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

El programa del presente informe posee dos partes, una de las cuales fue implementada en C y la otra en Assembly MIPS32.

La primera posee todas las funcionalidades que involucran la interpretación de las opciones ingresadas por línea de comando, la apertura, validación y cierre de los archivos a utilizar y la escritura de los eventuales errores por *stderr*.

Por otro lado, la segunda implementa la codificación y decodificación de los archivos, a través de las etiquetas *encode* y *decode*, respectivamente. Para lograr esto, a su vez, se hizo uso de las funciones auxiliares *byte_encoder* para la primera de estas funcionalidades y *correrReferencia*, *byte_decoder* para la segunda.

2. Stack Frames

A continuación se mostrarán los *Stack Frames* respectivos a cada función implementada en Assembly. 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 descripta 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'.

2.1. byte_encoder

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

2.2. 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

2.3. correrReferencia

8	a0
4	fp
0	gp

2.4. 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

2.5. 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

3. Comandos de compilación

3.1. Makefile

```
all: tp1

b16: b16.h
	gcc -Wall -c b16.S

tp1: b16
	gcc -Wall b16.o -o tp1 tp1.c

clean:
	rm b16.o tp1
```

4. Pruebas realizadas

4.1. Primeras pruebas

```
prueba="Archivo vacio"
touch /tmp/zero.txt
./tp1 -a encode -i /tmp/zero.txt -o /tmp/zero.txt.b16
longitud=‘ls -la /tmp/zero.txt.b16 | awk ’{print $5}’‘
```

```
if [[ $longitud -eq 0 ]] ; then echo "ok: $prueba"; else echo "ERROR: $prueba" ; fi
rm -f /tmp/zero.txt /tmp/zero.txt.b16

prueba="Codificacion de 'M' por entrada estandar"
hexa=`echo -n M | ./tp1`
if [[ "$hexa" == "4D" ]] ; then echo "ok: $prueba"; else echo "ERROR: $prueba" ; fi

prueba="Codificacion de 'Ma' por entrada estandar"
hexa=`echo -n Ma | ./tp1`
if [[ "$hexa" == "4D61" ]] ; then echo "ok: $prueba"; else echo "ERROR: $prueba" ; fi

prueba="Codificacion de 'Man' por entrada estandar"
hexa=`echo -n Man | ./tp1`
if [[ "$hexa" == "4D616E" ]] ; then echo "ok: $prueba"; else echo "ERROR: $prueba" ;
    fi

prueba="Codificacion y decodificacion de 'Man' por entrada estandar"
mensaje=`echo -n Man | ./tp1 | ./tp1 -a decode`
if [[ "$mensaje" == "Man" ]] ; then echo "ok: $prueba"; else echo "ERROR: $prueba" ;
    fi

prueba="Verificacion bit a bit de codificacion y decodificacion de xyz\n"
esperado="0000000    x   y   z  \n
0000004"
resultado=`echo xyz | ./tp1 | ./tp1 -a decode | od -t c`
if [[ "$resultado" == "$esperado" ]] ; then echo "ok: $prueba"; else echo "ERROR:
    $prueba" ; fi
```

4.2. Prueba de archivos aleatorios

```
n=1;
while :; do
 head -c $n </dev/urandom >/tmp/in.bin;
 ./tp1 -a encode -i /tmp/in.bin -o /tmp/out.b16;
 ./tp1 -a decode -i /tmp/out.b16 -o /tmp/out.bin;

        if diff /tmp/in.bin /tmp/out.bin; then :; else
                echo ERROR: $n;
                break;

        fi

echo ok: $n;

        n="`expr $n + 1`";

rm -f /tmp/in.bin /tmp/out.b16 /tmp/out.bin

done
```

5. Código Fuente

5.1. Código fuente C tp1.c

```
#include <stdio.h>
#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, &
                option_index);
66          while ( option != -1 ) {
67
68                  switch (option) {
69                          case 'v':
70                                          printf("66.20-Organizacion de Computadoras TP
                                                Version 0.0\n");
71                                          return 1;
72                                          break;
73                          case 'h':
74                                          imprimirAyuda();
75                                          return 1;
76                                          break;
77                          case 'i':
78                                          (*finput) = fopen(optarg,"r");
79                                          if ((*finput) == NULL) {
80                                                  fprintf(stderr,"Error al abrir el
                                                        archivo input %s\n",optarg);
81                                                  exit(4);
82                                     }
83                                     break;
84                          case 'o':
85                                          (*foutput) = fopen(optarg, "w");
86                                          if ((*foutput) == NULL) {
87                                                  fprintf(stderr,"Error al abrir el
                                                        archivo output %s \n",optarg);
88                                                  exit(5);
89                                          }
90                                                  break;
91                          case 'a':
92                                          comprobarAction(optarg);
93                                          break;
94                          default:
95                                          break;
96                  }
97
98                  option = getopt_long ( argc, argv, "vhi:o:a:", long_options, &
                        option_index);
99      }
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 }
```

5.2. Código de b16.h

```
#ifndef _b16_H_
#define _b16_H_

extern const char* b16_errmsg[];

extern int encode(int infd, int outfd);
extern int decode(int infd, int outfd);

#endif //_b16_H
```

5.3. Código assembly MIPS b16.S

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

#####   ENCODING    #####
  .text
  .align  2
  .globl  byte_encoder
  .ent  byte_encoder

byte_encoder:    # void byte_encoder (char* valorHexa, unsigned int numInt)

########## STACK FRAME ##########
#define BE_FSIZE 16

########## CALLER ARGS ##########
#define BE_FRAME_A1 20
#define BE_FRAME_A0 16

##########     SRA     ##########
#define BE_FRAME_FP 12
#define BE_FRAME_GP 8

##########     LTA     ##########
#define BE_FRAME_LNIBBLE 4
#define BE_FRAME_HNIBBLE 0

##########     ABA     ##########
# no hay por ser funcion leaf

  .frame  $fp, BE_FSIZE, ra   # (2 SRA + 2 LTA) * 4 bytes
  subu  sp, sp, BE_FSIZE

  sw  $fp, BE_FRAME_FP(sp)    # guardo fp en BE_FRAME_FP + sp
  sw  gp, BE_FRAME_GP(sp)     # guardo gp en BE_FRAME_GP + sp
  move  $fp, sp         # llevo fp a la pos del sp

  # Argumento de funcion
  sw  a0, BE_FRAME_A0($fp)    # a0: char* valorHexa
  sw  a1, BE_FRAME_A1($fp)    # a1: unsigned int numInt

  andi  t0, a1, 0xf0      # t0 = highNibble de numInt
  sra t0, t0, 4         # highNibble >> 4
  sw  t0, BE_FRAME_HNIBBLE($fp) # guardo la variable local en el SFrame

  andi  t1, a1, 0x0f      # t1 = lowNibble de numInt
  sw  t1, BE_FRAME_LNIBBLE($fp) # guardo la variable local t1 en el stack frame
```

```
47
48      lb  t2, vecHexa(t0)       # t2 = vecHexa[t0]; -> t2: primer caracter hexa
49      sb  t2, 0(a0)         # a0[0] = t2; es decir: valorHexa[0] = t2
50
51      lw  a0, BE_FRAME_A0($fp)    # tengo a0 nuevamente char* valorHexa (por seguridad)
52
53      lb  t3, vecHexa(t1)       # t3 = vecHexa[t1]; -> t3: segundo caracter hexa
54      sb  t3, 1(a0)         # a0[1] = t3; es decir: valorHexa[1] = t3
55
56      lw  $fp, BE_FRAME_FP(sp)    # recupero fp
57      addu  sp, sp, BE_FSIZE    # destruyo el stack frame
58
59      jr  ra
60
61      .end  byte_encoder
62      .size byte_encoder, .-byte_encoder
63
64
65
66      .align  2
67      .globl  encode
68      .ent  encode
69
70  encode:               # int encode (int infd, int outfd)
71
72  ########## STACK FRAME ##########
73  #define ENC_FSIZE 48
74
75  ########## CALLER ARGS ##########
76  #define ENC_FRAME_A1 52
77  #define ENC_FRAME_A0 48
78
79  ##########     SRA     ##########
80  # se agrega un word de padding
81  #define ENC_FRAME_RA 40
82  #define ENC_FRAME_FP 36
83  #define ENC_FRAME_GP 32
84
85  ##########     LTA     ##########
86  # el siguiente string tiene dos caracteres
87  #define ENC_FRAME_STRING 24
88  #define ENC_FRAME_BYTES 20
89  #define ENC_FRAME_CARACTER 16
90
91  ##########     ABA     ##########
92  #define ENC_FRAME_ARG3 12
93  #define ENC_FRAME_ARG2 8
94  #define ENC_FRAME_ARG1 4
95  #define ENC_FRAME_ARG0 0
96
97      .frame  $fp, ENC_FSIZE, ra
98      subu  sp, sp, ENC_FSIZE
99      .cprestore  ENC_FRAME_GP
100     sw  ra, ENC_FRAME_RA(sp)
101     sw  $fp, ENC_FRAME_FP(sp)
102     move  $fp, sp
103
104     sw  a0, ENC_FRAME_A0($fp)
105     sw  a1, ENC_FRAME_A1($fp)
106     sw  zero, ENC_FRAME_CARACTER($fp) # caracter = 0
107
```

```
read_y_loop:
  addu a1,$fp,ENC_FRAME_CARACTER    # a1 = &caracter
  li  a2,1              # a2 = 1, para leer un byte
  li  v0, SYS_read          # read: a0=infd, a1=&caracter, a2=1
  syscall               # en v0 = cantidad de bytes que leo o negativo si hubo error
  bltz  v0, error_read        # salto si hubo un error de lectura

  bgtz  v0, while_encode      # entro al while si es > 0 (si es 0, es eof)
  b return_encode         # salta en caso de que sea menor o igual a 0

while_encode:
  sw  v0, ENC_FRAME_BYTES($fp)    # salvo v0 por llamada de funcion de byte_encoder
  addu a0, $fp, ENC_FRAME_STRING    # a0 = $fp + ENC_FRAME_STRING (inicio del char*)
  lw  a1, ENC_FRAME_CARACTER($fp)   # a1 = caracter leido
  la  t9, byte_encoder        # carga en t9 donde esta byte_encoder
  jal t9                # salta a byte_encoder

  lw  a0, ENC_FRAME_A1($fp)     # en a0 tengo outfd
  addu a1, $fp, ENC_FRAME_STRING    # a1 = $fp + ENC_FRAME_STRING (inicio del char*)
  li  a2, 2             # cargo en a2 el 2, para escribir dos bytes
  li  v0, SYS_write         # llamo a write
  syscall
  bltz  v0, error_write       # si es menor a 0, hubo un error de escritura

  lw  a0, ENC_FRAME_A0($fp)     # a0 = infd
  b read_y_loop

error_write:
  sub v0, zero, 2           # error2: -2
  sw  v0, ENC_FRAME_BYTES($fp)
  b return_encode

error_read:
  sub v0, zero, 1           # error1: -1
  sw  v0, ENC_FRAME_BYTES($fp)

return_encode:
  lw  v0, ENC_FRAME_BYTES($fp)
  lw  ra, ENC_FRAME_RA(sp)
  lw  $fp, ENC_FRAME_FP(sp)
  addu  sp,sp, ENC_FSIZE

  jr  ra

  .end  encode
  .size encode, .-encode

  .rdata
  .align  2
  .size vecHexa, 16
vecHexa:
  .byte 48        #'0'
  .byte 49        #'1'
  .byte 50        #'2'
  .byte 51        #'3'
  .byte 52        #'4'
  .byte 53        #'5'
  .byte 54        #'6'
  .byte 55        #'7'
  .byte 56        #'8'
  .byte 57        #'9'
```

```
    .byte 65        #'A'
    .byte 66        #'B'
    .byte 67        #'C'
    .byte 68        #'D'
    .byte 69        #'E'
    .byte 70        #'F'

#####   DECODING    #####
    .text
    .align  2
    .globl  correrReferencia
    .ent  correrReferencia

correrReferencia:         # int correrReferencia  (int numInt)

########## STACK FRAME ##########
#define CR_FSIZE 8

########## CALLER ARGS ##########
#define CR_FRAME_A0 8

##########     SRA     ##########
#define CR_FRAME_FP 4
#define CR_FRAME_GP 0

    .frame  $fp, CR_FSIZE, ra
    subu  sp, sp, CR_FSIZE
    sw  $fp, CR_FRAME_FP(sp)
    sw  gp, CR_FRAME_GP(sp)
    move  $fp,sp

    sw  a0, CR_FRAME_A0($fp)    # En a0 tengo el parametro numInt

    slt t0, a0, 58          # si numInt < 58 -> t0 = 1
    sgt t1, a0, 47          # si numInt > 47 -> t1 = 1
    and t0, t0, t1          # si t0 and t1 = 1 -> t0 = 1
    beqz t0, comparacion2     # si no esta en ese rango se compara en siguiente
    lw v0, CR_FRAME_A0($fp)     # se almacena en v0 el a0=numInt
    sub v0, v0, 48          # se tiene en v0 = numInt - 48
    b return

comparacion2:
    slt t0, a0, 71          # idem al anterior con otro rango
    sgt t1, a0, 64
    and t0, t0, t1
    beqz t0, comparacion3
    lw v0, CR_FRAME_A0($fp)
    sub v0, v0, 55
    b return

comparacion3:
    slt t0, a0, 103         # idem al anterior con otro rango
    sgt t1, a0, 96
    and t0, t0, t1
    beqz t0,error_caracterNoHexa
    lw v0, CR_FRAME_A0($fp)
    sub v0, v0, 87
    b return

error_caracterNoHexa:
    sub v0, zero, 3         # error3: -3
```

```
230
231 return:
232   lw  $fp, CR_FRAME_FP(sp)
233   addu  sp, sp, CR_FSIZE
234   jr  ra
235   .end  correrReferencia
236   .size correrReferencia, .-correrReferencia
237
238
239
240   .align  2
241   .globl  byte_decoder
242   .ent  byte_decoder
243
244 byte_decoder:           # int byte_decoder (int numPri, int numSeg)
245
246 ########## STACK FRAME ##########
247 #define BD_FSIZE 40
248
249 ########## CALLER ARGS ##########
250 #define BD_FRAME_A1 44
251 #define BD_FRAME_A0 40
252
253 ##########     SRA     ##########
254 # se agrega un word de padding
255 #define BD_FRAME_RA 32
256 #define BD_FRAME_FP 28
257 #define BD_FRAME_GP 24
258
259 ##########     LTA     ##########
260 #define BD_FRAME_LNIBBLE 20
261 #define BD_FRAME_HNIBBLE 16
262
263 ##########     ABA     ##########
264 #define BD_FRAME_ARG3 12
265 #define BD_FRAME_ARG2 8
266 #define BD_FRAME_ARG1 4
267 #define BD_FRAME_ARG0 0
268
269   .frame  $fp, BD_FSIZE, ra
270   subu  sp, sp, BD_FSIZE
271   .cprestore BD_FRAME_GP
272
273   sw  ra, BD_FRAME_RA(sp)
274   sw  $fp, BD_FRAME_FP(sp)
275   move  $fp, sp
276
277   sw  a0, BD_FRAME_A0($fp)      # en a0 tengo numPri
278   sw  a1, BD_FRAME_A1($fp)      # en a1 tengo numSeg
279
280   la  t9, correrReferencia
281   jal t9
282   sw  v0, BD_FRAME_HNIBBLE($fp)     # highNibble = correrReferencia (numPri)
283   bltz  v0, returnValor       # si es menor a 0 -> error, fin decode
284
285   lw  a0, BD_FRAME_A1($fp)      # cargo en a0 el numSeg
286   la  t9, correrReferencia
287   jal t9
288   sw  v0, BD_FRAME_LNIBBLE($fp)   # lowNibble = correrReferencia(numSeg)
289   bltz  v0, returnValor       # si es menor a 0 -> error, fin decode
290
```

```
291     lw    t0, BD_FRAME_HNIBBLE($fp)
292     sll   t0, t0, 4         # t0 = highNibble << 4
293     andi  t0,t0,0xf0          # aseguro ceros en nibble menos significativo
294     lw    t1, BD_FRAME_LNIBBLE($fp) # t1 = lowNibble
295     andi  t1, t1, 0xf         # aseguro ceros en nibble mas significativo
296     or    v0, t0, t1          # v0 = highNibble | lowNibble
297
298   returnValor:
299     lw  ra, BD_FRAME_RA(sp)
300     lw  $fp, BD_FRAME_FP(sp)
301     addu  sp, sp, BD_FSIZE
302     jr  ra
303     .end  byte_decoder
304     .size byte_decoder, .-byte_decoder
305
306
307
308     .align  2
309     .globl  decode
310     .ent  decode
311
312   decode:
313
314   ########## STACK FRAME ##########
315   #define DEC_FSIZE 48
316
317   ########## CALLER ARGS ##########
318   #define DEC_FRAME_A1 52
319   #define DEC_FRAME_A0 48
320
321   ##########     SRA     ##########
322   # se agrega un word de padding
323   #define DEC_FRAME_RA 40
324   #define DEC_FRAME_FP 36
325   #define DEC_FRAME_GP 32
326
327   ##########     LTA     ##########
328   #define DEC_FRAME_C 28
329   #define DEC_FRAME_CARACTER2 24
330   #define DEC_FRAME_BYTES 20
331   #define DEC_FRAME_CARACTER 16
332
333   ##########     ABA     ##########
334   #define DEC_FRAME_ARG3 12
335   #define DEC_FRAME_ARG2 8
336   #define DEC_FRAME_ARG1 4
337   #define DEC_FRAME_ARG0 0
338
339     .frame  $fp, DEC_FSIZE, ra      # 56
340     subu  sp, sp, DEC_FSIZE
341     .cprestore DEC_FRAME_GP       # 40
342     sw  ra, DEC_FRAME_RA(sp)      # 48
343     sw  $fp, DEC_FRAME_FP(sp)     # 44
344     sw  gp, DEC_FRAME_GP(sp)      # 40
345     move  $fp, sp
346
347     sw  a0, DEC_FRAME_A0($fp)     # en a0 tengo infd
348     sw  a1, DEC_FRAME_A1($fp)     # en a1 tengo outfd
349     sw  zero, DEC_FRAME_C($fp)      # c = 0
350     sw  zero, DEC_FRAME_CARACTER2($fp)  # caracter2 = 0
351     sw  zero, DEC_FRAME_CARACTER($fp) # caracter = 0
```

```
352
353 read_y_loop_decode:
354   addu a1, $fp, DEC_FRAME_CARACTER  # a1 = &caracter
355   li  a2, 1             # cargo a2 con 1, para leer un byte
356   li  v0, SYS_read          # llama a read(a0,a1,a2) -> resultado en v0
357   syscall
358   bltz  v0, error_read_decode   # si v0 < 0 hubo error en lectura
359
360   bgtz  v0, while_decode      # entro al while si es > 0 (si es 0, es eof)
361   b return_decode
362
363 while_decode:
364   lw  a0, DEC_FRAME_A0($fp)     # a0 = infd
365   addu  a1, $fp, DEC_FRAME_CARACTER2  # a1 = &caracter2
366   li  a2, 1             # a2 = 1, para leer un byte
367   li  v0, SYS_read          # llama a read(a0,a1,a2) -> resultado v0
368   syscall
369   bltz  v0, error_read_decode   # si v0 < 0, hubo error de lectura
370   sw  v0, DEC_FRAME_BYTES($fp)    # bytesLeidos = v0
371
372   lw  a0, DEC_FRAME_CARACTER($fp)   # a0 = caracter
373   lw  a1, DEC_FRAME_CARACTER2($fp)  # a1 = caracter2
374   la  t9, byte_decoder
375   jal t9
376   sw  v0, DEC_FRAME_C($fp)      # c = byte_encoder(a0,a1)
377   bltz  v0, return_decode     # si c < 0 -> error: caracter no hexa
378
379   lw  a0, DEC_FRAME_A1($fp)     # a0 = outfd
380   addu a1, $fp, DEC_FRAME_C     # a1 = &c
381   li  a2, 1             # a2 = 1, para escribir un byte
382   li  v0, SYS_write         # llama a write(outfd,&c,1) -> resultado v0
383   syscall
384   bltz  v0, error_write_decode    # si v0 < 0, hubo error de escritura
385
386   lw  a0, DEC_FRAME_A0($fp)     # a0 = infd
387   b read_y_loop_decode
388
389 error_read_decode:
390   sub v0, zero, 1           # error1: -1
391   b return_decode
392
393 error_write_decode:
394   sub v0, zero, 2           # error2: -2
395
396 return_decode:
397   lw  ra, DEC_FRAME_RA(sp)
398   lw  $fp, DEC_FRAME_FP(sp)
399   addu  sp, sp, DEC_FSIZE
400   jr  ra
401   .end  decode
402   .size decode, .-decode
403
404
405
406 .globl  b16_errmsg
407   .rdata
408   .align  2
409
410 #########   b16_errmg   #########
411
412 b16_errmsg: .word noerror, error1, error2, error3
```

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

6. Conclusiones

Realizando este trabajo práctico se logró 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 cuestas de la pérdida total de la independencia de la arquitectura a utilizar.