
UBA FACULTAD DE INGENIERÍA

66.20 Organización de Computadoras

Trabajo Práctico 1

2^{do} Cuatrimestre 2017

Integrantes:

Rodriguez Longhi, Federico	93336
federico.rlonghi@gmail.com	
Deciancio, Nicolás Andrés	92150
nicodec.89@hotmail.com	

Índice

1. Introducción	2
2. Documentación	2
2.1. Diagrama de Proceso	3
3. Compilación	3
4. Pruebas	3
4.1. Corridas de Prueba	4
5. Análisis	4
6. Conclusión	5
A. Stacks de Funciones	6
A.1. Stack palindromo	6
A.2. Stack testpal	7
B. Código en C	7
C. Función Palíndromo en MIPS	10
D. Función testpal	15
E. Función testpal en C	16
F. Enunciado	17

1. Introducción

Este trabajo práctico, consistió principalmente en la codificación de la función palíndromo en código mips, implementada previamente en el tp0 en lenguaje C. Además, se reestructuro el método de input/output del programa para poder limitar la cantidad de acceso a escritura y lectura. Esto se logró con la utilización de buffers, tanto de escritura como de lectura. Los cuales se leen/escriben bien cuando se llenan o cuando se llega al final de la entrada y no hay que analizar más palabras.

Con esto tomamos como objetivo también poder hacer un análisis del costo de las llamadas a sistema. La idea es ver cuanto pierde un programa en performance al realizar estas llamadas.

Todo esto, nos obligó a interiorizarnos mucho más con el conjunto de instrucciones assembly para mips 32, y el uso de syscalls. Además, vimos de forma práctica, como llamar a funciones de archivos .s desde código C. El uso de un debugger fue de gran importancia para todo el desarrollo del programa en mips. A continuación, se ve la documentación del programa, forma de compilarlo, casos de prueba armados, análisis del problema y su solución, y el código fuente del programa.

2. Documentación

El uso del programa se compone de las siguientes opciones que le son pasadas por parámetro:

- `-h` o `--help`: muestra la ayuda.
- `-V` o `--version`: muestra la versión.
- `-i` o `--input`: recibe como parámetro un archivo de texto como entrada. En caso de que no usar esta opción, se toma como entrada la entrada estándar.
- `-o` o `--output`: recibe como parámetro un archivo de texto como salida. En caso de que no usar esta opción, se toma como salida la salida estándar.
- `-I` o `--ibuf-bytes`: recibe como parámetro el tamaño en bytes del buffer de entrada.
- `-O` o `--iobuf-bytes`: recibe como parámetro el tamaño en bytes del buffer de salida.

2.1. Diagrama de Proceso

A continuación se muestra un diagrama del proceso que recorre el programa, haciendo mención de los buffers que se utilizan y las acciones de comunicación de datos entre ellos.

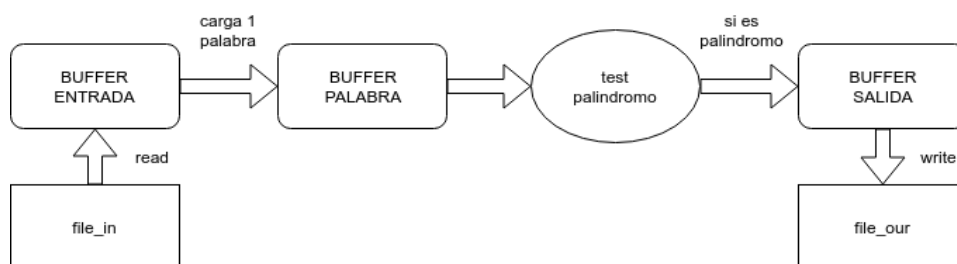


Figura 1: Diagrama del proceso de comunicación entre buffers

3. Compilación

Dentro del directorio raíz se encuentra un Makefile. Ejecutando `make` se compilara el programa y se generará el ejecutable `tp1`.

4. Pruebas

Para las pruebas se proveen de dos scripts que las ejecutan. Se utilizaron las mismas pruebas que para el `tp0`, solo que fueron corridas varias veces variando los parametros `-O` y `-I` del programa.

El primer script `test.sh` ejecuta los ejemplos del enunciado.

El segundo script `test_p.sh` ejecuta las pruebas propias. Este archivo esta diseñado para poder agregar pruebas de forma sencilla, simplemente se debe agregar una linea en el sector de pruebas de la siguiente manera:

```
make_test <nombre><entrada de texto><salida esperada>
```

Este script crea los archivos correspondientes en la carpeta `tests` (dentro del directorio sobre el cual se ejecuta). Los archivos creados son de la forma:

- `test-<nombre del test>_in`: archivo de entrada
- `test-<nombre del test>_out`: archivo de salida generado por el programa
- `test-<nombre del test>_expected`: archivo de salida esperado

4.1. Corridas de Prueba

A continuación se muestran las corridas de prueba generadas por el script:

```
1
2  Compiling Source
3  Compilation Success
4  Starting Tests
5
6  Test: one_letter_a
7  Test passed
8
9  Test: empty_file
10 Test passed
11
12 Test: no_palindroms
13 Test passed
14
15 Test: todos_palindromos
16 Test passed
17
18 Test: varias_lineas
19 Test passed
20
21 Test: all_letters
22 Test passed
23
24 Test: case_sensitive
25 Test passed
26
27 Test: numbers_and_letters
28 Test passed
29
30 Test: text_with_dash
31 Test passed
32
33 -----
34 All 9 tests passed!!!
35 -----
```

5. Análisis

Hemos utilizado este programa para analizar los tiempos de las llamadas al kernel. Las syscalls son llamadas al sistema, lo que requiere un cambio de entorno (de modo usuario a modo kernel), y esto consume mayor tiempo que si no se hiciese un cambio de entorno.

Como se provee un buffer de entrada y de salida al modificar el tamaño de estos, estamos disminuyendo o aumentando los syscalls read y write, es decir, cuanto mayor el tamaño del buffer menor cantidad de syscalls y

viceversa.

Lo que esperamos obtener como resultado es que al tener un buffer pequeño (por ejemplo de 1 byte) la cantidad de syscalls (la cantidad de syscalls read va a ser la cantidad de letras(bytes) que tenga el archivo) sea mucha, y por lo tanto obtendremos un tiempo de ejecución mayor que el tiempo de ejecución del mismo programa pero con un buffer mayor (por ejemplo de 1000 bytes, si el archivo es menor o igual al tamaño del buffer haremos un solo read). El costo de tener un buffer de mayor tamaño es el espacio utilizado por el programa. Esta claro que cuanto mayor sea el buffer mayor sera la memoria requerida por el programa.

Para realizar este análisis utilizamos un archivo de texto como entrada de un tamaño de N bytes. Luego medimos los tiempos de ejecución del programa. A continuación se muestra una tabla con los resultados:

Tiempos de ejecución

Tamaño Buffer Entrada	Tamaño Buffer Salida	Tiempo Ejecucion
1	1	-
10	10	-
100	100	-
1000	1000	-
10	1	-
100	1	-
1000	1	-
1	10	-
1	100	-
1	1000	-

6. Conclusión

De este trabajo práctico nos llevamos principalmente que es muy importante la experiencia y conocimiento de las instrucciones de Assembly para mips 32 y como usarlas eficientemente. También, pudimos ver por nuestros propios ojos como puede llegar a afectar al tiempo de ejecución de un programa la cantidad de syscalls que se realizan. Acotando las syscalls de read y de write con la utilización de un buffer de escritura y otro de lectura, vimos un speed up global del programa. Como dijimos previamente, cada syscall implica un cambio de entorno de modo usuario a modo kernel. Y si se repite consistentemente este 'switch', el tiempo de ejecución del programa se ve fuertemente afectado. Por esto, siempre que se pueda acotar la cantidad de llamadas al sistema de un programa, se debería tratar de optimizarlas. En este caso particular, se acotó esa cantidad usando los buffers. Fue muy importante también el uso de un debugger para poder arreglar problemas durante la codificación.

Apéndice

A. Stacks de Funciones

A continuación se muestran como fueron conformados los stacks de las funciones programadas en MIPS según lo dispuesto por la ABI.

A.1. Stack palindromo

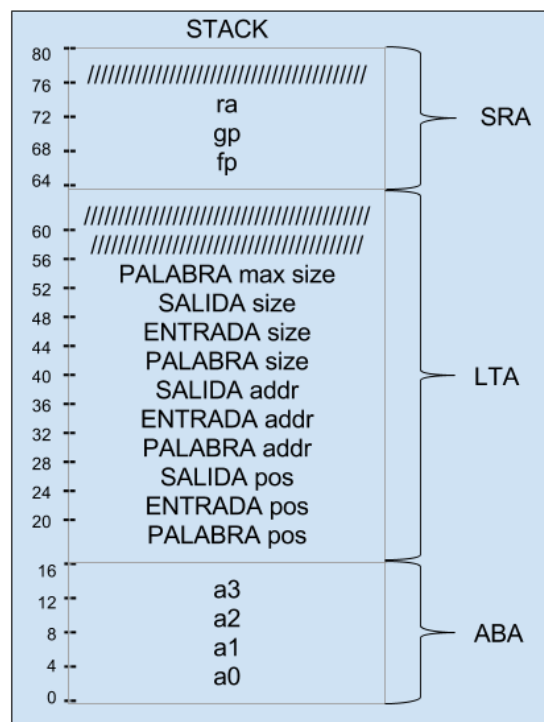


Figura 2: Stack de la función `int palindromo(int, size_t, int, size_t)`

A.2. Stack testpal

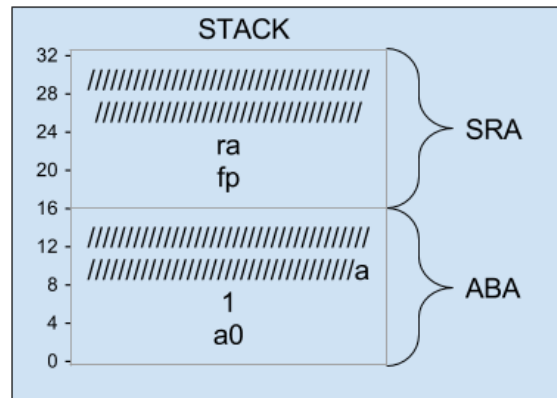


Figura 3: Stack de la función `int testpal(char *, size_t)`

B. Código en C

El siguiente código corresponde al cuerpo principal del programa. Se encarga de procesar los parámetros, abrir los archivos, llamar a la función palíndromo (escrita en MIPS) y luego cerrar los archivos correspondientes.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #include <unistd.h>
5  #include <getopt.h>
6  #include <errno.h>
7  #include <string.h>
8
9  extern size_t mystrlen(const char*);
10 extern int palindrome(int, size_t, int, size_t);
11
12 /* imprimir el uso de tp0 */
13 void print_usage() {
14     printf("Usage: tp0 -i [input_file] -o [output_file]\n");
15 }
16
17 /* imprimir la pagina de ayuda */
18 void print_help() {
19     printf("\tUsage:\n"
20           "\t\ttp0 -h\n"
21           "\t\ttp0 -V\n"
22           "\t\ttp0 [options]\n"
23           "\tOptions:\n"
24           "\t\t-V, --version\tPrint version and quit.\n"
25           "\t\t-h, --help\tPrint this information.\n"
26           "\t\t-i, --input\tLocation of the input file.\n"
27           "\t\t-o, --output\tLocation of the output file.\n")

```

```
28         "\t\t-I, --ibuf-bytes\tByte count of the input buffer.\n"
29         "\t\t-O, --obuf-bytes\tByte count of the output buffer\n"
30         "\tExamples:\n"
31         "\t\tftp0 -i ~/input -o ~/output\n");
32     }
33
34     /* imprimir la version del programa */
35     void print_version(){
36         printf("tp1 1.0\n");
37     }
38
39     int main(int argc, char *argv[]) {
40
41         int opt= 0;
42
43         int help = -1;
44         int version = -1;
45         int input = -1;
46         int output = -1;
47         int ibuf = 1;
48         int obuf = 1;
49
50         char *input_filename = NULL;
51         char *output_filename = NULL;
52
53         // especificacion de las opciones
54         static struct option long_options[] = {
55             {"help",          no_argument,          0,  'h' },
56             {"version",       no_argument,          0,  'V' },
57             {"input",         required_argument, 0,  'i' },
58             {"output",        required_argument, 0,  'o' },
59             {"ibuf-bytes",    required_argument, 0,  'I' },
60             {"obuf-bytes",    required_argument, 0,  'O' },
61             {0,               0,                   0,  0   },
62         };
63
64         int long_index = 0;
65
66         // evaluacion de los parametros enviados al programa
67         while ((opt = getopt_long(argc, argv, "hVui:o:I:O:",
68             long_options, &long_index )) != -1) {
69             switch (opt) {
70                 case 'h' :
71                     help = 0;
72                     break;
73                 case 'V' :
74                     version = 0;
75                     break;
76                 case 'i' :
77                     input = 0;
78                     input_filename = optarg;
79                     break;
```

```
80         case 'o' :
81             output = 0;
82             output_filename = optarg;
83             break;
84         case 'I':
85             ibuf = atoi(optarg);
86             break;
87         case 'O':
88             obuf = atoi(optarg);
89             break;
90         case '?':
91             exit(1);
92         default:
93             print_usage();
94             exit(EXIT_FAILURE);
95     }
96 }
97
98 // procesamiento de los parametros
99 if (help == 0) {
100     print_help();
101     exit(0);
102 }
103 else if (version == 0) {
104     print_version();
105     exit(0);
106 }
107
108 /* Si no se recibe parametro de ayuda o version se ejecuta
109    el programa */
110
111 // estableciendo los archivos de entrada y salida
112 FILE *input_file = stdin;
113 FILE *output_file = stdout;
114
115 if (input == 0){
116     input_file = fopen(input_filename,"r");
117     if (input_file == NULL) {
118         printf ("can't open input file, errno = %d\n",
119                 errno);
120         return 1;
121     }
122 }
123 if (output == 0){
124     output_file = fopen(output_filename,"w");
125     if (output_file == NULL) {
126         printf ("Can't open output file, errno = %d\n",
127                 errno);
128         return 1;
129     }
130 }
131
132 /* ejecucion del programa */
133 char *msg = "Estoy probando mips!!!\n";
```

```
131 write(1, msg, mystrlen(msg));
132
133 int file_in = fileno(input_file);
134 int file_out = fileno(output_file);
135
136 printf("file_in: %i\nibuf: %zu\nfile_out: %i\nobuf: %zu\n",
137        file_in, ibuf, file_out, obuf);
138
139
140 int a = palindrome(file_in, ibuf, file_out, obuf);
141
142
143 printf("a: %i\n", a);
144
145
146 if (input == 0){
147     fclose(input_file);
148 }
149 if (output == 0){
150     fclose(output_file);
151 }
152
153 printf("sali bien\n");
154
155 return 0;
156 }
```

C. Función Palíndromo en MIPS

Esta función escrita en MIPS es la que corresponde a la pedida en el enunciado, en ella se llama a mymalloc y a testpal (descrita en la próxima sección).

```
1 # palindrome.S - ver tp1.c.
2 #
3 # $Date: 2017/09/24 17:12:06 $
4
5 #include <sys/syscall.h>
6 #include <mips/regdef.h>
7
8 .text
9 .align 2
10
11 .globl palindrome
12 .ent palindrome
13
14 palindrome:
15     .frame $fp, 76, ra
16     .set noreorder
17     .cpload t9
18     .set reorder
19
```

```
20     subu    sp, sp, 76
21
22     .cprestore 32
23
24     sw     ra, 72(sp)
25     sw     $fp, 68(sp)
26     sw     gp, 64(sp)
27
28     sw     a0, 0(sp)      #file_in
29     sw     a1, 4(sp)      #ibuf
30     sw     a2, 8(sp)      #file_out
31     sw     a3, 12(sp)     #obuf
32
33     move   $fp, sp
34
35     #####
36     #carga los buffers necesarios para operar
37     #buffer_entrada -> empty
38     #buffer_word -> empty
39     #bufer_salida -> empty
40
41     li     a0, 100        # cargo el parametro para mymalloc
42     sw     a0, 44(sp)     # guardo en el stack el tamaño maximo
                           del buffer palabra
43     jal    mymalloc      # llamo a la funcion
44     sw     v0, 20(sp)     # guardo la direccion de memoria
                           reservada para el buffer de PALABRA
45     add    t0, zero, zero
46     sw     t0, 16(sp)     # guardo el tamaño del buffer (
                           incialmente vacio)
47
48     lw     a0, 4(sp)     # cargo el parametro para mymalloc
49     jal    mymalloc      # llamo a la funcion
50     sw     v0, 24(sp)     # guardo la direccion de memoria
                           reservada para el buffer de ENTRADA
51     add    t0, zero, zero
52     sw     t0, 36(sp)     # guardo el tamaño del buffer (
                           incialmente vacio)
53
54     lw     a0, 12(sp)    # cargo el parametro para mymalloc
55     jal    mymalloc      # llamo a la funcion
56     sw     v0, 28(sp)    # guardo la direccion de memoria
                           reservada para el buffer de SALIDA
57     add    t0, zero, zero
58     sw     t0, 40(sp)     # guardo el tamaño del buffer (
                           incialmente vacio)
59
60     #####
61
62 leerArchivo:
63     #read file_in -> buffer_entrada (I Bytes)
64
65     li     v0, SYS_read   # ver dentro de <sys/syscall.h>.
66     lw     a0, 0(sp)      # a0: file descriptor number.
```

```
67    lw  a1, 24(sp)      # a1: data pointer.
68    lw  a2, 4(sp)       # a2: available space.
69    syscall
70
71    bne a3, zero, read_error    #verifico error de lectura
72
73    beqz v0, palindromoReturn # si read no lee nada finalizo el
    programa
74
75    add t0, zero, zero # en t0 cargo el indice inicial del
    buffer
76    sw  t0, 48(sp)
77
78    sw  v0, 36(sp)
79
80
81 leoByte:
82    lw  t9, 36(sp)
83    beq t9, zero, leerArchivo # si buffer_entrada.size == 0 =>
    leerArchivo
84
85    # c = leo un byte <- buffer_entrada
86    lw  t0, 48(sp)
87    lw  t1, 24(sp) # cargo la direccion del buffer de entrada
    en t1
88    add t1, t1, t0 # a la direccion le sumo el indice
89    lb  t2, 0(t1) # c = t2, guardo en t2 el byte
90
91    #buffer_entrada.size --
92    sub  t9, t9, 1
93    sw  t9, 36(sp)
94
95    # si c no es alfanumerico tengo que testear
96    # si lo que hay en el buffer de word es palindromo
97    slti t3, t2, 48
98    sne  t4, t2, 45
99    add  t3, t4, t3
100    beq  t3, 2, test_palindromo
101    sgt  t3, t2, 57
102    slti t4, t2, 65
103    add  t3, t3, t4
104    beq  t3, 2, test_palindromo
105    sgt  t3, t2, 90
106    slti t4, t2, 97
107    add  t3, t3, t4
108    sne  t4, t2, 95
109    add  t3, t3, t4
110    beq  t3, 3, test_palindromo
111    slti t3, t2, 123
112    beqz t3, test_palindromo
113
114    # sino: pongo a c en el buffer de palabra (voy construyendo
    la palabra)
115    lw  t7, 20(sp) # en t7 tengo la direccion del buffer
```

```

PALABRA
116 lw t8, 16(sp) # guardo en t8 el tamaño del buffer PALABRA
117 add t7, t7, t8 # en t7 la direccion del char a guardar
118 sb t2, 0(t7)
119
120 add t0, t0, 1
121 # buffer_word.size ++
122 addi t8, 1
123 sw t8, 16(sp)
124 addi t0, t0, 1 # al indice le sumo 1
125 sw t0, 48(sp)
126
127 # (ver que el buffer word no se llene)
128 # (si se llena pedir mas memoria, hacer swap con memoria
    pedida y memoria vieja)
129
130 b leoByte # vuelvo a leer un byte
131
132 test_palindromo:
133 #me fijo si buffer_word es palindromo
134 #si es palindromo => esPalindromoTrue
135 #sino => leoByte
136
137 li a0, 1
138 li a1, 2
139 jal testpal
140 bnez v0, leoByte # si no es palindromo no hago nada y
    vuelvo a intentar leer un byte
141 # si v0 = 0 entonces sigo adelante
142
143 esPalindromoTrue:
144 # buffer_word -> buffer_salida
145 # (hay que fijarse que buffer salida tenga espacio suficiente
    )
146 # si se llena buffer salida => escribirArchivo
147 # buffer_salida + '\n'
148 # buffer_salida.size ++
149
150 lw t0, 20(sp) # en t0 addr de buffer PALABRA
151 lw t1, 28(sp) # en t1 addr de buffer SALIDA
152 lw t2, 16(sp) # en t2 tam buffer PALABRA
153 lw t3, 40(sp) # en t3 tam buffer SALIDA
154 add t4, zero, zero # en t4 posicion caracter PALABRA (i =
    0)
155
156 addu t1, t1, t3 # en t1 posicion para el proximo
    caracter (t1 = j)
157
158 llenarBufferSalida:
159 add t0, t0, t4
160 lb t5, 0(t0) # t5 = PALABRA[i]
161 sb t5, 0(t1) # SALIDA[j] = PALABRA[i]
162 lw t6, 12(sp) # t6 tamaño maximo buffer SALIDA
163 addi t3, t3, 1 # j + 1
```

```
164      sw      t3, 40(sp)      # guardo el tamaño del buffer SALIDA
                                en el stack
165
166      beq      t6, t3, escribirArchivo # si se llena el buffer
                                escribo (ojo con los indices)
167
168      addi     t4, t4, 1      # i + 1
169
170      #me fijo si no se termino la palabra
171      blt t4, t2, llenarBufferSalida
172
173      #sino => se termino la palabra -> guardo \n en buffer
                                salida
174      add t7, zero, 10      # guardo en t7 el valor 10 ('\n')
175      add t1, t1, t3      # addr SALIDA + j (SALIDA [j])
176      sb t7, 0(t1)      # SALIDA[j] = '\n'
177      addi     t3, t3, 1      # j + 1
178      sw      t3, 40(sp)
179
180      beq      t6, t3, escribirArchivo # si se llena el buffer
                                escribo (ojo con los indices)
181
182      b llenarBufferSalida
183
184 escribirArchivo:
185      # write buffer_salida -> file_out
186      li v0, SYS_write
187      lw a0, 8(sp)      #carga en a0 el fno del archivo de
                                salida
188      lw a1, 28(sp)      #carga en a1 la direccion del buffer de
                                salida
189      lw a2, 40(sp)      #carga en a2 el tamaño del buffer
190      syscall
191
192      bne a3, zero, write_error      #verifico error de escritura
193
194      add t0, zero, zero
195      sw t0, 40(sp)      #reinicio el tamaño del buffer a 0
196
197      b leoByte      #vuelvo a intentar leer bytes del
                                buffer de entrada
198
199 write_error:
200 read_error:
201      li v0, SYS_exit
202      li a0, 1
203      syscall
204
205 palindromoReturn:
206      move v0, zero
207      move sp, $fp
208
209      lw ra, 72(sp)
210      lw $fp, 68(sp)
```



```
211     lw    gp, 64(sp)
212
213     addu  sp, sp, 76
214     j     ra
215     .end  palindrome
```

D. Función testpal

Hemos desarrollado una función testpal.S la cual recibe en a0 la dirección de la palabra a evaluar, en a1 la longitud de la palabra y devuelve en v0: 0 si la palabra es palíndromo o 1 si no lo es.

```
1  #include <sys/syscall.h>
2  #include <mips/regdef.h>
3
4  .text
5  .align 2
6
7  .globl testpal
8  .ent testpal
9  testpal:
10     .frame $fp, 24, ra
11     .set noreorder
12     .cpload t9
13     .set reorder
14     subu sp, sp, 24
15     .cprestore 32
16
17     sw    ra, 20(sp)
18     sw    $fp, 16(sp)
19     sw    gp, 12(sp)
20
21     move  $fp, sp
22
23     add   t9, zero, zero # i = 0
24
25 test:
26     slti  t0, a1, 2      # si a1 (len palabra) < 2 => es
                           palindromo
27     bne   t0, zero, returnTrue
28
29     add   a0, a0, t9      # palabra[i]
30     lb    t1, 0(a0)      # t1 = palabra[i]
31
32     move  t8, t1
33     jal   to_lowercase   # lo paso a lowercase
34     move  t1, t8          # t1 = tolower(palabra[i])
35
36
37     addi  t2, a1, -1      # t2 = len - 1
38     add   t2, t2, a0
39     lb    t2, 0(t2)      # t2 = palabra[len-1]
```

```
40
41     move    t8, t2
42     jal     to_lowercase
43     move    t2, t8                # t2 = tolower(palabra[len -
                                   1])
44
45     bne     t1, t2, esFalse      # si t1 != t2 no es palindromo
46
47     #sino actualizo los indices
48
49     sub     a1, a1, 2            # len = len - 2
50     addi    t9, t9, 1           # i = i + 1
51
52     j test
53
54 esFalse:
55     # si es false devuelvo 1
56     addi    v0, zero, 1
57     j return
58
59 esTrue:
60     # si es true devuelvo 0
61     add     v0, zero, zero
62     j return
63
64 to_lowercase:
65     sgt     t7, t8, 64
66     slti    t6, t8, 91
67     add     t6, t7, t6
68     beq     t6, 2, is_upper
69     #move    v0, t8
70     j       ra
71
72 is_upper:
73     add     t8, t8, 32
74     j       ra
75
76 return:
77     move    sp, $fp
78     lw      ra, 20(sp)
79     lw      $fp, 16(sp)
80     lw      gp, 12(sp)
81     addu    sp, sp, 24
82     j       ra
83     .end    testpal
```

E. Función testpal en C

La siguiente función es el equivalente en C de testpal.S

```
1 int testpal(char * palabra, int len)
2 {
```

```
3     if (len < 2) {  
4         return true;  
5     }  
6  
7     char primero = tolower(palabra[0])  
8     char ultimo = tolower(palabra[len - 1])  
9  
10    if (primero == ultimo)  
11    {  
12        return testpal(palabra + 1, len - 2);  
13    }  
14  
15    return false;  
16 }
```

F. Enunciado

El enunciado se encuentra anexado al final de este documento.