U.B.A. FACULTAD DE INGENIERÍA

Departamento de Electrónica
Organización de Computadoras 66-20
Ing. Informática

Trabajo práctico Nº01
Assembly Mips

Apellido y Nombre:	*Padrón:*
Cabrera, Jorge	93310
Capolupo, Mauro	90283
Serra, Diego Adrián	92354

Fecha de Entrega : 03/05/2016
Fecha de Aprobación :
Calificación :
Firma de Aprobación :

1 Diseño e implementación del programa

El programa permite obtener el producto de dos matrices cuadradas a partir de un stream de datos. En esta oportunidad tanto la función encargada de realizar la multiplicación como la que la imprime se implementaron en MIPS.

2 Comando(s) para compilar el programa en NetBSD

gcc -Wall -g -o tp1 tp1.c tp1.S
cat file_name.txt ‖./*tp*1

3 Pruebas

3.1 Prueba 1

El archivo prueba.txt es
2 1 2 3 4 5 6 7 8
3 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
respresenta las operaciones

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ l & \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \times \begin{pmatrix} 10 & 11 & 12 \\ 13 & 14 & 15 \\ 16 & 17 & 18 \end{pmatrix} = \begin{pmatrix} 84 & 90 & 96 \\ 201 & 216 & 231 \\ 318 & 342 & 366 \end{pmatrix}$$

por salida estandar se vera
2 19 22 43 50
3 84 90 96 201 216 231 318 342 366

3.2 Prueba 2

El archivo prueba.txt es
2 1 2 3 4 5 6 7
respresenta la operacion

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & - \end{pmatrix}$$

por salida estandar se vera
Cantidad incorrecta de parametros para matriz de dimension 2.

3.3 Prueba 3

El archivo de prueba.txt es
2 a 2 3 4 5 6 7 8

respresenta las operaciones
$\begin{pmatrix} a & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$

por salida estandar se vera
Lectura de caracter no valido.
Cantidad incorrecta de parametros para matriz de dimension 2.

4 Codigo

```
1
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  #define MAX_LONG 20
7
8  extern int print_string(int fp, char* s);
9
10 int cantProcesos = 1;
11
12 typedef struct matrix {
13    size_t rows;
14    size_t cols;
15    float *array;
16 } matrix_t;
17
18 extern void matrix_multiply(matrix_t* m1, matrix_t* m2, matrix_t*↩
       m3);
19
20 // Constructor de matrix_t
21 matrix_t* create_matrix(size_t rows, size_t cols) {
22    matrix_t* matrix = malloc(sizeof(matrix_t));
23    if(matrix == NULL){
24       return NULL;
25    }
26    matrix->rows = rows;
27    matrix->cols = cols;
28    matrix->array = (float*)calloc(rows * cols, sizeof(float));
29    if (matrix->array == NULL){
30       return NULL;
31    }
32    return matrix;
33 }
34
35 // Destructor de matrix_t
36 void destroy_matrix(matrix_t* m) {
37    free(m->array);
38    m->array = NULL;
39    free(m);
40    m=NULL;
41 }
```

```
42
43 // Imprime matrix_t sobre el file pointer fp en el formato ↩
       solicitado
44 // por el enunciado
45 int print_matrix(FILE* fp, matrix_t* m) {
46    int i = 0;
47    char strNum [MAX_LONG];
48    char* spc = " \0";
49
50    int intCol = (int) (m->cols);
51    sprintf(strNum, "%d ", intCol);
52    print_string(fileno(fp),strNum);
53    while (i < (m->cols) * (m->cols)) {
54       float flNum = m->array[i];
55       sprintf(strNum, "%f", flNum);
56       print_string(fileno(fp),strNum);
57       print_string(fileno(fp),spc);
58       i++;
59    }
60    fprintf(fp, "\n");
61    return 0;
62 }
63
64 void show_help(){
65    printf("Usage:\n");
66    printf("\t tp0 -h \n");
67    printf("\t tp0 -V \n");
68    printf("\t tp0 < in_file > out_file \n");
69    printf("Options:\n");
70    printf("\t -V, --version \t Print version and quit. \n");
71    printf("\t -h, --help \t Print this information and quit. \n")↩
          ;
72    printf("Examples:\n");
73    printf("\t tp0 < in.txt > out.txt \n");
74    printf("\t cat in.txt | tp0 > out.txt \n");
75 }
76
77 void show_version(){
78    printf("version xx \n");
79 }
80
81
82 int leerTamanio(int* cont, int* err) {
83    int n=0;
84    int resp;
85    resp = scanf("%d", &n);
86
87    if (resp == EOF){
88       *cont=0;
89       return 0;
90    }else if (n <= 0){
91       *err = 1;
92       fprintf(stderr, "no se pudo obtener ñtamaio de matrix. Fila↩
             : %d\n",cantProcesos);
93       return 0;
94    };
95    return n;
```

```
}

char* readString(int* errRead) {
    char c;
    char *string;
    int continuar = 1;
    c = getchar();
    while (c == 32){ // blank
      c = getchar();
    }
    if ((c>=48 && c<=57) || (c>=45 && c<=46) || (c==43) || (c==101)){ //numeros | - | . | + | e |
      char aux[1];
      aux[0] = c;
      aux[1] = '\0';
      string = (char*) malloc((strlen(aux)+1)*sizeof(char));
      strcpy(string, aux);
    }else if(c == 10){ // newline
      return NULL;
    }else{
      *errRead = 1;
      fprintf(stderr,"Lectura de caracter no valido. Linea: %d\n",cantProcesos);
      return NULL;
    };
    do{
        c = getchar();
        if ((c>=48 && c<=57) || (c>=45 && c<=46) || (c==43) || (c==101)){
         char aux[1];
         aux[0] = c;
         aux[1] = '\0';
            string = (char*)realloc(string, (strlen(string)+strlen(aux)+1)*sizeof(char));
            strcat(string, aux);
        }else if ((c == 32) || (c == 10)){
         continuar = 0;
        }else{
         *errRead = 1;
         fprintf(stderr,"Lectura de caracter no valido. Linea: %d\n",cantProcesos);
         free(string);
         return NULL;
        };
    }while(continuar);
    return string;
}


void fillMatrix(int tam, matrix_t *matrix, int* err) {
         char *token;
         int i = 0;
         int validNumber = 0;
         int errFill = 0;
            token = readString(&errFill);
            while ((token != NULL) && (i < (tam*tam)) && !errFill){
```

```
147                 float d;
148                 validNumber = 0;
149                 validNumber = sscanf(token, "%g", &d);
150                 if (validNumber > 0 ) {
151                    matrix->array[i]=d;
152                    i++;
153                 }else{
154                    fprintf(stderr,"Numero con formato incorrecto. ↩
                           Linea: %d\n", cantProcesos);
155                    errFill = 1;
156                 }
157                 free(token);
158                 if (i != (tam*tam)){
159                    token = readString(&errFill);
160                 }
161              }
162              if ((token == NULL) && (i < (tam*tam))) {
163                 fprintf(stderr,"Cantidad incorrecta de parametros ↩
                       para matriz de dimension %d. Linea: %d\n", tam↩
                       , cantProcesos);
164                 errFill = 1;
165              }
166              if (errFill){ *err = errFill; };
167 }
168
169 int main(int argc, char **argv) {
170
171    if (argc > 1) {
172       if ((strcmp(argv[1], "-h") == 0) || (strcmp(argv[1], "--↩
              help") == 0)) {
173          show_help();
174          return 0;
175       } else if ((strcmp(argv[1], "-V") == 0)
176             || (strcmp(argv[1], "--version") == 0)) {
177          show_version();
178          return 0;
179       }else{
180          printf("Paramatro incorrecto. Ingrese -h para ayuda.\n")↩
                 ;
181          return 0;
182       };
183    }
184
185    matrix_t* matrix_a=NULL;
186    matrix_t* matrix_b=NULL;
187    matrix_t* matrix_c=NULL;
188    int continuar = 1;
189    int err = 0;
190    size_t n = leerTamanio(&continuar,&err);
191    while(continuar && !err){
192       matrix_a = create_matrix(n,n);
193       matrix_b = create_matrix(n,n);
194       fillMatrix(n,matrix_a, &err);
195       if (!err){
196          fillMatrix(n,matrix_b, &err);
197       }
198       if (!err){
```

```
199             matrix_c = create_matrix(n,n);
200             matrix_multiply(matrix_a, matrix_b, matrix_c);
201             print_matrix(stdout, matrix_c);
202             if (matrix_c != NULL) { destroy_matrix(matrix_c); };
203         }
204         if (matrix_a != NULL) { destroy_matrix(matrix_a); };
205         if (matrix_b != NULL) { destroy_matrix(matrix_b); };
206 
207         cantProcesos++;
208         if (!err){
209             n = leerTamanio(&continuar, &err);
210         }
211     }
212     return EXIT_SUCCESS;
213 }
```

```
# óFuncin matrix_multiply

/*matrix_t* matrix_multiply(matrix_t* m1, matrix_t* m2) {
   int m1_index = 0;
   int m2_index = 0;
   int m2_aux = 0;
   int index = 0;
   matrix_t* result = create_matrix(m1->rows, m1->cols);

   for (index = 0; index < m1->rows * m1->cols;) {
      m1_index = (index / m1->cols) * m1->rows;
      result->array[index]=0;
      for (m2_aux = 0; m2_aux < m2->rows;) {
         result->array[index] += m1->array[m1_index] * m2->array[↩
             m2_index];
         m2_aux++;
         m1_index++;
         m2_index += m2->rows;
      }
      index++;
      m2_index = index % m2->cols;
      m2_aux = 0;
   }
   return result;
}*/

#include <mips/regdef.h>

.text
.abicalls
.align 2
.globl matrix_multiply
.ent matrix_multiply

matrix_multiply:

.frame $fp,40,ra

#bloque de codigo para PIC
.set noreorder
.cpload t9
.set reorder

#creo el stack frame
subu sp, sp, 40

.cprestore 24              #sw gp, sp(24) / lw gp, 24(sp).

#salvado de callee-saved regs en SRA
sw $fp, 28(sp)
sw ra, 32(sp)

move $fp, sp               #uso fp en vez de sp.

#salvo los args fuera del stack frame.
sw a0, 40($fp)
sw a1, 44($fp)
```

```
57 sw a2, 48($fp)
58
59 sw zero, 0($fp)            #En 0(sp) tengo int m1_index = 0
60 sw zero, 4($fp)               #En 4(sp) tengo int m2_index = 0
61 sw zero, 8($fp)            #En 8(sp) tengo int m2_aux = 0
62 sw zero, 12($fp)           #En 12(sp) tengo int index = 0
63
64 first_for:
65 #for (index = 0; index < m1->rows * m1->cols;)
66 lw t0, 0($fp)              #m1_index
67 lw t1, 4($fp)              #m2_index
68 lw t2, 8($fp)              #m2_aux
69 lw t3, 12($fp)                #index
70
71 lw t7, 0(a0)
72 mul t4, t7, t7            #rows * cols
73
74 bge t3,t4, exit_first_for        #si m1_index es mayor que (rows ↩
       * cols) finalizo el loop
75
76 #m1_index = (index / m1->cols) * m1->rows;
77 divu t5, t3, t7            #m1_index = (index / m1->cols) * m1->↩
       rows;
78 mulo t0, t5, t7
79 sw t0, 0($fp)              #guardo m1_index
80
81 #result->array[index]=0;
82 lw t7, 8(a2)
83 sll t0, t3, 2
84 add t5, t7, t0
85
86 sw zero, 8($fp)            #En 8(sp) tengo int m2_aux = 0
87
88 li.s $f4, 0
89 b second_for
90
91 second_for:
92 #for (m2_aux = 0; m2_aux < m2->rows;) {
93 lw a1, 44($fp)                #recupero m2
94 lw t7, 0(a1)
95 lw t2, 8($fp)
96 bge t2, t7, exit_second_for
97 b multiply
98
99 multiply:
100 #result->array[index] += m1->array[m1_index] * m2->array[m2_index↩
       ];
101 lw t0, 0($fp)              #m1_index
102 lw t1, 4($fp)              #m2_index
103 lw t2, 8($fp)              #m2_aux
104 lw t3, 12($fp)                #index
105
106 lw t7, 8(a0)
107 sll t6, t0, 2
108 add t7, t7, t6                #m1->array[m1_index]
109 l.s $f12, 0(t7)
110
```

```
111 lw t7, 8(a1)
112 sll t6, t1, 2
113 add t7, t7, t6                #m2->array[m2_index]
114 l.s $f14, 0(t7)
115
116 mul.s $f2, $f12, $f14           #m1->array[m1_index] * m2->array↩
    [m2_index]
117 add.s $f4, $f2, $f4
118
119 addi t2, t2, 1                 #m2_aux++
120 sw t2, 8($fp)
121 addi t0, t0, 1                 #m1_index++
122 sw t0, 0($fp)
123
124 lw t0, 44($fp)
125 lw t7, 0(t0)                #m2->rows
126 add t1, t1, t7                #m2_index += m2->rows;
127 sw t1, 4($fp)
128
129 b second_for
130
131 exit_second_for:
132 s.s $f4, 0(t5)
133 lw t3, 12($fp)                 #index
134 addi t3, 1
135 sw t3, 12($fp)
136
137 #m2_index = index % m2->cols;
138 lw t1, 44($fp)
139 addi t1, t1, 4                 #m2->cols;
140 lw t7,0(t1)
141 div t0, t3, t7
142 mfhi t0
143 sw t0, 4($fp)              #piso el valor de m2_index
144 #sw zero, 8($fp)           #m2_aux=0
145
146 b first_for
147
148 exit_first_for:
149 #guardo el valor de retorno
150 lw t0, 48($fp)                 #result
151 add v0, zero, t0
152 #destruyo el stack frame
153 lw gp, 24(sp)
154 lw $fp, 28(sp)
155 lw ra, 32(sp)
156 addu sp, sp, 40
157 j ra
158 .end matrix_multiply
```

```
#Funcion print_string

#include <mips/regdef.h>
#include <sys/syscall.h>

   .text                       # segmento de texto del programa

   .abicalls
   .align 2                    # alineacion 2^2

   .globl   print_string
   .ent  print_string
print_string:
   # debugging info: descripcion del stack frame
   .frame   $fp, 40, ra             # $fp: registro usado como ↩
       frame pointer
                               # 40: ñtamaodel stack frame
                               # ra: registro que almacena el ↩
                                   return address
   # bloque para codigo PIC
   .set  noreorder                 # apaga reordenamiento de ↩
       instrucciones
   .cpload t9                      # directiva usada para codigo ↩
       PIC
   .set  reorder                   # enciende reordenamiento de ↩
       instrucciones

   # creo stack frame
   subu  sp, sp, 40                # 4 (SRA) + 2 (LTA) + 4 (ABA)

   # directiva para codigo PIC
   .cprestore 24                   # inserta aqui "sw gp, 24(sp)",
                               # mas "lw gp, 24(sp)" luego de cada↩
                                   jal.
                               # salvado de callee-saved regs en ↩
                                   SRA
   sw $fp, 28(sp)
   sw ra, 32(sp)

   # de aqui al fin de la funcion uso $fp en lugar de sp.
   move  $fp, sp

   # salvo 1er arg (siempre)
   sw a0, 40($fp)                  #file descriptor
   sw a1, 44($fp)                  #char* a string

bpsc1:
   move a0, a1                     # traigo char* a a0 para entrada↩
        de mystrlen
   #obtengo largo del string mystrlen
   la t9, mystrlen                 # mystrlen(name[r])
   jal   ra, t9
   #guardo resultado e LRA
   sw    v0, 16($fp)

bpsc2:
   # cargo argumentos y nro de syscall
```

```
50    lw      a2, 16($fp)           # t2 -> cantidad de caracteres a↩
          imprimir
51    lw     a1, 44($fp)
52    lw    a0, 40($fp)
53    li v0, SYS_write
54    syscall
55
56    # return;
57    # restauro callee-saved regs
58    lw gp, 24(sp)
59    lw $fp, 28(sp)
60    lw ra, 32(sp)
61    # destruyo stack frame
62    addu  sp, sp, 40
63    # vuelvo a funcion llamante
64    jr ra
65
66    .end  print_string
67    .size print_string,.-print_string
68
69    .ent  mystrlen
70
71 mystrlen:
72    .frame   $fp, 16, ra
73    .set  noreorder
74    .cpload t9
75    .set  reorder
76
77    # creo stack frame
78    subu  sp, sp, 16              # 2 (SRA) + 2 (LTA)
79    .cprestore 8                  # sw gp, 8(sp)
80    sw $fp, 12(sp)
81    move  $fp, sp
82
83    # salvo 1er arg (siempre)
84    sw a0, 16($fp)                ## redundante
85
86    # for (i=0; s[i] != 0; i++)
87    move  t0, zero             # i=0: t0, fp+0
88    sw t0, 0($fp)                 # i: t0
89 _for_loop:
90    # condicion de corte: s[i] != 0
91    lw a0, 16($fp)                ## redundante
92    lw t0, 0($fp)                 ## redundante
93    addu  t1, a0, t0              # s[i]: t1
94    lb t1, 0(t1)                  ## lb, NO lw!
95    beq   t1, zero, _end_for
96
97    lw t0, 0($fp)  # i++          ## redundante
98    addu  t0, t0, 1
99    sw t0, 0($fp)                 ## redundante
100   j  _for_loop
101
102 _end_for:
103   lw v0, 0($fp)                 ## podria ser un move v0, t0
104   lw gp, 8(sp)
105   lw $fp, 12(sp)
```

```
106     addu  sp, sp, 16
107     jr ra
108
109     .end  mystrlen
110     .size mystrlen,. - mystrlen
```

5 Conclusiones

Se han puesto en práctica conocimientos sobre la arquitectura MIPS, herramientas para su desarrollo - gdb -,como así también el manejo de registros y stack de momoria. En este caso no hemos observado una mejora al traducir las funciones mencionadas a MIPS, por el contrario los tiempos se han visto incrementado. Posiblemente a la ineficiente forma en la que los datos se piden en el programa traducido.

6 Enunciado

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 1: assembly MIPS
1^{er} cuatrimestre de 2016

`$Date: 2016/04/12 01:45:28 $`

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto 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 descripto 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

Se reutilizará mayormente el programa del TP anterior, en el cual se reescribirá completamente la función `matrix_multiply` en Assembly MIPS32.

Además, también deberá escribirse completamente en Assembly MIPS32 la función `print_string`, que responda al siguiente prototipo:

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

La función `print_matrix`, escrita en C, deberá adaptarse para utilizar esta función para imprimir la matriz, en lugar de la función C que se utilizó en el TP anterior (probablemente `fprintf`).

Notar que el primer argumento de `print_string` (`fd`) consiste en un file descriptor (`int`) y no un file pointer (`FILE*`).

Ante un error, la función deberá devolver -1. Caso contrario, devolverá el número de bytes escritos al archivo especificado.

El objetivo de este requisito es acceder al syscall `SYS_write` (ver [2]).

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 y Assembly MIPS32 según corresponda.
- Este enunciado.

7. Fechas

Fecha de vencimiento: martes 3/5/2016.

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/`).

[2] WRITE(2) NetBSD System Calls Manual.
(`http://netbsd.gw.com/cgi-bin/man-cgi?pwrite+2+NetBSD-current`)