

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 GX-emul provisto por la cátedra, como en la versión de Linux (Knoppix, Red-Hat, 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.

$N \times M$ $a_{1,1}$ $a_{1,2}$... $a_{1,M}$ $a_{2,1}$ $a_{2,2}$... $a_{2,M}$... $a_{N,1}$ $a_{N,2}$... $a_{N,M}$

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 índices de fila y columna respectivamente. El fin de línea es el carácter `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`.

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 estándar (`stdout`) en el mismo formato presentado anteriormente, hasta que llegue al final del archivo de entrada (EOF). Ante un error, el programa deberá informar la situación inmediatamente (por `stderr`) y detener su ejecución. Tener en cuenta que también se considera 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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <getopt.h>
4 #include <string.h>
5 #include "multiplicar.h"
6
7 #define MAX_LINE_LENGTH 512
8 #define MAX_DIMENSION_LENGTH 4
9 #define EXIT_OK 0
10 #define EXIT_ERROR 1
11
12 enum ACCION {
13     EMPTY,
14     HELP,
15     VERSION,
16     MULTIPLICAR,
17     ERROR
18 };
19
20 void multiplicarMatriz();
21 enum ACCION procesarArgumentos(int argc, char** argv);
22
23 int main(int argc, char **argv) {
24     enum ACCION comando = procesarArgumentos(argc, argv);
25     switch (comando) {
26         case HELP:
27             printf("Usage:\n"
28                 "\t\t -h\n"
29                 "\t\t -V\n"
30                 "\t\t < in_file > out_file\n"
31                 "Options:\n"
32                 "\t -V, --version\tPrint version and quit.\n"
33                 "\t -h, --help\tPrint this information.\n"
34                 "Examples:\n"
35                 "\t\t < in.txt > out.txt\n"
36                 "\t\t cat in.txt | tp0 > out.txt\n");
37             break;
38         case VERSION:
39             printf("tp0 v0.1\n");
40             break;
41         case MULTIPLICAR:
42             multiplicarMatriz();
43             break;
44         case ERROR:
45         default:
46             return EXIT_ERROR;
47     }
48     return EXIT_OK;
49 }
50
51
52 void imprimirMatriz(double* matriz, int filas, int columnas) {
53     printf("%dx%d ", filas, columnas);
54     int i;
55     for (i = 0; i < filas * columnas; i++) {
56         printf("%g ", matriz[i]);
57     }
58     printf("\n");
59 }
60
```

```

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
71         (char));
72     if (original == NULL) {
73         fprintf(stderr, " No hay mÃ¡s memoria.\n");
74         return NULL;
75     }
76     original[originalSize] = *toAppend;
77     return original;
78 }
79
80 void leerDimension(int* filas, int* columnas) {
81     int newChar;
82     char c;
83     char* buffer = (char *) malloc(sizeof(char) *
84         MAX_DIMENSION_LENGTH);
85     int i = 0, total = 0;
86     while((newChar = getchar()) != EOF) {
87         c = (char) newChar;
88         if (c == 'x') {
89             *filas = atoi(buffer);
90             memset(buffer, 0, strlen(buffer));
91             i = 0;
92             continue;
93         }
94         if(c == ' ') {
95             *columnas = atoi(buffer);
96             break;
97         }
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) * MAXLINELENGTH);
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 >= MAXLINELENGTH) {
173             char* oldBuffer = buffer;
174             buffer = append(buffer, total, &c);
175             if (buffer == NULL) {
176                 free(oldBuffer);
177                 fprintf(stderr, "%s\n", "No hay más memoria para guardar
la matriz.");
178             }
179             exit(EXIT_ERROR);
180         }

```



```

181     buffer[i] = c;
182     i++;
183 }
184 free(buffer);
185
186
187 if (elementos < filas * columnas) {
188     fprintf(stderr, "%s\n", "Dimension no compatible con datos de
189         matriz.");
190     exit(EXIT_ERROR);
191 }
192
193 if (newChar == EOF)
194     exit(EXIT_OK);
195 }
196
197 double* crearMatriz(int filas, int columnas) {
198     double* matriz = (double*) malloc(sizeof(double) * filas *
199         columnas);
200     return matriz;
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                 ");
219             free(matrizA);
220             free(matrizB);
221             exit(EXIT_ERROR);
222         }
223
224         double* matrizC = crearMatriz(filasA, columnasB);
225
226         multiplicarMatrices(filasA, matrizB, matrizC, matrizA,
227             columnasB, columnasA);
228
229         imprimirMatriz(matrizC, filasA, columnasB);
230
231         free(matrizA);
232         free(matrizB);
233         free(matrizC);
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
242        por argc y argv
243     * mas info: http://www.gnu.org/software/libc/manual/html\_node/Using-Getopt.html#Using-Getopt
244     * La cadena "hVbri:o:" indica que h, V no tienen argumentos.
245     */
246     int c;
247     while ((c = getopt(argc, argv, "hV")) != -1) {
248         switch (c) {
249             case 'h':
250                 comando = HELP;
251                 break;
252             case 'V':
253                 comando = VERSION;
254                 break;
255             default:
256                 comando = ERROR;
257                 break;
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: tamaño
53 #      stack frame, ra: return address
54 #
55 # bloque para c\ódigo PIC
56 # .set noreorder # apaga reordenamiento de
57 #      instrucciones
58 # .cload t9 # directiva usada para c\ódigo PIC
59 # .set reorder # enciende reordenamiento de
60 #      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
68                               "lw gp, 4(sp)" luego de cada jal.
69
70      addiu    t0, a0, 0      # lw t0, cantf-c-i
71      lw       t1, 32(sp)     # lw t1, cantc-c-j
72      lw       t2, 36(sp)     # lw t2, cantc-a-k
73
74
75      #no hago una etiqueta para i porque nunca volveremos a
76      #hacer i =0 para
77      #Las filas
78      li       s0, 0          # i = 0; initialize 1st for loop
79
80      loop1:
81      # Al contrario para las columnas de B se hace una corrida
82      # por cada fila
83      #Varias veces a este punto
84      li       s1, 0          # j = 0; restart 2nd for loop
85
86      loop2:
87      # Por cada valor de C deberemos iterar sobre todos los
88      # valores de K
89      #
90      li       s2, 0          # k = 0; restart 3rd for loop
91
92      #Operaciones con C
93      mul      t3, s0, t1      # en t3 la cantidad de columnas de
94      #C
95      addu     t3, t3, s1      # en t3 sumo el valor de j-> la
96      #posicion en el array
97      sll      t3, t3, 3       # el offset en bytes en el array
98      addu     t3, a2, t3      # cargo en t3 la direccion de C{i}{
99      #j}
100     sw       zero, 0(t3)     # inicializo C{i}{j} con cero
101     l.d      $f4, 0(t3)      # inicializo en t4 en cero que serÃ¡ el
102     #valor de C{i}
103
104     #Operaciones con A
105     loop3:
106     mul      t7, s0, t2      # en t7 la cantidad de columnas de
107     #A "K"
108     addu     t7, t7, s2      # ahora le sumamos K y obtenemos la
109     #posicion en el arra
110     sll      t7, t7, 3       # multiplcamos por 4 y obtenemos el
111     #offset de A[i][k]
112     addu     t7, a3, t7      # obtenemos la direccion de A[i][k]
113     l.d      $f8, 0(t7)     # guardamos en t8 el valor de A[i][
114     #k]
115
116     # Ahora con B
117     mul      t5, s2, t1      # t5 = k * 4 (size of row of b)
118     addu     t5, t5, s1      # t5 = k * size(row) + j
119     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	jlr	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 uno, 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.

7 Conclusiones

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