Facultad de Ingeniería
Universidad de Buenos Aires

Asignatura: Organización de Computadoras

TP0: Infraestructura básica

Alumnos:

Nombre	Padrón
Camila Serra	97422
Cynthia Marlene Gamarra Silva	92702
Gonzalo Almendro	80698

21 de octubre de 2020

Índice

Índice **1**

1. Enunciado del trabajo práctico **2**

2. Diseño e implementación **5**

3. Parámetros del programa **8**

4. Compilación del programa **8**

5. Pruebas realizadas **10**
5.1. Pruebas con archivo bash test-automatic.sh 10
5.1.1. Generales 11

6. Conclusiones **12**

Referencias **12**

A. Código fuente **13**
A.1. Código en C 13
A.1.1. main.c 13
A.1.2. constants.h 14
A.1.3. command.h 14
A.1.4. command.c 15
A.1.5. file.h 17
A.1.6. file.c 18
A.1.7. encode.h 19
A.1.8. encode.c 19
A.2. Código MIPS32 22
A.2.1. tp0.S 22

1. Enunciado del trabajo práctico

66:20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica

1. Objetivos

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa (y su correspondiente documentación) que resuelva el problema piloto que presentaremos más abajo.

2. Alcance

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

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 7), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4. Recursos

Usaremos el programa QEMU [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo Debian [2].

5. Base 64

La codificación base 64 [3] se creó para poder transmitir archivos binarios en medios que sólo admitían texto: 64 es la mayor potencia de 2 que se podía

representar sólo con caracteres ASCII imprimibles. Básicamente se tiene una tabla de conversión de combinaciones de 6 bits a caracteres ASCII, se 'corta' el archivo en secuencias de 6 bits y se transmiten los caracteres correspondientes a esas secuencias. Cada tres bytes de la secuencia original se generan cuatro caracteres base64; cuando la cantidad de bytes original no es múltiplo de tres, se adicionan caracteres '=' al final en cantidad necesaria.

6. Programa

El programa a escribir, en lenguaje C, recibirá un nombre de archivo (o el archivo mismo por `stdin`) y devolverá ese mismo archivo codificado en `base64` [3], o bien decodificado desde base64 si se utiliza la opción `-d`.

6.1. Ejemplos

Primero, usamos la opción `-h` para ver el mensaje de ayuda:

```
$ tp0 -h
Usage:
  tp0 -h
  tp0 -V
  tp0 [options]
Options:
  -V, --version     Print version and quit.
  -h, --help        Print this information.
  -o, --output      Path to output file.
  -i, --input       Path to input file.
  -d, --decode      Decode a base64-encoded file.
Examples:

  tp0 -i input.txt -o output.txt
```

Luego, lo usamos para codificar un pequeño fragmento de texto:

```
$ cat quijote.txt
En un lugar de La Mancha de cuyo nombre no quiero acordarme
$ ./tp0 -i quijote.txt -o qb64
$ cat qb64
RW4gdW4gbHVnYXIgZGUgTGEgTWFuY2hhIGRlIGN1eW8gbm9tYnJlIG5vIHF1aWVybyBhY29yZGFy
bWUK
```

Otra manera de ejecutarlo es a través de `stdin` y/o `stdout`:

```
cat quijote.txt | ./tp0
RW4gdW4gbHVnYXIgZGUgTGEgTWFuY2hhIGRlIGN1eW8gbm9tYnJlIG5vIHF1aWVybyBhY29yZGFy
bWUK
```

También se puede usar para decodificar:

```
$ ./tp0 -d -i qb64 -o texto
$ cat texto
En un lugar de La Mancha de cuyo nombre no quiero acordarme
```

7. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa;
- Las corridas de prueba, con los comentarios pertinentes;
- El código fuente, en lenguaje C;
- El código MIPS32 generado por el compilador;
- Este enunciado.

8. Fecha de entrega

La fecha de entrega es el jueves 22 de Octubre de 2020.

Referencias

[1] QEMU, `https://www.qemu.org/`.

[2] Debian, the Universal Operating System, `https://www.debian.org/`.

[3] Codificación base64, `https://es.wikipedia.org/wiki/Base64`

2. Diseño e implementación

El programa que contiene la lógica de codificador y decodificador se encuentra en el archivo ***encode.c***. El codificador transforma expresiones con caracteres ASCII en base64, mientras que el decodificador hace el proceso inverso. Ambas funciones se implementan con diversos ciclos y recorridos, así como utilización de instrucciones aritmético lógicas (sumas, &, shifts, etc) que relacionan la codificación base 64 con ASCII.

Se creó un achivo ***file.c*** para el manejo de los archivos a codificar/decodificar. Además se creó otro archivo llamado ***command.c*** cuya función es tomar los parámetros del programa y realizar las acciones pertinentes a la misma.

Específicamente el programa se estructura en los siguientes pasos:

- Análisis de las parámetros de la línea de comandos: se analizan las opciones ingresadas por la línea de comandos utilizando la función `getopt_long()`, la cual puede procesar cada opción que es leída de forma simplificada. Se extraen los argumentos de cada opción y se los guarda dentro de una estructura para su posterior acceso del tipo `command_options_st` cuya definición es

```
1     typedef struct {
2         File input_file;
3         File output_file;
4         const char *input_path;
5         const char *output_path;
6         char encode_option;
7         char error_condition;
8     } command_options_st;
9
```

  En caso de que no se encuentre alguna opción, se muestra el mensaje de ayuda al usuario para que identifique el prototipo de cómo debe ejecutar el programa.

- Validación de opciones: a medida que se va analizando cada opción de la línea de comandos, se valida cada una de ellas. Si se ingresó algún parámetro no válido para el programa o si se encuentró un error se lo informa al usuario por pantalla y se aborta la ejecución del programa. Se utiliza para ello se la función `show_error()` cuyo resultado es:

```
1     printf("Options:\n");
2     printf("\t-V,\t--version\tPrint version and quit.\n");
3     printf("\t-h,\t--help\t\tPrint this information.\n");
4     printf("\t-i,\t--input\t\tLocation of the input file.\n");
5     printf("\t-o,\t--output\tLocation of the output file.\n");
6     printf("\t-d,\t--decode\tDecode a base64-encoded file (default is encode).\n");
7     printf("Examples:\n");
8     printf("\ttp0 -i ~/input -o ~/output\n");
9     printf("\ttp0 --decode\n");
10
```

  Para el caso en que no hubo errores a la validación de los argumentos se procede a llamar a las funciones correspondientes a:

  - `Mensaje de ayuda:` Función `show_help()`
  - `Mensaje de versión:` Función `show_version()`
  - `Input file :` Función `set_input_file()` que guarda la entrada del archivo donde será leído el texto.
  - `Output file:` Función set_output_file() que guarda la entrada del archivo de salida donde se escribirá el texto codificado.
  - `Acción del programa a ejecutar:` Si se recibe la opción de decodificar se llama a la función set_decode() que setea la variable $opt->encode\_option$ indicando que es DECODE (está seteado en default ENCODE)

- Encode/Decode: una vez que se procesó correctamente las opciones de la línea de comandos se procede a llamar a la función `process()` que ejecutará la operación de ENCODE o DECODE dependiendo del argumento pasado en la línea de comandos. La operación de DECODE se ejecuta el siguiente código:

```
    while (!file_eof(&opt->input_file) && !has_errors(opt)) {
            unsigned int read = file_read(&opt->input_file, buf_encoded, 4);
            if (read > 0) {
                if (read != 4) {
                    set_error(opt, INVALID_FILE_LENGTH);
                    show_error(opt);
                } else {
                    ++count;
                    if (count == 18) { // 19 * 4 =  76 bytes
                        unsigned char aux;
                        file_read(&opt->input_file, &aux, 1);
                        count = 0;
                    }

                    if (Decode(buf_encoded, buf_decoded)) {
                        char aux = 0;
                        if (buf_encoded[2] == '=') {
                            ++aux;
                        }
                        if (buf_encoded[3] == '=') {
                            ++aux;
                        }

                        file_write(&opt->output_file, buf_decoded, 3 - aux);
                    } else {
                        set_error(opt, INVALID_CHARS);
                        show_error(opt);
                        unsigned int i;
                        for (i = 0; i < 4; ++i) {
                            fprintf(stderr, "%c", buf_encoded[i]);
                        }
                    }
                }

            }
        }
```

Básicamente lo que se realiza es la lectura del archivo para procesarlo teniendo en cuenta la longitud del archivo a procesar y el padding a decodificar. La función `Decode()` retorna un buffer de 3 caracteres con el decode de 4 caracteres en base64. Se debe cumplir:

Pre: el buffer input contiene 4 caracteres. El buffer output tiene por lo menos 3 caracteres

Post: retorna un buffer de 3 byte con los caracteres en ASCII. retorna 0 si error 1 si ok

La operación de ENCODE se ejecuta el siguiente código:

```
        while (!file_eof(&opt->input_file)) {
            memset(buf_decoded, 0, 3);
            unsigned int read = file_read(&opt->input_file, buf_decoded, 3);
            if (read > 0) {
                Encode(buf_decoded, read, buf_encoded);
                file_write(&opt->output_file, buf_encoded, 4);
                ++count;
                if (count == 18) // 19 * 4 =  76 bytes
                {
                    file_write(&opt->output_file, (unsigned char *) "\n", 1);
                    count = 0;
                }
            }
        }
```

Básicamente lo que se realiza es la lectura del archivo para procesarlo en la función `Encode()`e n donde recibe 3 caracteres en buffer y los convierte en 4 caracteres codificados en output. Se debe cumplir:

Pre: el buffer contiene length caracteres (1 a 3) y todos los caracteres son validos

Post: retorna un buffer de 4 byte con los caracteres en base64.

3. Parámetros del programa

Se detallan a continuación los parámetros del programa

- -h: Visualiza la ayuda del programa, en la que se indican los parámetros y sus objetivos.
- -V: Indica la versión del programa.
- -i: Archivo de entrada del programa.
- -o: Archivo de salida del programa.
- -d: Indica decodificación a base64.

Se indica a continuación detalles respecto a los parámetros:

- Si no se explicitan -i y -o, se utilizarán stdin y stdout, respectivamente.
- -V es una opción "show and quit". Si se explicita este parámetro, sólo se imprimirá la versión, aunque el resto de los parámetros se hayan explicitado.
- -h también es de tipo "show and quit" y se comporta de forma similar a -V.
- en caso de que se use la entrada estándar (con comando echo texto | ./tp0 -a encode) y luego se especifique un archivo de salida con -i, prevalecerá el establecido por parámetro.

4. Compilación del programa

Para obtener un ejecutable, se creó un archivo `makefile` cuyo contenido es:

```
CC = gcc
CFLAGS = -o0 -g -Wall -Werror -pedantic -std=c99

OBJECTS = command.o encode.o file.o
EXEC = tp0

all: $(EXEC)

command.o: command.c command.h
	$(CC) $(CFLAGS) -c command.c -o command.o

encode.o: encode.c encode.h
	$(CC) $(CFLAGS) -c encode.c -o encode.o

file.o: file.c file.h
	$(CC) $(CFLAGS) -c file.c -o file.o

$(EXEC): $(OBJECTS)
	$(CC) $(CFLAGS) $(OBJECTS) main.c -o $(EXEC) -lm


run: $(EXEC)
	./$(EXEC)


clean:
	rm -f *.o $(EXEC)

```

Para ejecutarlo, posicionarse en el directorio `src/` y ejecutar el siguiente comando:

```
$ make
```

Para proceder a la ejecución del programa, se debe llamar a:

```
$ ./tp0
```

seguido de los parámetros que se desee modificar, los cuales se indicaron en la sección 1.2.

En caso de ser entrada estándar (stdin) se podrá ejecutar de la siguiente forma:

```
$ echo texto | ./tp0 -a encode
```

También en este caso, se indican a continuación los parámetros a usar.

Para el caso de hacerlo en el programa QEMU que provee la cátedra, utilizando la máquina virtual que contiene el sistema operativo Debian, no se utilizó el archivo Makefile, la compilación se realizó con la herramienta `gcc`.

5. Pruebas realizadas

5.1. Pruebas con archivo bash test-automatic.sh

Para la ejecución del siguiente script se debe copiar, se debe ubicar el archivo ejecutable compilado dentro de la carpeta de test para que se ejecuten correctamente las pruebas. El script sería:

```
#!/bin/bash

echo "############################################################"
echo "#################### Tests Automaticos #####################"
echo "############################################################"
echo ""

echo "#-----------------------------------------------------------"
echo "#----------# COMIENZA Test 01 - Archivo Vacio #-----------"

touch ./test-out/zero.txt
./tp0 -i ./test-out/zero.txt -o ./test-out/zero-encoded.txt
ls -l ./test-out/zero-encoded.txt

if diff -b ./test-out/zero.txt ./test-out/zero-encoded.txt; then
        echo "[OK]";
else
        echo "[ERROR]";
fi

echo "#----------# TERMINA  Test 01 - Archivo Vacio #-----------"
echo "#-----------------------------------------------------------"
echo ""

echo "#----------# COMIENZA Test 02 - Stdin Input   #-----------"

echo -n Test02 > ./test-out/stdin-test.txt
ls -l ./test-out/stdin-test.txt

cat ./test-out/stdin-test.txt | ./tp0 -o ./test-out/stdin-encoded.txt
ls -l ./test-out/stdin-encoded.txt

cat ./test-out/stdin-encoded.txt | ./tp0 --decode -o ./test-out/stdin-decoded.txt
ls -l ./test-out/stdin-decoded.txt

if diff -b ./test-out/stdin-test.txt ./test-out/stdin-decoded.txt; then
        echo "[OK]";
else
        echo "[ERROR]";
fi


echo "#----------# TERMINA  Test 02 - Stdin Input   #-----------"
echo "#-----------------------------------------------------------"
echo ""


echo "#----------# COMIENZA Test 03 - Stdout Output  #-----------"

echo -n Test03 > ./test-out/stdout-test.txt
ls -l ./test-out/stdout-test.txt

./tp0 -i ./test-out/stdout-test.txt > ./test-out/stdout-encoded.txt
ls -l ./test-out/stdout-encoded.txt

./tp0 --decode -i ./test-out/stdout-encoded.txt > ./test-out/stdout-decoded.txt
ls -l ./test-out/stdout-decoded.txt

if diff -b ./test-out/stdout-test.txt ./test-out/stdout-decoded.txt; then
        echo "[OK]";
```

```
else
        echo "[ERROR]";
fi


echo "#----------# TERMINA  Test 02 - Stdout Output  #-----------"
echo "#------------------------------------------------------------"
echo ""



echo "#----------# COMIENZA Test 04 - Stdin Stdout  #-----------"

echo -n Test04 > ./test-out/stdin-stdout-test.txt
ls -l ./test-out/stdin-stdout-test.txt

cat ./test-out/stdin-stdout-test.txt | ./tp0 > ./test-out/stdin-stdout-encoded.txt
ls -l ./test-out/stdin-stdout-encoded.txt

cat ./test-out/stdin-stdout-encoded.txt | ./tp0 --decode > ./test-out/stdin-stdout-decoded.txt
ls -l ./test-out/stdin-stdout-decoded.txt

if diff -b ./test-out/stdin-stdout-test.txt ./test-out/stdin-stdout-decoded.txt; then
        echo "[OK]";
else
        echo "[ERROR]";
fi


echo "#----------# TERMINA  Test 04 - Stdin Stdout  #-----------"
echo "#------------------------------------------------------------"
echo ""
```

El cual no presenta errores en ninguna de las corridas llevadas a cabo.

Todas las pruebas que se presentan a continuación, están codificadas en los archivos de prueba ***.txt de forma que puedan ejecutarse y comprobar los resultados obtenidos.

Se indicaran a continuación lo siguiente: comandos para ejecutarlas, líneas de código que las componen y resultado esperado.

5.1.1. Generales

- Mensaje de ayuda

```
$$ ./tp0 -h   o ./tp0  --help
Options:
        -V,     --version       Print version and quit.
        -h,     --help          Print this information.
        -i,     --input         Location of the input file.
        -o,     --output        Location of the output file.
        -d,     --decode        Decode a base64-encoded file (default is encode).
Examples:
        tp0 -i ~/input -o ~/output
        tp0 --decode
```

- Mensaje de version

```
$ ./tp0 -V   o ./tp0  --version
Version: 1.0

```

6. Conclusiones

El trabajo práctico nos permitió desarrollar una API para procesar archivos transformándolos a su equivalente `base64` en lenguaje C. Además, nos permitió familiarizarnos con la arquitectura MIPS32 y el consecuente análisis y desarrollo de código assembler MIPS utilizando el programa QEMU para simular el entorno de desarrollo.

Referencias

[1] Base64 (Wikipedia) http://en.wikipedia.org/wiki/Base64

[2] Script QEMU, Script provisto por la práctica/

[3] QEMU, https://www.qemu.org/

[4] Kernighan, B. W. - Ritchie, D. M. - *C Programming Language* - 2nd edition - Prentice Hall - 1988.

[5] *GNU Make* - https://www.gnu.org/software/make/

[6] *Valgrind* - http://valgrind.org/

A. Código fuente

A.1. Código en C

A.1.1. main.c

```
#include <stddef.h>
#include <getopt.h>
#include "constants.h"
#include "command.h"

int main(int argc, char **argv) {
    struct option arg_long[] = {
            {"input",   required_argument, NULL, 'i'},
            {"output",  required_argument, NULL, 'o'},
            {"decode",  no_argument, NULL, 'd'},
            {"help",    no_argument, NULL, 'h'},
            {"version", no_argument, NULL, 'V'},
    };

    char arg_opt_str[] = "i:o:dhV";
    int arg_opt;
    int arg_opt_idx = 0;

    command_options_st cmd_options;
    // Default Values: encode, stdin as input, stdout as output, stderr as error output
    command_create(&cmd_options);

    char should_process = TRUE;
    while ((arg_opt = getopt_long(argc, argv, arg_opt_str, arg_long, &arg_opt_idx)) != -1
           &&
           should_process) {

        switch (arg_opt) {
            case 'i': {
                // Set Input File
                set_input_file(&cmd_options, optarg);
            }
                break;
            case 'o': {
                // Set Output File
                set_output_file(&cmd_options, optarg);
            }
                break;
            case 'h': {
                // Show Options
                show_help();
                should_process = FALSE;
            }
                break;
            case 'V': {
                // Show Version
                show_version();
                should_process = FALSE;
            }
                break;
            case 'd': {
                // Set Decode option
                set_decode(&cmd_options);
            }
                break;
            default: {
                // Set Error Condition = INVALID_ARGUMENT
                // [Gonzalo: We analize only valid arguments values here: i, o, h, V, d]
                set_error(&cmd_options, INVALID_ARGUMENT);
            }
                break;
        }
```

```
    }

    // Help or Version arguments, no processing
    if (!should_process) {
        return 0;
    }

    // Processs Encode/Decode if no errors found, otherwise show error message
    if (has_errors(&cmd_options)) {
        show_error(&cmd_options);
        return 1;
    }

    return process(&cmd_options);
}
```

A.1.2. constants.h

```
#ifndef __TP0_CONSTANTS_H__
#define __TP0_CONSTANTS_H__

// BOOLEAN VALUES
#define TRUE 1
#define FALSE 0

// RESULT VALUES
#define OK 0
#define ERROR 1

// PROCESS MODES
#define ENCODE 1
#define DECODE 0

// ERROR CODES
#define INVALID_ARGUMENT -1
#define NO_ARGUMENTS -2
#define EMPTY_INPUT -3
#define INVALID_FILE_LENGTH -4
#define INVALID_CHARS -5


#endif
```

A.1.3. command.h

```
#ifndef __COMMAND_H__
#define __COMMAND_H__

#include "file.h"
#include "constants.h"

typedef struct {
    File input_file;
    File output_file;
    const char *input_path;
    const char *output_path;
    char encode_option;
    char error_condition;
} command_options_st;

void command_create(command_options_st *opt);

void set_input_file(command_options_st *opt, const char *input);

void set_output_file(command_options_st *opt, const char *output);

void set_encode(command_options_st *opt);
```

```
23
24 void set_decode(command_options_st *opt);
25
26 void set_error(command_options_st *opt, char error_condition);
27
28 int has_errors(command_options_st *opt);
29
30 void show_error(command_options_st *opt);
31
32 void show_help();
33
34 void show_version();
35
36 char process(command_options_st *opt);
37
38 char _do_encode_decode(command_options_st *opt);
39
40 #endif
```

A.1.4. command.c

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdbool.h>
4  #include "command.h"
5  #include "encode.h"
6
7  void command_create(command_options_st *opt) {
8      create_file(&opt->input_file);
9      create_file(&opt->output_file);
10     opt->error_condition = OK;
11     opt->encode_option = ENCODE;
12     opt->input_path = 0;
13     opt->output_path = 0;
14 }
15
16 void set_input_file(command_options_st *opt, const char *input) {
17     opt->input_path = input;
18 }
19
20 void set_output_file(command_options_st *opt, const char *output) {
21     opt->output_path = output;
22 }
23
24 void set_encode(command_options_st *opt) {
25     opt->encode_option = ENCODE;
26 }
27
28 void set_decode(command_options_st *opt) {
29     opt->encode_option = DECODE;
30 }
31
32 void set_error(command_options_st *opt, char error_condition) {
33     opt->error_condition = error_condition;
34 }
35
36 int has_errors(command_options_st *opt) {
37     if (opt->error_condition != OK) {
38         return ERROR;
39     }
40     return OK;
41 }
42
43 void show_error(command_options_st *opt) {
44     char *error_message = NULL;
45     bool should_show_help = false;
46     if (opt->error_condition == INVALID_ARGUMENT) {
47         error_message = "Argumentos Invalidos!\n\n";
```

```
48          should_show_help = true;
49      } else if (opt->error_condition == NO_ARGUMENTS) {
50          error_message = "No se recibieron Argumentos!\n\n";
51          should_show_help = true;
52      } else if (opt->error_condition == INVALID_FILE_LENGTH) {
53          error_message = "Longitud de Archivo de Entrada Invalido!\n\n";
54      } else if (opt->error_condition == INVALID_CHARS) {
55          error_message = "Caracteres invalidos en Archivo Codificado!\n\n";
56      }
57
58      fprintf(stderr, "%s", error_message);
59      if (should_show_help) {
60          show_help();
61      }
62  }
63
64  void show_help() {
65      printf("Options:\n");
66      printf("\t-V,\t--version\tPrint version and quit.\n");
67      printf("\t-h,\t--help\t\tPrint this information.\n");
68      printf("\t-i,\t--input\t\tLocation of the input file.\n");
69      printf("\t-o,\t--output\tLocation of the output file.\n");
70      printf("\t-d,\t--decode\tDecode a base64-encoded file (default is encode).\n");
71      printf("Examples:\n");
72      printf("\ttp0 -i ~/input -o ~/output\n");
73      printf("\ttp0 --decode\n");
74  }
75
76  void show_version() {
77      printf("Version: 1.0\n");
78  }
79
80  char process(command_options_st *opt) {
81      if (open_file_read(&opt->input_file, opt->input_path) == ERROR) {
82          return ERROR;
83      }
84      if (open_file_write(&opt->output_file, opt->output_path) == ERROR) {
85          close_file(&opt->input_file);
86          return ERROR;
87      }
88
89      char result = _do_encode_decode(opt);
90      close_file(&opt->input_file);
91      close_file(&opt->output_file);
92      return result;
93  }
94
95  char _do_encode_decode(command_options_st *opt) {
96      unsigned char buf_decoded[3];
97      unsigned char buf_encoded[4];
98      unsigned char count = 0;
99
100     if (opt->encode_option == ENCODE) {
101         while (!file_eof(&opt->input_file)) {
102             memset(buf_decoded, 0, 3);
103             unsigned int read = file_read(&opt->input_file, buf_decoded, 3);
104             if (read > 0) {
105                 Encode(buf_decoded, read, buf_encoded);
106                 file_write(&opt->output_file, buf_encoded, 4);
107                 ++count;
108                 if (count == 18) // 19 * 4 =  76 bytes
109                 {
110                     file_write(&opt->output_file, (unsigned char *) "\n", 1);
111                     count = 0;
112                 }
113             }
114         }
```

```
115     }
116
117     if (opt->encode_option == DECODE) {
118         while (!file_eof(&opt->input_file) && !has_errors(opt)) {
119             unsigned int read = file_read(&opt->input_file, buf_encoded, 4);
120             if (read > 0) {
121                 if (read != 4) {
122                     set_error(opt, INVALID_FILE_LENGTH);
123                     show_error(opt);
124                 } else {
125                     ++count;
126                     if (count == 18) { // 19 * 4 =  76 bytes
127                         unsigned char aux;
128                         file_read(&opt->input_file, &aux, 1);
129                         count = 0;
130                     }
131
132                     if (Decode(buf_encoded, buf_decoded)) {
133                         char aux = 0;
134                         if (buf_encoded[2] == '=') {
135                             ++aux;
136                         }
137                         if (buf_encoded[3] == '=') {
138                             ++aux;
139                         }
140
141                         file_write(&opt->output_file, buf_decoded, 3 - aux);
142                     } else {
143                         set_error(opt, INVALID_CHARS);
144                         show_error(opt);
145                         unsigned int i;
146                         for (i = 0; i < 4; ++i) {
147                             fprintf(stderr, "%c", buf_encoded[i]);
148                         }
149                     }
150                 }
151
152             }
153         }
154     }
155
156     return opt->error_condition;
157 }
```

A.1.5. file.h

```
1  #ifndef __FILE_H__
2  #define __FILE_H__
3
4  #include <stdio.h>
5  #include "constants.h"
6
7
8  typedef struct {
9      FILE *file;
10     char eof;
11 } File;
12
13
14 void create_file(File *file);
15
16
17 char open_file_read(File *file, const char *route);
18
19
20 char open_file_write(File *file, const char *route);
21
22
```

```
int close_file(File *file);


unsigned int file_read(File *file, unsigned char *buffer, unsigned int length);


void file_write(File *file, unsigned char *buffer, unsigned int length);


int file_eof(File *file);

#endif
```

A.1.6. file.c

```
#include "file.h"
#include <stdlib.h>
#include <string.h>
#include <errno.h>

void create_file(File *file) {
    file->file = 0;
    file->eof = 0;
}

char open_file_read(File *file, const char *route) {
    if (route == NULL) {
        file->file = stdin;
    } else {
        file->file = fopen(route, "rb");
        if (file->file == NULL) {
            int err = errno;
            fprintf(stderr, "Error al abrir archivo: %s\n", strerror(err));
            return ERROR;
        }
    }
    return OK;
}


char open_file_write(File *file, const char *route) {
    if (route == NULL) {
        file->file = stdout;
    } else {
        file->file = fopen(route, "wb");
        if (file->file == NULL) {
            int err = errno;
            fprintf(stderr, "Error al abrir archivo: %s\n", strerror(err));
            return ERROR;
        }
    }
    return OK;
}


int close_file(File *file) {
    if (file->file == stdin || file->file == stdout) return OK;

    int result = fclose(file->file);
    if (result == EOF) {
        int err = errno;
        fprintf(stderr, "Error al cerrar archivo: %s\n", strerror(err));
        return ERROR;
    }
    return OK;
}


```

```
54 unsigned int file_read(File *file, unsigned char *buffer, unsigned int length) {
55     unsigned int result = 0;
56     if (!file_eof(file)) {
57         result = (unsigned int) fread(buffer, sizeof(char), length, file->file);
58         if (feof(file->file)) {
59             file->eof = 1;
60         }
61     }
62
63     return result;
64 }
65
66
67 void file_write(File *file, unsigned char *buffer, unsigned int length) {
68     fwrite(buffer, sizeof(char), length, file->file);
69 }
70
71
72 int file_eof(File *file) {
73     return file->eof;
74 }
```

A.1.7. encode.h

```
1  #ifndef __ENCODE_H__
2  #define __ENCODE_H__
3
4  /**
5   * Recibe 3 caracteres en buffer y los convierte en 4 caracteres codificados en output.
6   * Pre: el buffer contiene length caracteres (1 a 3) y todos los caracteres son validos
7   * Post: retorna un buffer de 4 byte con los caracteres en base64.
8   */
9  void Encode(const unsigned char *buffer, unsigned int length, unsigned char *output);
10
11 /**
12  * Devuelve un buffer de 3 caracteres recibiendo los 4 caracteres codificados en input.
13  * Pre:
14  * Post:
15  */
16 unsigned char Decode(unsigned char *buf_input, unsigned char *buf_output);
17
18 #endif // __ENCODE_H__
```

A.1.8. encode.c

```
1  #define BASE64_END '='
2  #define DECODE_ERROR 100
3
4  static unsigned char encoding_table[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
5                                           'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
6                                           'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
7                                           'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f',
8                                           'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
9                                           'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
10                                          'w', 'x', 'y', 'z', '0', '1', '2', '3',
11                                          '4', '5', '6', '7', '8', '9', '+', '/'};
12
13 static int encoding_table_size = 64;
14
15
16 void Encode(const unsigned char *buffer, unsigned int length, unsigned char *output) {
17     unsigned char b1 = buffer[0];
18     unsigned char b2 = buffer[1];
19     unsigned char b3 = buffer[2];
20     //I retrieve the first 6 bits and operate..
21     unsigned char b1aux = b1 >> 2;
22     //Recovered the first 6 bits, I look into the encoding in the table.
```

```
23      output[0] = encoding_table[(int) b1aux];
24      //I retrieve the next 6 bits.
25      unsigned char b2aux = b1 << 6;
26      b2aux = b2aux >> 2;
27      b2aux = b2aux | (b2 >> 4);
28      //I take look into the encoding table
29      output[1] = encoding_table[(int) b2aux];
30      output[2] = BASE64_END;
31      output[3] = BASE64_END;
32      if (length == 3) {
33          /*
34       * If I have 3 characters in the buffer I operate
35       * with the last 2 characters.
36       */
37          unsigned char b3aux = b3 >> 6;
38          unsigned char b3aux2 = b2 << 4;
39          b3aux2 = b3aux2 >> 2;
40          b3aux = b3aux | b3aux2;
41          //I take look into the encoding table.
42          output[2] = encoding_table[(int) b3aux];
43          unsigned char b4aux = b3 << 2;
44          b4aux = b4aux >> 2;
45          //I take look into the encoding table.
46          output[3] = encoding_table[(int) b4aux];
47      } else {
48          if (length == 2) {
49              /*
50               * In case of having only 2 characters in the buffer
51               * I recover the remaining character and place the end of the line(=)
52               */
53              unsigned char b3aux = b3 >> 6;
54              unsigned char b3aux2 = b2 << 4;
55              b3aux2 = b3aux2 >> 2;
56              b3aux = b3aux | b3aux2;
57              output[2] = encoding_table[(int) b3aux];
58          }
59      }
60  }
61
62  /**
63   * Returns the representation of the char in the table.
64   * pre: character is valid (belongs to table)
65   * is the character '='.
66   * post: returns the representation (int) of the character
67   * in the encoding table.
68   *
69   */
70  unsigned char DecodeChar(char character) {
71      unsigned char i;
72      for (i = 0; i < encoding_table_size; i++) {
73          if (encoding_table[i] == character) {
74              return i;
75          }
76      }
77      if (character == '=') {
78          return 0;
79      }
80      return DECODE_ERROR;
81  }
82
83  /**
84   * Returns a buffer with size 3 with the 4 character base64 decode.
85   * Pre: the input buffer contains 4 characters. The output buffer has at least 3 characters
86   * Post: returns a buffer with size 3 ASCII characters. returns 0 if error 1 if ok
87   */
88  unsigned char Decode(unsigned char *buf_input, unsigned char *buf_output) {
89      unsigned char chars[4];
```

```
90      unsigned int i;
91      for (i = 0; i < 4; ++i) {
92          chars[i] = DecodeChar(buf_input[i]);
93          if (chars[i] == DECODE_ERROR)
94              return 0;
95      }
96
97      unsigned char char1_aux = chars[0] << 2;
98      //Take the last 2 bits of char2
99      unsigned char char2_aux = chars[1] >> 4;
100     char1_aux = char1_aux | char2_aux;
101     buf_output[0] = char1_aux;
102
103     //Take the last 4b of char2 and the first 4b of char3
104     char1_aux = chars[1] << 4;
105     char2_aux = chars[2] >> 2;
106     char2_aux = char1_aux | char2_aux;
107     buf_output[1] = char2_aux;
108
109     //Take the last 2b of char3 + the bits of char4
110     char1_aux = chars[2] << 6;
111     buf_output[2] = char1_aux | chars[3];
112     return 1;
113 }
```

A.2. Código MIPS32

A.2.1. tp0.S

```
1
2  tp0:     file format elf32-tradbigmips
3
4
5  Disassembly of section .init:
6
7  00000b28 <_init>:
8   b28:   3c1c0002        lui     gp,0x2
9   b2c:   279c9d58        addiu   gp,gp,-25256
10  b30:   0399e021        addu    gp,gp,t9
11  b34:   27bdffe0        addiu   sp,sp,-32
12  b38:   afbc0010        sw      gp,16(sp)
13  b3c:   afbf001c        sw      ra,28(sp)
14  b40:   8f8280d8        lw      v0,-32552(gp)
15  b44:   10400004        beqz    v0,b58 <_init+0x30>
16  b48:   00000000        nop
17  b4c:   8f9980d8        lw      t9,-32552(gp)
18  b50:   0320f809        jalr    t9
19  b54:   00000000        nop
20  b58:   8fbf001c        lw      ra,28(sp)
21  b5c:   03e00008        jr      ra
22  b60:   27bd0020        addiu   sp,sp,32
23
24 Disassembly of section .text:
25
26 00000b70 <__start>:
27      b70:       03e00025        move    zero,ra
28      b74:       04110001        bal     b7c <__start+0xc>
29      b78:       00000000        nop
30      b7c:       3c1c0002        lui     gp,0x2
31      b80:       279c9d04        addiu   gp,gp,-25340
32      b84:       039fe021        addu    gp,gp,ra
33      b88:       0000f825        move    ra,zero
34      b8c:       8f848018        lw      a0,-32744(gp)
35      b90:       8fa50000        lw      a1,0(sp)
36      b94:       27a60004        addiu   a2,sp,4
37      b98:       2401fff8        li      at,-8
38      b9c:       03a1e824        and     sp,sp,at
39      ba0:       27bdffe0        addiu   sp,sp,-32
40      ba4:       8f87801c        lw      a3,-32740(gp)
41      ba8:       8f888020        lw      t0,-32736(gp)
42      bac:       afa80010        sw      t0,16(sp)
43      bb0:       afa20014        sw      v0,20(sp)
44      bb4:       afbd0018        sw      sp,24(sp)
45      bb8:       8f9980cc        lw      t9,-32564(gp)
46      bbc:       0320f809        jalr    t9
47      bc0:       00000000        nop
48
49 00000bc4 <hlt>:
50      bc4:       1000ffff        b       bc4 <hlt>
51      bc8:       00000000        nop
52      bcc:       00000000        nop
53
54 00000bd0 <deregister_tm_clones>:
55      bd0:       3c1c0002        lui     gp,0x2
56      bd4:       279c9cb0        addiu   gp,gp,-25424
57      bd8:       0399e021        addu    gp,gp,t9
58      bdc:       8f848028        lw      a0,-32728(gp)
59      be0:       8f828024        lw      v0,-32732(gp)
60      be4:       24842884        addiu   a0,a0,10372
61      be8:       24420003        addiu   v0,v0,3
62      bec:       00441023        subu    v0,v0,a0
63      bf0:       2c420007        sltiu   v0,v0,7
64      bf4:       14400005        bnez    v0,c0c <deregister_tm_clones+0x3c>
```

```
65      bf8:       8f9980f8        lw      t9,-32520(gp)
66      bfc:       13200003        beqz    t9,c0c <deregister_tm_clones+0x3c>
67      c00:       00000000        nop
68      c04:       03200008        jr      t9
69      c08:       00000000        nop
70      c0c:       03e00008        jr      ra
71      c10:       00000000        nop
72
73  00000c14 <register_tm_clones>:
74      c14:       3c1c0002        lui     gp,0x2
75      c18:       279c9c6c        addiu   gp,gp,-25492
76      c1c:       0399e021        addu    gp,gp,t9
77      c20:       8f848028        lw      a0,-32728(gp)
78      c24:       8f858024        lw      a1,-32732(gp)
79      c28:       24842884        addiu   a0,a0,10372
80      c2c:       00a42823        subu    a1,a1,a0
81      c30:       00052883        sra     a1,a1,0x2
82      c34:       000517c2        srl     v0,a1,0x1f
83      c38:       00452821        addu    a1,v0,a1
84      c3c:       00052843        sra     a1,a1,0x1
85      c40:       10a00005        beqz    a1,c58 <register_tm_clones+0x44>
86      c44:       8f9980a8        lw      t9,-32600(gp)
87      c48:       13200003        beqz    t9,c58 <register_tm_clones+0x44>
88      c4c:       00000000        nop
89      c50:       03200008        jr      t9
90      c54:       00000000        nop
91      c58:       03e00008        jr      ra
92      c5c:       00000000        nop
93
94  00000c60 <__do_global_dtors_aux>:
95      c60:       3c1c0002        lui     gp,0x2
96      c64:       279c9c20        addiu   gp,gp,-25568
97      c68:       0399e021        addu    gp,gp,t9
98      c6c:       27bdffe0        addiu   sp,sp,-32
99      c70:       afb00018        sw      s0,24(sp)
100     c74:       8f908028        lw      s0,-32728(gp)
101     c78:       afbc0010        sw      gp,16(sp)
102     c7c:       afbf001c        sw      ra,28(sp)
103     c80:       92022990        lbu     v0,10640(s0)
104     c84:       1440000d        bnez    v0,cbc <__do_global_dtors_aux+0x5c>
105     c88:       8f8280fc        lw      v0,-32516(gp)
106     c8c:       10400005        beqz    v0,ca4 <__do_global_dtors_aux+0x44>
107     c90:       8f82802c        lw      v0,-32724(gp)
108     c94:       8f9980fc        lw      t9,-32516(gp)
109     c98:       0320f809        jalr    t9
110     c9c:       8c440000        lw      a0,0(v0)
111     ca0:       8fbc0010        lw      gp,16(sp)
112     ca4:       8f998030        lw      t9,-32720(gp)
113     ca8:       27390bd0        addiu   t9,t9,3024
114     cac:       0411ffc8        bal     bd0 <deregister_tm_clones>
115     cb0:       00000000        nop
116     cb4:       24020001        li      v0,1
117     cb8:       a2022990        sb      v0,10640(s0)
118     cbc:       8fbf001c        lw      ra,28(sp)
119     cc0:       8fb00018        lw      s0,24(sp)
120     cc4:       03e00008        jr      ra
121     cc8:       27bd0020        addiu   sp,sp,32
122
123 00000ccc <frame_dummy>:
124     ccc:       3c1c0002        lui     gp,0x2
125     cd0:       279c9bb4        addiu   gp,gp,-25676
126     cd4:       0399e021        addu    gp,gp,t9
127     cd8:       8f828028        lw      v0,-32728(gp)
128     cdc:       27bdffe0        addiu   sp,sp,-32
129     ce0:       244427cc        addiu   a0,v0,10188
130     ce4:       afbc0010        sw      gp,16(sp)
131     ce8:       afbf001c        sw      ra,28(sp)
```

```
132      cec:       8c820000        lw      v0,0(a0)
133      cf0:       14400006        bnez    v0,d0c <frame_dummy+0x40>
134      cf4:       8f9980b0        lw      t9,-32592(gp)
135      cf8:       8f998030        lw      t9,-32720(gp)
136      cfc:       8fbf001c        lw      ra,28(sp)
137      d00:       27390c14        addiu   t9,t9,3092
138      d04:       1000ffc3        b       c14 <register_tm_clones>
139      d08:       27bd0020        addiu   sp,sp,32
140      d0c:       1320fffa        beqz    t9,cf8 <frame_dummy+0x2c>
141      d10:       00000000        nop
142      d14:       0320f809        jalr    t9
143      d18:       00000000        nop
144      d1c:       1000fff6        b       cf8 <frame_dummy+0x2c>
145      d20:       8fbc0010        lw      gp,16(sp)
146         ...
147
148 00000d30 <command_create>:
149 #include <string.h>
150 #include <stdbool.h>
151 #include "command.h"
152 #include "encode.h"
153
154 void command_create(command_options_st *opt) {
155      d30:       3c1c0002        lui     gp,0x2
156      d34:       279c9b50        addiu   gp,gp,-25776
157      d38:       0399e021        addu    gp,gp,t9
158      d3c:       27bdffe0        addiu   sp,sp,-32
159      d40:       afbf001c        sw      ra,28(sp)
160      d44:       afbe0018        sw      s8,24(sp)
161      d48:       03a0f025        move    s8,sp
162      d4c:       afbc0010        sw      gp,16(sp)
163      d50:       afc40020        sw      a0,32(s8)
164     create_file(&opt->input_file);
165      d54:       8fc20020        lw      v0,32(s8)
166      d58:       00402025        move    a0,v0
167      d5c:       8f828034        lw      v0,-32716(gp)
168      d60:       0040c825        move    t9,v0
169      d64:       0411037a        bal     1b50 <create_file>
170      d68:       00000000        nop
171      d6c:       8fdc0010        lw      gp,16(s8)
172     create_file(&opt->output_file);
173      d70:       8fc20020        lw      v0,32(s8)
174      d74:       24420008        addiu   v0,v0,8
175      d78:       00402025        move    a0,v0
176      d7c:       8f828034        lw      v0,-32716(gp)
177      d80:       0040c825        move    t9,v0
178      d84:       04110372        bal     1b50 <create_file>
179      d88:       00000000        nop
180      d8c:       8fdc0010        lw      gp,16(s8)
181     opt->error_condition = OK;
182      d90:       8fc20020        lw      v0,32(s8)
183      d94:       a0400019        sb      zero,25(v0)
184     opt->encode_option = ENCODE;
185      d98:       8fc20020        lw      v0,32(s8)
186      d9c:       24030001        li      v1,1
187      da0:       a0430018        sb      v1,24(v0)
188     opt->input_path = 0;
189      da4:       8fc20020        lw      v0,32(s8)
190      da8:       ac400010        sw      zero,16(v0)
191     opt->output_path = 0;
192      dac:       8fc20020        lw      v0,32(s8)
193      db0:       ac400014        sw      zero,20(v0)
194 }
195      db4:       00000000        nop
196      db8:       03c0e825        move    sp,s8
197      dbc:       8fbf001c        lw      ra,28(sp)
198      dc0:       8fbe0018        lw      s8,24(sp)
```

```
199       dc4:       27bd0020        addiu   sp,sp,32
200       dc8:       03e00008        jr      ra
201       dcc:       00000000        nop
202
203 00000dd0 <set_input_file>:
204
205 void set_input_file(command_options_st *opt, const char *input) {
206       dd0:       27bdfff8        addiu   sp,sp,-8
207       dd4:       afbe0004        sw      s8,4(sp)
208       dd8:       03a0f025        move    s8,sp
209       ddc:       afc40008        sw      a0,8(s8)
210       de0:       afc5000c        sw      a1,12(s8)
211     opt->input_path = input;
212       de4:       8fc20008        lw      v0,8(s8)
213       de8:       8fc3000c        lw      v1,12(s8)
214       dec:       ac430010        sw      v1,16(v0)
215 }
216       df0:       00000000        nop
217       df4:       03c0e825        move    sp,s8
218       df8:       8fbe0004        lw      s8,4(sp)
219       dfc:       27bd0008        addiu   sp,sp,8
220       e00:       03e00008        jr      ra
221       e04:       00000000        nop
222
223 00000e08 <set_output_file>:
224
225 void set_output_file(command_options_st *opt, const char *output) {
226       e08:       27bdfff8        addiu   sp,sp,-8
227       e0c:       afbe0004        sw      s8,4(sp)
228       e10:       03a0f025        move    s8,sp
229       e14:       afc40008        sw      a0,8(s8)
230       e18:       afc5000c        sw      a1,12(s8)
231     opt->output_path = output;
232       e1c:       8fc20008        lw      v0,8(s8)
233       e20:       8fc3000c        lw      v1,12(s8)
234       e24:       ac430014        sw      v1,20(v0)
235 }
236       e28:       00000000        nop
237       e2c:       03c0e825        move    sp,s8
238       e30:       8fbe0004        lw      s8,4(sp)
239       e34:       27bd0008        addiu   sp,sp,8
240       e38:       03e00008        jr      ra
241       e3c:       00000000        nop
242
243 00000e40 <set_encode>:
244
245 void set_encode(command_options_st *opt) {
246       e40:       27bdfff8        addiu   sp,sp,-8
247       e44:       afbe0004        sw      s8,4(sp)
248       e48:       03a0f025        move    s8,sp
249       e4c:       afc40008        sw      a0,8(s8)
250     opt->encode_option = ENCODE;
251       e50:       8fc20008        lw      v0,8(s8)
252       e54:       24030001        li      v1,1
253       e58:       a0430018        sb      v1,24(v0)
254 }
255       e5c:       00000000        nop
256       e60:       03c0e825        move    sp,s8
257       e64:       8fbe0004        lw      s8,4(sp)
258       e68:       27bd0008        addiu   sp,sp,8
259       e6c:       03e00008        jr      ra
260       e70:       00000000        nop
261
262 00000e74 <set_decode>:
263
264 void set_decode(command_options_st *opt) {
265       e74:       27bdfff8        addiu   sp,sp,-8
```

```
266      e78:       afbe0004        sw      s8,4(sp)
267      e7c:       03a0f025        move    s8,sp
268      e80:       afc40008        sw      a0,8(s8)
269     opt->encode_option = DECODE;
270      e84:       8fc20008        lw      v0,8(s8)
271      e88:       a0400018        sb      zero,24(v0)
272  }
273      e8c:       00000000        nop
274      e90:       03c0e825        move    sp,s8
275      e94:       8fbe0004        lw      s8,4(sp)
276      e98:       27bd0008        addiu   sp,sp,8
277      e9c:       03e00008        jr      ra
278      ea0:       00000000        nop
279
280  00000ea4 <set_error>:
281
282  void set_error(command_options_st *opt, char error_condition) {
283      ea4:       27bdfff8        addiu   sp,sp,-8
284      ea8:       afbe0004        sw      s8,4(sp)
285      eac:       03a0f025        move    s8,sp
286      eb0:       afc40008        sw      a0,8(s8)
287      eb4:       00a01025        move    v0,a1
288      eb8:       a3c2000c        sb      v0,12(s8)
289     opt->error_condition = error_condition;
290      ebc:       8fc20008        lw      v0,8(s8)
291      ec0:       93c3000c        lbu     v1,12(s8)
292      ec4:       a0430019        sb      v1,25(v0)
293  }
294      ec8:       00000000        nop
295      ecc:       03c0e825        move    sp,s8
296      ed0:       8fbe0004        lw      s8,4(sp)
297      ed4:       27bd0008        addiu   sp,sp,8
298      ed8:       03e00008        jr      ra
299      edc:       00000000        nop
300
301  00000ee0 <has_errors>:
302
303  int has_errors(command_options_st *opt) {
304      ee0:       27bdfff8        addiu   sp,sp,-8
305      ee4:       afbe0004        sw      s8,4(sp)
306      ee8:       03a0f025        move    s8,sp
307      eec:       afc40008        sw      a0,8(s8)
308     if (opt->error_condition != OK) {
309      ef0:       8fc20008        lw      v0,8(s8)
310      ef4:       80420019        lb      v0,25(v0)
311      ef8:       10400004        beqz    v0,f0c <has_errors+0x2c>
312      efc:       00000000        nop
313         return ERROR;
314      f00:       24020001        li      v0,1
315      f04:       10000002        b       f10 <has_errors+0x30>
316      f08:       00000000        nop
317     }
318     return OK;
319      f0c:       00001025        move    v0,zero
320  }
321      f10:       03c0e825        move    sp,s8
322      f14:       8fbe0004        lw      s8,4(sp)
323      f18:       27bd0008        addiu   sp,sp,8
324      f1c:       03e00008        jr      ra
325      f20:       00000000        nop
326
327  00000f24 <show_error>:
328
329  void show_error(command_options_st *opt) {
330      f24:       3c1c0002        lui     gp,0x2
331      f28:       279c995c        addiu   gp,gp,-26276
332      f2c:       0399e021        addu    gp,gp,t9
```

```
f30:       27bdffd8        addiu   sp,sp,-40
f34:       afbf0024        sw      ra,36(sp)
f38:       afbe0020        sw      s8,32(sp)
f3c:       03a0f025        move    s8,sp
f40:       afbc0010        sw      gp,16(sp)
f44:       afc40028        sw      a0,40(s8)
    char *error_message = NULL;
f48:       afc00018        sw      zero,24(s8)
    bool should_show_help = false;
f4c:       a3c0001c        sb      zero,28(s8)
    if (opt->error_condition == INVALID_ARGUMENT) {
f50:       8fc20028        lw      v0,40(s8)
f54:       80430019        lb      v1,25(v0)
f58:       2402ffff        li      v0,-1
f5c:       14620008        bne     v1,v0,f80 <show_error+0x5c>
f60:       00000000        nop
        error_message = "Argumentos Invalidos!\n\n";
f64:       8f828030        lw      v0,-32720(gp)
f68:       244224f0        addiu   v0,v0,9456
f6c:       afc20018        sw      v0,24(s8)
        should_show_help = true;
f70:       24020001        li      v0,1
f74:       a3c2001c        sb      v0,28(s8)
f78:       1000001f        b       ff8 <show_error+0xd4>
f7c:       00000000        nop
    } else if (opt->error_condition == NO_ARGUMENTS) {
f80:       8fc20028        lw      v0,40(s8)
f84:       80430019        lb      v1,25(v0)
f88:       2402fffe        li      v0,-2
f8c:       14620008        bne     v1,v0,fb0 <show_error+0x8c>
f90:       00000000        nop
        error_message = "No se recibieron Argumentos!\n\n";
f94:       8f828030        lw      v0,-32720(gp)
f98:       24422508        addiu   v0,v0,9480
f9c:       afc20018        sw      v0,24(s8)
        should_show_help = true;
fa0:       24020001        li      v0,1
fa4:       a3c2001c        sb      v0,28(s8)
fa8:       10000013        b       ff8 <show_error+0xd4>
fac:       00000000        nop
    } else if (opt->error_condition == INVALID_FILE_LENGTH) {
fb0:       8fc20028        lw      v0,40(s8)
fb4:       80430019        lb      v1,25(v0)
fb8:       2402fffc        li      v0,-4
fbc:       14620006        bne     v1,v0,fd8 <show_error+0xb4>
fc0:       00000000        nop
        error_message = "Longitud de Archivo de Entrada Invalido!\n\n";
fc4:       8f828030        lw      v0,-32720(gp)
fc8:       24422528        addiu   v0,v0,9512
fcc:       afc20018        sw      v0,24(s8)
fd0:       10000009        b       ff8 <show_error+0xd4>
fd4:       00000000        nop
    } else if (opt->error_condition == INVALID_CHARS) {
fd8:       8fc20028        lw      v0,40(s8)
fdc:       80430019        lb      v1,25(v0)
fe0:       2402fffb        li      v0,-5
fe4:       14620004        bne     v1,v0,ff8 <show_error+0xd4>
fe8:       00000000        nop
        error_message = "Caracteres invalidos en Archivo Codificado!\n\n";
fec:       8f828030        lw      v0,-32720(gp)
ff0:       24422554        addiu   v0,v0,9556
ff4:       afc20018        sw      v0,24(s8)
    }

    fprintf(stderr, "%s", error_message);
ff8:       8f8280f0        lw      v0,-32528(gp)
ffc:       8c420000        lw      v0,0(v0)
```

```
    1000:       00402825        move    a1,v0
    1004:       8fc40018        lw      a0,24(s8)
    1008:       8f8280a0        lw      v0,-32608(gp)
    100c:       0040c825        move    t9,v0
    1010:       0320f809        jalr    t9
    1014:       00000000        nop
    1018:       8fdc0010        lw      gp,16(s8)
    if (should_show_help) {
    101c:       93c2001c        lbu     v0,28(s8)
    1020:       10400006        beqz    v0,103c <show_error+0x118>
    1024:       00000000        nop
        show_help();
    1028:       8f828038        lw      v0,-32712(gp)
    102c:       0040c825        move    t9,v0
    1030:       04110009        bal     1058 <show_help>
    1034:       00000000        nop
    1038:       8fdc0010        lw      gp,16(s8)
    }
}
    103c:       00000000        nop
    1040:       03c0e825        move    sp,s8
    1044:       8fbf0024        lw      ra,36(sp)
    1048:       8fbe0020        lw      s8,32(sp)
    104c:       27bd0028        addiu   sp,sp,40
    1050:       03e00008        jr      ra
    1054:       00000000        nop

00001058 <show_help>:

void show_help() {
    1058:       3c1c0002        lui     gp,0x2
    105c:       279c9828        addiu   gp,gp,-26584
    1060:       0399e021        addu    gp,gp,t9
    1064:       27bdffe0        addiu   sp,sp,-32
    1068:       afbf001c        sw      ra,28(sp)
    106c:       afbe0018        sw      s8,24(sp)
    1070:       03a0f025        move    s8,sp
    1074:       afbc0010        sw      gp,16(sp)
    printf("Options:\n");
    1078:       8f828030        lw      v0,-32720(gp)
    107c:       24442584        addiu   a0,v0,9604
    1080:       8f8280e0        lw      v0,-32544(gp)
    1084:       0040c825        move    t9,v0
    1088:       0320f809        jalr    t9
    108c:       00000000        nop
    1090:       8fdc0010        lw      gp,16(s8)
    printf("\t-V,\t--version\tPrint version and quit.\n");
    1094:       8f828030        lw      v0,-32720(gp)
    1098:       24442590        addiu   a0,v0,9616
    109c:       8f8280e0        lw      v0,-32544(gp)
    10a0:       0040c825        move    t9,v0
    10a4:       0320f809        jalr    t9
    10a8:       00000000        nop
    10ac:       8fdc0010        lw      gp,16(s8)
    printf("\t-h,\t--help\t\tPrint this information.\n");
    10b0:       8f828030        lw      v0,-32720(gp)
    10b4:       244425b8        addiu   a0,v0,9656
    10b8:       8f8280e0        lw      v0,-32544(gp)
    10bc:       0040c825        move    t9,v0
    10c0:       0320f809        jalr    t9
    10c4:       00000000        nop
    10c8:       8fdc0010        lw      gp,16(s8)
    printf("\t-i,\t--input\t\tLocation of the input file.\n");
    10cc:       8f828030        lw      v0,-32720(gp)
    10d0:       244425e0        addiu   a0,v0,9696
    10d4:       8f8280e0        lw      v0,-32544(gp)
    10d8:       0040c825        move    t9,v0
```

```
467     10dc:       0320f809        jalr    t9
468     10e0:       00000000        nop
469     10e4:       8fdc0010        lw      gp,16(s8)
470     printf("\t-o,\t--output\tLocation of the output file.\n");
471     10e8:       8f828030        lw      v0,-32720(gp)
472     10ec:       2444260c        addiu   a0,v0,9740
473     10f0:       8f8280e0        lw      v0,-32544(gp)
474     10f4:       0040c825        move    t9,v0
475     10f8:       0320f809        jalr    t9
476     10fc:       00000000        nop
477     1100:       8fdc0010        lw      gp,16(s8)
478     printf("\t-d,\t--decode\tDecode a base64-encoded file (default is encode).\n");
479     1104:       8f828030        lw      v0,-32720(gp)
480     1108:       24442638        addiu   a0,v0,9784
481     110c:       8f8280e0        lw      v0,-32544(gp)
482     1110:       0040c825        move    t9,v0
483     1114:       0320f809        jalr    t9
484     1118:       00000000        nop
485     111c:       8fdc0010        lw      gp,16(s8)
486     printf("Examples:\n");
487     1120:       8f828030        lw      v0,-32720(gp)
488     1124:       24442678        addiu   a0,v0,9848
489     1128:       8f8280e0        lw      v0,-32544(gp)
490     112c:       0040c825        move    t9,v0
491     1130:       0320f809        jalr    t9
492     1134:       00000000        nop
493     1138:       8fdc0010        lw      gp,16(s8)
494     printf("\ttp0 -i ~/input -o ~/output\n");
495     113c:       8f828030        lw      v0,-32720(gp)
496     1140:       24442684        addiu   a0,v0,9860
497     1144:       8f8280e0        lw      v0,-32544(gp)
498     1148:       0040c825        move    t9,v0
499     114c:       0320f809        jalr    t9
500     1150:       00000000        nop
501     1154:       8fdc0010        lw      gp,16(s8)
502     printf("\ttp0 --decode\n");
503     1158:       8f828030        lw      v0,-32720(gp)
504     115c:       244426a0        addiu   a0,v0,9888
505     1160:       8f8280e0        lw      v0,-32544(gp)
506     1164:       0040c825        move    t9,v0
507     1168:       0320f809        jalr    t9
508     116c:       00000000        nop
509     1170:       8fdc0010        lw      gp,16(s8)
510 }
511     1174:       00000000        nop
512     1178:       03c0e825        move    sp,s8
513     117c:       8fbf001c        lw      ra,28(sp)
514     1180:       8fbe0018        lw      s8,24(sp)
515     1184:       27bd0020        addiu   sp,sp,32
516     1188:       03e00008        jr      ra
517     118c:       00000000        nop
518
519 00001190 <show_version>:
520
521 void show_version() {
522     1190:       3c1c0002        lui     gp,0x2
523     1194:       279c96f0        addiu   gp,gp,-26896
524     1198:       0399e021        addu    gp,gp,t9
525     119c:       27bdffe0        addiu   sp,sp,-32
526     11a0:       afbf001c        sw      ra,28(sp)
527     11a4:       afbe0018        sw      s8,24(sp)
528     11a8:       03a0f025        move    s8,sp
529     11ac:       afbc0010        sw      gp,16(sp)
530     printf("Version: 1.0\n");
531     11b0:       8f828030        lw      v0,-32720(gp)
532     11b4:       244426b0        addiu   a0,v0,9904
533     11b8:       8f8280e0        lw      v0,-32544(gp)
```

```
    11bc:       0040c825        move    t9,v0
    11c0:       0320f809        jalr    t9
    11c4:       00000000        nop
    11c8:       8fdc0010        lw      gp,16(s8)
}
    11cc:       00000000        nop
    11d0:       03c0e825        move    sp,s8
    11d4:       8fbf001c        lw      ra,28(sp)
    11d8:       8fbe0018        lw      s8,24(sp)
    11dc:       27bd0020        addiu   sp,sp,32
    11e0:       03e00008        jr      ra
    11e4:       00000000        nop

000011e8 <process>:

char process(command_options_st *opt) {
    11e8:       3c1c0002        lui     gp,0x2
    11ec:       279c9698        addiu   gp,gp,-26984
    11f0:       0399e021        addu    gp,gp,t9
    11f4:       27bdffd8        addiu   sp,sp,-40
    11f8:       afbf0024        sw      ra,36(sp)
    11fc:       afbe0020        sw      s8,32(sp)
    1200:       03a0f025        move    s8,sp
    1204:       afbc0010        sw      gp,16(sp)
    1208:       afc40028        sw      a0,40(s8)
    if (open_file_read(&opt->input_file, opt->input_path) == ERROR) {
    120c:       8fc30028        lw      v1,40(s8)
    1210:       8fc20028        lw      v0,40(s8)
    1214:       8c420010        lw      v0,16(v0)
    1218:       00402825        move    a1,v0
    121c:       00602025        move    a0,v1
    1220:       8f82803c        lw      v0,-32708(gp)
    1224:       0040c825        move    t9,v0
    1228:       04110257        bal     1b88 <open_file_read>
    122c:       00000000        nop
    1230:       8fdc0010        lw      gp,16(s8)
    1234:       00401825        move    v1,v0
    1238:       24020001        li      v0,1
    123c:       14620004        bne     v1,v0,1250 <process+0x68>
    1240:       00000000        nop
        return ERROR;
    1244:       24020001        li      v0,1
    1248:       10000031        b       1310 <process+0x128>
    124c:       00000000        nop
    }
    if (open_file_write(&opt->output_file, opt->output_path) == ERROR) {
    1250:       8fc20028        lw      v0,40(s8)
    1254:       24430008        addiu   v1,v0,8
    1258:       8fc20028        lw      v0,40(s8)
    125c:       8c420014        lw      v0,20(v0)
    1260:       00402825        move    a1,v0
    1264:       00602025        move    a0,v1
    1268:       8f828040        lw      v0,-32704(gp)
    126c:       0040c825        move    t9,v0
    1270:       0411028b        bal     1ca0 <open_file_write>
    1274:       00000000        nop
    1278:       8fdc0010        lw      gp,16(s8)
    127c:       00401825        move    v1,v0
    1280:       24020001        li      v0,1
    1284:       1462000b        bne     v1,v0,12b4 <process+0xcc>
    1288:       00000000        nop
        close_file(&opt->input_file);
    128c:       8fc20028        lw      v0,40(s8)
    1290:       00402025        move    a0,v0
    1294:       8f828044        lw      v0,-32700(gp)
    1298:       0040c825        move    t9,v0
    129c:       041102c6        bal     1db8 <close_file>
```

```
601     12a0:       00000000        nop
602     12a4:       8fdc0010        lw      gp,16(s8)
603         return ERROR;
604     12a8:       24020001        li      v0,1
605     12ac:       10000018        b       1310 <process+0x128>
606     12b0:       00000000        nop
607     }
608
609     char result = _do_encode_decode(opt);
610     12b4:       8fc40028        lw      a0,40(s8)
611     12b8:       8f828048        lw      v0,-32696(gp)
612     12bc:       0040c825        move    t9,v0
613     12c0:       04110019        bal     1328 <_do_encode_decode>
614     12c4:       00000000        nop
615     12c8:       8fdc0010        lw      gp,16(s8)
616     12cc:       a3c20018        sb      v0,24(s8)
617     close_file(&opt->input_file);
618     12d0:       8fc20028        lw      v0,40(s8)
619     12d4:       00402025        move    a0,v0
620     12d8:       8f828044        lw      v0,-32700(gp)
621     12dc:       0040c825        move    t9,v0
622     12e0:       041102b5        bal     1db8 <close_file>
623     12e4:       00000000        nop
624     12e8:       8fdc0010        lw      gp,16(s8)
625     close_file(&opt->output_file);
626     12ec:       8fc20028        lw      v0,40(s8)
627     12f0:       24420008        addiu   v0,v0,8
628     12f4:       00402025        move    a0,v0
629     12f8:       8f828044        lw      v0,-32700(gp)
630     12fc:       0040c825        move    t9,v0
631     1300:       041102ad        bal     1db8 <close_file>
632     1304:       00000000        nop
633     1308:       8fdc0010        lw      gp,16(s8)
634     return result;
635     130c:       83c20018        lb      v0,24(s8)
636 }
637     1310:       03c0e825        move    sp,s8
638     1314:       8fbf0024        lw      ra,36(sp)
639     1318:       8fbe0020        lw      s8,32(sp)
640     131c:       27bd0028        addiu   sp,sp,40
641     1320:       03e00008        jr      ra
642     1324:       00000000        nop
643
644 00001328 <_do_encode_decode>:
645
646 char _do_encode_decode(command_options_st *opt) {
647     1328:       3c1c0002        lui     gp,0x2
648     132c:       279c9558        addiu   gp,gp,-27304
649     1330:       0399e021        addu    gp,gp,t9
650     1334:       27bdffc0        addiu   sp,sp,-64
651     1338:       afbf003c        sw      ra,60(sp)
652     133c:       afbe0038        sw      s8,56(sp)
653     1340:       03a0f025        move    s8,sp
654     1344:       afbc0010        sw      gp,16(sp)
655     1348:       afc40040        sw      a0,64(s8)
656     unsigned char buf_decoded[3];
657     unsigned char buf_encoded[4];
658     unsigned char count = 0;
659     134c:       a3c00018        sb      zero,24(s8)
660
661     if (opt->encode_option == ENCODE) {
662     1350:       8fc20040        lw      v0,64(s8)
663     1354:       80430018        lb      v1,24(v0)
664     1358:       24020001        li      v0,1
665     135c:       1462004b        bne     v1,v0,148c <_do_encode_decode+0x164>
666     1360:       00000000        nop
667         while (!file_eof(&opt->input_file)) {
```

```
668     1364:       10000040        b       1468 <_do_encode_decode+0x140>
669     1368:       00000000        nop
670             memset(buf_decoded, 0, 3);
671     136c:       27c20028        addiu   v0,s8,40
672     1370:       24060003        li      a2,3
673     1374:       00002825        move    a1,zero
674     1378:       00402025        move    a0,v0
675     137c:       8f8280c0        lw      v0,-32576(gp)
676     1380:       0040c825        move    t9,v0
677     1384:       0320f809        jalr    t9
678     1388:       00000000        nop
679     138c:       8fdc0010        lw      gp,16(s8)
680             unsigned int read = file_read(&opt->input_file, buf_decoded, 3);
681     1390:       8fc20040        lw      v0,64(s8)
682     1394:       27c30028        addiu   v1,s8,40
683     1398:       24060003        li      a2,3
684     139c:       00602825        move    a1,v1
685     13a0:       00402025        move    a0,v0
686     13a4:       8f82804c        lw      v0,-32692(gp)
687     13a8:       0040c825        move    t9,v0
688     13ac:       041102cb        bal     1edc <file_read>
689     13b0:       00000000        nop
690     13b4:       8fdc0010        lw      gp,16(s8)
691     13b8:       afc20020        sw      v0,32(s8)
692             if (read > 0) {
693     13bc:       8fc20020        lw      v0,32(s8)
694     13c0:       10400029        beqz    v0,1468 <_do_encode_decode+0x140>
695     13c4:       00000000        nop
696                 Encode(buf_decoded, read, buf_encoded);
697     13c8:       27c3002c        addiu   v1,s8,44
698     13cc:       27c20028        addiu   v0,s8,40
699     13d0:       00603025        move    a2,v1
700     13d4:       8fc50020        lw      a1,32(s8)
701     13d8:       00402025        move    a0,v0
702     13dc:       8f828050        lw      v0,-32688(gp)
703     13e0:       0040c825        move    t9,v0
704     13e4:       041100ce        bal     1720 <Encode>
705     13e8:       00000000        nop
706     13ec:       8fdc0010        lw      gp,16(s8)
707                 file_write(&opt->output_file, buf_encoded, 4);
708     13f0:       8fc20040        lw      v0,64(s8)
709     13f4:       24420008        addiu   v0,v0,8
710     13f8:       27c3002c        addiu   v1,s8,44
711     13fc:       24060004        li      a2,4
712     1400:       00602825        move    a1,v1
713     1404:       00402025        move    a0,v0
714     1408:       8f828054        lw      v0,-32684(gp)
715     140c:       0040c825        move    t9,v0
716     1410:       041102e6        bal     1fac <file_write>
717     1414:       00000000        nop
718     1418:       8fdc0010        lw      gp,16(s8)
719                 ++count;
720     141c:       93c20018        lbu     v0,24(s8)
721     1420:       24420001        addiu   v0,v0,1
722     1424:       a3c20018        sb      v0,24(s8)
723                 if (count == 18) // 19 * 4 =  76 bytes
724     1428:       93c30018        lbu     v1,24(s8)
725     142c:       24020012        li      v0,18
726     1430:       1462000d        bne     v1,v0,1468 <_do_encode_decode+0x140>
727     1434:       00000000        nop
728                 {
729                     file_write(&opt->output_file, (unsigned char *) "\n", 1);
730     1438:       8fc20040        lw      v0,64(s8)
731     143c:       24430008        addiu   v1,v0,8
732     1440:       24060001        li      a2,1
733     1444:       8f828030        lw      v0,-32720(gp)
734     1448:       244526c0        addiu   a1,v0,9920
```

```
735     144c:       00602025        move    a0,v1
736     1450:       8f828054        lw      v0,-32684(gp)
737     1454:       0040c825        move    t9,v0
738     1458:       041102d4        bal     1fac <file_write>
739     145c:       00000000        nop
740     1460:       8fdc0010        lw      gp,16(s8)
741                     count = 0;
742     1464:       a3c00018        sb      zero,24(s8)
743         while (!file_eof(&opt->input_file)) {
744     1468:       8fc20040        lw      v0,64(s8)
745     146c:       00402025        move    a0,v0
746     1470:       8f828058        lw      v0,-32680(gp)
747     1474:       0040c825        move    t9,v0
748     1478:       041102e9        bal     2020 <file_eof>
749     147c:       00000000        nop
750     1480:       8fdc0010        lw      gp,16(s8)
751     1484:       1040ffb9        beqz    v0,136c <_do_encode_decode+0x44>
752     1488:       00000000        nop
753                 }
754             }
755         }
756     }
757
758     if (opt->encode_option == DECODE) {
759     148c:       8fc20040        lw      v0,64(s8)
760     1490:       80420018        lb      v0,24(v0)
761     1494:       14400097        bnez    v0,16f4 <_do_encode_decode+0x3cc>
762     1498:       00000000        nop
763         while (!file_eof(&opt->input_file) && !has_errors(opt)) {
764     149c:       10000084        b       16b0 <_do_encode_decode+0x388>
765     14a0:       00000000        nop
766             unsigned int read = file_read(&opt->input_file, buf_encoded, 4);
767     14a4:       8fc20040        lw      v0,64(s8)
768     14a8:       27c3002c        addiu   v1,s8,44
769     14ac:       24060004        li      a2,4
770     14b0:       00602825        move    a1,v1
771     14b4:       00402025        move    a0,v0
772     14b8:       8f82804c        lw      v0,-32692(gp)
773     14bc:       0040c825        move    t9,v0
774     14c0:       04110286        bal     1edc <file_read>
775     14c4:       00000000        nop
776     14c8:       8fdc0010        lw      gp,16(s8)
777     14cc:       afc20024        sw      v0,36(s8)
778             if (read > 0) {
779     14d0:       8fc20024        lw      v0,36(s8)
780     14d4:       10400076        beqz    v0,16b0 <_do_encode_decode+0x388>
781     14d8:       00000000        nop
782                 if (read != 4) {
783     14dc:       8fc30024        lw      v1,36(s8)
784     14e0:       24020004        li      v0,4
785     14e4:       10620010        beq     v1,v0,1528 <_do_encode_decode+0x200>
786     14e8:       00000000        nop
787                     set_error(opt, INVALID_FILE_LENGTH);
788     14ec:       2405fffc        li      a1,-4
789     14f0:       8fc40040        lw      a0,64(s8)
790     14f4:       8f82805c        lw      v0,-32676(gp)
791     14f8:       0040c825        move    t9,v0
792     14fc:       0411fe69        bal     ea4 <set_error>
793     1500:       00000000        nop
794     1504:       8fdc0010        lw      gp,16(s8)
795                     show_error(opt);
796     1508:       8fc40040        lw      a0,64(s8)
797     150c:       8f828060        lw      v0,-32672(gp)
798     1510:       0040c825        move    t9,v0
799     1514:       0411fe83        bal     f24 <show_error>
800     1518:       00000000        nop
801     151c:       8fdc0010        lw      gp,16(s8)
```

```
802     1520:       10000063        b       16b0 <_do_encode_decode+0x388>
803     1524:       00000000        nop
804                 } else {
805                     ++count;
806     1528:       93c20018        lbu     v0,24(s8)
807     152c:       24420001        addiu   v0,v0,1
808     1530:       a3c20018        sb      v0,24(s8)
809                     if (count == 18) { // 19 * 4 =  76 bytes
810     1534:       93c30018        lbu     v1,24(s8)
811     1538:       24020012        li      v0,18
812     153c:       1462000c        bne     v1,v0,1570 <_do_encode_decode+0x248>
813     1540:       00000000        nop
814                         unsigned char aux;
815                         file_read(&opt->input_file, &aux, 1);
816     1544:       8fc20040        lw      v0,64(s8)
817     1548:       27c30030        addiu   v1,s8,48
818     154c:       24060001        li      a2,1
819     1550:       00602825        move    a1,v1
820     1554:       00402025        move    a0,v0
821     1558:       8f82804c        lw      v0,-32692(gp)
822     155c:       0040c825        move    t9,v0
823     1560:       0411025e        bal     1edc <file_read>
824     1564:       00000000        nop
825     1568:       8fdc0010        lw      gp,16(s8)
826                         count = 0;
827     156c:       a3c00018        sb      zero,24(s8)
828                     }
829
830                     if (Decode(buf_encoded, buf_decoded)) {
831     1570:       27c30028        addiu   v1,s8,40
832     1574:       27c2002c        addiu   v0,s8,44
833     1578:       00602825        move    a1,v1
834     157c:       00402025        move    a0,v0
835     1580:       8f828064        lw      v0,-32668(gp)
836     1584:       0040c825        move    t9,v0
837     1588:       04110114        bal     19dc <Decode>
838     158c:       00000000        nop
839     1590:       8fdc0010        lw      gp,16(s8)
840     1594:       10400022        beqz    v0,1620 <_do_encode_decode+0x2f8>
841     1598:       00000000        nop
842                         char aux = 0;
843     159c:       a3c00019        sb      zero,25(s8)
844                         if (buf_encoded[2] == '=') {
845     15a0:       93c3002e        lbu     v1,46(s8)
846     15a4:       2402003d        li      v0,61
847     15a8:       14620005        bne     v1,v0,15c0 <_do_encode_decode+0x298>
848     15ac:       00000000        nop
849                             ++aux;
850     15b0:       93c20019        lbu     v0,25(s8)
851     15b4:       24420001        addiu   v0,v0,1
852     15b8:       304200ff        andi    v0,v0,0xff
853     15bc:       a3c20019        sb      v0,25(s8)
854                         }
855                         if (buf_encoded[3] == '=') {
856     15c0:       93c3002f        lbu     v1,47(s8)
857     15c4:       2402003d        li      v0,61
858     15c8:       14620005        bne     v1,v0,15e0 <_do_encode_decode+0x2b8>
859     15cc:       00000000        nop
860                             ++aux;
861     15d0:       93c20019        lbu     v0,25(s8)
862     15d4:       24420001        addiu   v0,v0,1
863     15d8:       304200ff        andi    v0,v0,0xff
864     15dc:       a3c20019        sb      v0,25(s8)
865                         }
866
867                         file_write(&opt->output_file, buf_decoded, 3 - aux);
868     15e0:       8fc20040        lw      v0,64(s8)
```

```
869     15e4:       24440008        addiu   a0,v0,8
870     15e8:       83c20019        lb      v0,25(s8)
871     15ec:       24030003        li      v1,3
872     15f0:       00621023        subu    v0,v1,v0
873     15f4:       00401825        move    v1,v0
874     15f8:       27c20028        addiu   v0,s8,40
875     15fc:       00603025        move    a2,v1
876     1600:       00402825        move    a1,v0
877     1604:       8f828054        lw      v0,-32684(gp)
878     1608:       0040c825        move    t9,v0
879     160c:       04110267        bal     1fac <file_write>
880     1610:       00000000        nop
881     1614:       8fdc0010        lw      gp,16(s8)
882     1618:       10000025        b       16b0 <_do_encode_decode+0x388>
883     161c:       00000000        nop
884                         } else {
885                             set_error(opt, INVALID_CHARS);
886     1620:       2405fffb        li      a1,-5
887     1624:       8fc40040        lw      a0,64(s8)
888     1628:       8f82805c        lw      v0,-32676(gp)
889     162c:       0040c825        move    t9,v0
890     1630:       0411fe1c        bal     ea4 <set_error>
891     1634:       00000000        nop
892     1638:       8fdc0010        lw      gp,16(s8)
893                             show_error(opt);
894     163c:       8fc40040        lw      a0,64(s8)
895     1640:       8f828060        lw      v0,-32672(gp)
896     1644:       0040c825        move    t9,v0
897     1648:       0411fe36        bal     f24 <show_error>
898     164c:       00000000        nop
899     1650:       8fdc0010        lw      gp,16(s8)
900                             unsigned int i;
901                             for (i = 0; i < 4; ++i) {
902     1654:       afc0001c        sw      zero,28(s8)
903     1658:       10000011        b       16a0 <_do_encode_decode+0x378>
904     165c:       00000000        nop
905                                 fprintf(stderr, "%c", buf_encoded[i]);
906     1660:       8f8280f0        lw      v0,-32528(gp)
907     1664:       8c440000        lw      a0,0(v0)
908     1668:       8fc2001c        lw      v0,28(s8)
909     166c:       27c30018        addiu   v1,s8,24
910     1670:       00621021        addu    v0,v1,v0
911     1674:       90420014        lbu     v0,20(v0)
912     1678:       00802825        move    a1,a0
913     167c:       00402025        move    a0,v0
914     1680:       8f8280b4        lw      v0,-32588(gp)
915     1684:       0040c825        move    t9,v0
916     1688:       0320f809        jalr    t9
917     168c:       00000000        nop
918     1690:       8fdc0010        lw      gp,16(s8)
919                             for (i = 0; i < 4; ++i) {
920     1694:       8fc2001c        lw      v0,28(s8)
921     1698:       24420001        addiu   v0,v0,1
922     169c:       afc2001c        sw      v0,28(s8)
923     16a0:       8fc2001c        lw      v0,28(s8)
924     16a4:       2c420004        sltiu   v0,v0,4
925     16a8:       1440ffed        bnez    v0,1660 <_do_encode_decode+0x338>
926     16ac:       00000000        nop
927         while (!file_eof(&opt->input_file) && !has_errors(opt)) {
928     16b0:       8fc20040        lw      v0,64(s8)
929     16b4:       00402025        move    a0,v0
930     16b8:       8f828058        lw      v0,-32680(gp)
931     16bc:       0040c825        move    t9,v0
932     16c0:       04110257        bal     2020 <file_eof>
933     16c4:       00000000        nop
934     16c8:       8fdc0010        lw      gp,16(s8)
935     16cc:       14400009        bnez    v0,16f4 <_do_encode_decode+0x3cc>
```

```
936     16d0:       00000000        nop
937     16d4:       8fc40040        lw      a0,64(s8)
938     16d8:       8f828068        lw      v0,-32664(gp)
939     16dc:       0040c825        move    t9,v0
940     16e0:       0411fdff        bal     ee0 <has_errors>
941     16e4:       00000000        nop
942     16e8:       8fdc0010        lw      gp,16(s8)
943     16ec:       1040ff6d        beqz    v0,14a4 <_do_encode_decode+0x17c>
944     16f0:       00000000        nop
945
946             }
947         }
948     }
949
950     return opt->error_condition;
951     16f4:       8fc20040        lw      v0,64(s8)
952     16f8:       80420019        lb      v0,25(v0)
953 }
954     16fc:       03c0e825        move    sp,s8
955     1700:       8fbf003c        lw      ra,60(sp)
956     1704:       8fbe0038        lw      s8,56(sp)
957     1708:       27bd0040        addiu   sp,sp,64
958     170c:       03e00008        jr      ra
959     1710:       00000000        nop
960         ...
961
962 00001720 <Encode>:
963                                          '4', '5', '6', '7', '8', '9', '+', '/'};
964
965 static int encoding_table_size = 64;
966
967
968 void Encode(const unsigned char *buffer, unsigned int length, unsigned char *output) {
969     1720:       3c1c0002        lui     gp,0x2
970     1724:       279c9160        addiu   gp,gp,-28320
971     1728:       0399e021        addu    gp,gp,t9
972     172c:       27bdffe0        addiu   sp,sp,-32
973     1730:       afbe001c        sw      s8,28(sp)
974     1734:       03a0f025        move    s8,sp
975     1738:       afbc0000        sw      gp,0(sp)
976     173c:       afc40020        sw      a0,32(s8)
977     1740:       afc50024        sw      a1,36(s8)
978     1744:       afc60028        sw      a2,40(s8)
979     unsigned char b1 = buffer[0];
980     1748:       8fc20020        lw      v0,32(s8)
981     174c:       90420000        lbu     v0,0(v0)
982     1750:       a3c20008        sb      v0,8(s8)
983     unsigned char b2 = buffer[1];
984     1754:       8fc20020        lw      v0,32(s8)
985     1758:       90420001        lbu     v0,1(v0)
986     175c:       a3c20009        sb      v0,9(s8)
987     unsigned char b3 = buffer[2];
988     1760:       8fc20020        lw      v0,32(s8)
989     1764:       90420002        lbu     v0,2(v0)
990     1768:       a3c2000a        sb      v0,10(s8)
991     //I retrieve the first 6 bits and operate..
992     unsigned char b1aux = b1 >> 2;
993     176c:       93c20008        lbu     v0,8(s8)
994     1770:       00021082        srl     v0,v0,0x2
995     1774:       a3c2000b        sb      v0,11(s8)
996     //Recovered the first 6 bits, I look into the encoding in the table.
997     output[0] = encoding_table[(int) b1aux];
998     1778:       93c3000b        lbu     v1,11(s8)
999     177c:       8f828028        lw      v0,-32728(gp)
1000    1780:       244227e0        addiu   v0,v0,10208
1001    1784:       00621021        addu    v0,v1,v0
1002    1788:       90430000        lbu     v1,0(v0)
```

```
    178c:       8fc20028        lw      v0,40(s8)
    1790:       a0430000        sb      v1,0(v0)
    //I retrieve the next 6 bits.
    unsigned char b2aux = b1 << 6;
    1794:       93c20008        lbu     v0,8(s8)
    1798:       00021180        sll     v0,v0,0x6
    179c:       a3c2000c        sb      v0,12(s8)
    b2aux = b2aux >> 2;
    17a0:       93c2000c        lbu     v0,12(s8)
    17a4:       00021082        srl     v0,v0,0x2
    17a8:       a3c2000c        sb      v0,12(s8)
    b2aux = b2aux | (b2 >> 4);
    17ac:       93c20009        lbu     v0,9(s8)
    17b0:       00021102        srl     v0,v0,0x4
    17b4:       304300ff        andi    v1,v0,0xff
    17b8:       93c2000c        lbu     v0,12(s8)
    17bc:       00621025        or      v0,v1,v0
    17c0:       a3c2000c        sb      v0,12(s8)
    //I take look into the encoding table
    output[1] = encoding_table[(int) b2aux];
    17c4:       8fc20028        lw      v0,40(s8)
    17c8:       24420001        addiu   v0,v0,1
    17cc:       93c4000c        lbu     a0,12(s8)
    17d0:       8f838028        lw      v1,-32728(gp)
    17d4:       246327e0        addiu   v1,v1,10208
    17d8:       00831821        addu    v1,a0,v1
    17dc:       90630000        lbu     v1,0(v1)
    17e0:       a0430000        sb      v1,0(v0)
    output[2] = BASE64_END;
    17e4:       8fc20028        lw      v0,40(s8)
    17e8:       24420002        addiu   v0,v0,2
    17ec:       2403003d        li      v1,61
    17f0:       a0430000        sb      v1,0(v0)
    output[3] = BASE64_END;
    17f4:       8fc20028        lw      v0,40(s8)
    17f8:       24420003        addiu   v0,v0,3
    17fc:       2403003d        li      v1,61
    1800:       a0430000        sb      v1,0(v0)
    if (length == 3) {
    1804:       8fc30024        lw      v1,36(s8)
    1808:       24020003        li      v0,3
    180c:       14620026        bne     v1,v0,18a8 <Encode+0x188>
    1810:       00000000        nop
        /*
     * If I have 3 characters in the buffer I operate
     * with the last 2 characters.
     */
        unsigned char b3aux = b3 >> 6;
    1814:       93c2000a        lbu     v0,10(s8)
    1818:       00021182        srl     v0,v0,0x6
    181c:       a3c2000d        sb      v0,13(s8)
        unsigned char b3aux2 = b2 << 4;
    1820:       93c20009        lbu     v0,9(s8)
    1824:       00021100        sll     v0,v0,0x4
    1828:       a3c2000e        sb      v0,14(s8)
        b3aux2 = b3aux2 >> 2;
    182c:       93c2000e        lbu     v0,14(s8)
    1830:       00021082        srl     v0,v0,0x2
    1834:       a3c2000e        sb      v0,14(s8)
        b3aux = b3aux | b3aux2;
    1838:       93c3000d        lbu     v1,13(s8)
    183c:       93c2000e        lbu     v0,14(s8)
    1840:       00621025        or      v0,v1,v0
    1844:       a3c2000d        sb      v0,13(s8)
        //I take look into the encoding table.
        output[2] = encoding_table[(int) b3aux];
    1848:       8fc20028        lw      v0,40(s8)
```

```
1070      184c:       24420002        addiu   v0,v0,2
1071      1850:       93c4000d        lbu     a0,13(s8)
1072      1854:       8f838028        lw      v1,-32728(gp)
1073      1858:       246327e0        addiu   v1,v1,10208
1074      185c:       00831821        addu    v1,a0,v1
1075      1860:       90630000        lbu     v1,0(v1)
1076      1864:       a0430000        sb      v1,0(v0)
1077          unsigned char b4aux = b3 << 2;
1078      1868:       93c2000a        lbu     v0,10(s8)
1079      186c:       00021080        sll     v0,v0,0x2
1080      1870:       a3c2000f        sb      v0,15(s8)
1081          b4aux = b4aux >> 2;
1082      1874:       93c2000f        lbu     v0,15(s8)
1083      1878:       00021082        srl     v0,v0,0x2
1084      187c:       a3c2000f        sb      v0,15(s8)
1085          //I take look into the encoding table.
1086          output[3] = encoding_table[(int) b4aux];
1087      1880:       8fc20028        lw      v0,40(s8)
1088      1884:       24420003        addiu   v0,v0,3
1089      1888:       93c4000f        lbu     a0,15(s8)
1090      188c:       8f838028        lw      v1,-32728(gp)
1091      1890:       246327e0        addiu   v1,v1,10208
1092      1894:       00831821        addu    v1,a0,v1
1093      1898:       90630000        lbu     v1,0(v1)
1094      189c:       a0430000        sb      v1,0(v0)
1095              b3aux2 = b3aux2 >> 2;
1096              b3aux = b3aux | b3aux2;
1097              output[2] = encoding_table[(int) b3aux];
1098          }
1099      }
1100  }
1101      18a0:       1000001a        b       190c <Encode+0x1ec>
1102      18a4:       00000000        nop
1103          if (length == 2) {
1104      18a8:       8fc30024        lw      v1,36(s8)
1105      18ac:       24020002        li      v0,2
1106      18b0:       14620016        bne     v1,v0,190c <Encode+0x1ec>
1107      18b4:       00000000        nop
1108              unsigned char b3aux = b3 >> 6;
1109      18b8:       93c2000a        lbu     v0,10(s8)
1110      18bc:       00021182        srl     v0,v0,0x6
1111      18c0:       a3c20010        sb      v0,16(s8)
1112              unsigned char b3aux2 = b2 << 4;
1113      18c4:       93c20009        lbu     v0,9(s8)
1114      18c8:       00021100        sll     v0,v0,0x4
1115      18cc:       a3c20011        sb      v0,17(s8)
1116              b3aux2 = b3aux2 >> 2;
1117      18d0:       93c20011        lbu     v0,17(s8)
1118      18d4:       00021082        srl     v0,v0,0x2
1119      18d8:       a3c20011        sb      v0,17(s8)
1120              b3aux = b3aux | b3aux2;
1121      18dc:       93c30010        lbu     v1,16(s8)
1122      18e0:       93c20011        lbu     v0,17(s8)
1123      18e4:       00621025        or      v0,v1,v0
1124      18e8:       a3c20010        sb      v0,16(s8)
1125              output[2] = encoding_table[(int) b3aux];
1126      18ec:       8fc20028        lw      v0,40(s8)
1127      18f0:       24420002        addiu   v0,v0,2
1128      18f4:       93c40010        lbu     a0,16(s8)
1129      18f8:       8f838028        lw      v1,-32728(gp)
1130      18fc:       246327e0        addiu   v1,v1,10208
1131      1900:       00831821        addu    v1,a0,v1
1132      1904:       90630000        lbu     v1,0(v1)
1133      1908:       a0430000        sb      v1,0(v0)
1134  }
1135      190c:       00000000        nop
1136      1910:       03c0e825        move    sp,s8
```

```
1137     1914:       8fbe001c        lw      s8,28(sp)
1138     1918:       27bd0020        addiu   sp,sp,32
1139     191c:       03e00008        jr      ra
1140     1920:       00000000        nop
1141
1142 00001924 <DecodeChar>:
1143  * is the character '='.
1144  * post: returns the representation (int) of the character
1145  * in the encoding table.
1146  *
1147  */
1148 unsigned char DecodeChar(char character) {
1149     1924:       3c1c0002        lui     gp,0x2
1150     1928:       279c8f5c        addiu   gp,gp,-28836
1151     192c:       0399e021        addu    gp,gp,t9
1152     1930:       27bdffe8        addiu   sp,sp,-24
1153     1934:       afbe0014        sw      s8,20(sp)
1154     1938:       03a0f025        move    s8,sp
1155     193c:       afbc0000        sw      gp,0(sp)
1156     1940:       00801025        move    v0,a0
1157     1944:       a3c20018        sb      v0,24(s8)
1158     unsigned char i;
1159     for (i = 0; i < encoding_table_size; i++) {
1160     1948:       a3c00008        sb      zero,8(s8)
1161     194c:       10000010        b       1990 <DecodeChar+0x6c>
1162     1950:       00000000        nop
1163         if (encoding_table[i] == character) {
1164     1954:       93c30008        lbu     v1,8(s8)
1165     1958:       8f828028        lw      v0,-32728(gp)
1166     195c:       244227e0        addiu   v0,v0,10208
1167     1960:       00621021        addu    v0,v1,v0
1168     1964:       90420000        lbu     v0,0(v0)
1169     1968:       00401825        move    v1,v0
1170     196c:       83c20018        lb      v0,24(s8)
1171     1970:       14620004        bne     v1,v0,1984 <DecodeChar+0x60>
1172     1974:       00000000        nop
1173             return i;
1174     1978:       93c20008        lbu     v0,8(s8)
1175     197c:       10000012        b       19c8 <DecodeChar+0xa4>
1176     1980:       00000000        nop
1177     for (i = 0; i < encoding_table_size; i++) {
1178     1984:       93c20008        lbu     v0,8(s8)
1179     1988:       24420001        addiu   v0,v0,1
1180     198c:       a3c20008        sb      v0,8(s8)
1181     1990:       93c30008        lbu     v1,8(s8)
1182     1994:       8f828028        lw      v0,-32728(gp)
1183     1998:       8c422820        lw      v0,10272(v0)
1184     199c:       0062102a        slt     v0,v1,v0
1185     19a0:       1440ffec        bnez    v0,1954 <DecodeChar+0x30>
1186     19a4:       00000000        nop
1187         }
1188     }
1189     if (character == '=') {
1190     19a8:       83c30018        lb      v1,24(s8)
1191     19ac:       2402003d        li      v0,61
1192     19b0:       14620004        bne     v1,v0,19c4 <DecodeChar+0xa0>
1193     19b4:       00000000        nop
1194         return 0;
1195     19b8:       00001025        move    v0,zero
1196     19bc:       10000002        b       19c8 <DecodeChar+0xa4>
1197     19c0:       00000000        nop
1198     }
1199     return DECODE_ERROR;
1200     19c4:       24020064        li      v0,100
1201 }
1202     19c8:       03c0e825        move    sp,s8
1203     19cc:       8fbe0014        lw      s8,20(sp)
```

```
    19d0:       27bd0018        addiu   sp,sp,24
    19d4:       03e00008        jr      ra
    19d8:       00000000        nop

000019dc <Decode>:
/**
 * Returns a buffer with size 3 with the 4 character base64 decode.
 * Pre: the input buffer contains 4 characters. The output buffer has at least 3 characters
 * Post: returns a buffer with size 3 ASCII characters. returns 0 if error 1 if ok
 */
unsigned char Decode(unsigned char *buf_input, unsigned char *buf_output) {
    19dc:       3c1c0002        lui     gp,0x2
    19e0:       279c8ea4        addiu   gp,gp,-29020
    19e4:       0399e021        addu    gp,gp,t9
    19e8:       27bdffd0        addiu   sp,sp,-48
    19ec:       afbf002c        sw      ra,44(sp)
    19f0:       afbe0028        sw      s8,40(sp)
    19f4:       03a0f025        move    s8,sp
    19f8:       afbc0010        sw      gp,16(sp)
    19fc:       afc40030        sw      a0,48(s8)
    1a00:       afc50034        sw      a1,52(s8)
    unsigned char chars[4];
    unsigned int i;
    for (i = 0; i < 4; ++i) {
    1a04:       afc00018        sw      zero,24(s8)
    1a08:       1000001e        b       1a84 <Decode+0xa8>
    1a0c:       00000000        nop
        chars[i] = DecodeChar(buf_input[i]);
    1a10:       8fc30030        lw      v1,48(s8)
    1a14:       8fc20018        lw      v0,24(s8)
    1a18:       00621021        addu    v0,v1,v0
    1a1c:       90420000        lbu     v0,0(v0)
    1a20:       7c021420        seb     v0,v0
    1a24:       00402025        move    a0,v0
    1a28:       8f82806c        lw      v0,-32660(gp)
    1a2c:       0040c825        move    t9,v0
    1a30:       0411ffbc        bal     1924 <DecodeChar>
    1a34:       00000000        nop
    1a38:       8fdc0010        lw      gp,16(s8)
    1a3c:       00402025        move    a0,v0
    1a40:       8fc20018        lw      v0,24(s8)
    1a44:       27c30018        addiu   v1,s8,24
    1a48:       00621021        addu    v0,v1,v0
    1a4c:       a0440008        sb      a0,8(v0)
        if (chars[i] == DECODE_ERROR)
    1a50:       8fc20018        lw      v0,24(s8)
    1a54:       27c30018        addiu   v1,s8,24
    1a58:       00621021        addu    v0,v1,v0
    1a5c:       90430008        lbu     v1,8(v0)
    1a60:       24020064        li      v0,100
    1a64:       14620004        bne     v1,v0,1a78 <Decode+0x9c>
    1a68:       00000000        nop
            return 0;
    1a6c:       00001025        move    v0,zero
    1a70:       1000002e        b       1b2c <Decode+0x150>
    1a74:       00000000        nop
    for (i = 0; i < 4; ++i) {
    1a78:       8fc20018        lw      v0,24(s8)
    1a7c:       24420001        addiu   v0,v0,1
    1a80:       afc20018        sw      v0,24(s8)
    1a84:       8fc20018        lw      v0,24(s8)
    1a88:       2c420004        sltiu   v0,v0,4
    1a8c:       1440ffe0        bnez    v0,1a10 <Decode+0x34>
    1a90:       00000000        nop
    }

    unsigned char char1_aux = chars[0] << 2;
```

```
1271      1a94:       93c20020        lbu     v0,32(s8)
1272      1a98:       00021080        sll     v0,v0,0x2
1273      1a9c:       a3c2001c        sb      v0,28(s8)
1274      //Take the last 2 bits of char2
1275      unsigned char char2_aux = chars[1] >> 4;
1276      1aa0:       93c20021        lbu     v0,33(s8)
1277      1aa4:       00021102        srl     v0,v0,0x4
1278      1aa8:       a3c2001d        sb      v0,29(s8)
1279      char1_aux = char1_aux | char2_aux;
1280      1aac:       93c3001c        lbu     v1,28(s8)
1281      1ab0:       93c2001d        lbu     v0,29(s8)
1282      1ab4:       00621025        or      v0,v1,v0
1283      1ab8:       a3c2001c        sb      v0,28(s8)
1284      buf_output[0] = char1_aux;
1285      1abc:       8fc20034        lw      v0,52(s8)
1286      1ac0:       93c3001c        lbu     v1,28(s8)
1287      1ac4:       a0430000        sb      v1,0(v0)
1288
1289      //Take the last 4b of char2 and the first 4b of char3
1290      char1_aux = chars[1] << 4;
1291      1ac8:       93c20021        lbu     v0,33(s8)
1292      1acc:       00021100        sll     v0,v0,0x4
1293      1ad0:       a3c2001c        sb      v0,28(s8)
1294      char2_aux = chars[2] >> 2;
1295      1ad4:       93c20022        lbu     v0,34(s8)
1296      1ad8:       00021082        srl     v0,v0,0x2
1297      1adc:       a3c2001d        sb      v0,29(s8)
1298      char2_aux = char1_aux | char2_aux;
1299      1ae0:       93c3001c        lbu     v1,28(s8)
1300      1ae4:       93c2001d        lbu     v0,29(s8)
1301      1ae8:       00621025        or      v0,v1,v0
1302      1aec:       a3c2001d        sb      v0,29(s8)
1303      buf_output[1] = char2_aux;
1304      1af0:       8fc20034        lw      v0,52(s8)
1305      1af4:       24420001        addiu   v0,v0,1
1306      1af8:       93c3001d        lbu     v1,29(s8)
1307      1afc:       a0430000        sb      v1,0(v0)
1308
1309      //Take the last 2b of char3 + the bits of char4
1310      char1_aux = chars[2] << 6;
1311      1b00:       93c20022        lbu     v0,34(s8)
1312      1b04:       00021180        sll     v0,v0,0x6
1313      1b08:       a3c2001c        sb      v0,28(s8)
1314      buf_output[2] = char1_aux | chars[3];
1315      1b0c:       8fc20034        lw      v0,52(s8)
1316      1b10:       24420002        addiu   v0,v0,2
1317      1b14:       93c40023        lbu     a0,35(s8)
1318      1b18:       93c3001c        lbu     v1,28(s8)
1319      1b1c:       00831825        or      v1,a0,v1
1320      1b20:       306300ff        andi    v1,v1,0xff
1321      1b24:       a0430000        sb      v1,0(v0)
1322      return 1;
1323      1b28:       24020001        li      v0,1
1324  }
1325      1b2c:       03c0e825        move    sp,s8
1326      1b30:       8fbf002c        lw      ra,44(sp)
1327      1b34:       8fbe0028        lw      s8,40(sp)
1328      1b38:       27bd0030        addiu   sp,sp,48
1329      1b3c:       03e00008        jr      ra
1330      1b40:       00000000        nop
1331          ...
1332
1333  00001b50 <create_file>:
1334  #include "file.h"
1335  #include <stdlib.h>
1336  #include <string.h>
1337  #include <errno.h>
```

```
1338
1339 void create_file(File *file) {
1340     1b50:       27bdfff8        addiu   sp,sp,-8
1341     1b54:       afbe0004        sw      s8,4(sp)
1342     1b58:       03a0f025        move    s8,sp
1343     1b5c:       afc40008        sw      a0,8(s8)
1344     file->file = 0;
1345     1b60:       8fc20008        lw      v0,8(s8)
1346     1b64:       ac400000        sw      zero,0(v0)
1347     file->eof = 0;
1348     1b68:       8fc20008        lw      v0,8(s8)
1349     1b6c:       a0400004        sb      zero,4(v0)
1350 }
1351     1b70:       00000000        nop
1352     1b74:       03c0e825        move    sp,s8
1353     1b78:       8fbe0004        lw      s8,4(sp)
1354     1b7c:       27bd0008        addiu   sp,sp,8
1355     1b80:       03e00008        jr      ra
1356     1b84:       00000000        nop
1357
1358 00001b88 <open_file_read>:
1359
1360 char open_file_read(File *file, const char *route) {
1361     1b88:       3c1c0002        lui     gp,0x2
1362     1b8c:       279c8cf8        addiu   gp,gp,-29448
1363     1b90:       0399e021        addu    gp,gp,t9
1364     1b94:       27bdffd0        addiu   sp,sp,-48
1365     1b98:       afbf002c        sw      ra,44(sp)
1366     1b9c:       afbe0028        sw      s8,40(sp)
1367     1ba0:       afb00024        sw      s0,36(sp)
1368     1ba4:       03a0f025        move    s8,sp
1369     1ba8:       afbc0010        sw      gp,16(sp)
1370     1bac:       afc40030        sw      a0,48(s8)
1371     1bb0:       afc50034        sw      a1,52(s8)
1372     if (route == NULL) {
1373     1bb4:       8fc20034        lw      v0,52(s8)
1374     1bb8:       14400007        bnez    v0,1bd8 <open_file_read+0x50>
1375     1bbc:       00000000        nop
1376         file->file = stdin;
1377     1bc0:       8f8280c4        lw      v0,-32572(gp)
1378     1bc4:       8c430000        lw      v1,0(v0)
1379     1bc8:       8fc20030        lw      v0,48(s8)
1380     1bcc:       ac430000        sw      v1,0(v0)
1381     1bd0:       1000002b        b       1c80 <open_file_read+0xf8>
1382     1bd4:       00000000        nop
1383     } else {
1384         file->file = fopen(route, "rb");
1385     1bd8:       8f828030        lw      v0,-32720(gp)
1386     1bdc:       244526d0        addiu   a1,v0,9936
1387     1be0:       8fc40034        lw      a0,52(s8)
1388     1be4:       8f8280f4        lw      v0,-32524(gp)
1389     1be8:       0040c825        move    t9,v0
1390     1bec:       0320f809        jalr    t9
1391     1bf0:       00000000        nop
1392     1bf4:       8fdc0010        lw      gp,16(s8)
1393     1bf8:       00401825        move    v1,v0
1394     1bfc:       8fc20030        lw      v0,48(s8)
1395     1c00:       ac430000        sw      v1,0(v0)
1396         if (file->file == NULL) {
1397     1c04:       8fc20030        lw      v0,48(s8)
1398     1c08:       8c420000        lw      v0,0(v0)
1399     1c0c:       1440001c        bnez    v0,1c80 <open_file_read+0xf8>
1400     1c10:       00000000        nop
1401             int err = errno;
1402     1c14:       8f8280bc        lw      v0,-32580(gp)
1403     1c18:       0040c825        move    t9,v0
1404     1c1c:       0320f809        jalr    t9
```

```
1405      1c20:       00000000        nop
1406      1c24:       8fdc0010        lw      gp,16(s8)
1407      1c28:       8c420000        lw      v0,0(v0)
1408      1c2c:       afc20018        sw      v0,24(s8)
1409              fprintf(stderr, "Error al abrir archivo: %s\n", strerror(err));
1410      1c30:       8f8280f0        lw      v0,-32528(gp)
1411      1c34:       8c500000        lw      s0,0(v0)
1412      1c38:       8fc40018        lw      a0,24(s8)
1413      1c3c:       8f8280dc        lw      v0,-32548(gp)
1414      1c40:       0040c825        move    t9,v0
1415      1c44:       0320f809        jalr    t9
1416      1c48:       00000000        nop
1417      1c4c:       8fdc0010        lw      gp,16(s8)
1418      1c50:       00403025        move    a2,v0
1419      1c54:       8f828030        lw      v0,-32720(gp)
1420      1c58:       244526d4        addiu   a1,v0,9940
1421      1c5c:       02002025        move    a0,s0
1422      1c60:       8f8280c8        lw      v0,-32568(gp)
1423      1c64:       0040c825        move    t9,v0
1424      1c68:       0320f809        jalr    t9
1425      1c6c:       00000000        nop
1426      1c70:       8fdc0010        lw      gp,16(s8)
1427              return ERROR;
1428      1c74:       24020001        li      v0,1
1429      1c78:       10000002        b       1c84 <open_file_read+0xfc>
1430      1c7c:       00000000        nop
1431          }
1432      }
1433      return OK;
1434      1c80:       00001025        move    v0,zero
1435  }
1436      1c84:       03c0e825        move    sp,s8
1437      1c88:       8fbf002c        lw      ra,44(sp)
1438      1c8c:       8fbe0028        lw      s8,40(sp)
1439      1c90:       8fb00024        lw      s0,36(sp)
1440      1c94:       27bd0030        addiu   sp,sp,48
1441      1c98:       03e00008        jr      ra
1442      1c9c:       00000000        nop
1443
1444  00001ca0 <open_file_write>:
1445
1446
1447  char open_file_write(File *file, const char *route) {
1448      1ca0:       3c1c0002        lui     gp,0x2
1449      1ca4:       279c8be0        addiu   gp,gp,-29728
1450      1ca8:       0399e021        addu    gp,gp,t9
1451      1cac:       27bdffd0        addiu   sp,sp,-48
1452      1cb0:       afbf002c        sw      ra,44(sp)
1453      1cb4:       afbe0028        sw      s8,40(sp)
1454      1cb8:       afb00024        sw      s0,36(sp)
1455      1cbc:       03a0f025        move    s8,sp
1456      1cc0:       afbc0010        sw      gp,16(sp)
1457      1cc4:       afc40030        sw      a0,48(s8)
1458      1cc8:       afc50034        sw      a1,52(s8)
1459      if (route == NULL) {
1460      1ccc:       8fc20034        lw      v0,52(s8)
1461      1cd0:       14400007        bnez    v0,1cf0 <open_file_write+0x50>
1462      1cd4:       00000000        nop
1463          file->file = stdout;
1464      1cd8:       8f8280b8        lw      v0,-32584(gp)
1465      1cdc:       8c430000        lw      v1,0(v0)
1466      1ce0:       8fc20030        lw      v0,48(s8)
1467      1ce4:       ac430000        sw      v1,0(v0)
1468      1ce8:       1000002b        b       1d98 <open_file_write+0xf8>
1469      1cec:       00000000        nop
1470      } else {
1471          file->file = fopen(route, "wb");
```

```
1472     1cf0:       8f828030        lw      v0,-32720(gp)
1473     1cf4:       244526f0        addiu   a1,v0,9968
1474     1cf8:       8fc40034        lw      a0,52(s8)
1475     1cfc:       8f8280f4        lw      v0,-32524(gp)
1476     1d00:       0040c825        move    t9,v0
1477     1d04:       0320f809        jalr    t9
1478     1d08:       00000000        nop
1479     1d0c:       8fdc0010        lw      gp,16(s8)
1480     1d10:       00401825        move    v1,v0
1481     1d14:       8fc20030        lw      v0,48(s8)
1482     1d18:       ac430000        sw      v1,0(v0)
1483         if (file->file == NULL) {
1484     1d1c:       8fc20030        lw      v0,48(s8)
1485     1d20:       8c420000        lw      v0,0(v0)
1486     1d24:       1440001c        bnez    v0,1d98 <open_file_write+0xf8>
1487     1d28:       00000000        nop
1488             int err = errno;
1489     1d2c:       8f8280bc        lw      v0,-32580(gp)
1490     1d30:       0040c825        move    t9,v0
1491     1d34:       0320f809        jalr    t9
1492     1d38:       00000000        nop
1493     1d3c:       8fdc0010        lw      gp,16(s8)
1494     1d40:       8c420000        lw      v0,0(v0)
1495     1d44:       afc20018        sw      v0,24(s8)
1496             fprintf(stderr, "Error al abrir archivo: %s\n", strerror(err));
1497     1d48:       8f8280f0        lw      v0,-32528(gp)
1498     1d4c:       8c500000        lw      s0,0(v0)
1499     1d50:       8fc40018        lw      a0,24(s8)
1500     1d54:       8f8280dc        lw      v0,-32548(gp)
1501     1d58:       0040c825        move    t9,v0
1502     1d5c:       0320f809        jalr    t9
1503     1d60:       00000000        nop
1504     1d64:       8fdc0010        lw      gp,16(s8)
1505     1d68:       00403025        move    a2,v0
1506     1d6c:       8f828030        lw      v0,-32720(gp)
1507     1d70:       244526d4        addiu   a1,v0,9940
1508     1d74:       02002025        move    a0,s0
1509     1d78:       8f8280c8        lw      v0,-32568(gp)
1510     1d7c:       0040c825        move    t9,v0
1511     1d80:       0320f809        jalr    t9
1512     1d84:       00000000        nop
1513     1d88:       8fdc0010        lw      gp,16(s8)
1514             return ERROR;
1515     1d8c:       24020001        li      v0,1
1516     1d90:       10000002        b       1d9c <open_file_write+0xfc>
1517     1d94:       00000000        nop
1518         }
1519     }
1520     return OK;
1521     1d98:       00001025        move    v0,zero
1522 }
1523     1d9c:       03c0e825        move    sp,s8
1524     1da0:       8fbf002c        lw      ra,44(sp)
1525     1da4:       8fbe0028        lw      s8,40(sp)
1526     1da8:       8fb00024        lw      s0,36(sp)
1527     1dac:       27bd0030        addiu   sp,sp,48
1528     1db0:       03e00008        jr      ra
1529     1db4:       00000000        nop
1530
1531 00001db8 <close_file>:
1532
1533
1534 int close_file(File *file) {
1535     1db8:       3c1c0002        lui     gp,0x2
1536     1dbc:       279c8ac8        addiu   gp,gp,-30008
1537     1dc0:       0399e021        addu    gp,gp,t9
1538     1dc4:       27bdffd0        addiu   sp,sp,-48
```

```
1539      1dc8:       afbf002c        sw      ra,44(sp)
1540      1dcc:       afbe0028        sw      s8,40(sp)
1541      1dd0:       afb00024        sw      s0,36(sp)
1542      1dd4:       03a0f025        move    s8,sp
1543      1dd8:       afbc0010        sw      gp,16(sp)
1544      1ddc:       afc40030        sw      a0,48(s8)
1545      if (file->file == stdin || file->file == stdout) return OK;
1546      1de0:       8fc20030        lw      v0,48(s8)
1547      1de4:       8c430000        lw      v1,0(v0)
1548      1de8:       8f8280c4        lw      v0,-32572(gp)
1549      1dec:       8c420000        lw      v0,0(v0)
1550      1df0:       10620007        beq     v1,v0,1e10 <close_file+0x58>
1551      1df4:       00000000        nop
1552      1df8:       8fc20030        lw      v0,48(s8)
1553      1dfc:       8c430000        lw      v1,0(v0)
1554      1e00:       8f8280b8        lw      v0,-32584(gp)
1555      1e04:       8c420000        lw      v0,0(v0)
1556      1e08:       14620004        bne     v1,v0,1e1c <close_file+0x64>
1557      1e0c:       00000000        nop
1558      1e10:       00001025        move    v0,zero
1559      1e14:       1000002a        b       1ec0 <close_file+0x108>
1560      1e18:       00000000        nop
1561
1562      int result = fclose(file->file);
1563      1e1c:       8fc20030        lw      v0,48(s8)
1564      1e20:       8c420000        lw      v0,0(v0)
1565      1e24:       00402025        move    a0,v0
1566      1e28:       8f8280ac        lw      v0,-32596(gp)
1567      1e2c:       0040c825        move    t9,v0
1568      1e30:       0320f809        jalr    t9
1569      1e34:       00000000        nop
1570      1e38:       8fdc0010        lw      gp,16(s8)
1571      1e3c:       afc20018        sw      v0,24(s8)
1572      if (result == EOF) {
1573      1e40:       8fc30018        lw      v1,24(s8)
1574      1e44:       2402ffff        li      v0,-1
1575      1e48:       1462001c        bne     v1,v0,1ebc <close_file+0x104>
1576      1e4c:       00000000        nop
1577          int err = errno;
1578      1e50:       8f8280bc        lw      v0,-32580(gp)
1579      1e54:       0040c825        move    t9,v0
1580      1e58:       0320f809        jalr    t9
1581      1e5c:       00000000        nop
1582      1e60:       8fdc0010        lw      gp,16(s8)
1583      1e64:       8c420000        lw      v0,0(v0)
1584      1e68:       afc2001c        sw      v0,28(s8)
1585          fprintf(stderr, "Error al cerrar archivo: %s\n", strerror(err));
1586      1e6c:       8f8280f0        lw      v0,-32528(gp)
1587      1e70:       8c500000        lw      s0,0(v0)
1588      1e74:       8fc4001c        lw      a0,28(s8)
1589      1e78:       8f8280dc        lw      v0,-32548(gp)
1590      1e7c:       0040c825        move    t9,v0
1591      1e80:       0320f809        jalr    t9
1592      1e84:       00000000        nop
1593      1e88:       8fdc0010        lw      gp,16(s8)
1594      1e8c:       00403025        move    a2,v0
1595      1e90:       8f828030        lw      v0,-32720(gp)
1596      1e94:       244526f4        addiu   a1,v0,9972
1597      1e98:       02002025        move    a0,s0
1598      1e9c:       8f8280c8        lw      v0,-32568(gp)
1599      1ea0:       0040c825        move    t9,v0
1600      1ea4:       0320f809        jalr    t9
1601      1ea8:       00000000        nop
1602      1eac:       8fdc0010        lw      gp,16(s8)
1603          return ERROR;
1604      1eb0:       24020001        li      v0,1
1605      1eb4:       10000002        b       1ec0 <close_file+0x108>
```

```
1606     1eb8:       00000000        nop
1607     }
1608     return OK;
1609     1ebc:       00001025        move    v0,zero
1610 }
1611     1ec0:       03c0e825        move    sp,s8
1612     1ec4:       8fbf002c        lw      ra,44(sp)
1613     1ec8:       8fbe0028        lw      s8,40(sp)
1614     1ecc:       8fb00024        lw      s0,36(sp)
1615     1ed0:       27bd0030        addiu   sp,sp,48
1616     1ed4:       03e00008        jr      ra
1617     1ed8:       00000000        nop
1618
1619 00001edc <file_read>:
1620
1621
1622 unsigned int file_read(File *file, unsigned char *buffer, unsigned int length) {
1623     1edc:       3c1c0002        lui     gp,0x2
1624     1ee0:       279c89a4        addiu   gp,gp,-30300
1625     1ee4:       0399e021        addu    gp,gp,t9
1626     1ee8:       27bdffd8        addiu   sp,sp,-40
1627     1eec:       afbf0024        sw      ra,36(sp)
1628     1ef0:       afbe0020        sw      s8,32(sp)
1629     1ef4:       03a0f025        move    s8,sp
1630     1ef8:       afbc0010        sw      gp,16(sp)
1631     1efc:       afc40028        sw      a0,40(s8)
1632     1f00:       afc5002c        sw      a1,44(s8)
1633     1f04:       afc60030        sw      a2,48(s8)
1634     unsigned int result = 0;
1635     1f08:       afc00018        sw      zero,24(s8)
1636     if (!file_eof(file)) {
1637     1f0c:       8fc40028        lw      a0,40(s8)
1638     1f10:       8f828058        lw      v0,-32680(gp)
1639     1f14:       0040c825        move    t9,v0
1640     1f18:       04110041        bal     2020 <file_eof>
1641     1f1c:       00000000        nop
1642     1f20:       8fdc0010        lw      gp,16(s8)
1643     1f24:       1440001a        bnez    v0,1f90 <file_read+0xb4>
1644     1f28:       00000000        nop
1645         result = (unsigned int) fread(buffer, sizeof(char), length, file->file);
1646     1f2c:       8fc20028        lw      v0,40(s8)
1647     1f30:       8c420000        lw      v0,0(v0)
1648     1f34:       00403825        move    a3,v0
1649     1f38:       8fc60030        lw      a2,48(s8)
1650     1f3c:       24050001        li      a1,1
1651     1f40:       8fc4002c        lw      a0,44(s8)
1652     1f44:       8f8280e4        lw      v0,-32540(gp)
1653     1f48:       0040c825        move    t9,v0
1654     1f4c:       0320f809        jalr    t9
1655     1f50:       00000000        nop
1656     1f54:       8fdc0010        lw      gp,16(s8)
1657     1f58:       afc20018        sw      v0,24(s8)
1658         if (feof(file->file)) {
1659     1f5c:       8fc20028        lw      v0,40(s8)
1660     1f60:       8c420000        lw      v0,0(v0)
1661     1f64:       00402025        move    a0,v0
1662     1f68:       8f8280d4        lw      v0,-32556(gp)
1663     1f6c:       0040c825        move    t9,v0
1664     1f70:       0320f809        jalr    t9
1665     1f74:       00000000        nop
1666     1f78:       8fdc0010        lw      gp,16(s8)
1667     1f7c:       10400004        beqz    v0,1f90 <file_read+0xb4>
1668     1f80:       00000000        nop
1669             file->eof = 1;
1670     1f84:       8fc20028        lw      v0,40(s8)
1671     1f88:       24030001        li      v1,1
1672     1f8c:       a0430004        sb      v1,4(v0)
```

```
1673            }
1674        }
1675
1676        return result;
1677        1f90:       8fc20018        lw      v0,24(s8)
1678    }
1679        1f94:       03c0e825        move    sp,s8
1680        1f98:       8fbf0024        lw      ra,36(sp)
1681        1f9c:       8fbe0020        lw      s8,32(sp)
1682        1fa0:       27bd0028        addiu   sp,sp,40
1683        1fa4:       03e00008        jr      ra
1684        1fa8:       00000000        nop
1685
1686    00001fac <file_write>:
1687
1688
1689    void file_write(File *file, unsigned char *buffer, unsigned int length) {
1690        1fac:       3c1c0002        lui     gp,0x2
1691        1fb0:       279c88d4        addiu   gp,gp,-30508
1692        1fb4:       0399e021        addu    gp,gp,t9
1693        1fb8:       27bdffe0        addiu   sp,sp,-32
1694        1fbc:       afbf001c        sw      ra,28(sp)
1695        1fc0:       afbe0018        sw      s8,24(sp)
1696        1fc4:       03a0f025        move    s8,sp
1697        1fc8:       afbc0010        sw      gp,16(sp)
1698        1fcc:       afc40020        sw      a0,32(s8)
1699        1fd0:       afc50024        sw      a1,36(s8)
1700        1fd4:       afc60028        sw      a2,40(s8)
1701        fwrite(buffer, sizeof(char), length, file->file);
1702        1fd8:       8fc20020        lw      v0,32(s8)
1703        1fdc:       8c420000        lw      v0,0(v0)
1704        1fe0:       00403825        move    a3,v0
1705        1fe4:       8fc60028        lw      a2,40(s8)
1706        1fe8:       24050001        li      a1,1
1707        1fec:       8fc40024        lw      a0,36(s8)
1708        1ff0:       8f8280e8        lw      v0,-32536(gp)
1709        1ff4:       0040c825        move    t9,v0
1710        1ff8:       0320f809        jalr    t9
1711        1ffc:       00000000        nop
1712        2000:       8fdc0010        lw      gp,16(s8)
1713    }
1714        2004:       00000000        nop
1715        2008:       03c0e825        move    sp,s8
1716        200c:       8fbf001c        lw      ra,28(sp)
1717        2010:       8fbe0018        lw      s8,24(sp)
1718        2014:       27bd0020        addiu   sp,sp,32
1719        2018:       03e00008        jr      ra
1720        201c:       00000000        nop
1721
1722    00002020 <file_eof>:
1723
1724
1725    int file_eof(File *file) {
1726        2020:       27bdfff8        addiu   sp,sp,-8
1727        2024:       afbe0004        sw      s8,4(sp)
1728        2028:       03a0f025        move    s8,sp
1729        202c:       afc40008        sw      a0,8(s8)
1730        return file->eof;
1731        2030:       8fc20008        lw      v0,8(s8)
1732        2034:       80420004        lb      v0,4(v0)
1733    }
1734        2038:       03c0e825        move    sp,s8
1735        203c:       8fbe0004        lw      s8,4(sp)
1736        2040:       27bd0008        addiu   sp,sp,8
1737        2044:       03e00008        jr      ra
1738        2048:       00000000        nop
1739        204c:       00000000        nop
```

```
00002050 <main>:
#include <stddef.h>
#include <getopt.h>
#include "constants.h"
#include "command.h"

int main(int argc, char **argv) {
    2050:       3c1c0002        lui     gp,0x2
    2054:       279c8830        addiu   gp,gp,-30672
    2058:       0399e021        addu    gp,gp,t9
    205c:       27bdff58        addiu   sp,sp,-168
    2060:       afbf00a4        sw      ra,164(sp)
    2064:       afbe00a0        sw      s8,160(sp)
    2068:       03a0f025        move    s8,sp
    206c:       afbc0018        sw      gp,24(sp)
    2070:       afc400a8        sw      a0,168(s8)
    2074:       afc500ac        sw      a1,172(s8)
    struct option arg_long[] = {
    2078:       8f828028        lw      v0,-32728(gp)
    207c:       27c30028        addiu   v1,s8,40
    2080:       24422830        addiu   v0,v0,10288
    2084:       24040050        li      a0,80
    2088:       00803025        move    a2,a0
    208c:       00402825        move    a1,v0
    2090:       00602025        move    a0,v1
    2094:       8f8280ec        lw      v0,-32532(gp)
    2098:       0040c825        move    t9,v0
    209c:       0320f809        jalr    t9
    20a0:       00000000        nop
    20a4:       8fdc0018        lw      gp,24(s8)
            {"decode",  no_argument, NULL, 'd'},
            {"help",    no_argument, NULL, 'h'},
            {"version", no_argument, NULL, 'V'},
    };

    char arg_opt_str[] = "i:o:dhV";
    20a8:       8f828030        lw      v0,-32720(gp)
    20ac:       8c432748        lw      v1,10056(v0)
    20b0:       24422748        addiu   v0,v0,10056
    20b4:       8c420004        lw      v0,4(v0)
    20b8:       afc30078        sw      v1,120(s8)
    20bc:       afc2007c        sw      v0,124(s8)
    int arg_opt;
    int arg_opt_idx = 0;
    20c0:       afc00080        sw      zero,128(s8)

    command_options_st cmd_options;
    // Default Values: encode, stdin as input, stdout as output, stderr as error output
    command_create(&cmd_options);
    20c4:       27c20084        addiu   v0,s8,132
    20c8:       00402025        move    a0,v0
    20cc:       8f828070        lw      v0,-32656(gp)
    20d0:       0040c825        move    t9,v0
    20d4:       0411fb16        bal     d30 <command_create>
    20d8:       00000000        nop
    20dc:       8fdc0018        lw      gp,24(s8)

    char should_process = TRUE;
    20e0:       24020001        li      v0,1
    20e4:       a3c20020        sb      v0,32(s8)
    while ((arg_opt = getopt_long(argc, argv, arg_opt_str, arg_long, &arg_opt_idx)) != -1
    20e8:       10000048        b       220c <main+0x1bc>
    20ec:       00000000        nop
           &&
           should_process) {

```

```
1807            switch (arg_opt) {
1808        20f0:       8fc20024        lw      v0,36(s8)
1809        20f4:       2442ffaa        addiu   v0,v0,-86
1810        20f8:       2c43001a        sltiu   v1,v0,26
1811        20fc:       1060003a        beqz    v1,21e8 <main+0x198>
1812        2100:       00000000        nop
1813        2104:       00021880        sll     v1,v0,0x2
1814        2108:       8f828030        lw      v0,-32720(gp)
1815        210c:       24422750        addiu   v0,v0,10064
1816        2110:       00621021        addu    v0,v1,v0
1817        2114:       8c420000        lw      v0,0(v0)
1818        2118:       005c1021        addu    v0,v0,gp
1819        211c:       00400008        jr      v0
1820        2120:       00000000        nop
1821                case 'i': {
1822                    // Set Input File
1823                    set_input_file(&cmd_options, optarg);
1824        2124:       8f8280a4        lw      v0,-32604(gp)
1825        2128:       8c430000        lw      v1,0(v0)
1826        212c:       27c20084        addiu   v0,s8,132
1827        2130:       00602825        move    a1,v1
1828        2134:       00402025        move    a0,v0
1829        2138:       8f828074        lw      v0,-32652(gp)
1830        213c:       0040c825        move    t9,v0
1831        2140:       0411fb23        bal     dd0 <set_input_file>
1832        2144:       00000000        nop
1833        2148:       8fdc0018        lw      gp,24(s8)
1834                }
1835                    break;
1836        214c:       1000002f        b       220c <main+0x1bc>
1837        2150:       00000000        nop
1838                case 'o': {
1839                    // Set Output File
1840                    set_output_file(&cmd_options, optarg);
1841        2154:       8f8280a4        lw      v0,-32604(gp)
1842        2158:       8c430000        lw      v1,0(v0)
1843        215c:       27c20084        addiu   v0,s8,132
1844        2160:       00602825        move    a1,v1
1845        2164:       00402025        move    a0,v0
1846        2168:       8f828078        lw      v0,-32648(gp)
1847        216c:       0040c825        move    t9,v0
1848        2170:       0411fb25        bal     e08 <set_output_file>
1849        2174:       00000000        nop
1850        2178:       8fdc0018        lw      gp,24(s8)
1851                }
1852                    break;
1853        217c:       10000023        b       220c <main+0x1bc>
1854        2180:       00000000        nop
1855                case 'h': {
1856                    // Show Options
1857                    show_help();
1858        2184:       8f828038        lw      v0,-32712(gp)
1859        2188:       0040c825        move    t9,v0
1860        218c:       0411fbb2        bal     1058 <show_help>
1861        2190:       00000000        nop
1862        2194:       8fdc0018        lw      gp,24(s8)
1863                    should_process = FALSE;
1864        2198:       a3c00020        sb      zero,32(s8)
1865                }
1866                    break;
1867        219c:       1000001b        b       220c <main+0x1bc>
1868        21a0:       00000000        nop
1869                case 'V': {
1870                    // Show Version
1871                    show_version();
1872        21a4:       8f82807c        lw      v0,-32644(gp)
1873        21a8:       0040c825        move    t9,v0
```

```
1874    21ac:       0411fbf8        bal     1190 <show_version>
1875    21b0:       00000000        nop
1876    21b4:       8fdc0018        lw      gp,24(s8)
1877                should_process = FALSE;
1878    21b8:       a3c00020        sb      zero,32(s8)
1879            }
1880                break;
1881    21bc:       10000013        b       220c <main+0x1bc>
1882    21c0:       00000000        nop
1883            case 'd': {
1884                // Set Decode option
1885                set_decode(&cmd_options);
1886    21c4:       27c20084        addiu   v0,s8,132
1887    21c8:       00402025        move    a0,v0
1888    21cc:       8f828080        lw      v0,-32640(gp)
1889    21d0:       0040c825        move    t9,v0
1890    21d4:       0411fb27        bal     e74 <set_decode>
1891    21d8:       00000000        nop
1892    21dc:       8fdc0018        lw      gp,24(s8)
1893            }
1894                break;
1895    21e0:       1000000a        b       220c <main+0x1bc>
1896    21e4:       00000000        nop
1897            default: {
1898                // Set Error Condition = INVALID_ARGUMENT
1899                // [Gonzalo: We analize only valid arguments values here: i, o, h, V, d]
1900                set_error(&cmd_options, INVALID_ARGUMENT);
1901    21e8:       27c20084        addiu   v0,s8,132
1902    21ec:       2405ffff        li      a1,-1
1903    21f0:       00402025        move    a0,v0
1904    21f4:       8f82805c        lw      v0,-32676(gp)
1905    21f8:       0040c825        move    t9,v0
1906    21fc:       0411fb29        bal     ea4 <set_error>
1907    2200:       00000000        nop
1908    2204:       8fdc0018        lw      gp,24(s8)
1909            }
1910                break;
1911    2208:       00000000        nop
1912    while ((arg_opt = getopt_long(argc, argv, arg_opt_str, arg_long, &arg_opt_idx)) != -1
1913    220c:       27c40028        addiu   a0,s8,40
1914    2210:       27c30078        addiu   v1,s8,120
1915    2214:       27c20080        addiu   v0,s8,128
1916    2218:       afa20010        sw      v0,16(sp)
1917    221c:       00803825        move    a3,a0
1918    2220:       00603025        move    a2,v1
1919    2224:       8fc500ac        lw      a1,172(s8)
1920    2228:       8fc400a8        lw      a0,168(s8)
1921    222c:       8f8280d0        lw      v0,-32560(gp)
1922    2230:       0040c825        move    t9,v0
1923    2234:       0320f809        jalr    t9
1924    2238:       00000000        nop
1925    223c:       8fdc0018        lw      gp,24(s8)
1926    2240:       afc20024        sw      v0,36(s8)
1927    2244:       8fc30024        lw      v1,36(s8)
1928    2248:       2402ffff        li      v0,-1
1929    224c:       10620004        beq     v1,v0,2260 <main+0x210>
1930    2250:       00000000        nop
1931            &&
1932    2254:       83c20020        lb      v0,32(s8)
1933    2258:       1440ffa5        bnez    v0,20f0 <main+0xa0>
1934    225c:       00000000        nop
1935        }
1936    }
1937
1938    // Help or Version arguments, no processing
1939    if (!should_process) {
1940    2260:       83c20020        lb      v0,32(s8)
```

```
1941      2264:       14400004        bnez    v0,2278 <main+0x228>
1942      2268:       00000000        nop
1943          return 0;
1944      226c:       00001025        move    v0,zero
1945      2270:       1000001b        b       22e0 <main+0x290>
1946      2274:       00000000        nop
1947      }
1948
1949      // Processs Encode/Decode if no errors found, otherwise show error message
1950      if (has_errors(&cmd_options)) {
1951      2278:       27c20084        addiu   v0,s8,132
1952      227c:       00402025        move    a0,v0
1953      2280:       8f828068        lw      v0,-32664(gp)
1954      2284:       0040c825        move    t9,v0
1955      2288:       0411fb15        bal     ee0 <has_errors>
1956      228c:       00000000        nop
1957      2290:       8fdc0018        lw      gp,24(s8)
1958      2294:       1040000b        beqz    v0,22c4 <main+0x274>
1959      2298:       00000000        nop
1960          show_error(&cmd_options);
1961      229c:       27c20084        addiu   v0,s8,132
1962      22a0:       00402025        move    a0,v0
1963      22a4:       8f828060        lw      v0,-32672(gp)
1964      22a8:       0040c825        move    t9,v0
1965      22ac:       0411fb1d        bal     f24 <show_error>
1966      22b0:       00000000        nop
1967      22b4:       8fdc0018        lw      gp,24(s8)
1968          return 1;
1969      22b8:       24020001        li      v0,1
1970      22bc:       10000008        b       22e0 <main+0x290>
1971      22c0:       00000000        nop
1972      }
1973
1974      return process(&cmd_options);
1975      22c4:       27c20084        addiu   v0,s8,132
1976      22c8:       00402025        move    a0,v0
1977      22cc:       8f828084        lw      v0,-32636(gp)
1978      22d0:       0040c825        move    t9,v0
1979      22d4:       0411fbc4        bal     11e8 <process>
1980      22d8:       00000000        nop
1981      22dc:       8fdc0018        lw      gp,24(s8)
1982  }
1983      22e0:       03c0e825        move    sp,s8
1984      22e4:       8fbf00a4        lw      ra,164(sp)
1985      22e8:       8fbe00a0        lw      s8,160(sp)
1986      22ec:       27bd00a8        addiu   sp,sp,168
1987      22f0:       03e00008        jr      ra
1988      22f4:       00000000        nop
1989          ...
1990
1991  00002300 <__libc_csu_init>:
1992      2300:       3c1c0002        lui     gp,0x2
1993      2304:       279c8580        addiu   gp,gp,-31360
1994      2308:       0399e021        addu    gp,gp,t9
1995      230c:       27bdffc8        addiu   sp,sp,-56
1996      2310:       8f998088        lw      t9,-32632(gp)
1997      2314:       afbc0010        sw      gp,16(sp)
1998      2318:       afb50030        sw      s5,48(sp)
1999      231c:       00c0a825        move    s5,a2
2000      2320:       afb4002c        sw      s4,44(sp)
2001      2324:       00a0a025        move    s4,a1
2002      2328:       afb30028        sw      s3,40(sp)
2003      232c:       00809825        move    s3,a0
2004      2330:       afb20024        sw      s2,36(sp)
2005      2334:       afb0001c        sw      s0,28(sp)
2006      2338:       afbf0034        sw      ra,52(sp)
2007      233c:       0411f9fa        bal     b28 <_init>
```

```
    2340:       afb10020        sw      s1,32(sp)
    2344:       8fbc0010        lw      gp,16(sp)
    2348:       8f90808c        lw      s0,-32628(gp)
    234c:       8f928090        lw      s2,-32624(gp)
    2350:       02509023        subu    s2,s2,s0
    2354:       00129083        sra     s2,s2,0x2
    2358:       12400009        beqz    s2,2380 <__libc_csu_init+0x80>
    235c:       00008825        move    s1,zero
    2360:       8e190000        lw      t9,0(s0)
    2364:       26310001        addiu   s1,s1,1
    2368:       02a03025        move    a2,s5
    236c:       02802825        move    a1,s4
    2370:       0320f809        jalr    t9
    2374:       02602025        move    a0,s3
    2378:       1651fff9        bne     s2,s1,2360 <__libc_csu_init+0x60>
    237c:       26100004        addiu   s0,s0,4
    2380:       8fbf0034        lw      ra,52(sp)
    2384:       8fb50030        lw      s5,48(sp)
    2388:       8fb4002c        lw      s4,44(sp)
    238c:       8fb30028        lw      s3,40(sp)
    2390:       8fb20024        lw      s2,36(sp)
    2394:       8fb10020        lw      s1,32(sp)
    2398:       8fb0001c        lw      s0,28(sp)
    239c:       03e00008        jr      ra
    23a0:       27bd0038        addiu   sp,sp,56

000023a4 <__libc_csu_fini>:
    23a4:       03e00008        jr      ra
    23a8:       00000000        nop
    23ac:       00000000        nop

Disassembly of section .MIPS.stubs:

000023b0 <_MIPS_STUBS_>:
    23b0:       8f998010        lw      t9,-32752(gp)
    23b4:       03e07825        move    t7,ra
    23b8:       0320f809        jalr    t9
    23bc:       24180033        li      t8,51
    23c0:       8f998010        lw      t9,-32752(gp)
    23c4:       03e07825        move    t7,ra
    23c8:       0320f809        jalr    t9
    23cc:       24180031        li      t8,49
    23d0:       8f998010        lw      t9,-32752(gp)
    23d4:       03e07825        move    t7,ra
    23d8:       0320f809        jalr    t9
    23dc:       24180030        li      t8,48
    23e0:       8f998010        lw      t9,-32752(gp)
    23e4:       03e07825        move    t7,ra
    23e8:       0320f809        jalr    t9
    23ec:       2418002f        li      t8,47
    23f0:       8f998010        lw      t9,-32752(gp)
    23f4:       03e07825        move    t7,ra
    23f8:       0320f809        jalr    t9
    23fc:       2418002e        li      t8,46
    2400:       8f998010        lw      t9,-32752(gp)
    2404:       03e07825        move    t7,ra
    2408:       0320f809        jalr    t9
    240c:       2418002d        li      t8,45
    2410:       8f998010        lw      t9,-32752(gp)
    2414:       03e07825        move    t7,ra
    2418:       0320f809        jalr    t9
    241c:       2418002b        li      t8,43
    2420:       8f998010        lw      t9,-32752(gp)
    2424:       03e07825        move    t7,ra
    2428:       0320f809        jalr    t9
    242c:       2418002a        li      t8,42
    2430:       8f998010        lw      t9,-32752(gp)
```

```
    2434:       03e07825        move    t7,ra
    2438:       0320f809        jalr    t9
    243c:       24180029        li      t8,41
    2440:       8f998010        lw      t9,-32752(gp)
    2444:       03e07825        move    t7,ra
    2448:       0320f809        jalr    t9
    244c:       24180028        li      t8,40
    2450:       8f998010        lw      t9,-32752(gp)
    2454:       03e07825        move    t7,ra
    2458:       0320f809        jalr    t9
    245c:       24180026        li      t8,38
    2460:       8f998010        lw      t9,-32752(gp)
    2464:       03e07825        move    t7,ra
    2468:       0320f809        jalr    t9
    246c:       24180025        li      t8,37
    2470:       8f998010        lw      t9,-32752(gp)
    2474:       03e07825        move    t7,ra
    2478:       0320f809        jalr    t9
    247c:       24180023        li      t8,35
    2480:       8f998010        lw      t9,-32752(gp)
    2484:       03e07825        move    t7,ra
    2488:       0320f809        jalr    t9
    248c:       24180021        li      t8,33
    2490:       8f998010        lw      t9,-32752(gp)
    2494:       03e07825        move    t7,ra
    2498:       0320f809        jalr    t9
    249c:       2418001e        li      t8,30
        ...

Disassembly of section .fini:

000024b0 <_fini>:
    24b0:       3c1c0002        lui     gp,0x2
    24b4:       279c83d0        addiu   gp,gp,-31792
    24b8:       0399e021        addu    gp,gp,t9
    24bc:       27bdffe0        addiu   sp,sp,-32
    24c0:       afbc0010        sw      gp,16(sp)
    24c4:       afbf001c        sw      ra,28(sp)
    24c8:       8fbf001c        lw      ra,28(sp)
    24cc:       03e00008        jr      ra
    24d0:       27bd0020        addiu   sp,sp,32
```