stderr, "Matriz invÃlida.\n" );
    if(matrix!=NULL){
        free((*matrix).array);
        free(matrix);
        matrix = NULL;
    }
    exit(EXIT_FAILURE);
}

}
return matrix;
}

// // Multiplica las matrices en m1 y m2
// void matrix_multiply(matrix_t* m1, matrix_t* m2, matrix_t*
//     matrizResultado ){

//     int pos = 0;
//     int i,j,k, m;
//     int dimension = (*matrizResultado).rows;

//     for( i=0 ; i < dimension*dimension; i=i+dimension){
//         for(k=0; k < dimension ; k++){
//             double sum = 0.0;
//             m = i;
//             // printf("Entro for de K con k = %d\n",k );
//             for(j=k; j < dimension*dimension; j=j+
// dimension){
//                 sum+= (*m1).array[m]*(*m2).array[j];
//                 m++;
//             }
//             (*matrizResultado).array[pos] = sum;
//             pos++;
//         }
//     }

// }

// }

// Destructor de matrix_t
void destroy_matrix(matrix_t* matriz){
    if(matriz != NULL){
        free((*matriz).array);
        free(matriz);
        matriz = NULL;
    }
}

```

```

// Imprime matrix_t
int print_matrix(FILE* fp, matrix_t* matrix){
    int dim = (*matrix).rows;

    char* string = (char*) malloc(200*sizeof(char)); //buffer
    200 bytes
    if(string == NULL){
        fprintf(stderr, "Error_malloc\n");
        exit(EXIT_FAILURE);
    }
    sprintf(string, "%d", dim); // carga la dimension en el
    string

    int bufferDouble = 10*sizeof(char); //buffer 10 bytes
    char* stringAux = (char*) malloc(bufferDouble);
    if(stringAux == NULL){
        fprintf(stderr, "Error_malloc\n");
        exit(EXIT_FAILURE);
    }
    for(int i=0;i<dim*dim;i++){
        snprintf(stringAux, bufferDouble, "%d", (*matrix).
            array[i]);
        strcat(string, stringAux); // concatena el elemento
            actual en el string
    }
    strcat(string, "\n");

    int fd = fileno(fp); // Obtiene el file descriptor a
    partir del file pointer
    if (fd == -1) {
        fprintf(stderr, "Error_file_descriptor");
        free(string);
        free(stringAux);
        exit(EXIT_FAILURE);
    }

    if(print_string(fd, string) == -1){
        fprintf(stderr, "Error_write_syscall");
        free(string);
        free(stringAux);
        exit(EXIT_FAILURE);
    }
    free(string);
    free(stringAux);

    return 0;
}

/* ssize_t print_string(int fd, char* str){

    ssize_t retorno = 0;

```

```

        retorno = write(fd, str, strlen(str)); // llamado a
            syscall
        if(retorno < 0){
            return -1;
        }
        return retorno;
    }

*/
// argc == argument count, argv== argument vector
int main(int argc, char *argv[]) {

    if (!checkArguments(argc, argv)){
        return 1; // Error en los argumentos
    }else{
        do{
            int dimension = readMatrixDimension();
            if(!feof(stdin)){
                return 0;
            }
            if(dimension > 0){
                matrix_t* matrix1 = create_matrix(dimension,
                    dimension);
                matrix1 = readMatrix(matrix1);
                matrix_t* matrix2 = create_matrix(dimension,
                    dimension);
                matrix2 = readMatrix(matrix2);
                matrix_t* matrizResultado = create_matrix(
                    dimension, dimension);
                matrix_multiply(matrix1, matrix2,
                    matrizResultado);
                print_matrix(stdout, matrizResultado);

                //Borra las matrices creadas
                destroy_matrix(matrix1);
                destroy_matrix(matrix2);
                destroy_matrix(matrizResultado);
            }else{
                fprintf(stderr, "Error: \_Matriz\_inv\'alida\_para
                    \_multiplicaci\'on\_n");
                exit(EXIT_FAILURE);
            }
        }while(!feof(stdin));
    }
    return 0;
}

```

7. Código MIPS32

7.1. tp0.S

```
#include <mips/regdef.h>
#include <sys/syscall.h>
    .text
    .abicalls
    .align 2
    .globl matrix_multiply
    .ent    matrix_multiply
matrix_multiply:

    .frame $fp, 48, ra
    .set    noreorder
    .cload t9
    .set    reorder
    subu sp, sp, 48
    .cprestore 44
    sw $fp, 40(sp)
    move $fp, sp

    sw      a0, 48($fp) #guardo dir m1
    sw      a1, 52($fp) #guardo dir m2
    sw      a2, 56($fp) #guardo dir out

    lw      t0, 48($fp) #t0= dir matrix m1
    li      t1, 0
    addu    t2, t0, t1 #carga valor de rows
    lw      t2, 0(t2)
    sw      t2, 0($fp) #guardo valor de rows
    addiu   t1, t1, 8
    addu    t3, t0, t1 #carga dir de array de m1
    lw      t3, 0(t3)
    sw      t3, 4($fp) #guardo dir de array de m1

    lw      t0, 52($fp) #t0= dir matrix m2
    li      t1, 0
    addiu   t1, t1, 8
    addu    t3, t0, t1 #carga dir de array de m2
    lw      t3, 0(t3)
    sw      t3, 8($fp) #guardo dir de array de m2

    lw      t0, 56($fp) #t0= dir matrix out
    li      t1, 0
    addiu   t1, t1, 8
    addu    t3, t0, t1 #carga dir de array de out
    lw      t3, 0(t3)
    sw      t3, 12($fp) #guardo dir de array de out

    li      t0, 0 #carga valor el valor 0 en t0
    sw      t0, 16($fp) #pos = 0
    sw      t0, 20($fp) # i = 0
```

```

sw      t0,28($fp) # k = 0

lw      t1,0($fp) #t1=rows
mul     t1,t1,t1 #t1=rows*rows
sw      t1,36($fp) #cantidad de elementos matriz 1

f1:     lw      t0, 20($fp) #t0= i
        lw      t1, 36($fp) #t1 = cantidad elementos
        bge     t0,t1,endif1 #i< cantidad elementos

f2:     lw      t0,28($fp) #t0 = k
        lw      t1,0($fp) #t1= rows
        bge     t0,t1,endif2 #k<rows
        li.d    $f0, 0 #f0=sum=0

        lw      t1,20($fp) #t1 = i
        sw      t1, 32($fp) #n=i
        lw      t2, 28($fp) #t2=k
        sw      t2, 24($fp) #j=k

f3:     lw      t2, 24($fp) #t2 =j
        lw      t3, 36($fp) #t3=cantidad elementos
        bge     t2,t3,endif3 #j< cantidad elementos

        lw      t3, 4($fp) #t3=m1
        lw      t1,32($fp) #t1=m
        sll     t1,t1,3 #t1= m*8
        addu    t3,t3,t1 #t3 = &m1[m]

        lw      t4, 8($fp) #t4=m2
        lw      t2, 24($fp) #t2=j
        sll     t2, t2, 3 #t2 = j*8
        addu    t4,t4,t2 #t4= &m2[j]

        l.d     $f2,0(t3) #f1 = m1[m]
        l.d     $f4,0(t4) #f2= m2[j]
        mul.d   $f2,$f2,$f4 #m1[m]*m2[j]
        add.d   $f0,$f0,$f2

        lw      t1,32($fp) #carga en t1 el valor de m
        addi    t1,t1,1 #n = m+1
        sw      t1,32($fp) # actualizo m

        lw      t1,24($fp) #t1=j
        lw      t2,0($fp) #t2 =rows
        addu    t1,t1,t2 #t1= j+rows
        sw      t1,24($fp) #j= j+rows
        j       f3

endif3: lw      t1, 12($fp) #t1=out
        lw      t2, 16($fp) #t2=pos
        sll     t2,t2,3 #t2=pos*8
        addu    t1,t1,t2 # t1= &out[pos]
        s.d     $f0,0(t1) #out[pos]=sum

```

```

        lw      t2,16($fp) #t2=pos
        addi    t2,t2,1 #pos++
        sw      t2,16($fp) #guardo pos

        lw      t0,28($fp) #t0 = k
        addi    t0,t0,1 #k++
        sw      t0,28($fp)

    j          f2
endf2:    lw      t0, 20($fp) #t0 = i
        lw      t1,0($fp) #t1 =rows
        addu    t0,t0,t1 # i=i+rows
        sw      t0,20($fp)
        li      t0, 0
        sw      t0, 28($fp)
        j          f1

endf1:
    move sp, $fp
    lw $fp, 40(sp) #recupero fp
    addu sp,sp,48 #restablezco stack frame
    jr ra
    .end matrix_multiply

    .align 2
    .globl print_string
    .ent print_string
print_string:
    .frame $fp, 16, ra    #frame multiplo de 8 bytes
    .set noreorder
    .cpload t9            #se usa t9 para referenciar data
        global
    .set reorder

    subu sp,sp,16        #reservo el stackframe
    .cpstore 8           #lw y sw del gp en 8(sp)
        automatico

    sw $fp, 4(sp)        #guarda fp
    sw ra, 12(sp)
    move $fp, sp         #fp=sp
    sw a0, 16($fp)       #guarda larg (file descriptor)
    sw a1, 20($fp)       #guarda 2arg (string)
    move a0, a1          #a0=string carga la direcc del
        string como primer arg para llamar a strlen
    la t9, strlen        #carga direccion de la funcion
        strlen
    jal t9               #llama a strlen retorno en v0
    lw a0, 16($fp)       #carga el file descriptor como 1
        argumento
    move a2, v0          #carga el retorno de strlen como
        3 argumento
    li v0, SYS-write     #carga syscall en v0

```



```

        syscall                #llamado a SYS_write
        beqz a3, ok            #a3=0 --> syscall OK, a3=1 -->
                                syscall error
error:  li v0, -1              #si error devuelve -1
ok:     #result syscall == v0 == #bytes
        escritos
        lw gp, 8(sp)           #restore gp
        lw $fp, 4(sp)          #restore fp
        lw ra, 12(sp)
        addu sp, sp, 16        #restore stack frame
        jr ra                  #salta a la direccion siguiente
                                a la que la haya llamado
.end print_string

```

8. Stack Frame

Dado que la función `matrix_multiply` es una función leaf no tuvimos necesidad de reservar espacio en el stack frame para el ABA, tampoco tuvimos que salvar el registro `ra`. Desde la dirección 0 hasta 36 almacenamos las variables auxiliares utilizadas para los cálculos, de 40 a 44 almacenamos `fp` y `gp`. Por otro lado, para el caso de `print_string` al ser una función non-leaf (llama a otra función, `strlen`) hay necesidad de guardar el `ra` ya que se modifica y también el `fp` y `gp` como en todas las funciones.

A continuación se muestran los stack frames de las dos funciones implementadas:

Stack frame anterior	Stack frame anterior
28(\$fp) a3	60(\$fp) a3
24(\$fp) a2	56(\$fp) matrix* out
20(\$fp) str	52(\$fp) matrix* m2
16(\$fp) fd	48(\$fp) matrix* m1
Stack frame print_string	Stack frame matrix_multiply
12(\$fp) ra	44(\$fp) fp
8(\$fp) gp	40(\$fp) gp
4(\$fp) fp	36(\$fp) cant. elem.
0(\$fp) padding	32(\$fp) m
	28(\$fp) k
	24(\$fp) j
	20(\$fp) i
	16(\$fp) pos
	12(\$fp) out
	8(\$fp) m2
	4(\$fp) m1
	0(\$fp) dimension

Referencias

- [1] MIPS ABI: Function Calling Convention, Organización de computadoras - 66.20 (archivo func call conv.pdf: <http://groups.yahoo.com/groups/orga-comp/Material/>).