

# Trabajo Práctico N° 1: Assembly MIPS

Martinez Ariel, *Padrón Nro. 88573*  
arielcorreofiuba@gmail.com.ar

Nestor Huallpa, *Padrón Nro. 88614*  
huallpa.nestor@gmail.com

1° Entrega: 20/10/2015

2do. Cuatrimestre de 2015

66.20 Organización de Computadoras

Facultad de Ingeniería, Universidad de Buenos Aires

## Resumen

En el presente trabajo práctico se describirán todos los pasos y conclusiones relacionadas al desarrollo e implementación de multiplicación de matrices de número reales, representados en punto flotante de doble precisión.

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Implementación</b>	<b>3</b>
2.1. Lenguaje . . . . .	3
2.2. Descripción del programa . . . . .	3
2.2.1. Errores posibles . . . . .	3
2.3. Desarrollo de actividades . . . . .	4
2.4. Corridas de pruebas . . . . .	5
<b>3. El código fuente, en lenguaje C</b>	<b>6</b>
<b>4. Funcion multiplicar, en lenguaje Assembly MIPS</b>	<b>9</b>
<b>5. Conclusiones</b>	<b>12</b>
<b>6. Enunciado del trabajo practico</b>	<b>13</b>

## 1. Introducción

El objetivo del presente trabajo práctico es familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI.

## 2. Implementación

### 2.1. Lenguaje

Como lenguaje de implementación se eligió ANSI C [?] ya que el mismo permite una alta portabilidad entre diferentes plataformas. El desarrollo del programa se realizó usando un editor de texto (gedit,vim, kwrite) y compilando los archivos fuente con GCC que viene en linux. Para compilar, ejecutar el siguiente comando:

```
$ make
```

### 2.2. Descripción del programa

Cuando se pasa un nombre como argumento, se verifica que dicho nombre que está haciendo referencia a un archivo (.txt) y no a un archivo de directorio. Una vez hecha la verificación el programa se dispone a leer cada una de las líneas desde el principio. Cada par de líneas leídas como válidas se las cargan en memoria dinámica para su posterior multiplicación, luego el resultado de la multiplicación se lo imprime por salida estándar (stdout). La función `main` se encuentra en `tp1.c` y se encarga de interpretar las opciones y argumentos. En caso de ser una opción, como ayuda o versión, se imprime el mensaje correspondiente y finaliza la ejecución. Cuando no es una opción de ayuda o versión, se procede a procesar los datos de entrada. La salida de estas funciones proveen un código de error que sirve como salida del programa. Los mensajes de versión y ayuda se imprimen por stdout y el programa finaliza devolviendo 0 (cero) al sistema. Los mensajes de error se imprimen por la salida de errores (stderr) y el programa finaliza devolviendo 1 (uno) al sistema.

#### 2.2.1. Errores posibles

1. El procesamiento de la entrada estándar causó el agotamiento del heap.
2. La invocación del programa es incorrecta.
3. Alguno de los archivos es inexistente.

Se contemplan otros errores gracias al uso de la variable externa `errno`. Cuando ocurre un error inesperado, el mismo es informado por stderr y finaliza el programa liberando la memoria que se había solicitado hasta el momento. (con la función `perror()`).

### 2.3. Desarrollo de actividades

1. Se tomó el código fuente generado por el tp0, y se realizaron algunos cambios de implementación en la multiplicación de las matrices. Se dejó de utilizar un arreglo de punto a puntero de doubles y se ahora utiliza un puntero a doubles. También se separo la parte computacional en una funcion llamada multiplicacion.

```
void multiplicar(double* m_a_datos, double* m_b_datos, double* matriz_res,
               int m_a_cantFil, int m_a_cantCol, int m_b_cantCol) {
    int i,j,k = 0;
    int indiceA = 0;
    int indiceB = 0;
    int indiceC = 0;
    double suma = 0.0;

    for (i=0; i<m_a_cantFil; i++)
    {
        for (j=0; j<m_b_cantCol; j++)
        {
            suma = 0.0;
            for (k=0; k<m_a_cantCol; k++)
            {
                indiceA = (i*m_a_cantCol) + k;
                indiceB = j + k*(m_b_cantCol);

                suma = suma + (m_a_datos[indiceA] * m_b_datos[indiceB]);
            }
            indiceC = i*m_b_cantCol + j;
            matriz_res[indiceC] = suma;
        }
    }
}
```

2. Una vez separada la función multiplicación. Primero que nada, se definio el stack frame a reservar y las variables involucradas. Luego, se realizó la codificación en Assembly MIPS, en el archivo multiplicar.S.

Stack Frame:

Tamaño Frame	56 Bytes
52	gp
48	fp
44	
40	indiceC
36	indiceB
32	indiceA
28	k
24	j
20	i
16	matriz_res
12	
8	
4	
0	

3. Para crear el presente informe en formato PDF usando L<sup>A</sup>T<sub>E</sub>X [?] en Linux, ingresar los siguientes comandos:

***\$ pdflatex tp1.tex tp1.pdf***

Para automatizar las pruebas se crearon los siguientes 3 script: `test_suit.sh`, `gen_matriz.sh` y `gen_test.sh`

Contiene una gran batería de pruebas que se pueden ejecutar con el siguiente comando.

se tiene que observar el siguiente resultado

## 2. Prueba 2

```
$ source gen_test.sh
```

```
root@:/home/root/tp0# cat entrada/testMatriz1.txt | ./tp0
ERROR:MEMORIA INSUFICIENTE PARA MATRIZ B
root@:/home/root/tp0#
```

### 3. El código fuente, en lenguaje C

```
/**
 * 6620 - Organizacion del computador
 * Trabajo Practico 1
 * Alumnos:
 *      88614 - Nestor Huallpa
 *      88573 - Ariel Martinez
 */

#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>

typedef struct {
    int cantFil;
    int cantCol;
    double* datos;
} matriz;

extern void multiplicar(double* m_a_datos, double* m_b_datos, double* matriz_res,
                       int m_a_cantFil, int m_a_cantCol, int m_b_cantCol);

/**
 * Imprime cada elemento de array por stdout
 */
void imprimirElementos(double* arreglo, int n)
{
    int i=0;
    for (i=0; i<n; i++) {
        double elemento = arreglo[i];
        printf("%4.2lf", elemento);
    }
}

int multiplicarMatrices(matriz* m_a, matriz* m_b)
{
    int m_a_cantFil = (*m_a).cantFil;
    int m_b_cantCol = (*m_b).cantCol;

    double* matriz_resultado = ((double*) malloc(m_a_cantFil*m_b_cantCol*sizeof(double)));
    if (matriz_resultado == NULL) {
        return 1;
    }

    printf(" %dX%d", m_a_cantFil, m_b_cantCol);
    multiplicar((*m_a).datos, (*m_b).datos, matriz_resultado, (*m_a).cantFil, (*m_a).cantCol, (*m_b).cantCol);
    imprimirElementos(matriz_resultado, m_a_cantFil*m_b_cantCol);
    free(matriz_resultado);
    printf("\n");
    return 0;
}

double* mallocMatrizDouble(int cantFila, int cantCol)
{
    int i;
    double* datos = ((double*) malloc(cantFila*cantCol*sizeof(double)));
    if (datos == NULL)
    {
        return NULL;
    }
    else
    {
        for (i=0; i<cantFila*cantCol; i++)
        {
            datos[i] = 0.0;
        }
    }
    return datos;
}

void liberarMemoria(matriz* p_m)
{
    if (p_m != NULL)
    {
        free((*p_m).datos);
        (*p_m).datos=NULL;
    }
}

void manejarArgumentosEntrada(int argc, char** argv)
{
    int siguiente_opcion;

    /* Una cadena que lista las opciones cortas validas */
    const char* const op_cortas = "hV";

    /* Una estructura de varios arrays describiendo los valores largos */
    const struct option op_largos[] =
    {
        { "help",          0, NULL, 'h' },
        { "version",       0, NULL, 'V' },
        { NULL,             0, NULL, 0 }
    };
};
```



```

if (m_b.cantFil<0)
{
    fprintf(stderr, "ERROR: _FILA_INGRESADA_INVALIDA_PARA_MATRIZ_B\n");
    liberarMemoria(&m_a);
    exit(1);
}
if (m_b.cantCol<0)
{
    fprintf(stderr, "ERROR: _COLUMNA_INGRESADA_INVALIDA_PARA_MATRIZ_B\n");
    liberarMemoria(&m_a);
    exit(1);
}

if (m_b.cantFil==0 || m_b.cantCol == 0){
    fprintf(stderr, "ERROR: _MATRIZ_B_NO_INGRESADA_\n" );
    liberarMemoria(&m_a);
    exit(1);
}

/* se aloja memoria para la matriz b */
m_b.datos = mallocMatrizDouble(m_b.cantFil, m_b.cantCol);
if (m_b.datos==NULL)
{
    fprintf(stderr, "ERROR:MEMORIA_INSUFICIENTE_PARA_MATRIZ_B\n");
    liberarMemoria(&m_a);
    exit(1);
}

for (i=0;i<m_b.cantFil;i++)
{
    for (j=0;j<m_b.cantCol;j++)
    {
        int indice = (i*m_b.cantCol) + j;
        if ( (dato = fgetc(fp)) == '\n'){
            fprintf(stderr, "ERROR: _FALTAN_ELEMENTOS_EN_MATRIZ_B\n" );
            liberarMemoria(&m_a);
            liberarMemoria(&m_b);
            exit(1);
        }
        if (fscanf(fp, "%d", &m_b.datos[indice]) == 0)
        {
            fprintf(stderr, "ERROR: _ELEMENTO_INCORRECTO_EN_LA_MATRIZ_B\n");
            liberarMemoria(&m_a);
            liberarMemoria(&m_b);
            exit(1);
        }
    }
}

if (m_a.cantCol != m_b.cantFil)
{
    fprintf(stderr, "ERROR:NO_SE_PUEDEN_MULTIPLICAR_LAS_MATRICES_DEBIDO_A_SUS_DIMENSIONES\n");
    liberarMemoria(&m_a);
    liberarMemoria(&m_b);
    exit(1);
}
else
{
    if (multiplicarMatrices(&m_a, &m_b) != 0) {
        fprintf(stderr, "ERROR:MEMORIA_INSUFICIENTE_PARA_MATRIZ_A\n");
        liberarMemoria(&m_a);
        liberarMemoria(&m_b);
        exit(1);
    }
}

liberarMemoria(&m_a);
liberarMemoria(&m_b);
}

return 0;
}

```



#### 4. Funcion multiplicar, en lenguaje Assembly MIPS

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

        .text
        .align      2
        .global     multiplicar
        .ent         multiplicar

multiplicar:
        .frame      $fp,56,ra
        .set        noreorder
        .cpload     t9
        .set        reorder

        subu        sp,sp,56
        sw          $fp,48($sp)
        sw          gp,52($sp)
        move        $fp,sp
        sw          a0,56($fp)      #Guardo parametros
        sw          a1,60($fp)
        sw          a2,64($fp)
        sw          a3,68($fp)
        sw          a2,16($fp)
        li          t0,0
        sw          t0,20($fp)      #i=0
        li          t0,0
        sw          t0,24($fp)      #j=0
        li          t0,0
        sw          t0,28($fp)      #k=0
        li          t0,0
        sw          t0,32($fp)      #indiceA=0
        li          t0,0
        sw          t0,36($fp)      #indiceB=0
        li          t0,0
        sw          t0,40($fp)      #indiceC=0

fori:
        lw          t0,20($fp)      #t0=i
        lw          t1,68($fp)      #t1=m_a_cantFil
        bge         t0,t1,retornar
        li          t0,0
        sw          t0,24($fp)      #j=0

forj:
        lw          t0,24($fp)      #t0=j
        lw          t1,76($fp)      #t1=m_b_cantFil
        bge         t0,t1,nexti
        li          t0,0
        sw          t0,28($fp)      #k=0
        li.d        $f0,0.0        #suma=0.0

fork:
        lw          t0,28($fp)      #t0=k
```

```

lw      t1,72($fp)      #t1=m_a_cantCol
bge     t0,t1,nextj

lw      t2,20($fp)      #t2=i
lw      t3,72($fp)      #t1=m_a_cantCol
mul    t2,t2,t3         #t2=i*m_a_cantCol
lw      t4,28($fp)      #t4=k
addu    t2,t2,t4         #t2=i*m_a_cantCol + k
sw      t2,32($fp)      #indiceA = i*m_a_cantCol + k

lw      t2,28($fp)      #t2=k
lw      t3,76($fp)      #t3=m_b_cantCol
mul    t2,t2,t3         #t2=k*m_b_cantCol
lw      t4,24($fp)      #t4=j
addu    t2,t2,t4         #t2=j+k*m_b_cantCol
sw      t2,36($fp)      #indiceB=j+k*m_b_cantCol

lw      t2,56($fp)      #t2=m_a_datos
lw      t3,32($fp)      #t3=indiceA
sll     t3,t3,3
addu    t4,t3,t2         #t4=&m_a_datos[indiceA]

lw      t5,60($fp)      #t5=m_b_datos
lw      t6,36($fp)      #t6=indiceB
sll     t6,t6,3
addu    t7,t6,t5         #t7=&m_b_datos[indiceB]

l.d     $f2,0(t4)        #f2=m_a_datos[indiceA]
l.d     $f4,0(t7)        #f4=m_b_datos[indiceB]
mul.d   $f2,$f2,$f4      #f2=m_a_datos[indiceA]*m_b_datos[indiceB]
add.d   $f0,$f0,$f2      #f0=suma+m_a_datos[indiceA]*m_b_datos[indiceB]

nextk:
lw      t0,28($fp)      #t0=k
addiu   t0,t0,1          #t0=k+1
sw      t0,28($fp)
j       fork

nextj:
lw      t2,20($fp)      #t2=i
lw      t3,76($fp)      #t3=m_b_cantCol
mul    t2,t2,t3         #t2=i*m_b_cantCol
lw      t3,24($fp)      #t3=j
addu    t2,t2,t3         #t2=i*m_b_cantCol+j
sw      t2,40($fp)      #indiceC=i*m_b_cantCol+j

lw      t2,64($fp)      #t2=matriz_res
lw      t3,40($fp)      #t3=indiceC
sll     t3,t3,3
addu    t4,t3,t2         #t4=&matriz_res[indiceC]
s.d     $f0,0(t4)        #matriz_res[indiceC]=suma

lw      t0,24($fp)      #t0=j
addiu   t0,t0,1          #t0=j+1

```

```

        sw      t0,24($fp)
        j       forj

nexti:
        lw      t0,20($fp)      #t0=i
        addiu   t0,t0,1        #t0=i+1
        sw      t0,20($fp)
        j       fori

retornar:
        move    sp,$fp
        lw      $fp,48(sp)
        lw      gp,52(sp)
        addu    sp,sp,56
        j       ra
        .end    multiplicar

```

## 5. Conclusiones

1. Si bien lo solicitado por el programa no era excesivamente difícil, la realización completa del TP llevó cierta dificultad al tener que realizarlo en el contexto solicitado: alta portabilidad, desarrollo en C, e informe hecho en L<sup>A</sup>T<sub>E</sub>X [?].
2. Resulta muy útil analizar previamente como queda el stack frame para poder programar en assembly MIPS.
3. Nos familiarizamos con la forma de manejar los registros en mips y también con las instrucciones de punto flotante. Se tuvo en cuenta la longitud de un double para poder iterar el arreglo de doubles.

## **6.    Enunciado del trabajo practico**

Universidad de Buenos Aires - FIUBA  
66.20 Organización de Computadoras  
Trabajo práctico 1: assembly MIPS  
2<sup>do</sup> cuatrimestre de 2015

\$Date: 2015/09/29 14:34:09 \$

## 1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descrito 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 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

## 4. Descripción

En este trabajo práctico implementaremos el programa descrito en el TP anterior, utilizando el conjunto de instrucciones MIPS, y aplicando la convención de llamadas a funciones explicada en clase [1].

### 4.1. Implementación

En su versión mas simple, cuando es invocado sin argumentos, el programa debe tomar datos provenientes de `stdin` e imprimir el resultado en `stdout`, implementando la funcionalidad descrita en el trabajo práctico inicial.

**Componentes.** La implementación deberá dividirse en tres secciones complementarias:

- **Arranque y configuración:** desde `main()` pasando por el procesamiento de las opciones y configuración del entorno de ejecución, incluyendo la apertura de los archivos de la línea de comando. Este componente deberá realizarse en lenguaje C.
- **Entrada/salida:** lectura y escritura de los *streams* de entrada (matrices), detección de errores asociados. Este componente también debe ser realizado en lenguaje C.
- **Procesamiento.** Recibe un entorno completamente configurado y las matrices a procesar (ya en memoria), calcula y retorna la matriz producto de acuerdo a la descripción del TP anterior. Esta parte del deberá escribirse íntegramente en assembly MIPS.

## 5. Pruebas

Es condición necesaria para la aprobación del trabajo práctico diseñar, implementar y documentar un conjunto completo de pruebas que permita validar el funcionamiento del programa. Asimismo deberán incluirse los casos de prueba correspondientes al TP anterior.

### 5.1. Interfaz

La interfaz de uso del programa coincide con la del primer TP.

## 6. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa.
- Documentación relevante al proceso de compilación: cómo obtener el ejecutable a partir de los archivos fuente.
- Las corridas de prueba, con los comentarios pertinentes.
- El código fuente, en lenguaje C.
- Este enunciado.

## 7. Fechas

Fecha de vencimiento: martes 20/10/2015.

## 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/>).