Trabajo Practico N^0 1 "assembly MIPS"

Samanta Loiza, *Padrón Nro. 91.935*
samiloiza@gmail.com
Victoria Pérez Bustos, *Padrón Nro. 92.060*
vickyperezbustos@gmail.com

2do. Cuatrimestre de 2015
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

Índice

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. 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*:

```
4x2 3 2 1 4 4 1 2 4
3x1 4 3 2
```

- *charTest.txt*

```
2x3 1 a 3 4 5 6.1
3x2 1 0 0 0 0 1
3x3 1 2 3 4 a 6.1 3 2 1
3x1 1 1 0
```

- *test2.txt*

```
2x2 -2.0 1.0 3.0 4.0
2x3 1.0 4.0 6.0 2.0 5.0 3.0
2x3 1.0 2.0 3.0 4.0 5.0 6.0
3x1 -1.0 -0.0 0.0
```

- *testHigh.txt*

```
8x13 32.93 -23.65 -23.67 -91.48 -79.55 -29.34 -65.75 42.30 -72.10
-79.12 4.81 11.78 96.60 35.64 98.57 -90.94 -95.57 74.64 80.47
58.21 30.85 47.17 95.71 30.31 56.66 -68.6 -19.25 -55.22 -30.71
71.39 4.2 2.27 -75.42 76.24 81.24 47.37 -68.34 -6.36 88.85 -56.34
71.64 -85.19 -43.13 -77.39 -38.79 -9.82 -38.28 23.74 -59.19 79.23
-11.37 84.33 -52.16 4.42 65.16 32.63 -26.90 -96.4 67.25 -30.66
56.66 73.72 -36.25 -17.12 -63.90 -45.87 -4.9 74.46 -35.78 13.42
27.41 61.45 -36.60 41.65 63.24 -68.47 -94.43 -89.39 -13.60 81.17
-72.87 16.4 12.37 -57.21 -97.96 -67.91 -43.15 65.12 10.17 -88.63
-54.46 18.21 -61.46 -33.68 -42.78 45.71 34.66 -82.5 -1.0 11.2
-45.72 20.3 -73.75 67.95
13x21 -100.55 57.23 92.56 44.15 30.98 21.29 -88.35 -16.16 -50.51
14.53 -56.70 95.21 43.0 73.83 -39.29 -93.54 -72.85 48.0 -62.98
36.51 -86.9 -72.0 97.74 -35.18 84.92 26.1 -42.9 -42.62 -68.15
45.93 53.25 50.85 46.4 96.65 -41.83 1.27 46.86 19.85 79.28 -3.83
96.12 -99.44 54.78 -89.90 -62.31 -59.65 89.92 9.48 39.33 -93.80
-31.91 -77.41 -84.59 14.55 24.44 -63.86 12.65 54.87 -40.19 89.70
-99.52 -55.68 -35.91 7.37 42.59 -24.97 -24.21 -77.35 -14.17 -7.72
29.87 -25.39 70.71 -11.31 36.85 -68.18 45.54 -39.57 -55.92 76.84
1.35 15.57 22.87 95.91 1.17 21.89 53.38 -87.94 -42.15 98.60 -31.53
56.82 -63.76 86.80 53.86 16.66 74.78 -71.92 -55.20 -95.30 88.80
-48.48 90.93 -10.48 -26.48 30.51 80.6 -91.56 -6.23 -65.75 80.75
36.95 -98.73 76.27 -55.63 24.16 -48.96 -36.81 26.64 46.36 -100.86
-81.74 -16.49 52.71 -57.64 76.71 10.78 46.29 4.69 -62.73 88.78
28.50 -55.18 56.13 45.77 -71.32 -39.14 -41.90 -5.0 44.4 -71.38
5.87 29.31 32.62 -51.82 82.38 68.43 -2.77 2.63 -52.58 -4.76 95.55
-68.76 29.45 55.92 85.66 40.45 34.32 47.84 49.33 49.93 68.13 -58.47
52.82 -99.17 81.23 -9.20 -31.61 -34.90 97.81 4.79 15.14 -52.52
67.9 9.5 26.18 31.48 40.86 -35.0 -16.89 -64.31 53.1 70.81 46.48
31.33 -44.30 -20.24 -54.41 68.94 65.6 -78.73 -39.35 -96.58 -65.34
58.24 57.66 -53.19 -14.81 40.51 30.54 -26.49 -52.12 -83.7 -41.86
97.69 -93.6 -45.89 -40.44 -76.2 40.23 -99.59 39.17 -81.65 -49.39
70.34 -7.60 -75.49 43.40 -57.26 77.24 -54.42 -7.81 12.63 -35.44
-45.22 2.11 -58.60 -62.89 -44.15 12.42 -8.51 -49.82 90.11 96.17
-44.93 92.71 73.41 -85.95 21.27 -6.73 -52.28 55.44 -12.95 67.71
-62.82 90.50 -98.40 -47.83 -43.74 19.36 10.38 36.88 -20.63 -46.4
77.23 -2.14 64.94 55.62 -88.53 9.91 24.44 87.11 -41.52 -10.64
63.46 13.80 -38.3 13.73 19.70 -92.47 5.11 -94.88 78.60
```

- *matrix.dat*

```
2x14 -11.75 -3.14 77.18 -63.18 71.66 19.68 9.31 37.60 50.7 -9.19
96.24 61.39 31.97 -65.46 72.45 -92.50 -16.21 -27.88 -52.56 67.62
53.48 40.39 79.73 -90.13 97.2 74.26 -92.54 22.40
14x2 -2.49 17.82 -48.65 -6.40 -16.88 -86.71 31.12 66.43 -79.14
-2.69 -26.32 -6.92 -86.26 95.93 -44.74 11.92 -49.84 52.50 -46.33
3.83 71.18 32.71 22.63 -15.24 -17.28 2.74 58.89 19.19
```

4.1. 1^o Prueba

En esta prueba probamos que la matriz sea válida.

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

4.2. 2^o Prueba

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

4.3. 3^o Prueba

```
   cat fileTest/badDim.txt | ./tp0
      Matrices incorrectas para la multiplicacion.
```

4.4. 4^o Prueba

```
   ./tp0 <fileTest/test2.txt
      2x3 4.000000 13.000000 15.000000 11.000000 32.000000 30.000000
      2x1 1.000000 4.000000
```

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*:

```
  2x3 0.000000 -3.000000 -9.000000 11.000000 32.000000 30.000000
  2x1 -1.000000 -4.000000
```

4.5. 5^o Prueba

```
   ./tp0 <fileTest/testHigh.txt
      2x14 -11.75 -3.14 77.18 -63.18 71.66 19.68 9.31 37.60 50.7 -9.19
96.24 61.39 31.97 -65.46 72.45 -92.50 -16.21 -27.88 -52.56 67.62 53.48
40.39 79.73 -90.13 97.2 74.26 -92.54 22.40
      14x2 -2.49 17.82 -48.65 -6.40 -16.88 -86.71 31.12 66.43 -79.14
-2.69 -26.32 -6.92 -86.26 95.93 -44.74 11.92 -49.84 52.50 -46.33 3.83
```

```
71.18 32.71 22.63 -15.24 -17.28 2.74 58.89 19.19
```

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*:
  `2x2 -10030.273500 -6395.865400 11404.737000 12786.414000`

4.6. 6º Prueba

```
./tp0 <fileTest/matrix.dat
      8x21 -11126.102100 -13288.959200 14515.208100 19749.962800
25497.151500 -15510.146300 -11002.804600 1356.348200 4685.098000
-2270.241700 -4513.409600 5307.925900 18084.590000 8114.853800
7958.038500 -3593.734300 -7100.608200 -7465.797700 2179.386100
-4157.455500 -14581.579300 -13075.981400 33228.881400 -2712.679500
-5606.596600 -3230.548100 19307.949800 -6990.362500 -2344.408900
11014.834700 4354.955500 22121.771600 -4110.066200 11046.958500
-10222.682400 4631.623000 -9962.778200 -14260.687100 -9389.894900
10041.257700 10358.347200 -8052.816000 9367.293600 -6869.304900
6301.062600 -1325.058800 -17047.037600 7131.426700 9996.525200
6518.406600 8570.843500 20938.290600 4204.119000 1598.901400 -10762.966700
10713.900100 10805.423300 -1888.683500 13200.619800 -27634.733400
15982.620900 -27132.058200 22442.698800 -19526.702700 9371.658900
1430.324800 20060.790300 9699.134400 -11461.632100 -16328.800600
7273.621700 7421.763700 13765.782100 -2585.405800 -5880.748100
6301.986500 3789.907800 -182.741000 17597.761200 3470.997000 -7475.954700
-1975.874300 7259.536900 1096.580500 15213.553000 -12607.289300
-6791.444100 -11276.486900 -3731.605300 -3951.318500 14057.188500
445.232600 -3934.666000 -8437.663700 -6215.945900 -4348.649200
-15917.913500 -7572.770200 -4821.899700 31876.786000 6177.989500
14908.010800 -19396.803600 4033.729900 5297.832900 26458.546800
-14019.077900 -20083.530500 -8709.405700 -18190.707200 3378.558100
28828.020700 -15658.368400 4282.556400 -19319.328700 -3142.463200
-9943.825400 -19210.755800 -4856.559500 -3920.227000 24638.641700
-1524.851600 20527.216700 -11214.408300 -3865.828200 966.593100
-642.579800 -9057.347100 -7302.655200 12632.864200 -31096.193400
-14967.308200 10667.310400 -7508.581300 -12768.603800 20971.060000
11957.837000 6529.868800 -5415.885600 -12439.035200 -14079.185800
20498.193100 1230.620800 10347.122700 1518.970500 9219.912300
-19963.684000 -2481.169700 -8094.811300 4297.159000 6324.583000
-10982.571000 4643.400800 639.486400 2974.991000 -3690.021600
14405.244800 -4133.857600 384.514400 679.589300 7313.605900 -2110.321700
-8029.942900 2581.634600 -15189.109700 11596.942200 -17695.780500
-1148.915400
```

Con esta prueba multiplicamos dos pares de matrices y mostramos por pantalla el resultado obtenido.

5. Conclusiones

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

6. Codigo 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>


#define SIZE_MAT 50

void printHelp();
void printLinea(int fil, int col, double* matrix);
void finally(double* m1,double* m2,double* out);

double* leerMatriz(int* fil, int* col);

int  checkArguments(int,char*[]);

extern void multiplicarMatrices(int f1, int c1, int f2,
    int c2, double* m1, double* m2, double* out);

//void multiplicarMatrices(int f1, int c1, int f2, int c2,
     double* m1, double* m2, double* out);


#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\t Print version and quit. \n"
            " -h, --help \n\t Print this information and quit
                .\n"
            " Examples: \n"
            "  ./tp0 < in.txt > out.txt \n"
            "  cat in.txt | ./tp0 > out.txt \n";

    printf("%s", help);
}

void printLinea(int fil, int col, double* matrix){
    printf("%dx%d ", fil, col);
    if (ferror(stdout)){
        fprintf(stderr, "Error printing stdout\n");
        free(matrix);
        matrix = NULL;
        exit(EXIT_FAILURE);
    }
    for(int i=0;i<fil*col;i++){
        printf(" %f",matrix[i]);
        if (ferror(stdout)){
            fprintf(stderr, "Error printing stdout\n");
            free(matrix);
            matrix = NULL;
            exit(EXIT_FAILURE);
        }
    }
    printf("\n");
    if (ferror(stdout)){
        fprintf(stderr, "Error printing stdout\n");
        free(matrix);
        matrix = NULL;
        exit(EXIT_FAILURE);
    }
}
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;
}

// void multiplicarMatrices(int f1, int c1, int f2, int c2,
    double* m1, double* m2, double* out) {
//
//  int pos = 0;
//  int i,j,k, m;
//  for( i =0 ; i < f1*c1; i=i+c1){
//      for(k=0; k < c2 ; k++){
//          double sum = 0.0;
//          m = i;
//          // printf("Entro for de K con k = %d\n",k );
//          for(j=k; j < f2*c2; j=j+c2){
//              sum+= m1[m]*m2[j];
//              m++;
//          }
//          out[pos] = sum;
//          pos++;
//      }
//  }
// }
//

double* leerMatriz(int* fil, int* col ){
    char x;
    double* m = NULL;
    double value = 0.0;
    int cantNums;
    if(( fscanf(stdin, "%d %c %d", fil, &x, col )==3) && !feof
        (stdin)){
        if (ferror (stdin)){
            printf ("Error reading stdin\n");
            if(m!=NULL){
                free(m);
                m = NULL;
            }
            exit(EXIT_FAILURE);
        }
        cantNums = (*fil) * (*col);
        m = (double*) malloc(sizeof(double)*cantNums);
        if(m == NULL){
            fprintf(stderr, "Error malloc \n");
        }

        for(int j =0;j<cantNums; j++){
            if(fscanf(stdin, "%lf", &value) ==1){
                if (ferror (stdin)){
                    printf ("Error reading stdin\n");
                    if(m!=NULL){
                        free(m);
```

```
                        m = NULL;
                    }
                    exit(EXIT_FAILURE);
                }
                m[j] = value;
            }else{
                fprintf(stderr, "Matriz invÃ¡lida \n");
                if(m!=NULL){
                    free(m);
                    m = NULL;
                }
                exit(EXIT_FAILURE);
            }

        }


    }else{
        if(m!=NULL){
            free(m);
            m = NULL;
        }
        exit(EXIT_FAILURE);
    }

    return m;
}

void finally(double* m1,double* m2,double* out){
    if(m1 != NULL){
        free(m1);
        m1 = NULL;
    }
    if(m2 != NULL){
        free(m2);
        m2 = NULL;
    }
    if(out !=NULL){
        free(out);
        out = NULL;
    }
}

 int main(int argc, char *argv[])
{
    if (!checkArguments(argc,argv)){
        return 1;
    }else{
        do{
            int fil1 = 0, col1 = 0, fil2 = 0, col2 = 0;
            double* m1 = NULL;
            double* m2 = NULL;
            double* out = NULL;
```

```
            m1 = leerMatriz(&fil1 , &col1 );
            m2 = leerMatriz(&fil2 , &col2 );

            if (col1== fil2){
                out = (double*) malloc(sizeof(double)*fil1*
                    col2);
                if(out == NULL){
                    fprintf(stderr, "Error malloc \n");
                }
                multiplicarMatrices(fil1 , col1 , fil2 , col2 , m1
                    , m2, out);
                printLinea(fil1 , col2 , out);


                finally(m1,m2,out);
            }else{
                fprintf(stderr,"Matrices incorrectas para la
                    multiplicacion. \n");
                finally(m1,m2,out);
                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  multiplicarMatrices
    .ent    multiplicarMatrices
multiplicarMatrices:

        .frame  $fp, 32, ra
        .set    noreorder
        .cpload t9
        .set    reorder
        subu sp, sp,64
        .cprestore 56
        sw $fp, 60(sp)
        move $fp,sp

        sw      a0, 0($fp) #guardo f1, c1, f2,c2
        sw      a1, 4($fp)
        sw      a2, 8($fp)
        sw      a3, 12($fp)

        lw      a0, 0($fp) #f1 LTA #cargo en registros los
            argumentos recibidos por a funciÂ´on
        lw      a1,4($fp) #c1 LTA
        lw      a2,8($fp) #f2 LTA
        lw      a3,12($fp) #c2 LTA
        lw      t0, 80($fp) #dir M1
        lw      t1, 84($fp) #dir M2
        lw      t2,88($fp) #dir outs

        sw      t0, 16($fp) #guardo en SF la direccion de m1,
            m2, out
        sw      t1, 20($fp)
        sw      t2, 24($fp)

        lw      t0,0($fp) #t0=f1
        lw      t1,4($fp) #t1=c1
        lw      t2,8($fp) #t2=f2
        lw      t3,12($fp) #t3=c2

        li      t0,0 #cargo valor el valor 0 en t0
        sw      t0,28($fp) #pos = 0
        sw      t0,32($fp) # i = 0
        lw      t1,0($fp) #t1=f1
        lw      t2,4($fp) #t2=c1
        mul     t1,t1,t2 #t1=f1*c1
        sw      t1,48($fp) #cantidad de elementos matriz 1
```

```
        lw      t1,8($fp) #t1=f2
        lw      t2,12($fp) #t2=c2
        mul     t1,t1,t2 #t1=f2*c2
        sw      t1,52($fp) #cantidad de elementros matriz 2

f1:     lw      t0, 32($fp)  #t0= i
        lw      t1, 48($fp)  #t1 = f1*c1
        bge     t0,t1,endf1  #i< f1*c1
        li      t0,0
        sw      t0,40($fp) # k=0
f2:     lw      t0,40($fp) #t0 = k
        lw      t1,12($fp)  #t1= c2
        bge     t0,t1,endf2   #k<c2
        li.d    $f0,0.00000  #f0=sum=0
        lw      t1,32($fp) #t1 = i
        sw      t1, 44($fp) #m =i
        lw      t2, 40($fp) #t2=k
        sw      t2, 36($fp) #j=k
f3:     lw      t2, 36($fp) #t2 =j
        lw      t3, 52($fp) #t3=f2*c2
        bge     t2,t3,endf3 #j< f2*c2
        lw      t1,44($fp) #t1=m
        lw      t3, 16($fp) #t3=m1
        sll     t1,t1,3 #t1=m*8 (8bytes)
        addu    t3,t3,t1  #t3 = &m1[m]
        lw      t4, 20($fp) #t4=m2
        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
        add.d   $f0,$f0,$f2
        lw      t1,44($fp) #cargo en t1 el valor de m
        addi    t1,t1,1 #m = m+1
        sw      t1,44($fp) # actualizo m
        lw      t1,36($fp) #t1=j
        lw      t2,12($fp)  #t2 =c2
        addu    t1,t1,t2  #t1= j+c2
        sw      t1,36($fp) #j= j+C2
        j       f3
endf3:  lw      t1, 24($fp) #t1=out
        lw      t2, 28($fp) #t2=pos
        sll     t2,t2,3  #t2=pos*8
        addu    t1,t1,t2 # t1= &out[pos]
        s.d     $f0,0(t1)  #out[pos]=sum
        lw      t2,28($fp) #t2=pos
        addi    t2,t2,1 #pos++
        sw      t2,28($fp)
        lw      t0,40($fp)
        addi    t0,t0,1 #k++
        sw      t0,40($fp)
        j       f2
endf2:  lw      t0, 32($fp) #t0 = i
        lw      t1,4($fp) #t1 =c1
```

```
        addu    t0,t0,t1 # i=i+c1
        sw      t0,32($fp)
        j       f1


endf1:  move sp, $fp
    lw $fp, 60(sp) #recupero fp
    addu sp,sp,64 #restablezco stack frame
    jr ra
    .end multiplicarMatrices
```

8. Stack Frame

Dado que la función implementada 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 56 almacenamos las variables auxiliares utilizadas para los cálculos, de 56 a 64 almacenamos fp y gp.

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/).