# Trabajo Práctico 1: Conjunto de instrucciones MIPS

Hernán González, *Padrón Nro. 79.460*
gonzalezhg@yahoo.com.ar

Pablo Magnaghi, *Padrón Nro. 88.126*
pablomagnaghi@gmail.com

Enzo Guagnini, *Padrón Nro. 88.325*
enzog_m@hotmail.com

1er. Cuatrimestre de 2011
66.20 Organización de Computadoras – Práctica Martes
Facultad de Ingeniería, Universidad de Buenos Aires

03/05/2011

# 1. Introducción

Este artículo se refiere a la implementación parcial de un programa desarrollado en lenguaje assembly MIPS, el cual es una versión simplificada del comando `tr` de UNIX del trabajo anterior. Para esto, se proveerá el código en C de referencia de dicho comando, que consta fundamentalmente de los archivos *main.c*, *tr.c* y *tr.h*.

# 2. Enunciado

## 2.1. 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.

## 2.2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

## 2.3. Requisitos

El informe deberá ser entregado personalmente, por escrito, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes. Además, es necesario que el trabajo práctico incluya (entre otras cosas), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

## 2.4. Descripción

En este trabajo, se reimplementará parcialmente en assembly MIPS la versión minimalista del comando tr de UNIX del trabajo anterior [1]. Para esto, se proveerá el cdóigo en C de referencia de dicho comando, que consta fundamentalmente de los archivos *main.c*, *tr.c* y *tr.h*. Los alumnos deberán reescribir en assembly MIPS32 las funciones definidas en *tr.c*, respetando la interfaz definida en *tr.h*, de forma tal que *tr.c* será reemplazado por código MIPS32 equivalente contenido en *tr.S*, de manera transparente para *main.c* (es decir, *main.c* se mantendrá sin modificaciones, y se linkeará con *tr.S* en lugar de *tr.c*, para formar el ejecutable final).

## 2.5. Implementación

El programa a implementar deber' a satisfacer algunos requerimientos m'inimos, que detallamos a continuaci'on:

### 2.5.1. ABI

Será necesario que el código presentado utilice la ABI explicada en la clase del martes 12/4/2011 [2].

#### 2.5.2. Casos de prueba

Es necesario que la implementación propuesta pase todos los casos incluidos tanto en el enunciado del trabajo anterior [1] como en el conjunto de pruebas suministrado en el informe del trabajo, los cuales deberán estar debidamente documentados y justificados.

#### 2.5.3. Documentación

El informe deberá incluir una descripción detallada de las técnicas y procesos de desarrollo y debugging empleados, ya que forman parte de los objetivos principales del trabajo.

### 2.6. Informe

El informe deberá incluir:

- Este enunciado.
- Documentación relevante al diseño, desarrollo y debugging del programa.
- Las corridas de prueba, (sección 5.2) y sus correspondientes comentarios.
- El código fuente completo, en dos formatos, digitalizado e impreso en papel.

### 2.7. Fechas

La fecha de entrega, es el martes 3/05. La fecha de vencimiento, 17/05.

## 3. Documentación

La realización de este informe se hizo con la herramienta TEX / LATEX. Tanto el arhivo en formato pdf como el archivo .tex se encuentran en la carpeta doc del cd entregado.

## 4. Técnicas y procesos de desarrollo y debugging empleados

Para la compilación del trabajo se fue trabajando con las funciones que contiene el *tr.c* de manera separada reimplementandolas parcialmente en assembly para encontrar posibles errores, es decir, primero empezamos traduciendo a MIPS la función *void tr_ds(char*, char*)* y se probó que el programa funcionara en forma correcta pasando los casos de prueba pertinentes y luego se prosiguió con *void tr_d(char*)* siguiendo la misma metodología hasta cubrir todas las funciones. Una vez que compilaban todas de manera correcta y pasaban los casos de prueba se junto todo el código assembly en el archivo *tr.S* y se realizó un testeo general igual que en el tp0 y las respuestas fueron las esperadas.

Para la obtención del ejecutable se ejecutó lo siguiente:
*gcc -o tp1a main.c tr.S*

## 5. Casos de prueba

Los casos de prueba utilizados son los que figuran a continuación.

```
$ echo ‘‘3.14159192’’ | ./tp1a 1 9
3.94959992

$ echo ‘‘aba’’ | ./tp1a aba 123
323

$ echo ‘‘aa bb cc’’ | ./tp1a -s ab
a b cc

$ echo ‘‘aa bb cc’’ | ./tp1a -s ab 12
1 2 cc

$ echo ‘‘Hola mundo’’ | ./tp1a -d Ho
la mund

$ echo ‘‘aabbcc’’ | ./tp1a -ds a b
bcc

$ echo ‘‘aaaa’’ | ./tp1a -s a b
b

$ echo ‘‘bbb bbb’’ | ./tp1a -s b
b b

$echo ‘‘bbbaaabb bbabbbbbbb’’ | ./tp1a -ds b a
a a

$echo ‘‘f’’ | ./tp1a aba def
f

$echo ‘‘abaco’’ | ./tp1a -s a
abaco

$echo ‘‘abbbbbabbaaaco’’ | ./tp1a -s a b
bco

$echo ‘‘bbbbbbaaaaaaabbb’’ | ./tp1a -ds a b
b
$echo ‘‘abaco baco abaco’’ | ./tp1a a b
bbbco bbco bbbco
```

# 6. Código fuente

A contiuación se detalla el código fuente implementado en lenguaje C que se encuentra en carpeta code del cd entregado que consta fundamentalmente de los archivos *main.c*, *tr.h* y *tr.c*.

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <assert.h>
4  #include <unistd.h>
5  #include <stdlib.h>
6  #include "tr.h"
7
8  static void usage(void);
9
10 int
11 main(argc, argv)
12     int argc;
13     char **argv;
14 {
15     int ch, dflag, sflag, isset2;
16
17     dflag = sflag = 0;
18     while ((ch = getopt(argc, argv, "ds")) != -1)
19         switch((char)ch) {
20         case 'd':
21             dflag = 1;
22             break;
23         case 's':
24             sflag = 1;
25             break;
26         case '?':
27         default:
28             usage();
29         }
30     argc -= optind;
31     argv += optind;
32
33     switch(argc) {
34     case 0:
35     default:
36         usage();
37         /* NOTREACHED */
38     case 1:
39         isset2 = 0;
40         break;
41     case 2:
42         isset2 = 1;
43         break;
44     }
45
46
47     if (dflag && sflag) {
48         if (!isset2) usage();
49         tr_ds(argv[0], argv[1]);
50     }
51     else if (dflag) {
52         if (isset2) usage();
53         tr_d(argv[0]);
54     }
55     else if (sflag && !isset2) {
56         tr_s1(argv[0]);
57     }
58     else if (sflag && isset2) {
59         tr_s2(argv[0], argv[1]);
60     }
61     else if (isset2) {
62         tr(argv[0], argv[1]);
63     }
64     else {
65         assert(0);
66     }
67
```

```
68      exit(0);
69
70  }
71
72  static void
73  usage()
74  {
75      (void)fprintf(stderr, "usage: tr [-s] string1 string2\n");
76      (void)fprintf(stderr, "       tr -d string1\n");
77      (void)fprintf(stderr, "       tr -s string1\n");
78      (void)fprintf(stderr, "       tr -ds string1 string2\n");
79      exit(1);
80  }
```

```
1   #ifndef TR_H_
2   #define TR_H_
3
4   void tr_ds(char*, char*);
5   void tr_d(char*);
6   void tr_s1(char*);
7   void tr_s2(char*, char*);
8   void tr(char*, char*);
9
10  #endif
```

```
1   #include <sys/cdefs.h>
2   #include <sys/types.h>
3
4   #include <err.h>
5   #include <stdio.h>
6   #include <stdlib.h>
7   #include <unistd.h>
8   #include <string.h>
9   #include <assert.h>
10
11  #include <limits.h>
12  #define NCHARS   (UCHAR_MAX + 1)      /* Number of possible characters. */
13  #define OOBCH    (UCHAR_MAX + 1)      /* Out of band character value. */
14
15  #include "tr.h"
16
17  static int map1[NCHARS] = {
18      0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,      /* ASCII */
19      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
20      0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
21      0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
22      0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
23      0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f,
24      0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
25      0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f,
26      0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47,
27      0x48, 0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f,
28      0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57,
29      0x58, 0x59, 0x5a, 0x5b, 0x5c, 0x5d, 0x5e, 0x5f,
30      0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
31      0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
32      0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
33      0x78, 0x79, 0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f,
34      0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
35      0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f,
36      0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
37      0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f,
38      0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
39      0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf,
40      0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
41      0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
42      0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
43      0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf,
44      0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7,
45      0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf,
46      0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
47      0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
48      0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
49      0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
```

```
50  }, map2[NCHARS];
51
52  static void setup(int *, char *, char *);
53  static void myputchar(char);
54  static int mygetchar(void);
55
56  /*
57   * tr -ds string1 string2
58   * Delete all characters in string1.
59   * Squeeze all characters in string2.
60   */
61  void
62  tr_ds(set1, set2)
63      char* set1;
64      char* set2;
65  {
66      int lastch, ch;
67
68      setup(map1, set1, 0);
69      setup(map2, set2, 0);
70
71      for (lastch = OOBCH; (ch = mygetchar()) != EOF;)
72          if (!map1[ch] && (!map2[ch] || lastch != ch)) {
73              lastch = ch;
74              myputchar(ch);
75          }
76  }
77
78  /*
79   * tr -d string1
80   * Delete all characters in string1.
81   */
82  void
83  tr_d(set1)
84      char* set1;
85  {
86      int ch;
87
88      setup(map1, set1, 0);
89
90      while ((ch = mygetchar()) != EOF)
91          if (!map1[ch])
92              myputchar(ch);
93  }
94
95  /*
96   * tr -s string1
97   * Squeeze all characters in string1.
98   */
99  void
100 tr_s1(set1)
101     char* set1;
102 {
103     int lastch, ch;
104
105     setup(map1, set1, 0);
106
107     for (lastch = OOBCH; (ch = mygetchar()) != EOF;)
108         if (!map1[ch] || lastch != ch) {
109             lastch = ch;
110             myputchar(ch);
111         }
112 }
113
114 /*
115  * tr [-s] string1 string2
116  * Replace all characters in string1 with
117  * the character in the same position in string2.  If the -s option is
118  * specified, squeeze all the characters in string2.
119  */
120 void
121 tr_s2(set1, set2)
122     char* set1;
123     char* set2;
```

```
124 {
125     int lastch, ch;
126 
127     if (!*set2)
128         errx(1, "empty set2");
129 
130     /* If string2 runs out of characters, use the last one specified. */
131     setup(map1, set1, set2);
132 
133     setup(map2, set2, 0);
134 
135     for (lastch = OOBCH; (ch = mygetchar()) != EOF;) {
136         ch = map1[ch];
137         if (!map2[ch] || lastch != ch) {
138             lastch = ch;
139             myputchar(ch);
140         }
141     }
142 }
143 
144 void
145 tr(set1, set2)
146     char* set1;
147     char* set2;
148 {
149     int ch;
150 
151     if (!*set2)
152         errx(1, "empty set2");
153 
154     /* If string2 runs out of characters, use the last one specified. */
155     setup(map1, set1, set2);
156 
157     while ((ch = mygetchar()) != EOF)
158         myputchar(map1[ch]);
159 }
160 
161 static void
162 setup(map, set1, set2)
163     int *map;
164     char *set1;
165     char *set2;
166 {
167     char lastch;
168 
169     if (!set1)
170         assert(0);
171 
172     if (!set2) {
173         memset(map, 0, NCHARS * sizeof(int));
174         while(*set1)
175             map[(unsigned char)*set1++]=1;
176         return;
177     }
178 
179     // set2 not empty => translate
180     while(*set1 && *set2)
181         map[(unsigned char)*set1++] = *set2++;
182 
183     // set1 longer that set2 => use last set2 char till set1's end
184     lastch = *(set2-1);
185     while(*set1)
186         map[(unsigned char)*set1++] = lastch;
187 }
188 
189 static void
190 myputchar(ch)
191     char ch;
192 {
193     if (putchar(ch) == EOF)
194         errx(1, "error writing to stdout");
195 }
196 
197 static int
```

```
198  mygetchar()
199  {
200      int ch;
201      if ((ch = getchar()) == EOF && ferror(stdin))
202          errx(1, "error reading from stdin");
203      return ch;
204  }
```

## 6.1. Código MIPS

El siguiente es el código assembly que reimplementa parcialmente en assembly MIPS la versión minimalista del comando tr, este archivo se encuentra en la carpeta code del cd entregado:

```
1  #include <mips/regdef.h>
2  #include <sys/syscall.h>
3  #SETUP
4  #define SO_SIZE 40
5  #define SO_SET2 48
6  #define SO_SET1 44
7  #define SO_MAP  40
8  #define SO_RA   32
9  #define SO_FP   28
10 #define SO_GP   24
11 #define SO_LASTCH 16
12 #SETUP
13 #myputchar
14 #define PC_SIZE 32
15 #define PC_CH 32
16 #define PC_RA 24
17 #define PC_FP 20
18 #define PC_GP 16
19 #myputchar
20 #mygetchar
21 #define GC_SIZE 40
22 #define GC_RA 32
23 #define GC_FP 28
24 #define GC_GP 24
25 #define GC_CH 16
26 #mygetchar
27 #tr
28 #define TR_SIZE 40
29 #define TR_SET2 44
30 #define TR_SET1 40
31 #define TR_RA   32
32 #define TR_FP   28
33 #define TR_GP   24
34 #define TR_CH   16
35 #tr
36 #tr_s2
37 #define TR_S2_SIZE   40
38 #define TR_S2_SET2   44
39 #define TR_S2_SET1   40
40 #define TR_S2_RA     32
41 #define TR_S2_FP     28
42 #define TR_S2_GP     24
43 #define TR_S2_CH     20
44 #define TR_S2_LASTCH     16
45 #tr_s2
46 #tr_s1
47 #define TR_S1_SIZE   40
48 #define TR_S1_SET1   40
49 #define TR_S1_RA     32
50 #define TR_S1_FP     28
51 #define TR_S1_GP     24
52 #define TR_S1_CH     20
53 #define TR_S1_LASTCH     16
54 #tr_s1
55 #tr_d
56 #define TR_D_SIZE    40
57 #define TR_D_SET1    40
58 #define TR_D_RA      32
59 #define TR_D_FP      28
60 #define TR_D_GP      24
61 #define TR_D_CH      16
62 #tr_d
63 #tr_ds
64 #define TR_DS_SIZE   40
65 #define TR_DS_SET2   44
66 #define TR_DS_SET1   40
67 #define TR_DS_RA     32
```

```
68  #define TR_DS_FP      28
69  #define TR_DS_GP      24
70  #define TR_DS_CH      20
71  #define TR_DS_LASTCH      16
72  #tr_ds
73      .data
74      .align  2
75      .type   map1, @object
76      .size   map1, 1024
77  map1:
78      .word   0
79      .word   1
80      .word   2
81      .word   3
82      .word   4
83      .word   5
84      .word   6
85      .word   7
86      .word   8
87      .word   9
88      .word   10
89      .word   11
90      .word   12
91      .word   13
92      .word   14
93      .word   15
94      .word   16
95      .word   17
96      .word   18
97      .word   19
98      .word   20
99      .word   21
100     .word   22
101     .word   23
102     .word   24
103     .word   25
104     .word   26
105     .word   27
106     .word   28
107     .word   29
108     .word   30
109     .word   31
110     .word   32
111     .word   33
112     .word   34
113     .word   35
114     .word   36
115     .word   37
116     .word   38
117     .word   39
118     .word   40
119     .word   41
120     .word   42
121     .word   43
122     .word   44
123     .word   45
124     .word   46
125     .word   47
126     .word   48
127     .word   49
128     .word   50
129     .word   51
130     .word   52
131     .word   53
132     .word   54
133     .word   55
134     .word   56
135     .word   57
136     .word   58
137     .word   59
138     .word   60
139     .word   61
140     .word   62
141     .word   63
```

```
142         .word   64
143         .word   65
144         .word   66
145         .word   67
146         .word   68
147         .word   69
148         .word   70
149         .word   71
150         .word   72
151         .word   73
152         .word   74
153         .word   75
154         .word   76
155         .word   77
156         .word   78
157         .word   79
158         .word   80
159         .word   81
160         .word   82
161         .word   83
162         .word   84
163         .word   85
164         .word   86
165         .word   87
166         .word   88
167         .word   89
168         .word   90
169         .word   91
170         .word   92
171         .word   93
172         .word   94
173         .word   95
174         .word   96
175         .word   97
176         .word   98
177         .word   99
178         .word   100
179         .word   101
180         .word   102
181         .word   103
182         .word   104
183         .word   105
184         .word   106
185         .word   107
186         .word   108
187         .word   109
188         .word   110
189         .word   111
190         .word   112
191         .word   113
192         .word   114
193         .word   115
194         .word   116
195         .word   117
196         .word   118
197         .word   119
198         .word   120
199         .word   121
200         .word   122
201         .word   123
202         .word   124
203         .word   125
204         .word   126
205         .word   127
206         .word   128
207         .word   129
208         .word   130
209         .word   131
210         .word   132
211         .word   133
212         .word   134
213         .word   135
214         .word   136
215         .word   137
```

```
216         .word   138
217         .word   139
218         .word   140
219         .word   141
220         .word   142
221         .word   143
222         .word   144
223         .word   145
224         .word   146
225         .word   147
226         .word   148
227         .word   149
228         .word   150
229         .word   151
230         .word   152
231         .word   153
232         .word   154
233         .word   155
234         .word   156
235         .word   157
236         .word   158
237         .word   159
238         .word   160
239         .word   161
240         .word   162
241         .word   163
242         .word   164
243         .word   165
244         .word   166
245         .word   167
246         .word   168
247         .word   169
248         .word   170
249         .word   171
250         .word   172
251         .word   173
252         .word   174
253         .word   175
254         .word   176
255         .word   177
256         .word   178
257         .word   179
258         .word   180
259         .word   181
260         .word   182
261         .word   183
262         .word   184
263         .word   185
264         .word   186
265         .word   187
266         .word   188
267         .word   189
268         .word   190
269         .word   191
270         .word   192
271         .word   193
272         .word   194
273         .word   195
274         .word   196
275         .word   197
276         .word   198
277         .word   199
278         .word   200
279         .word   201
280         .word   202
281         .word   203
282         .word   204
283         .word   205
284         .word   206
285         .word   207
286         .word   208
287         .word   209
288         .word   210
289         .word   211
```

```
290       .word   212
291       .word   213
292       .word   214
293       .word   215
294       .word   216
295       .word   217
296       .word   218
297       .word   219
298       .word   220
299       .word   221
300       .word   222
301       .word   223
302       .word   224
303       .word   225
304       .word   226
305       .word   227
306       .word   228
307       .word   229
308       .word   230
309       .word   231
310       .word   232
311       .word   233
312       .word   234
313       .word   235
314       .word   236
315       .word   237
316       .word   238
317       .word   239
318       .word   240
319       .word   241
320       .word   242
321       .word   243
322       .word   244
323       .word   245
324       .word   246
325       .word   247
326       .word   248
327       .word   249
328       .word   250
329       .word   251
330       .word   252
331       .word   253
332       .word   254
333       .word   255
334  #----------------------------tr_ds-------------------------------
335       .text
336       .align  2
337       .globl  tr_ds
338       .ent    tr_ds
339  tr_ds:
340       .frame  $fp,TR_DS_SIZE,ra   # Tamanio 48
341       .set noreorder              # Bloque de codigo PIC
342       .cpload t9
343       .set    reorder
344       subu    sp,sp,TR_DS_SIZE
345       .cprestore TR_DS_GP     # Equivale a sw gp,TR_DS_GP(sp)
346       sw  ra,TR_DS_RA(sp)     # Se guardan los Calle Saved Registers
347       sw  $fp,TR_DS_FP(sp)
348
349       move    $fp,sp          # De ahora en mas se usa FP para direccionar
              el stack
350       sw  a0,TR_DS_SET1($fp)  # Se almacenan los argumentos en la ABA del
            caller
351       sw  a1,TR_DS_SET2($fp)
352       la  a0,map1         # Preparo llamada a setup
353       lw  a1,TR_DS_SET1($fp)
354       move    a2,zero
355       jal     setup           # Llamo a setup(map1, set1, 0);
356       la  a0,map2         # Preparo llamada a setup
357       lw  a1,TR_DS_SET2($fp)
358       move    a2,zero
359       jal     setup           # Llamo a setup(map1, set2, 0);
360       li  t0,256
361       sw  t0,TR_DS_LASTCH($fp)    # lastch=oobch
```

```
362 trdsfor:
363     jal mygetchar
364     sw  v0,TR_DS_CH($fp)    # ch=getchar()
365     li  t0,-1
366     beq v0,t0,fintrds       # si es eof termina
367     lw  t0,TR_DS_CH($fp)    # t0 = ch
368     sll t0,t0,2         # t0 = ch * 4 offset de map1
369     la  t1,map1         # t1 = map1
370     addu    t1,t1,t0        # t1 = map1 + offset
371     lw  t0,0(t1)        # t0 = map1[ch]
372     bne t0,zero,trdsfor     # si no es 0 vuelve al for
373     lw  t0,TR_DS_CH($fp)    # t0 = ch
374     sll t0,t0,2         # t0 = ch * 4 offset de map1
375     la  t1,map2         # t1 = map2
376     addu    t1,t1,t0        # t1 = map2 + offset
377     lw  t0,0(t1)        # t0 = map2[ch]
378     beq t0,zero,trdsif      # si es 0 entra al if
379     lw  t0,TR_DS_CH($fp)    # t0 = ch
380     lw  t1,TR_DS_LASTCH($fp)    # t1 = lastch
381     bne t0,t1,trdsif
382     b   trdsfor
383 trdsif:
384     lw  t0,TR_DS_CH($fp)
385     sw  t0,TR_DS_LASTCH($fp)    # lastch = ch
386     lb  a0,TR_DS_CH($fp)
387     jal myputchar
388     b   trdsfor
389 fintrds:
390     move    sp,$fp
391     lw  ra,TR_DS_RA(sp)
392     lw  $fp,TR_DS_FP(sp)
393     addu    sp,sp,TR_DS_SIZE
394     j   ra
395     .end    tr_ds
396 #-----------------------------tr_d-------------------------------
397     .align  2
398     .globl  tr_d
399     .ent    tr_d
400 tr_d:
401     .frame  $fp,TR_D_SIZE,ra
402     .set noreorder
403     .cpload t9
404     .set    reorder
405     subu    sp,sp,TR_D_SIZE
406     .cprestore TR_D_GP      # Equivale a sw gp,TR_D_GP(sp)
407     sw  ra,TR_D_RA(sp)      # Se guardan los Calle Sabed Registers ra,fp
            ,gp
408     sw  $fp,TR_D_FP(sp)
409
410     move    $fp,sp          # De ahora en mas se usa FP para direccionar
            el stack
411     sw  a0,TR_D_SET1($fp)   # Se almacenan los argumentos en la ABA del
            caller
412     sw  zero,TR_D_CH($fp)
413     la  a0,map1         # Preparo llamada a setup
414     lw  a1,TR_S1_SET1($fp)
415     move    a2,zero
416     jal     setup           # Llamo a setup(map1, set1, 0);
417 trdwh:
418     jal mygetchar
419     sw  v0,TR_D_CH($fp)     # ch = mygetchar()
420     li  t0,-1
421     beq v0,t0,fintrd        # si es eof termina
422     lw  t0,TR_D_CH($fp)     # t0 = ch
423     sll t0,t0,2         # t0 = ch * 4 offset de map1
424     la  t1,map1         # t1 = map1
425     addu    t1,t1,t0        # t1 = map1 + offset
426     lw  t0,0(t1)        # t0 = map1[ch]
427     bne t0,zero,trdwh       # si no es 0 salta al while
428     lw  a0,TR_D_CH($fp)
429     jal myputchar       # myputchar(ch)
430     b   trdwh
431 fintrd:
432     move    sp,$fp
```

```
433     lw   ra,TR_D_RA(sp)
434     lw   $fp,TR_D_FP(sp)
435     addu     sp,sp,TR_D_SIZE
436     j    ra
437     .end     tr_d
438 #------------------------------tr_s1--------------------------------
439     .align   2
440     .globl   tr_s1
441     .ent     tr_s1
442 tr_s1:
443     .frame   $fp,TR_S1_SIZE,ra
444     .set noreorder
445     .cpload t9
446     .set     reorder
447     subu     sp,sp,TR_S1_SIZE
448     .cprestore TR_S1_GP      # Equivale a sw gp,TR_S1_GP(sp)
449     sw   ra,TR_S1_RA(sp)     # Se guardan los Calle Sabed Registers ra,fp
            ,gp
450     sw   $fp,TR_S1_FP(sp)
451
452     move     $fp,sp          # De ahora en mas se usa FP para direccionar
             el stack
453     sw   a0,TR_S1_SET1($fp)  # Se almacenan los argumentos en la ABA del
            caller
454     sw   zero,TR_S1_LASTCH($fp)
455     sw   zero,TR_S1_CH($fp)
456     la   a0,map1         # Preparo llamada a setup
457     lw   a1,TR_S1_SET1($fp)
458     move     a2,zero
459     jal      setup           # Llamo a setup(map1, set1, 0);
460     li   t0,256
461     sw   t0,TR_S2_LASTCH($fp)    # lastch=oobch
462 trs1for:
463     jal mygetchar
464     sw   v0,TR_S1_CH($fp)    # ch=getchar()
465     li   t0,-1
466     beq v0,t0,fintrs1        # si es eof termina
467     lw   t0,TR_S1_CH($fp)    # t0 = ch
468     sll t0,t0,2          # t0 = ch *4 offset del map1
469     la   t1,map1         # t1 = map1
470     addu     t1,t1,t0        # t1 = map1 + offset
471     lw   t0,0(t1)        # t0 = map1[ch]
472     beq t0,zero,trs1if       # si es 0 entra al if
473     lw   t0,TR_S1_CH($fp)    # t0 = ch
474     lw   t1,TR_S1_LASTCH($fp)    # t1 = lastch
475     bne t0,t1,trs1if         # si son diferentes entran al if
476     b    trs1for
477 trs1if:
478     lw   t0,TR_S1_CH($fp)
479     sw   t0,TR_S1_LASTCH($fp)    #lastch = ch
480     lb   a0,TR_S1_CH($fp)
481     jal myputchar
482     b    trs1for
483 fintrs1:
484     move     sp,$fp
485     lw   ra,TR_S1_RA(sp)
486     lw   $fp,TR_S1_FP(sp)
487     addu     sp,sp,TR_S1_SIZE
488     j    ra
489     .end     tr_s1
490 #------------------------------tr_s2--------------------------------
491     .align   2
492     .globl   tr_s2
493     .ent     tr_s2
494 tr_s2:
495     .frame   $fp,TR_S2_SIZE,ra
496     .set noreorder
497     .cpload t9
498     .set     reorder
499     subu     sp,sp,TR_S2_SIZE
500     .cprestore TR_S2_GP      # Equivale a sw gp,TR_S2_GP(sp)
501     sw   ra,TR_S2_RA(sp)     # guardo ra,fp,gp
502     sw   $fp,TR_S2_FP(sp)
503
```

```
504       move    $fp,sp
505       sw  a0,TR_S2_SET1($fp)
506       sw  a1,TR_S2_SET2($fp)
507       sw  zero,TR_S2_LASTCH($fp)
508       sw  zero,TR_S2_CH($fp)
509       lw  t0,TR_S2_SET2($fp)  # t0 = set2
510       lb  t0,0(t0)        # t0 = * set2
511       bne t0,zero,segtrs2     # si no hay error salta,sino
512       li  v0,SYS_write        # imprime mensaje de error
513       li  a0,1                # a0: file descriptor number.
514       la  a1,msgerrset2       # a1: output data pointer.
515       li  a2,11           # a2: output byte size.
516       syscall
517       b   fintrs2
518   segtrs2:
519       la  a0,map1         # preparo llamada a setup(map1,set1,set2)
520       lw  a1,TR_SET1($fp)
521       lw  a2,TR_SET2($fp)
522       jal     setup
523       la  a0,map2         # preparo llamada a setup(map2,set2,0)
524       lw  a1,TR_SET2($fp)
525       li  a2,0
526       jal     setup
527       li  t0,256
528       sw  t0,TR_S2_LASTCH($fp)    # lastch=oobch
529   trs2for:
530       jal mygetchar
531       sw  v0,TR_S2_CH($fp)    # ch=getchar()
532       li  t0,-1
533       beq v0,t0,fintrs2       # si es eof termina
534       lw  t0,TR_S2_CH($fp)    # t0 = ch
535       sll t0,t0,2         # t0 = t0 * 4 offset del map
536       la  t1,map1         # t1 = map1
537       addu    t1,t1,t0        # t1 = map + offset
538       lw  t0,0(t1)        # t0 = map1[ch]
539       sw  t0,TR_S2_CH($fp)    # ch = map1[ch]
540       lw  t0,TR_S2_CH($fp)    # t0 = ch
541       sll t0,t0,2         # t0 = ch *4 offset del map2
542       la  t1,map2         # t1 = map2
543       addu    t1,t1,t0        # t1 = map2 + offset
544       lw  t0,0(t1)        # t0 = map2[ch]
545       beq t0,zero,trs2if      # si es 0 entra al if
546       lw  t0,TR_S2_CH($fp)    # t0 = ch
547       lw  t1,TR_S2_LASTCH($fp)    # t1 = lastch
548       bne t0,t1,trs2if
549       b   trs2for
550   trs2if:
551       lw  t0,TR_S2_CH($fp)
552       sw  t0,TR_S2_LASTCH($fp)    # lastch = ch
553       lb  a0,TR_S2_CH($fp)
554       jal myputchar
555       b   trs2for
556   fintrs2:
557       move    sp,$fp
558       lw  ra,TR_S2_RA(sp)
559       lw  $fp,TR_S2_FP(sp)
560       addu    sp,sp,TR_S2_SIZE
561       j   ra
562       .end    tr_s2
563
564       .rdata
565   msgerrset2:
566       .ascii  "empty_set2\000" #strlen = 11
567
568   #----------------------------tr------------------------------
569       .text
570       .align  2
571       .globl  tr
572       .ent    tr
573   tr:
574       .frame  $fp,TR_SIZE,ra
575       .set noreorder
576       .cpload t9
577       .set    reorder
```

```
578       subu    sp,sp,TR_SIZE
579       .cprestore TR_GP        # Equivale sw gp,TR_GP(sp)
580       sw  ra,TR_RA(sp)        # guardo ra,fp,gp
581       sw  $fp,TR_FP(sp)
582
583       move    $fp,sp
584       sw  a0,TR_SET1($fp)
585       sw  a1,TR_SET2($fp)
586       sw  zero,TR_CH($fp)
587       lw  t0,TR_SET2($fp)     # t0 = set2
588       lb  t0,0(t0)        # t0 = *set2
589       bne t0,zero,segtr       # si no hay error salta a segtr
590       li  v0,SYS_write        # en caso de error imprimo que hubo error en
              set2
591       li  a0,1                # a0: file descriptor number.
592       la  a1,msgerrset2       # a1: output data pointer.
593       li  a2,11           # a2: output byte size.
594       syscall
595       b   fintr
596   segtr:
597       la  a0,map1         # preparo llamada a setup
598       lw  a1,TR_SET1($fp)
599       lw  a2,TR_SET2($fp)
600       jal     setup
601   trwh1:
602       jal mygetchar       # llamo a mygetchar, devuelve en v0
603       sw  v0,TR_CH($fp)       # ch = getchar()
604       move    t0,v0
605       li  t1,-1
606       beq t0,t1,fintr     # si es eof termina
607       sll t0,t0,2         # t0 = ch *4 = offset del map
608       la  t1,map1         # t1 = map
609       addu    t1,t1,t0        # t1 = map + offsetmap
610       lb  a0,0(t1)        # a0 = map1[ch]
611       jal myputchar
612       b   trwh1
613   fintr:
614       move    sp,$fp
615       lw  ra,TR_RA(sp)
616       lw  $fp,TR_FP(sp)
617       addu    sp,sp,TR_SIZE
618       j   ra
619       .end    tr
620   #-----------------------------setup--------------------------------
621       .align  2
622       .ent    setup
623   setup:
624       .frame  $fp,SO_SIZE,ra
625       .set noreorder
626       .cpload t9
627       .set    reorder
628       subu    sp,sp,SO_SIZE
629       .cprestore SO_GP        # Equivale a sw gp,SO_GP(sp)
630       sw  ra,SO_RA(sp)        # guardo ra,fp,gp
631       sw  $fp,SO_FP(sp)
632
633       move    $fp,sp
634       sw  a0,SO_MAP(sp)       # guardo los 3 argumentos
635       sw  a1,SO_SET1(sp)
636       sw  a2,SO_SET2(sp)
637       lw  t0,SO_SET1($fp)     # primer if(!set1)
638       beq t0,zero,finsetup    # si se cumple termina el setup sino sigo
              con el segundo if
639       lw  t0,SO_SET2($fp)
640       bne t0,zero,swh2
641       li  t0,0            # t0 = 0 i
642       li  t1,256          # t1 = 255
643   memset:
644       sll     t3,t0,2         # t3 = i*4 offset
645       lw  t2,SO_MAP($fp)      # t2 = map
646       addu    t2,t2,t3        # t2 = map + offset
647       sw  zero,0(t2)      # map[t0] = 0
648       addiu   t0,t0,1         # t0 += 1
649       blt t0,t1,memset
```

```
650 swh1:                   # while1 adentro de if1
651     lw   t0,SO_SET1($fp)
652     lb   t0,0(t0)
653     beq  t0,zero,finsetup     # si es 0 hace return
654     lw   t0,SO_SET1($fp)      # t0 = set1
655     lbu  t1,0(t0)         # t1 = *set1
656     sll  t1,t1,2          # t1 = *set1 *4 para obtener offset dentro de
            map
657     lw   t2,SO_MAP($fp)       # t2 = map
658     addu     t2,t2,t1         # t2 = map + offsetmap
659     li   t1,1             # t1 = 1
660     sw   t1,0(t2)         # map[(unsigned char) *set1] = 1
661     addu     t0,t0,1          # set1++
662     sw   t0,SO_SET1($fp)
663     b    swh1
664 swh2:                   # while(*set1 && *set2)
665     lw   t0,SO_SET1($fp)
666     lb   t0,0(t0)         # t0 = *set1
667     beq  t0,zero,nswh2
668     lw   t1,SO_SET2($fp)
669     lb   t1,0(t1)         # t1 = *set2
670     beq  t1,zero,nswh2
671     lw   t0,SO_SET1($fp)      # t0 = set1
672     lbu  t1,0(t0)         # t1 = *set1
673     lw   t2,SO_SET2($fp)      # t2 = set2
674     lbu  t3,0(t2)         # t3 = *set2
675     lw   t4,SO_MAP($fp)       # t4 = map
676     sll  t1,t1,2          # $t1 = *set1 *4 offset del map
677     addu     t4,t4,t1         # t4 = map + offsetdelmap
678     sw   t3,0(t4)         # map[(unsigned char) *set1] = *set2
679     addu     t0,t0,1          # set1++
680     sw   t0,SO_SET1($fp)
681     addu     t2,t2,1          # set2++
682     sw   t2,SO_SET2($fp)
683     b    swh2
684 nswh2:
685     lw   t0,SO_SET2($fp)      # t0 = set2
686     addu     t0,t0,-1         # t0 = set2 - 1
687     lbu  t0,0(t0)         # t0 = *set2
688     sb   t0,SO_LASTCH($fp)
689 swh3:
690     lw   t0,SO_SET1($fp)      # t0 = set1
691     lb   t0,0(t0)         # t0 = *set1
692     beq  t0,zero,finsetup
693     lw   t0,SO_SET1($fp)      # t0 = set1
694     lbu  t1,0(t0)         # t1 = *set1
695     sll  t1,t1,2          # $t1 = *set1 *4 offset del map
696     lw   t2,SO_MAP($fp)       # t2 = map
697     addu     t2,t2,t1         # t2 = map + offsetmap
698     lb   t1,SO_LASTCH($fp)    # t1 = lastch
699     sw   t1,0(t2)         # map[(unsigned char)*set1] = lastch
700     addu     t0,t0,1          # set1++
701     sw   t0,SO_SET1($fp)
702     b    swh3
703 finsetup:
704     move     sp,$fp
705     lw   ra,SO_RA(sp)
706     lw   $fp,SO_FP(sp)
707     addu     sp,sp,SO_SIZE
708     j    ra
709     .end     setup
710
711     .rdata
712 msgerr1:
713     .ascii   "error␣writing␣to␣stdout\000" #strlen = 24
714
715 #----------------------------myputchar-------------------------------
716     .text
717     .align   2
718     .ent     myputchar
719 myputchar:
720     .frame   $fp,PC_SIZE,ra
721     .set noreorder
722     .cpload t9
```

```
723       .set    reorder
724       subu    sp,sp,PC_SIZE
725       .cprestore PC_GP        # Equivale a sw gp,PC_GP(sp)
726       sw  ra,PC_RA(sp)        # guardo ra,fp,gp
727       sw  $fp,PC_FP(sp)
728
729       move    $fp,sp
730       sb  a0,PC_CH($fp)       # Guardo ch
731       li  v0, SYS_write
732       li  a0,1                # a0: file descriptor number.
733       la  a1,PC_CH($fp)       # a1: output data pointer.
734       li  a2,1                # a2: output byte size.
735       syscall
736       beq a3,zero,finmyputchar
737       li  v0,SYS_write        # en caso de error imprimo que hubo error de
              escritura
738       li  a0,1                # a0: file descriptor number.
739       la  a1,msgerr1      # a1: output data pointer.
740       li  a2,24               # a2: output byte size.
741       syscall
742   finmyputchar:
743       move    sp,$fp
744       lw  ra,PC_RA(sp)
745       lw  $fp,PC_FP(sp)
746       addu    sp,sp,PC_SIZE
747       j   ra
748       .end    myputchar
749
750       .rdata
751   msgerr2:
752       .ascii  "error reading from stdin\000" #strlen = 25
753   #----------------------------mygetchar-------------------------------
754       .text
755       .align  2
756       .ent    mygetchar
757   mygetchar:
758       .frame  $fp,GC_SIZE,ra
759       .set noreorder
760       .cpload t9
761       .set    reorder
762       subu    sp,sp,GC_SIZE
763       .cprestore GC_GP        # Equivale sw gp,GC_GP(sp)
764       sw  ra,GC_RA(sp)        # guardo ra,fp,gp
765       sw  $fp,GC_FP(sp)
766
767       move    $fp,sp
768       sw  zero,GC_CH($fp)     # inicializo ch
769       li  v0, SYS_read
770       li  a0, 0               # a0: file descriptor number.
771       la  a1,GC_CH($fp)       # a1: data pointer.
772       li  a2, 1               # a2: available space.
773       syscall             # ch = getchar()
774       bne a3,zero,readerror   # si a3 no es 0 hubo error
775       beq v0,zero,mygeteof    # si v0 = 0 es eof retornara -1
776       li  t0, 1
777       bne v0,t0,readerror     # si se leyo una cantidad diferente a 1 es
              error
778       b   finmygetchar
779   readerror:
780       li  v0,SYS_write        # en caso de error imprimo que hubo error de
              escritura
781       li  a0,1                # a0: file descriptor number.
782       la  a1,msgerr2      # a1: output data pointer.
783       li  a2,25               # a2: output byte size.
784       syscall
785       b   finmygetchar
786   mygeteof:
787       li  t0,-1
788       sw  t0,GC_CH($fp)       # en caso de eof retornara -1 en v0
789   finmygetchar:
790       lw  v0,GC_CH($fp)
791       move    sp,$fp
792       lw  ra,GC_RA(sp)
793       lw  $fp,GC_FP(sp)
```

```
794         addu    sp,sp,GC_SIZE
795         j   ra
796         .end    mygetchar
797         .local  map2
798         .comm   map2,1024,4
```

# 7. Conclusiones

El presente logró familiarizarnos a distintas herramientas que se utilizan en la materia. Entre las actividades realizadas podemos resaltar:

- Realizar código en assembly para la arquitectura MIPS.
- Aprender a utilizar la ABI.
- Desarrollar el presente informe en LATEX.