Trabajo Práctico nro. 1: assembly MIPS

Luis Arancibia, *Padrón 88744*
aran.com.ar@gmail.com

Matias Cerrotta, *Padrón 89992*
matias.cerrotta@gmail.com

Gabriel La Torre, *Padrón 87796*
latorregab@gmail.com

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

Abstract

Este trabajo práctico nro. 1 busca familiarizar a los alumnos con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema piloto que se presentará más abajo.

Se mostrará el código C y el código Assembly generado para la correcta resolución.

El programa correrá tanto en NetBSD/pmax, usando el simulador GXemul provisto por la cátedra, como en la versión de Linux (Knoppix, RedHat, Debian, Ubuntu) usada para correr el simulador, Linux/i386.

1 Introducción

El programa, a escribir en lenguaje C, deberá multiplicar matrices de números reales, representados en punto flotante de doble precisión. Las matrices a multiplicar ingresarán por entrada estándar (*stdin*), donde cada línea describe una matriz completa en formato de texto.

NxM a1,1 a1,2 ... a1,M a2,1 a2,2 ... a2,M ... aN,1 aN,2 ... aN,M

La línea anterior representa a la matriz A de N filas y M columnas. Los elementos de la matriz A son los a x,y , siendo x e y los indices de fila y columna respectivamente 1 . El fin de línea es el caracter newline. Los componentes de la lńea 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.

Por ejemplo, dada la siguiente matriz:
1 2 3
4 5 6

Su representación sería:
2x3 1 2 3 4 5 6

Por cada par de matrices que se presenten en su entrada, el programa deberá multiplicarlas y presentar el resultado por su salida esándar (stdout) en el mismo formato presentado anteriormente, hasta que llegue al final del archivo de entrada (EOF). Ante un error, el progama deberá informar la situación inmediatamente (por stderr) y detener su ejecución. Tener en cuenta que también se condidera un error que a la entrada se presenten matrices de dimensiones incompatibles entre sí para su multiplicación.

2 Documentación

2.1 Compilación

El programa se compilará con la siguiente instrucción para utilizar la implementación en C:

```
gcc -Wall -O0 -o tp1 tp1.c multiplicar.c
```

Y con la siguiente instrucción para la implementación en Assembly:

```
gcc -Wall -O0 -o tp1 tp1.c multiplicar.S
```

Los tests se ejecutarán con el siguiente script:

```
./tests.sh
```

Ejemplo de la salida de ejecución:

```
$ ./tests.sh
Tests #0 success_normal:  OK
Tests #1 success_espacios:  OK
Tests #2 error_dimension:  OK
Tests #3 error_dimension_caracter_invalido:  OK
Tests #4 error_dimension_cero:  OK
Tests #5 error_matriz1:  OK
Tests #6 error_matriz2:  OK
```

2.2 Utilización

Opciones de ejecución:

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

Ejemplos de ejecución:

```
Examples:
./tp1 -h
./tp1 -V
./tp1 < in_file > out_file
./tp1 < in.txt > out.txt
cat in.txt | tp0 > out.txt
```

A continuación se presenta un ejemplo de prueba:

```
$ cat in.txt
2x3 1 2 3 4 5 6.1
3x2 1 0 0 0 0 1
3x3 1 2 3 4 5 6.1 3 2 1
3x1 1 1 0

$ cat in.txt | ./tp0
2x2 1 3 4 6.1
3x1 3 9 5
```

3 Casos de pruebas

Se crearon los siguientes casos de pruebas:

1. Caso normal.
2. Utilizando espacios entre elementos de matriz.
3. Dimensiones de matrices incompatibles.
4. Dimensiones de matrices con valores no alfanuméricos.
5. Dimensiones de matrices con valores inválidos. -
6. Elementos de más con respecto a la dimensión.
7. Elementos de menos con respecto a la dimensión.

4 Código fuente

```c
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <string.h>
#include "multiplicar.h"

#define MAX_LINE_LENGTH 512
#define MAX_DIMENSION_LENGTH 4
#define EXIT_OK 0
#define EXIT_ERROR 1

enum ACCION {
  EMPTY,
  HELP,
  VERSION,
  MULTIPLICAR,
  ERROR
};

void multiplicarMatriz();
enum ACCION procesarArgumentos(int argc, char** argv);

int main(int argc, char **argv) {
  enum ACCION comando = procesarArgumentos(argc, argv);
  switch (comando) {
    case HELP:
      printf("Usage:\n"
      "\ttp0 -h\n"
      "\ttp0 -V\n"
      "\ttp0 < in_file > out_file\n"
      "Options:\n"
      "\t-V, --version\tPrint version and quit.\n"
      "\t-h, --help\tPrint this information.\n"
      "Examples:\n"
      "\ttp0 < in.txt > out.txt\n"
      "\tcat in.txt | tp0 > out.txt\n");
      break;
    case VERSION:
      printf("tp0 v0.1\n");
      break;
    case MULTIPLICAR:
      multiplicarMatriz();
      break;
    case ERROR:
    default:
      return EXIT_ERROR;
  }
  return EXIT_OK;
}


void imprimirMatriz(double* matriz, int filas, int columnas) {
  printf("%dx%d ", filas, columnas);
  int i;
  for (i = 0; i < filas * columnas; i++) {
      printf("%g ", matriz[i]);
  }
  printf("\n");
}

```

```
61 // Devuelve null sino hay memoria
62 char* append(char* original, int originalSize, char* toAppend) {
63   if (original == NULL) {
64     original = malloc(sizeof(toAppend));
65   }
66   if (original == NULL) {
67     //printf("out of memory\n");
68     return NULL;
69   }
70   original = (char *) realloc(original, (originalSize + 1) * sizeof
       (char));
71   if (original == NULL) {
72     fprintf(stderr, " No hay mÃ¡s memoria.\n");
73     return NULL;
74   }
75   original[originalSize] = *toAppend;
76   return original;
77 }
78
79
80 void leerDimension(int* filas, int* columnas) {
81   int newChar;
82   char c;
83   char* buffer = (char *) malloc(sizeof(char) *
       MAX_DIMENSION_LENGTH);
84   int i = 0, total = 0;
85   while((newChar = getchar()) != EOF) {
86     c = (char) newChar;
87
88     if (c == 'x') {
89       *filas = atoi(buffer);
90       memset(buffer,0,strlen(buffer));
91       i = 0;
92       continue;
93     }
94
95     if(c == ' ') {
96       *columnas = atoi(buffer);
97       break;
98     }
99
100    if (c != '\n' && (c < '0' || c > '9')) {
101      free(buffer);
102      fprintf(stderr, "Dimension incorrecta.\n");
103      exit(EXIT_ERROR);
104    }
105
106    if(total >= MAX_DIMENSION_LENGTH) {
107      char* oldBuffer = buffer;
108      buffer = append(buffer, total,&c);
109      if (buffer == NULL) {
110        free(oldBuffer);
111        fprintf(stderr, " No hay mÃ¡s memoria.\n");
112        exit(EXIT_ERROR);
113      }
114    } else {
115      buffer[i] = c;
116    }
117    i++;
118    total++;
119  }
120  free(buffer);
```

```
121
122   if (*filas == 0 || *columnas == 0) {
123     fprintf(stderr, "Dimension incorrecta.\n");
124     exit(EXIT_ERROR);
125   }
126
127   if (newChar == EOF)
128     exit(EXIT_OK);
129 }
130
131
132 void leerMatriz(double* matriz, int filas, int columnas) {
133   int newChar;
134   char c;
135   char* buffer = (char *) malloc(sizeof(char) * MAX_LINE_LENGTH);
136   int i = 0, total = 0, elementos = 0, fila = 0, columna = 0;
137   while((newChar = fgetc(stdin)) != EOF) {
138     if (newChar == EOF) {
139       break;
140     }
141     c = (char) newChar;
142     total++;
143
144     if (elementos >= filas * columnas) {
145       fprintf(stderr, "%s\n", "Dimension no compatible con datos de
              matriz.");
146       free(buffer);
147       exit(EXIT_ERROR);
148     }
149
150     if (c == ' ' || c == '\n') {
151       // elimino espacios consecutivos
152       if (strlen(buffer) == 0)
153         continue;
154
155       elementos++;
156       matriz[(fila*columnas)+columna] = atof(buffer);
157       memset(buffer,0,strlen(buffer));
158       i = 0;
159       if (columna >= columnas - 1) {
160         fila++;
161         columna = 0;
162       } else {
163         columna++;
164       }
165
166       if(c == '\n')
167         break;
168
169       continue;
170     }
171
172     if(total >= MAX_LINE_LENGTH) {
173       char* oldBuffer = buffer;
174       buffer = append(buffer, total, &c);
175       if (buffer == NULL) {
176         free(oldBuffer);
177         fprintf(stderr, "%s\n", "No hay mÃ¡s memoria para guardar
              la matriz.");
178       }
179       exit(EXIT_ERROR);
180     }
```

```
181
182       buffer[i] = c;
183       i++;
184     }
185     free(buffer);
186
187     if (elementos < filas * columnas) {
188       fprintf(stderr, "%s\n", "Dimension no compatible con datos de
              matriz.");
189       exit(EXIT_ERROR);
190     }
191
192     if (newChar == EOF)
193       exit(EXIT_OK);
194 }
195
196
197 double* crearMatriz(int filas, int columnas) {
198   double* matriz = (double*) malloc(sizeof(double) * filas *
          columnas);
199   return matriz;
200 }
201
202
203 void multiplicarMatriz() {
204
205   while (1) {
206     int filasA, columnasA;
207     leerDimension(&filasA, &columnasA);
208     double* matrizA = crearMatriz(filasA, columnasA);
209     leerMatriz(matrizA, filasA, columnasA);
210
211     int filasB, columnasB;
212     leerDimension(&filasB, &columnasB);
213     double* matrizB = crearMatriz(filasB, columnasB);
214     leerMatriz(matrizB, filasB, columnasB);
215
216     if (columnasA != filasB) {
217       fprintf(stderr, "%s\n", "Dimensiones incorrectas de matrices.
              ");
218       free(matrizA);
219       free(matrizB);
220       exit(EXIT_ERROR);
221     }
222
223     double* matrizC = crearMatriz(filasA, columnasB);
224
225     multiplicarMatrices(filasA, matrizB, matrizC, matrizA,
226      columnasB, columnasA);
227
228     imprimirMatriz(matrizC, filasA, columnasB);
229
230     free(matrizA);
231     free(matrizB);
232     free(matrizC);
233   }
234 }
235
236
237 enum ACCION procesarArgumentos(int argc, char** argv) {
238   // valores por defecto
239   enum ACCION comando = EMPTY;
```

```
240
241     /* La funcion getopt obtiene el siguiente argumento especificado
            por argc y argv
242     * mas info: http://www.gnu.org/software/libc/manual/html_node/
            Using-Getopt.html#Using-Getopt
243     * La cadena "hVbri:o:" indica que h, V no tienen argumentos.
244     */
245    int c;
246    while ((c = getopt(argc, argv, "hV")) != -1) {
247      switch (c) {
248        case 'h':
249          comando = HELP;
250          break;
251        case 'V':
252          comando = VERSION;
253          break;
254        default:
255          comando = ERROR;
256          break;
257      }
258    }
259
260    if (comando == EMPTY)
261      comando = MULTIPLICAR;
262
263    return comando;
264  }
```

```
1  void multiplicarMatrices(int filasA, double* matrizB, double*
       resultado, double* matrizA, int columnasB, int columnasA) {
2    int i, j, k;
3    for (i = 0; i < filasA; ++i) {
4        for (j = 0 ; j < columnasB ; ++j) {
5          resultado[i*columnasB+j]=0;
6            for ( k = 0; k < columnasA; ++k) {
7              resultado[i*columnasB+j] = (resultado[i*columnasB+j] +
                   (matrizA[i*columnasA+k] * matrizB[k*columnasB+j]));
8            }
9      }
10   }
11 }
```

5 Código $MIPS^{TM}$

```
1  #
2  # Aplicacion que implementa la funcion multiplicar.
3  #
4  #
5  ###############
6  # Stack frame #
7  ###############
8  #
9  # ------------ 16
10 #       xxx
11 # ------------ 12
12 #       ra
13 # ------------ 8
14 #       gp
15 # ------------ 4
16 #       fp
17 # ------------ 0
18 #
19 ################################
20 # Detalle de registros usados: #
21 ################################
22 #       a0: cant filas A
23 #       a1: puntero a matriz B
24 #       a2: puntero a matriz C
25 #       a3: puntero a matriz A
26 #
27 #       s0: temporal usada para calculos
28 #       s1: temporal usada para calculos
29 #       s2: temporal usada para calculos
30 #       t1: temporal usada para calculos
31 #       t2: temporal usada para calculos
32 #       t3: temporal usada para calculos
33 #       t5: temporal usada para calculos
34 #       t7: temporal usada para calculos
35 #
36 #       f4: temporal usada para calculos float
37 #       f6: temporal usada para calculos float
38 #       f8: temporal usada para calculos float
39 #
40
41 # include <mips/regdef.h>
42
43         .text
44         .align 2
45
46         .globl multiplicarMatrices
47         .ent multiplicarMatrices
48
49 multiplicarMatrices:
50
51         # debugging info: descripcion del stack frame
52         .frame  $fp, 16, ra     # $fp: frame pointer, 16: tamanio
               stack frame, ra: return address
53
54         # bloque para c\'odigo PIC
55         .set    noreorder       # apaga reordenamiento de
               instrucciones
56         .cpload t9              # directiva usada para c\'odigo PIC
57         .set    reorder         # enciende reordenamiento de
               instrucciones
```

```
58
59          subu    sp, sp, 16      # creo stack frame
60          sw      $fp, 0(sp)      # guardo valor de fp
61          sw      gp, 4(sp)       # guardo valor de gp
62          sw      ra, 8(sp)       # guardo valor de ra
63          move    $fp, sp
64
65
66          # directiva para c\'odigo PIC
67          .cprestore 4            # inserta aqui "sw gp, 4(sp)", mas
                "lw gp, 4(sp)" luego de cada jal.
68
69          addiu   t0, a0, 0         # lw t0, cantf_c_i
70          lw      t1, 32(sp)        # lw t1, cantc_c_j
71          lw      t2, 36(sp)        # lw t2, cantc_a_k
72  #       ------------------------------------------------------------------------


73          #no hago una estiqueta para i porque nunca volveremos a
                hacer i =0 para
74          #Las filas
75          li      s0, 0           # i = 0; initialize 1st for loop
76  loop1:
77          # Al contrario para las columnas de B se hace una corrida
                por cada fila
78          #Varias veces a este punto
79          li      s1, 0           # j = 0; restart 2nd for loop
80  loop2:
81          # Por cada valor de C deberemos iterar sobre todos los
                valores de K
82          #
83          li      s2, 0           # k = 0; restart 3rd for loop
84
85  #Operaciones con C
86          mul     t3, s0, t1      # en t3 la cantidad de columnas de
                c
87          addu    t3, t3, s1      # en t3 sumo el valor de j-> la
                posicion en el array
88          sll     t3, t3, 3       #  el offset en bytes en el array
89          addu    t3, a2, t3      # cargo en t3 la direccion de C{i}{
                j}
90          sw      zero, 0(t3)     # inicializo C{i}{j} con cero
91    l.d     $f4, 0(t3)
92          #add t4, zero, zero # inicializo en t4 en cero que serÃ¡ el
                valor de C{i}
93
94  #Operaciones con A
95  loop3:
96          mul     t7, s0, t2      # en t7 la cantidad de columnas de
                A "K"
97          addu    t7, t7, s2      # ahora le sumamos K y obtenemos la
                posicion en el arra
98          sll     t7, t7, 3       # multiplcamos por 4 y obtenemos el
                offset de A[i][k]
99          addu    t7, a3, t7      # obtenemos la direccion de A[i][k]
100         l.d     $f8, 0(t7)      # guardamos en t8 el valor de A[i][
                k]
101
102 # Ahora con B
103         mul     t5, s2, t1      # t5 = k * 4 (size of row of b)
104         addu    t5, t5, s1      # t5 = k * size(row) + j
105         sll     t5, t5, 3       # t5 = byte offset off [k][j]
```

```
106         addu    t5, a1, t5      # t5 = byte address of b[k][j]
107         l.d     $f6, 0(t5)
108
109         mul.d   $f6, $f8, $f6   # t6 = a[i][k] * b[k][j]
110         add.d   $f4, $f4, $f6   # t4 = c[i][j] + a[i][k] * b[k][j]
111
112         s.d     $f4, 0(t3)      # c[i][j] = t4  guardo el valor
                calculado en c[i][j]
113
114         addiu   s2, s2, 1       # k = k + 1
115         bne     s2, t2, loop3   # if (k != 4) go to loop3
116
117         addiu   s1, s1, 1       # j = j + 1
118         bne     s1, t1, loop2   # if (j != 4) go to loop2
119
120         addiu   s0, s0, 1       # i = i + 1
121         bne     s0, t0, loop1   # if (i != 32) go to loop1
122
123 #   ------------------------------------------------------------------------
    |
124         lw      $fp, 0(sp)      # reestablesco valor de fp
125         lw      gp, 4(sp)       # reestablesco valor de gp
126         lw      ra, 8(sp)       # reestablesco valor de ra
127         addi    sp, sp, 16      # destruyo stack frame
128         jalr    ra
129
130         .end multiplicarMatrices
```

6 Explicación del Stack

Lo primero que hacemos es restarle 16 al stack pointer. Puesto que queremos guardar el ra (return address), el gp (global pointer) y el fp (frame pointer), estos tres valores nos ocuparán 4 bytes cada una, o sea que necesitaríamos 12 bytes posiciones, pero por convención, el mínimo que debe ocupar el stack es 16 bytes. Por eso restamos 16 bytes al valor del stack pointer y no 12. Una vez que tenemos el stack en su nueva posición, procedemos a guardar los valores del fp, gp y ra en las posiciones relativas al sp, 0, 4 y 8, respectivamente. Luego cargamos en el frame pointer el valor actual del stack pointer para que apunten a la misma dirección de memoria. Posteriormente, hay que notar que al levantar los parámetros que no entraron en a0, a1, a2 y a3. Estos se hayan a partir, por convención, a partir de la posición 16 relativamente al valor con el que recibimos el sp, en este caso, como descendimos 16 bytes, debemos ir a buscar el valor del quinto parámetro en la posición 32 y el sexto en la posición 36. Luego, el código se ejecuta normalmente, en base a los valores recibidos y posteriormente, antes de salir de la rutina, se restauran los valores que se habían guardado en el stack. Dejando el fp, ra y el gp con los valores con los que los recibimos, para que el programa que nos llamó pueda continuar normalmente su ejecución.

7 Conclusiones

*** HACER Completar conclusiones del código Assembly generado por el compilador vs. el código que nosotros generamos