

UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

66.20 ORGANIZACIÓN DE COMPUTADORAS

Trabajo Práctico 1

Integrantes:

Daniel FERNANDEZ - 93083

Nicolas ORTOLEVA - 93196

Maximiliano SCHULTHEIS - 93285



6 de Mayo de 2014

Índice

1. Diseño e implementación	2
2. Comandos de compilación	2
2.1. Makefile	2
3. Pruebas realizadas	2
3.1. Primeras pruebas	2
3.2. Prueba de archivos aleatorios	3
4. Código Fuente	3
4.1. Código fuente C tp1.c	3
4.2. Código de b16.h	5
4.3. Código assembly MIPS b16.S	5
4.4. Código assembly MIPS b16.S	13
5. Stack Frame	21
5.1. Stack Frame byte encoder	21
5.2. Stack Frame encode	21
5.3. Stack Frame correr Referencia	21
5.4. Stack Frame byte decoder	22
5.5. Stack Frame decode	22
6. Conclusiones	22

1. Diseño e implementación

2. Comandos de compilación

2.1. Makefile

para entregar/makefile

```
1 para entregar/makefile
2 all: tp1
3
4 b16c: b16c.h
5         gcc -Wall -c b16c.c
6
7 b16S: b16.h
8         gcc -Wall -c b16.S
9
10 tp1: b16c b16S
11         gcc -Wall b16.o b16c.o -o tp1 tp1.c
12
13 clean:
14         rm b16c.o b16.o tp1para entregar/makefile
```

3. Pruebas realizadas

3.1. Primeras pruebas

```
1 prueba="Archivo vacio"
2 touch /tmp/zero.txt
3 ./tp1 -a encode -i /tmp/zero.txt -o /tmp/zero.txt.b16
4 longitud='ls -la /tmp/zero.txt.b16 | awk '{print $5}''
5 if [[ $longitud -eq 0 ]] ; then echo "ok: $prueba"; else echo "ERROR: $prueba" ; fi
6 rm -f /tmp/zero.txt /tmp/zero.txt.b16
7
8 prueba="Codificacion de 'M' por entrada estandar"
9 hexa='echo -n M | ./tp1'
10 if [[ "$hexa" == "4D" ]] ; then echo "ok: $prueba"; else echo "ERROR: $prueba" ; fi
11
12 prueba="Codificacion de 'Ma' por entrada estandar"
13 hexa='echo -n Ma | ./tp1'
14 if [[ "$hexa" == "4D61" ]] ; then echo "ok: $prueba"; else echo "ERROR: $prueba" ; fi
15
16 prueba="Codificacion de 'Man' por entrada estandar"
17 hexa='echo -n Man | ./tp1'
18 if [[ "$hexa" == "4D616E" ]] ; then echo "ok: $prueba"; else echo "ERROR: $prueba" ;
19     fi
20
21 prueba="Codificacion y decodificacion de 'Man' por entrada estandar"
22 mensaje='echo -n Man | ./tp1 | ./tp1 -a decode'
23 if [[ "$mensaje" == "Man" ]] ; then echo "ok: $prueba"; else echo "ERROR: $prueba" ;
24     fi
25
26 prueba="Verificacion bit a bit de codificacion y decodificacion de xyz\n"
27 esperado="0000000    x    y    z    \n
28 0000004"
29 resultado='echo xyz | ./tp1 | ./tp1 -a decode | od -t c'
30 if [[ "$resultado" == "$esperado" ]] ; then echo "ok: $prueba"; else echo "ERROR:
31     $prueba" ; fi
```

3.2. Prueba de archivos aleatorios

```
1 n=1;
2 while ;; do
3     head -c $n </dev/urandom >/tmp/in.bin;
4     ./tp1 -a encode -i /tmp/in.bin -o /tmp/out.b16;
5     ./tp1 -a decode -i /tmp/out.b16 -o /tmp/out.bin;
6
7     if diff /tmp/in.bin /tmp/out.bin; then ;; else
8         echo ERROR: $n;
9         break;
10
11     fi
12
13 echo ok: $n;
14
15     n="'expr $n + 1'";
16
17 rm -f /tmp/in.bin /tmp/out.b16 /tmp/out.bin
18
19 done
```

4. Código Fuente

4.1. Código fuente C tp1.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <getopt.h>
4 #include <string.h>
5 #include <stdbool.h>
6
7 #include "b16.h"
8
9 static bool encoderActivo = true;
10
11 static struct option long_options[] = {
12     {"version", no_argument, 0, 'v'},
13     {"help", no_argument, 0, 'h'},
14     {"input", required_argument, 0, 'i'},
15     {"output", required_argument, 0, 'o'},
16     {"action", required_argument, 0, 'a'},
17     {0, 0, 0, 0}
18 };
19
20 int procesarArchivos (FILE* finput, FILE* foutput) {
21     int infd = fileno (finput);
22     int outfd = fileno (foutput);
23     int resultado = 0;
24     if (encoderActivo) resultado = encode (infd, outfd);
25     else resultado = decode (infd, outfd);
26
27     if (finput != stdin) fclose(finput);
28     if (foutput != stdout) fclose(foutput);
```

```

29     return resultado;
30 }
31
32 void escribir_error (int errorcode) {
33     int error = (-1) * errorcode;
34     fprintf(stderr, "%s", b16_errmsg[error]);
35     exit (error);
36 }
37
38 void comprobarAction (char* optarg) {
39     if ( strcmp (optarg, "encode") == 0 ) {
40         encoderActivo = true;
41     }
42     if ( strcmp (optarg, "decode") == 0 ) {
43         encoderActivo = false;
44     }
45 }
46
47 void imprimirAyuda () {
48     printf("Usage:\n");
49     printf("\t ./tp0 -h\n");
50     printf("\t ./tp0 -v\n");
51     printf("Options:\n");
52     printf("\t -v, --version, Shows the version of TP. \n");
53     printf("\t -h, --help , Show help \n");
54     printf("\t -i, --input, Location of the input file\n");
55     printf("\t -o, --output, Location of the output file\n");
56     printf("\t -a, --action, Program action: encode (default) or decode \n");
57     printf("Example: \n");
58     printf("\t ./tp0 -a encode -i /input -o /output -h\n");
59     printf("\t ./tp0 -a decode\n");
60 }
61
62 int opciones (int argc , char** argv, FILE** finput, FILE** foutput) {
63
64     int option_index = 0;
65     int option = getopt_long ( argc, argv, "vhi:o:a:", long_options, &
66         option_index);
67     while ( option != -1 ) {
68
69         switch (option) {
70             case 'v':
71                 printf("66.20-Organizacion de Computadoras TP
72                     Version 0.0\n");
73                 return 1;
74                 break;
75             case 'h':
76                 imprimirAyuda();
77                 return 1;
78                 break;
79             case 'i':
80                 (*finput) = fopen(optarg,"r");
81                 if ((*finput) == NULL) {
82                     fprintf(stderr,"Error al abrir el
83                         archivo input %s\n",optarg);
84                     exit(4);
85                 }
86                 break;
87             case 'o':
88                 (*foutput) = fopen(optarg, "w");
89                 if ((*foutput) == NULL) {

```

```

87             fprintf(stderr,"Error al abrir el
88                 archivo output %s \n",optarg);
89             exit(5);
90         }
91         break;
92     case 'a':
93         comprobarAction(optarg);
94         break;
95     default:
96         break;
97     }
98     option = getopt_long ( argc, argv, "vhi:o:a:", long_options, &
99         option_index);
100 }
101 return 0;
102 }
103
104 int main (int argc, char** argv) {
105     FILE* finput = stdin;
106     FILE* foutput = stdout;
107     int opcion = opciones (argc, argv, &finput, &foutput);
108
109     if (opcion == 0) {
110         int resultado = procesarArchivos (finput, foutput);
111         if (resultado < 0) escribir_error(resultado);
112     }
113     return 0;
114 }

```

4.2. Código de b16.h

```

1  #ifndef _b16_H_
2  #define _b16_H_
3
4  extern const char* b16_errmsg[];
5
6  extern int encode(int infd, int outfd);
7  extern int decode(int infd, int outfd);
8
9  #endif //_b16_H

```

4.3. Código assembly MIPS b16.S

```

1  #include<mips/regdef.h>
2  #include<sys/syscall.h>
3
4  #####          ENCODE          #####
5      .text
6      .align 2
7      .globl byte_encoder
8      .ent    byte_encoder
9
10 byte_encoder:                                     # void byte_encoder (char*
    valorHexa, unsigned int numInt)

```

```

11 ##### STACK FRAME #####
12 #define BE_FSIZE 16
13
14 ##### CALLER ARGS #####
15 #define BE_FRAME_A1 20
16 #define BE_FRAME_A0 16
17
18 ##### SRA #####
19 #define BE_FRAME_FP 12
20 #define BE_FRAME_GP 8
21
22 ##### LTA #####
23 #define BE_FRAME_LNIBBLE 4
24 #define BE_FRAME_HNIBBLE 0
25
26
27 .frame $fp, BE_FSIZE, ra # (2 SRA + 2 LTA) * 4 bytes; no hay
28     ABA por ser funcion leaf
29     subu    sp, sp, BE_FSIZE
30
31     sw      $fp, BE_FRAME_FP(sp) # guardo fp en BE_FRAME_FP + sp
32     sw      gp, BE_FRAME_GP(sp)  # guardo gp en BE_FRAME_GP +
33     sp
34     move    $fp, sp             # llevo fp a la pos del sp
35
36     # Argumento de funcion
37     sw      a0, BE_FRAME_A0($fp) # a0: char* valorHexa
38     sw      a1, BE_FRAME_A1($fp) # a1: unsigned int numInt
39
40     andi    t0, a1, 0xf0         # t0 = highNibble de numInt
41     sra     t0, t0, 4             # highNibble >> 4
42     sw      t0, BE_FRAME_HNIBBLE($fp) # guardo la variable local t0 en el
43     stack frame
44
45     andi    t1, a1, 0x0f         # t1 = lowNibble de numInt
46     sw      t1, BE_FRAME_LNIBBLE($fp) # guardo la variable local t1 en el
47     stack frame
48
49     lb      t2, vecHexa(t0)      # t2 = vecHexa[t0]; -> t2:
50     primer caracter hexa
51     sb      t2, 0(a0)            # a0[0] = t2; es
52     decir: valorHexa[0] = t2
53
54     lw      a0, BE_FRAME_A0($fp) # tengo a0 nuevamente char* valorHexa
55     (por seguridad)
56
57     lb      t3, vecHexa(t1)      # t3 = vecHexa[t1]; -> t3:
58     segundo caracter hexa
59     sb      t3, 1(a0)            # a0[1] = t3; es
60     decir: valorHexa[1] = t3
61
62     lw      $fp, BE_FRAME_FP(sp) # recupero fp
63     addu    sp, sp, BE_FSIZE     # destruyo el stack frame
64
65     jr      ra
66
67     .end    byte_encoder
68     .size   byte_encoder, .-byte_encoder
69
70
71
72

```

```

63         .align    2
64         .globl    encode
65         .ent      encode
66
67 encode:                                     # int encode (int
        infd, int outfd)
68
69 ##### STACK FRAME #####
70 #define ENC_FSIZE 48
71
72 ##### CALLER ARGS #####
73 #define ENC_FRAME_A1 52
74 #define ENC_FRAME_A0 48
75
76 #####          SRA          #####
77 # se agrega un word de padding
78 #define ENC_FRAME_RA 40
79 #define ENC_FRAME_FP 36
80 #define ENC_FRAME_GP 32
81
82 #####          LTA          #####
83 # el siguiente string tiene dos caracteres
84 #define ENC_FRAME_STRING 24
85 #define ENC_FRAME_BYTES 20
86 #define ENC_FRAME_CHARACTER 16
87
88 #####          ABA          #####
89 #define ENC_FRAME_ARG3 12
90 #define ENC_FRAME_ARG2 8
91 #define ENC_FRAME_ARG1 4
92 #define ENC_FRAME_ARG0 0
93
94         .frame    $fp, ENC_FSIZE, ra
95         subu      sp, sp, ENC_FSIZE
96         .cprestore    ENC_FRAME_GP
97         sw        ra, ENC_FRAME_RA(sp)
98         sw        $fp, ENC_FRAME_FP(sp)
99         move      $fp, sp
100
101         sw        a0, ENC_FRAME_A0($fp)
102         sw        a1, ENC_FRAME_A1($fp)
103         sw        zero, ENC_FRAME_CHARACTER($fp)    # character = 0
104
105 read_y_loop:
106         addu      a1,$fp,ENC_FRAME_CHARACTER        # a1 = &caracter
107         li        a2,1                                # a2 = 1,
108         para leer un byte
109         li        v0,    SYS_read                    # tengo los 3
110         argumentos del read: a0=infd, a1=&caracter, a2=1
111         syscall                                      # en v0 tengo
112         la cantidad de bytes que leo o negativo si hubo error
113         bltz      v0, error_read                      # salto si hubo un error de
114         lectura
115
116         bgtz      v0, while_encode                    # entro al while si es mayor
117         a 0 (en este caso si es 0, es eof)
118         b         return_encode                        # salta en caso de
119         que sea menor o igual a 0
120
121 while_encode:

```



```

116      sw      v0, ENC_FRAME_BYTES($fp)                # salvo v0 por llamada de
          funcion de byte_encoder
117      addu a0, $fp, ENC_FRAME_STRING                # cargo en a0 la direccion $fp +
          ENC_FRAME_STRING (comienzo del char*)
118      lw      a1, ENC_FRAME_CHARACTER($fp)          # a1 = caracter leído
119      la      t9, byte_encoder                      # carga en t9 donde
          esta byte_encoder
120      jal     t9                                    #
          salta a byte_encoder
121
122      lw      a0, ENC_FRAME_A1($fp)                  # en a0 tengo outfd
123      addu a1, $fp, ENC_FRAME_STRING                # cargo en a1 la direccion $fp +
          ENC_FRAME_STRING (comienzo del char*)
124      li      a2, 2                                # cargo en a2
          el 2, para escribir dos bytes
125      li      v0, SYS_write                          # llamo a write
126      syscall
127      bltz    v0, error_write                        # si es menor a 0, hubo un
          error de escritura
128
129      lw      a0, ENC_FRAME_A0($fp)                  # a0 = infd
130      b read_y_loop
131
132  error_write:
133      sub      v0, zero, 2                          # error2: -2
134      sw      v0, ENC_FRAME_BYTES($fp)
135      b return_encode
136
137  error_read:
138      sub      v0, zero, 1                          # error1: -1
139      sw      v0, ENC_FRAME_BYTES($fp)
140
141  return_encode:
142      lw v0, ENC_FRAME_BYTES($fp)
143      lw ra, ENC_FRAME_RA(sp)
144      lw $fp, ENC_FRAME_FP(sp)
145      addu sp, sp, ENC_FSIZE
146
147      jr      ra
148
149      .end     encode
150      .size    encode, .-encode
151
152      .rdata
153      .align   2
154      .size    vecHexa, 16
155  vecHexa:
156      .byte    48                                #'0'
157      .byte    49                                #'1'
158      .byte    50                                #'2'
159      .byte    51                                #'3'
160      .byte    52                                #'4'
161      .byte    53                                #'5'
162      .byte    54                                #'6'
163      .byte    55                                #'7'
164      .byte    56                                #'8'
165      .byte    57                                #'9'
166      .byte    65                                #'A'
167      .byte    66                                #'B'
168      .byte    67                                #'C'
169      .byte    68                                #'D'

```

```

170         .byte    69                                #'E'
171         .byte    70                                #'F'
172
173 #####          DECODE #####
174         .text
175         .align   2
176         .globl   correrReferencia
177         .ent     correrReferencia
178
179 correrReferencia:                                # int correrReferencia (int
        numInt)
180
181 ##### STACK FRAME #####
182 #define CR_FSIZE 8
183
184 ##### CALLER ARGS #####
185 #define CR_FRAME_A0 8
186
187 #####          SRA          #####
188 #define CR_FRAME_FP 4
189 #define CR_FRAME_GP 0
190
191         .frame    $fp, CR_FSIZE, ra
192         subu      sp, sp, CR_FSIZE
193         sw        $fp, CR_FRAME_FP(sp)
194         sw        gp, CR_FRAME_GP(sp)
195         move      $fp, sp
196
197         sw        a0, CR_FRAME_A0($fp)            # En a0 tengo el parametro numInt
198
199         slt t0, a0, 58                                # si numInt < 58 -> t0 = 1
200         sgt t1, a0, 47                                # si numInt > 47 -> t1 = 1
201         and t0, t0, t1                                # si t0 and t1 = 1 -> t0 = 1
202         beqz t0, comparacion2                        # si no esta en ese rango se compara
        en siguiente
203         lw v0, CR_FRAME_A0($fp)                    # se almacena en v0 el a0=numInt
204         sub v0, v0, 48                                # se tiene en v0 = numInt -
        48
205         b return
206
207 comparacion2:
208         slt t0, a0, 71                                # idem al anterior con otro
        rango
209         sgt t1, a0, 64
210         and t0, t0, t1
211         beqz t0, comparacion3
212         lw v0, CR_FRAME_A0($fp)
213         sub v0, v0, 55
214         b return
215
216 comparacion3:
217         slt t0, a0, 103                                # idem al anterior con otro
        rango
218         sgt t1, a0, 96
219         and t0, t0, t1
220         beqz t0, error_caracterNoHexa
221         lw v0, CR_FRAME_A0($fp)
222         sub v0, v0, 87
223         b return
224
225 error_caracterNoHexa:

```

```

226         sub        v0, zero, 3                                # error3: -3
227
228     return:
229         lw          $fp, CR_FRAME_FP(sp)
230         addu        sp, sp, CR_FSIZE
231         jr          ra
232         .end        correrReferencia
233         .size       correrReferencia, .-correrReferencia
234
235
236
237         .align      2
238         .globl      byte_decoder
239         .ent        byte_decoder
240
241     byte_decoder:                                             # int byte_decoder (int
        numPri, int numSeg)
242
243     ##### STACK FRAME #####
244     #define BD_FSIZE 40
245
246     ##### CALLER ARGS #####
247     #define BD_FRAME_A1 44
248     #define BD_FRAME_A0 40
249
250     #####          SRA          #####
251     # se agrega un word de padding
252     #define BD_FRAME_RA 32
253     #define BD_FRAME_FP 28
254     #define BD_FRAME_GP 24
255
256     #####          LTA          #####
257     #define BD_FRAME_LNIBBLE 20
258     #define BD_FRAME_HNIBBLE 16
259
260     #####          ABA          #####
261     #define BD_FRAME_ARG3 12
262     #define BD_FRAME_ARG2 8
263     #define BD_FRAME_ARG1 4
264     #define BD_FRAME_ARG0 0
265
266         .frame       $fp, BD_FSIZE, ra
267         subu         sp, sp, BD_FSIZE
268         .cprestore   BD_FRAME_GP
269
270         sw           ra, BD_FRAME_RA(sp)
271         sw           $fp, BD_FRAME_FP(sp)
272         move         $fp, sp
273
274         sw           a0, BD_FRAME_A0($fp)                    # en a0 tengo numPri
275         sw           a1, BD_FRAME_A1($fp)                    # en a1 tengo numSeg
276
277         la           t9, correrReferencia
278         jal          t9
279         sw           v0, BD_FRAME_HNIBBLE($fp)               # highNibble =
        correrReferencia (numPri)
280         bltz         v0, returnValor                          # si es menor a 0 -> error,
        fin decode
281
282         lw           a0, BD_FRAME_A1($fp)                    # cargo en a0 el numSeg
283         la           t9, correrReferencia

```

```

284     jal      t9
285     sw        v0, BD_FRAME_LNIBBLE($fp)           # lowNibble =
286     correrReferencia(numSeg)
287     bltz      v0, returnValor                     # si es menor a 0 -> error,
288     fin decode
289
290     lw        t0, BD_FRAME_HNIBBLE($fp)
291     sll       t0, t0, 4                           # t0 =
292     highNibble << 4
293     andi      t0, t0, 0xf0                         # aseguro ceros en
294     nibble menos significativo
295     lw        t1, BD_FRAME_LNIBBLE($fp)           # t1 = lowNibble
296     andi      t1, t1, 0xf                          # aseguro ceros en
297     nibble mas significativo
298     or        v0, t0, t1                           # v0 =
299     highNibble | lowNibble
300
301 returnValor:
302     lw        ra, BD_FRAME_RA(sp)
303     lw        $fp, BD_FRAME_FP(sp)
304     addu      sp, sp, BD_FSIZE
305     jr        ra
306     .end      byte_decoder
307     .size     byte_decoder, .-byte_decoder
308
309     .align    2
310     .globl    decode
311     .ent      decode
312
313 decode:
314
315 ##### STACK FRAME #####
316 #define DEC_FSIZE 48
317
318 ##### CALLER ARGS #####
319 #define DEC_FRAME_A1 52
320 #define DEC_FRAME_A0 48
321
322 ##### SRA #####
323 # se agrega un word de padding
324 #define DEC_FRAME_RA 40
325 #define DEC_FRAME_FP 36
326 #define DEC_FRAME_GP 32
327
328 ##### LTA #####
329 #define DEC_FRAME_C 28
330 #define DEC_FRAME_CHARACTER2 24
331 #define DEC_FRAME_BYTES 20
332 #define DEC_FRAME_CHARACTER 16
333
334 ##### ABA #####
335 #define DEC_FRAME_ARG3 12
336 #define DEC_FRAME_ARG2 8
337 #define DEC_FRAME_ARG1 4
338 #define DEC_FRAME_ARG0 0
339
340     .frame    $fp, DEC_FSIZE, ra                   # 56
341     subu      sp, sp, DEC_FSIZE
342     .cprestore DEC_FRAME_GP                        # 40

```

```

339      sw      ra, DEC_FRAME_RA(sp)                # 48
340      sw      $fp, DEC_FRAME_FP(sp)                # 44
341      sw      gp, DEC_FRAME_GP(sp)                # 40
342      move    $fp, sp
343
344      sw      a0, DEC_FRAME_A0($fp)                # en a0 tengo infd
345      sw      a1, DEC_FRAME_A1($fp)                # en a1 tengo outfd
346      sw      zero, DEC_FRAME_C($fp)              # c = 0
347      sw      zero, DEC_FRAME_CHARACTER2($fp)      # character2 = 0
348      sw      zero, DEC_FRAME_CHARACTER($fp)       # character = 0
349
350 read_y_loop_decode:
351     addu a1, $fp, DEC_FRAME_CHARACTER            # a1 = &character
352     li      a2, 1                                # cargo a2
353     con 1, para leer un byte
354     li      v0, SYS_read                          # llama a read(a0,a1,
355     a2) -> resultado en v0
356     syscall
357     bltz    v0, error_read_decode                # si v0 < 0 hubo error en lectura
358     bgtz    v0, while_decode                     # entro al while si es mayor
359     a 0 (en este caso si es 0, es eof)
360     b       return_decode
361
362 while_decode:
363     lw      a0, DEC_FRAME_A0($fp)                # a0 = infd
364     addu a1, $fp, DEC_FRAME_CHARACTER2            # a1 = &character2
365     li      a2, 1                                # a2 = 1,
366     para leer un byte
367     li      v0, SYS_read                          # llama a read(a0,a1,
368     a2) -> resultado v0
369     syscall
370     bltz    v0, error_read_decode                # si v0 < 0, hubo error de lectura
371     sw      v0, DEC_FRAME_BYTES($fp)             # bytesLeidos = v0
372
373     lw      a0, DEC_FRAME_CHARACTER($fp)          # a0 = character
374     lw      a1, DEC_FRAME_CHARACTER2($fp)         # a1 = character2
375     la      t9, byte_decoder
376     jal     t9
377     sw      v0, DEC_FRAME_C($fp)                  # c = byte_encoder(a0
378     ,a1)
379     bltz    v0, return_decode                    # si c < 0 -> error: character
380     no hexa
381
382     lw      a0, DEC_FRAME_A1($fp)                # a0 = outfd
383     addu a1, $fp, DEC_FRAME_C                    # a1 = &c
384     li      a2, 1                                # a2 = 1,
385     para escribir un byte
386     li      v0, SYS_write                          # llama a write(outfd
387     ,&c,1) -> resultado v0
388     syscall
389     bltz    v0, error_write_decode                # si v0 < 0, hubo error de escritura
390
391     lw      a0, DEC_FRAME_A0($fp)                # a0 = infd
392     b       read_y_loop_decode
393
394 error_read_decode:
395     sub     v0, zero, 1                            # error1: -1
396     b       return_decode
397
398 error_write_decode:

```

```

391         sub        v0, zero, 2                                # error2: -2
392
393 return_decode:
394         lw         ra, DEC_FRAME_RA(sp)
395         lw         $fp, DEC_FRAME_FP(sp)
396         addu       sp, sp, DEC_FSIZE
397         jr         ra
398         .end       decode
399         .size      decode, .-decode
400
401
402
403 .globl  b16_errmsg
404         .rdata
405         .align     2
406
407 #####                b16_errmg                #####
408
409 b16_errmsg: .word noerror, error1, error2, error3
410
411         .size      b16_errmsg, 16
412         .align     0
413
414 noerror: .asciiiz "Sin Errores\n\000"
415 error1:  .asciiiz "Error al leer el archivo de entrada\n\000"
416 error2:  .asciiiz "Error al escribir el archivo de salida\n\000"
417 error3:  .asciiiz "Contiene caracteres que no pertenecen al codigo Hexa\n\000"

```

4.4. Código assembly MIPS b16.S

```

1  #include<mips/regdef.h>
2  #include<sys/syscall.h>
3
4  #####                ENCODE                #####
5
6  .text
7  .align 2
8  .globl byte_encoder
9  .ent   byte_encoder
10
11 byte_encoder:                                # void byte_encoder (char*
12     valorHexa, unsigned int numInt)
13
14 ##### STACK FRAME #####
15 #define BE_FSIZE 16
16
17 ##### CALLER ARGS #####
18 #define BE_FRAME_A1 20
19 #define BE_FRAME_A0 16
20
21 #####                SRA                #####
22 #define BE_FRAME_FP 12
23 #define BE_FRAME_GP 8
24
25 #####                LTA                #####
26 #define BE_FRAME_LNIBBLE 4
27 #define BE_FRAME_HNIBBLE 0
28
29 .frame $fp, BE_FSIZE, ra                    # (2 SRA + 2 LTA) * 4 bytes; no hay
30     ABA por ser funcion leaf

```

```

28     subu    sp, sp, BE_FSIZE
29
30     sw      $fp, BE_FRAME_FP(sp)           # guardo fp en BE_FRAME_FP + sp
31     sw      gp, BE_FRAME_GP(sp)           # guardo gp en BE_FRAME_GP +
32     sp                                           # llevo fp a la pos del sp
33
34     # Argumento de funcion
35     sw      a0, BE_FRAME_A0($fp)           # a0: char* valorHexa
36     sw      a1, BE_FRAME_A1($fp)           # a1: unsigned int numInt
37
38     andi    t0, a1, 0xf0                    # t0 = highNibble de numInt
39     sra     t0, t0, 4                        # highNibble >> 4
40     sw      t0, BE_FRAME_HNIBBLE($fp)       # guardo la variable local t0 en el
41     stack frame
42
43     andi    t1, a1, 0x0f                    # t1 = lowNibble de numInt
44     sw      t1, BE_FRAME_LNIBBLE($fp)       # guardo la variable local t1 en el
45     stack frame
46
47     lb      t2, vecHexa(t0)                 # t2 = vecHexa[t0]; -> t2:
48     primer caracter hexa
49     sb      t2, 0(a0)                       # a0[0] = t2; es
50     decir: valorHexa[0] = t2
51
52     lw      a0, BE_FRAME_A0($fp)           # tengo a0 nuevamente char* valorHexa
53     (por seguridad)
54
55     lb      t3, vecHexa(t1)                 # t3 = vecHexa[t1]; -> t3:
56     segundo caracter hexa
57     sb      t3, 1(a0)                       # a0[1] = t3; es
58     decir: valorHexa[1] = t3
59
60     lw      $fp, BE_FRAME_FP(sp)           # recupero fp
61     addu    sp, sp, BE_FSIZE               # destruyo el stack frame
62
63     jr      ra
64
65     .end    byte_encoder
66     .size   byte_encoder, .-byte_encoder
67
68
69     .align  2
70     .globl  encode
71     .ent    encode
72
73     encode:                                     # int encode (int
74         infd, int outfd)
75
76     ##### STACK FRAME #####
77     #define ENC_FSIZE 48
78
79     ##### CALLER ARGS #####
80     #define ENC_FRAME_A1 52
81     #define ENC_FRAME_A0 48
82
83     ##### SRA #####
84     # se agrega un word de padding
85     #define ENC_FRAME_RA 40
86     #define ENC_FRAME_FP 36

```

```

80 #define ENC_FRAME_GP 32
81
82 #####          LTA          #####
83 # el siguiente string tiene dos caracteres
84 #define ENC_FRAME_STRING 24
85 #define ENC_FRAME_BYTES 20
86 #define ENC_FRAME_CHARACTER 16
87
88 #####          ABA          #####
89 #define ENC_FRAME_ARG3 12
90 #define ENC_FRAME_ARG2 8
91 #define ENC_FRAME_ARG1 4
92 #define ENC_FRAME_ARG0 0
93
94     .frame    $fp, ENC_FSIZE, ra
95     subu      sp, sp, ENC_FSIZE
96     .cprestore    ENC_FRAME_GP
97     sw        ra, ENC_FRAME_RA(sp)
98     sw        $fp, ENC_FRAME_FP(sp)
99     move      $fp, sp
100
101     sw        a0, ENC_FRAME_A0($fp)
102     sw        a1, ENC_FRAME_A1($fp)
103     sw        zero, ENC_FRAME_CHARACTER($fp)    # caracter = 0
104
105 read_y_loop:
106     addu a1, $fp, ENC_FRAME_CHARACTER            # a1 = &caracter
107     li        a2, 1                                # a2 = 1,
108     para leer un byte
109     li        v0, SYS_read                        # tengo los 3
110     argumentos del read: a0=infd, a1=&caracter, a2=1
111     syscall                                        # en v0 tengo
112     la cantidad de bytes que leo o negativo si hubo error
113     bltz      v0, error_read                      # salto si hubo un error de
114     lectura
115
116     bgtz      v0, while_encode                    # entro al while si es mayor
117     a 0 (en este caso si es 0, es eof)
118     b         return_encode                      # salta en caso de
119     que sea menor o igual a 0
120
121 while_encode:
122     sw        v0, ENC_FRAME_BYTES($fp)            # salvo v0 por llamada de
123     funcion de byte_encoder
124     addu a0, $fp, ENC_FRAME_STRING                # cargo en a0 la direccion $fp +
125     ENC_FRAME_STRING (comienzo del char*)
126     lw        a1, ENC_FRAME_CHARACTER($fp)        # a1 = caracter leído
127     la        t9, byte_encoder                    # carga en t9 donde
128     esta byte_encoder
129     jal       t9                                  #
130     salta a byte_encoder
131
132     lw        a0, ENC_FRAME_A1($fp)                # en a0 tengo outfd
133     addu a1, $fp, ENC_FRAME_STRING                # cargo en a1 la direccion $fp +
134     ENC_FRAME_STRING (comienzo del char*)
135     li        a2, 2                                # cargo en a2
136     el 2, para escribir dos bytes
137     li        v0, SYS_write                        # llamo a write
138     syscall
139     bltz      v0, error_write                      # si es menor a 0, hubo un
140     error de escritura

```



```

128          lw      a0, ENC_FRAME_A0($fp)                # a0 = infd
129          b read_y_loop
130
131
132 error_write:
133         sub      v0, zero, 2                          # error2: -2
134         sw      v0, ENC_FRAME_BYTES($fp)
135         b return_encode
136
137 error_read:
138         sub      v0, zero, 1                          # error1: -1
139         sw      v0, ENC_FRAME_BYTES($fp)
140
141 return_encode:
142         lw      v0, ENC_FRAME_BYTES($fp)
143         lw      ra, ENC_FRAME_RA(sp)
144         lw      $fp, ENC_FRAME_FP(sp)
145         addu    sp, sp, ENC_FSIZE
146
147         jr      ra
148
149         .end     encode
150         .size    encode, .-encode
151
152         .rdata
153         .align   2
154         .size    vecHexa, 16
155 vecHexa:
156         .byte    48                                #'0'
157         .byte    49                                #'1'
158         .byte    50                                #'2'
159         .byte    51                                #'3'
160         .byte    52                                #'4'
161         .byte    53                                #'5'
162         .byte    54                                #'6'
163         .byte    55                                #'7'
164         .byte    56                                #'8'
165         .byte    57                                #'9'
166         .byte    65                                #'A'
167         .byte    66                                #'B'
168         .byte    67                                #'C'
169         .byte    68                                #'D'
170         .byte    69                                #'E'
171         .byte    70                                #'F'
172
173 #####          DECODE #####
174         .text
175         .align   2
176         .globl   correrReferencia
177         .ent     correrReferencia
178
179 correrReferencia:                                # int correrReferencia (int
180         numInt)
181
182 #####          STACK FRAME #####
183 #define CR_FSIZE 8
184
185 #####          CALLER ARGS #####
186 #define CR_FRAME_A0 8
187
188 #####          SRA #####

```

```

188 #define CR_FRAME_FP 4
189 #define CR_FRAME_GP 0
190
191     .frame    $fp, CR_FSIZE, ra
192     subu     sp, sp, CR_FSIZE
193     sw       $fp, CR_FRAME_FP(sp)
194     sw       gp, CR_FRAME_GP(sp)
195     move     $fp, sp
196
197     sw       a0, CR_FRAME_A0($fp)                # En a0 tengo el parametro numInt
198
199     slt t0, a0, 58                                # si numInt < 58 -> t0 = 1
200     sgt t1, a0, 47                                # si numInt > 47 -> t1 = 1
201     and t0, t0, t1                                # si t0 and t1 = 1 -> t0 = 1
202     beqz t0, comparacion2                        # si no esta en ese rango se compara
        en siguiente
203     lw v0, CR_FRAME_A0($fp)                      # se almacena en v0 el a0=numInt
204     sub v0, v0, 48                                # se tiene en v0 = numInt -
        48
205     b return
206
207 comparacion2:
208     slt t0, a0, 71                                # idem al anterior con otro
        rango
209     sgt t1, a0, 64
210     and t0, t0, t1
211     beqz t0, comparacion3
212     lw v0, CR_FRAME_A0($fp)
213     sub v0, v0, 55
214     b return
215
216 comparacion3:
217     slt t0, a0, 103                               # idem al anterior con otro
        rango
218     sgt t1, a0, 96
219     and t0, t0, t1
220     beqz t0, error_caracterNoHexa
221     lw v0, CR_FRAME_A0($fp)
222     sub v0, v0, 87
223     b return
224
225 error_caracterNoHexa:
226     sub     v0, zero, 3                          # error3: -3
227
228 return:
229     lw      $fp, CR_FRAME_FP(sp)
230     addu    sp, sp, CR_FSIZE
231     jr      ra
232     .end    correrReferencia
233     .size   correrReferencia, .-correrReferencia
234
235
236
237     .align  2
238     .globl  byte_decoder
239     .ent    byte_decoder
240
241 byte_decoder:                                     # int byte_decoder (int
        numPri, int numSeg)
242
243 ##### STACK FRAME #####

```

```

244 #define BD_FSIZE 40
245
246 ##### CALLER ARGS #####
247 #define BD_FRAME_A1 44
248 #define BD_FRAME_A0 40
249
250 ##### SRA #####
251 # se agrega un word de padding
252 #define BD_FRAME_RA 32
253 #define BD_FRAME_FP 28
254 #define BD_FRAME_GP 24
255
256 ##### LTA #####
257 #define BD_FRAME_LNIBBLE 20
258 #define BD_FRAME_HNIBBLE 16
259
260 ##### ABA #####
261 #define BD_FRAME_ARG3 12
262 #define BD_FRAME_ARG2 8
263 #define BD_FRAME_ARG1 4
264 #define BD_FRAME_ARG0 0
265
266 .frame $fp, BD_FSIZE, ra
267 subu sp, sp, BD_FSIZE
268 .cprestore BD_FRAME_GP
269
270 sw ra, BD_FRAME_RA(sp)
271 sw $fp, BD_FRAME_FP(sp)
272 move $fp, sp
273
274 sw a0, BD_FRAME_A0($fp) # en a0 tengo numPri
275 sw a1, BD_FRAME_A1($fp) # en a1 tengo numSeg
276
277 la t9, correrReferencia
278 jal t9
279 sw v0, BD_FRAME_HNIBBLE($fp) # highNibble =
    correrReferencia (numPri)
280 bltz v0, returnValor # si es menor a 0 -> error,
    fin decode
281
282 lw a0, BD_FRAME_A1($fp) # cargo en a0 el numSeg
283 la t9, correrReferencia
284 jal t9
285 sw v0, BD_FRAME_LNIBBLE($fp) # lowNibble =
    correrReferencia(numSeg)
286 bltz v0, returnValor # si es menor a 0 -> error,
    fin decode
287
288 lw t0, BD_FRAME_HNIBBLE($fp)
289 sll t0, t0, 4 # t0 =
    highNibble << 4
290 andi t0, t0, 0xf0 # aseguro ceros en
    nibble menos significativo
291 lw t1, BD_FRAME_LNIBBLE($fp) # t1 = lowNibble
292 andi t1, t1, 0xf # aseguro ceros en
    nibble mas significativo
293 or v0, t0, t1 # v0 =
    highNibble | lowNibble
294
295 returnValor:
296 lw ra, BD_FRAME_RA(sp)

```

```

297     lw      $fp, BD_FRAME_FP(sp)
298     addu    sp, sp, BD_FSIZE
299     jr      ra
300     .end    byte_decoder
301     .size    byte_decoder, .-byte_decoder
302
303
304
305     .align  2
306     .globl  decode
307     .ent    decode
308
309 decode:
310
311 ##### STACK FRAME #####
312 #define DEC_FSIZE 48
313
314 ##### CALLER ARGS #####
315 #define DEC_FRAME_A1 52
316 #define DEC_FRAME_A0 48
317
318 #####      SRA      #####
319 # se agrega un word de padding
320 #define DEC_FRAME_RA 40
321 #define DEC_FRAME_FP 36
322 #define DEC_FRAME_GP 32
323
324 #####      LTA      #####
325 #define DEC_FRAME_C 28
326 #define DEC_FRAME_CHARACTER2 24
327 #define DEC_FRAME_BYTES 20
328 #define DEC_FRAME_CHARACTER 16
329
330 #####      ABA      #####
331 #define DEC_FRAME_ARG3 12
332 #define DEC_FRAME_ARG2 8
333 #define DEC_FRAME_ARG1 4
334 #define DEC_FRAME_ARG0 0
335
336     .frame   $fp, DEC_FSIZE, ra                # 56
337     subu    sp, sp, DEC_FSIZE
338     .cprestore DEC_FRAME_GP                    # 40
339     sw      ra, DEC_FRAME_RA(sp)                # 48
340     sw      $fp, DEC_FRAME_FP(sp)                # 44
341     sw      gp, DEC_FRAME_GP(sp)                # 40
342     move    $fp, sp
343
344     sw      a0, DEC_FRAME_A0($fp)                # en a0 tengo infd
345     sw      a1, DEC_FRAME_A1($fp)                # en a1 tengo outfd
346     sw      zero, DEC_FRAME_C($fp)                # c = 0
347     sw      zero, DEC_FRAME_CHARACTER2($fp)      # character2 = 0
348     sw      zero, DEC_FRAME_CHARACTER($fp)        # character = 0
349
350 read_y_loop_decode:
351     addu    a1, $fp, DEC_FRAME_CHARACTER        # a1 = &character
352     li      a2, 1                                # cargo a2
353     con 1, para leer un byte
354     li      v0, SYS_read                        # llama a read(a0,a1,
355     a2) -> resultado en v0
356     syscall
357     bltz    v0, error_read_decode                # si v0 < 0 hubo error en lectura

```

```

356      bgtz      v0, while_decode                # entro al while si es mayor
357      a 0 (en este caso si es 0, es eof)
358      b        return_decode
359
360  while_decode:
361      lw        a0, DEC_FRAME_A0($fp)            # a0 = infd
362      addu      a1, $fp, DEC_FRAME_CHARACTER2    # a1 = &caracter2
363      li        a2, 1                          # a2 = 1,
364      para leer un byte
365      li        v0, SYS_read                    # llama a read(a0,a1,
366      a2) -> resultado v0
367      syscall
368      bltz      v0, error_read_decode            # si v0 < 0, hubo error de lectura
369      sw        v0, DEC_FRAME_BYTES($fp)        # bytesLeidos = v0
370
371      lw        a0, DEC_FRAME_CHARACTER($fp)    # a0 = caracter
372      lw        a1, DEC_FRAME_CHARACTER2($fp)   # a1 = caracter2
373      la        t9, byte_decoder
374      jal       t9
375      sw        v0, DEC_FRAME_C($fp)            # c = byte_encoder(a0
376      ,a1)
377      bltz      v0, return_decode                # si c < 0 -> error: caracter
378      no hexa
379
380      lw        a0, DEC_FRAME_A1($fp)            # a0 = outfd
381      addu      a1, $fp, DEC_FRAME_C            # a1 = &c
382      li        a2, 1                          # a2 = 1,
383      para escribir un byte
384      li        v0, SYS_write                    # llama a write(outfd
385      ,&c,1) -> resultado v0
386      syscall
387      bltz      v0, error_write_decode            # si v0 < 0, hubo error de escritura
388
389      lw        a0, DEC_FRAME_A0($fp)            # a0 = infd
390      b        read_y_loop_decode
391
392  error_read_decode:
393      sub       v0, zero, 1                      # error1: -1
394      b        return_decode
395
396  error_write_decode:
397      sub       v0, zero, 2                      # error2: -2
398
399  return_decode:
400      lw        ra, DEC_FRAME_RA(sp)
401      lw        $fp, DEC_FRAME_FP(sp)
402      addu      sp, sp, DEC_FSIZE
403      jr        ra
404      .end      decode
405      .size     decode, .-decode
406
407  .globl b16_errmsg
408      .rdata
409      .align    2
410
411  #####          b16_errmg          #####
412
413  b16_errmsg: .word noerror, error1, error2, error3

```

```

410
411         .size b16_errmsg, 16
412         .align 0
413
414 noerror: .ascii "Sin Errores\n\000"
415 error1:  .ascii "Error al leer el archivo de entrada\n\000"
416 error2:  .ascii "Error al escribir el archivo de salida\n\000"
417 error3:  .ascii "Contiene caracteres que no pertenecen al codigo Hexa\n\000"

```

5. Stack Frame

A continuación se mostrarán los stack frames respectivos a cada función. Los casilleros en verde corresponden a los parámetros que serán pasados por la función caller, por lo que no son parte del stack frame de la función descrita en esa sección. En los casos que aparezca un '-', se trata de padding y está solamente para mantener el tamaño de cada área como un múltiplo de '8'.

5.1. Stack Frame byte encoder

20	a1
16	a0
12	fp
8	gp
4	LH
0	HL

5.2. Stack Frame encode

52	a1
48	a0
44	-
40	ra
36	fp
32	gp
28	s[1]
24	s[0]
20	byte
16	caracter
12	a3
8	a2
4	a1
0	a0

5.3. Stack Frame correr Referencia

8	a0
4	fp
0	gp

5.4. Stack Frame byte decoder

44	a1
40	a0
36	-
32	ra
28	fp
24	gp
20	LN
16	HL
12	a3
8	a2
4	a1
0	a0

5.5. Stack Frame decode

52	a1
48	a0
44	-
40	ra
36	fp
32	gp
28	c
24	caracter2
20	byte
16	caracter
12	a3
8	a2
4	a1
0	a0

6. Conclusiones

Realizando este trabajo práctico logramos terminar de afianzar los conocimientos adquiridos durante la cursada en lo que respecta a la programación en assembly. Ya que al comienzo de este trabajo se poseía la implementación en C del mismo programa, se pudo compilar el mismo a assembly y compararlo con el código propio. Se comprobó de esta forma que los stack frames creados por el compilador reservaban mucho más espacio del necesario. A su vez, se pudo notar que programando directamente en assembly, el usuario tiene mayor flexibilidad para optimizar el software que en lenguajes de mayor nivel, a costas de la pérdida total de la independencia de la arquitectura a utilizar.