Fiuba
Facultad de Ingeniería
Universidad de Buenos Aires

U.B.A. FACULTAD DE INGENIERÍA

Departamento de Electrónica

Organización de computadoras 66-20

TRABAJO PRÁCTICO #1

Conjunto de instrucciones MIPS

Curso: 2018 - 2do Cuatrimestre

Turno: Martes

GRUPO N°	
Integrantes	**Padrón**
Verón, Lucas	89341
Gamarra Silva, Cynthia Marlene	92702
Gatti, Nicolás	93570
Fecha de entrega:	16-10-2018
Fecha de aprobación:	
Calificación:	
Firma de aprobación:	

Observaciones:

Índice

1. Enunciado del trabajo práctico

66.20 Organización de Computadoras
Trabajo práctico 1: conjunto de instrucciones MIPS
`$Date: 2018/10/14 03:07:24 $`

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto en la sección 4.

2. Alcance

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

3. Requisitos

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

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 6), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada caso.

4. Descripción

En este trabajo, se reimplementará parcialmente en assembly MIPS el programa desarrollado en el trabajo práctico anterior [1].

Para esto, se requiere reescribir el programa, de forma tal que quede organizado de la siguiente forma:

- `main.c`: contendrá todo el código necesario para el procesamiento de las opciones de línea de comandos, apertura y cierre de archivos (de ser necesario), y reporte de errores (`stderr`). Desde aquí se llama a las funciones de encoding y decoding siguientes.

- `base64.S`: contendrá el código MIPS32 assembly con las funciones `base64_encode()` y `base64_decode()`, y las funciones y estructuras de datos auxiliares para realizar los cómputo de encoding y decoding, que los alumnos crean convenientes. También contendrá la definición en assembly de un vector equivalente al siguiente vector C: `const char* errmsg[]`. Dicho vector contendrá los mensajes de error que las funciones antes mencionadas puedan generar, y cuyo índice es el código de error devuelto por las mismas.
- Los header files pertinentes (al menos, `base64.h`, con los prototipos de las funciones mencionadas, a incluir en `main.c`), y la declaración del vector `extern const char* errmsg[]`).

A su vez, las funciones MIPS32 `base64_encode()` y `base64_decode()` antes mencionadas, coresponden a los siguientes prototipos C:

- `int base64_encode(int infd, int outfd)`
- `int base64_decode(int infd, int outfd)`

Ambas funciones reciben por `infd` y `outfd` los file descriptors correspondientes a los archivos de entrada y salida pre-abiertos por `main.c`, la primera función realizará el encoding a base 64 de su entrada, y la segunda función el decoding de base 64 se su entrada.

Ante un error, ambas funciones volverán con un código de error numérico (índice del vector de mensajes de error de `base64.h`), o cero en caso de realizar el procesamiento de forma exitosa.

5. Implementación

El programa a implementar deberá satisfacer algunos requerimientos mínimos, que detallamos a continuación:

5.1. ABI

Será necesario que el código presentado utilice la ABI explicada en clase ([2] y [3]).

5.2. Syscalls

Es importante aclarar que desde el código assembly no podrán llamarse funciones que no fueran escritas originalmente en assembly por los alumnos. Por lo contrario, desde el código C sí podrá (y deberá) invocarse código assembly.

Por ende, y atendiendo a lo planteado en la sección 4, los alumnos deberán invocar algunos de los system calls disponibles en NetBSD (en particular, `SYS_read` y `SYS_write`).

5.3. Casos de prueba

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

5.4. Documentación

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

6. Informe

El informe deberá incluir al menos las siguientes secciones:

- Documentación relevante al diseño, desarrollo y debugging del programa;
- Comando(s) para compilar el programa;
- Las corridas de prueba, (sección 5.3) con los comentarios pertinentes;
- El código fuente completoi, el cual deberá entregarse en formato digital compilable (incluyendo archivos de entrada y salida de pruebas);
- Este enunciado.

El informe deberá entregarse en formato impreso y digital.

7. Fechas

- Vencimiento: 30/10/2018.

Referencias

[1] Enunciado del primer trabajo práctico (TP0), primer cuatrimestre de 2018.

[2] System V application binary interface, MIPS RISC processor supplement (third edition). Santa Cruz Operations, Inc.

[3] MIPS ABI: Function Calling Convention, Organización de computadoras - 66.20 (archivo "func_call_conv.pdf", `http://groups.yahoo.com/groups/orga-comp/Material/`).

1.1. Diseño e implementación

Tomando como referencia el Trabajo Práctico #0 en donde el programa contenía la lógica tanto del codificador y decodificador y de otras funciones auxiliares, para este nuevo programa, se requirió re-escribirlo, de forma tal que quede organizado de la siguiente forma:

- main.c: contendrá todo el código necesario para el procesamiento de las opciones de línea de comandos, apertura y cierre de archivos (de ser necesario), y reporte de errores (stderr). Desde aquí se llama a las funciones de encoding y decoding siguientes.
- base64.S: contendrá el cádigo MIPS32 assembly con las funciones base64_encode() y base64_decode(), y las funciones y estructuras de datos auxiliares para realizar los cómputo de encoding y decoding, que los alumnos crean convenientes. También contendrá la definición en asembly de un vector equivalente al siguiente vector C: const char errmsg[]. Dicho vector contendrá los mensajes de error que las funciones antes mencionadas puedan generar, y cuyo índice es el código de error devuelto por las mismas.
- Los header files pertinentes (al menos, base64.h, con los prototipos de las funciones mencionadas, a incluir en ***main.c***), y la declaración del vector extern const char errmsg[]).

A su vez, las funciones MIPS32 base64 encode() y base64 decode() antes mencionadas, corresponden a los siguientes prototipos C:

```
1                int base64_encode(int infd, int outfd)
2                int base64_decode(int infd, int outfd)
3
```

Ambas funciones reciben por *infd* y *outfd* los *file descriptors* correspondientes a los archivos de entrada y salida pre-abiertos por ***main.c***, la primera función realizará el encoding a base 64 de su entrada, y la segunda función el decoding de base 64 de su entrada. Ante un error, ambas funciones volverán con un código de error numérico índice del vector de mensajes de error de ***base64.h***), o cero en caso de realizar el procesamiento de forma exitosa.

El programa implementado satisface los siguientes requerimientos, que se detallan a continuación:

- **ABI**
  El código presentado utilice la ABI explicada en clase([2] y [3]).
- **Syscalls**
  Se aclara que desde el código assembly no se llaman funciones que no son escritas originalmente en assembly. Por lo contrario, desde el código C sí se invoca código assembly, particularmente se invocan algunos de los system calls disponibles en NetBSD (en particular, ***SYS_read*** y ***SYS_write***).

Como en el Trabajo Práctico #0, 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 `CommandOptions` cuya definición es

```
1        typedef struct {
2                File input;
3                File output;
```

```
4                   const char* input_route;
5                   const char* output_route;
6                   char error;
7                   char encode_opt;
8           } CommandOptions;
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 `CommandErrArg()` cuyo resultado es:

```
1                   fprintf(stderr, "Invalid Arguments\n");
2                   fprintf(stderr,"Options:\n");
3                   fprintf(stderr,"  -V, --version    Print version and quit.\n");
4                   fprintf(stderr,"  -h, --help       Print this information.\n");
5                   fprintf(stderr,"  -i, --input      Location of the input file.\n
    ");
6                   fprintf(stderr,"  -o, --output     Location of the output file.\
    n");
7                   fprintf(stderr,"  -a, --action     Program action: encode (
    default) or decode.\n");
8                   fprintf(stderr,"Examples:\n");
9                   fprintf(stderr,"  tp0 -a encode -i ~/input -o ~/output\n");
10                  fprintf(stderr,"  tp0 -a decode\n");
11
```

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 `CommandVersion()`
  - `Mensaje de versión:` Función `CommandHelp()`
  - `Input file :` Función `CommandSetInput()` que guarda la entrada del archivo donde será leído el texto.
  - `Output file:` Función CommandSetOutput() que guarda la entrada del archivo de salida donde se escribirá el texto codificado.
  - `Acción del programa a ejecutar:` Función CommandSetEncodeOpt() que setea la variable $opt- > encode\_opt$ indicando si es una operación de ENCODE o DECODE respectivamente.

- Encode/Decode: una vez que se procesó correctamente las opciones de la línea de comandos se procede a llamar a las funciones correspondientes que ejecutarán la operación de ENCODE o DECODE dependiendo del argumento pasado en la línea de comandos. Como se especifico más arriba está parte del programa es implementada en lenguaje assembly MIPS y cumplen lo siguientes:

  - DECODE
    La operación de DECODE está implementada en el archivo ***decode.S*** que contiene una función `Decode()` que básicamente lo que realiza es la lectura del archivo para procesarlo

teniendo en cuenta la longitud del archivo a procesar y el padding a decodificar. Esta función recibe los files descriptor de entrada y salida procesándolo, según la ABI requerida y luego en la salida si no hubo errores se retorna cero sino se retorna un código de error númerico .

- ENCODE
  La operación de ENCODE está implementada en el archivo ***encode.S*** que contiene una función `Encode()` que básicamente lo que realiza es la lectura del archivo para procesarlo teniendo en cuenta la longitud del archivo a procesar y el padding a decodificar. Esta función recibe los files descriptor de entrada y salida procesándolo, según la ABI requerida y luego en la salida si no hubo errores se retorna cero sino se retorna un código de error númerico .

1.2. 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.
- -a: Acción a llevar a cabo: codificación o decodificación.

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 z 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.

1.3. Compilación del programa

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

```
$ gcc -std=c99 -Wall -o0 -g -o tp1 main.c base64.S
```

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

```
$ ./tp1
```

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 | ./tp1 -a encode
```

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

2. Pruebas realizadas

2.1. 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 "########################################"

mkdir ./outputs

echo "#---------# COMIENZA test ejercicio 0 archivo vacio #--------#"
touch ./outputs-aut/zero.txt
./tp1 -a encode -i ./outputs-aut/zero.txt -o ./outputs-aut/zero.txt.b64
ls -l ./outputs-aut/zero.txt.b64

if diff -b ./outputs-aut/zero.txt ./outputs-aut/zero_ok.txt; then
 echo "[OK]";
else echo ERROR;
fi

echo "#---------# FIN test ejercicio 0 archivo vacio #--------#"
echo "#-----------------------------------------------------#"
echo "#---------# COMIENZA test ejercicio 1 archivo vacio sin -a #--------#"

touch ./outputs-aut/zero.txt
./tp1 -i ./outputs-aut/zero.txt -o ./outputs-aut/zero.txt.b64
ls -l ./outputs-aut/zero.txt.b64

if diff -b ./outputs-aut/zero.txt ./outputs-aut/zero_ok.txt; then
 echo "[OK]";
else echo ERROR;
fi

echo "#---------# FIN test ejercicio 1 archivo vacio sin -a #--------#"
echo "#-------------------------------------------------------------#"
echo "#---------# COMIENZA test ejercicio 2 stdin y stdout #---------#"

echo -n Man | ./tp1 -a encode > ./outputs/outputEncode.txt
if diff -b ./outputs-aut/outputEncode-aut.txt ./outputs/outputEncode.txt; then echo
    "[OK]"; else
        echo ERROR;
fi

echo "#---------# FIN test ejercicio 2 stdin y stdout #--------#"
echo "#-----------------------------------------------------#"
echo "#---------# COMIENZA test ejercicio 3 stdin y stdout #--------#"

echo -n TWFu | ./tp1 -a decode > ./outputs/outputDecode.txt
if diff -b ./outputs-aut/outputDecode-aut.txt ./outputs/outputDecode.txt; then echo
    "[OK]"; else
        echo ERROR;
fi
```

```
49
50 echo "#---------# FIN test ejercicio 3 stdin y stdout #--------#"
51 echo "#-----------------------------------------------------#"
52 echo "#---------# COMIENZA test ejercicio 3 help sin parámetros #--------#"
53
54 ./tp1 > ./outputs/outputMenuHelp.txt
55 if diff -b ./outputs-aut/outputMenuHelp-aut.txt ./outputs/outputMenuH.txt; then echo
       "[OK]"; else
56         echo ERROR;
57 fi
58
59 echo "#---------# FIN test ejercicio 3 help sin parámetros #--------#"
60 echo "#-----------------------------------------------------#"
61 echo "#----------# COMIENZA test menu help (-h) #----------#"
62
63 ./tp1 -h > ./outputs/outputMenuH.txt
64
65 if diff -b ./outputs-aut/outputMenuHelp-aut.txt ./outputs/outputMenuH.txt; then echo
       "[OK]"; else
66         echo ERROR;
67 fi
68
69 echo "#----------# FIN test menu version (-h) #----------#"
70 echo "#-----------------------------------------------------#"
71 echo "#----------# COMIENZA test menu help (--help) #----------#"
72
73 ./tp1 --help > ./outputs/outputMenuHelp.txt
74
75 if diff -b ./outputs-aut/outputMenuHelp-aut.txt ./outputs/outputMenuHelp.txt; then
       echo "[OK]"; else
76                 echo ERROR;
77 fi
78
79 echo "#----------# FIN test menu version (--help) #----------#"
80 echo "#-----------------------------------------------------#"
81 echo "#----------# COMIENZA test menu version (-V) #----------#"
82
83 ./tp1 -V > ./outputs/outputMenuV.txt
84
85 if diff -b ./outputs-aut/outputMenuVersion-aut.txt ./outputs/outputMenuV.txt; then
       echo "[OK]"; else
86                 echo ERROR;
87 fi
88 echo "#----------# FIN test menu version (-V) #----------#"
89 echo "#-----------------------------------------------------#"
90 echo "#----------# COMIENZA test menu version (--version) #----------#"
91
92 ./tp1 --version > ./outputs/outputMenuVersion.txt
93
94 if diff -b ./outputs-aut/outputMenuVersion-aut.txt ./outputs/outputMenuVersion.txt;
       then echo "[OK]"; else
95                 echo ERROR;
96 fi
97 echo "#----------# FIN test menu version (--version) #----------#"
98 echo "#-----------------------------------------------------#"
99 echo "#---------# COMIENZA test ejercicio encode/decode #--------#"
100
101 echo xyz | ./tp1 -a encode | ./tp1 -a decode | od -t c
```

```
102
103 echo "#---------# FIN test ejercicio encode #--------#"
104 echo "#-----------------------------------------------------#"
105 echo "#---------# COMIENZA test ejercicio longitud maxima 76 #--------#"
106
107 yes | head -c 1024 | ./tp1 -a encode > ./outputs/outputSize76.txt
108
109 if diff -b ./outputs-aut/outputSize76-aut.txt ./outputs/outputSize76.txt; then echo
    "[OK]"; else
110                 echo ERROR;
111 fi
112
113 echo "#---------# FIN test ejercicio longitud maxima 76 #--------#"
114 echo "#-----------------------------------------------------#"
115 echo "#---------# COMIENZA test ejercicio decode 1024 #--------#"
116
117 yes | head -c 1024 | ./tp1 -a encode | ./tp1 -a decode | wc -c > ./outputs/
    outputSize1024.txt
118
119 if diff -b ./outputs-aut/outputSize1024-aut.txt ./outputs/outputSize1024.txt; then
    echo "[OK]"; else
120                 echo ERROR;
121 fi
122
123 echo "#---------# FIN test ejercicio decode 1024#--------#"
124 echo "#-----------------------------------------------------#"
125 echo "#---------# COMIENZA test ejercicio encode/decode random #--------#"
126
127 n=1;
128 while :; do
129 #while [$n -lt 10]; do
130 head -c $n </dev/urandom >/tmp/in.bin;
131 ./tp1 -a encode -i /tmp/in.bin -o /tmp/out.b64;
132 ./tp1 -a decode -i /tmp/out.b64 -o /tmp/out.bin;
133 if diff /tmp/in.bin /tmp/out.bin; then :; else
134 echo ERROR: $n;
135 break;
136 fi
137 echo [OK]: $n;
138 n=`expr $n + 1`;
139 rm -f /tmp/in.bin /tmp/out.b64 /tmp/out.bin
140 done
141
142 echo "#---------# FIN test ejercicio encode/decode random #--------#"
143 echo "#-----------------------------------------------------#"
144
145 echo "##########################################"
146 echo "####### FIN Tests automaticos  ###########"
147 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.

2.1.1. Generales

- Mensaje de ayuda

```
$ ./tp1 -h  o ./tp1 --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.
  -a, --action     Program action: encode (default) or decode.
Examples:
  tp1 -a encode -i ~/input -o ~/output
  tp1 -a decode
```

- Mensaje de version

```
$ ./tp1 -V  o ./tp1 --version
Version: 0.2

```

- Archivo de entrada no válido

```
$ ./tp1 -i archivoInvalido.txt

Invalid Arguments
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.
  -a, --action     Program action: encode (default) or decode.
Examples:
  tp1 -a encode -i ~/input -o ~/output
  tp1 -a decode


```

3. Conclusiones

El trabajo práctico nos permitió desarrollar una API para procesar archivos transformándolos a su equivalente `base64` en lenguaje C y, en parte, en lenguaje assembly MIPS para la codificación y decodificación de los archivos. Además, nos permitió familiarizarnos con las `syscalls` para el llamado de las funciones en lenguaje assembly y el consecuente análisis y desarrollo de código assembler MIPS utilizando el emulador GXemul.

Referencias

[1] Enunciado del primer trabajo práctico (TP0), primer cuatrimestre de 2018.

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

[3] The NetBSD project, http://www.netbsd.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/

[7] MIPS ABI: Function Calling, Convention Organización de computadoras(66.20) en archivo "func call conv.pdf" y enlace http://groups.yahoo.com/groups/orga-comp/Material/)

[8] System V application binary interface, MIPS RISC processor supplement (third edition). Santa Cruz Operations, Inc.

A. Código fuente

A.0.1. main.c

```
/**
 * Created by gatti2602 on 12/09/18.
 * Main
 */

#define FALSE 0
#define TRUE 1

#include <getopt.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <stdio.h>

#define CMD_ENCODE 1
#define CMD_DECODE 0
#define CMD_NOENCODE 2
#define FALSE 0
#define TRUE 1
#define ERROR 1
#define OK 0

#include "base64.h"

/*****************************
 * DECLARACION DE FUNCIONES *
 *****************************/

typedef struct{
    FILE* file;
    char eof;
} File;

typedef struct {
    File input;
    File output;
    const char* input_route;
    const char* output_route;
    char error;
    char encode_opt;
} CommandOptions;

/**
 * Inicializa TDA CommandOptions
 * Pre: Puntero a Command Options escribible
 * Post: CommandOptions Inicializados a valores por default
 * Valores default:
 *      input: stdin
 *      output stdout
 *      error: FALSE
 *      encode_opt: decode
 */
void CommandCreate(CommandOptions* opt);
```

```
54
55 /**
56  * Setea ruta de entrada
57  * Pre: ruta valida
58  * Post: ruta lista para abrir file
59  */
60 void CommandSetInput(CommandOptions* opt, const char* input);
61
62 /**
63  * Setea ruta de salida
64  * Pre: ruta valida
65  * Post: ruta lista para abrir file
66  */
67 void CommandSetOutput(CommandOptions* opt, const char* output);
68
69 /**Setea Command Option
70  * Pre: opt inicializado
71  * Post: Setea el encoding.
72  *       Si string no es encode/decode setea opt error flag.
73  */
74 void CommandSetEncodeOpt(CommandOptions* opt, const char* encode_opt);
75
76 /**
77  * Devuelve el flag de error
78  */
79 char CommandHasError(CommandOptions *opt);
80
81 /**
82  * Indica que hubo un error
83  */
84 void CommandSetError(CommandOptions *opt);
85
86 /**
87  * Ejecuta el comando
88  * Pre: Asume parametros previamente validados y ok
89  * Post: Ejecuta el comando generando la salida esperada
90  *        Devuelve 0 si error y 1 si OK.
91  */
92 char CommandProcess(CommandOptions* opt);
93
94 /**
95  * Help Command
96  * Imprime por salida estandar los distintos comandos posibles.
97  * Pre: N/A
98  * Post: N/A
99  */
100 void CommandHelp();
101
102 /**
103  * Imprime la ayuda por la salida de errores
104  */
105 void CommandErrArg();
106
107 /**
108  * Version Command
109  * Imprime por salida estandar la version del codigo
110  * Pre: N/A
111  * Post: N/A
```

```
112  */
113 void CommandVersion();
114 
115 /**
116  * Recibe los archivos abiertos y debe ejecutar la operacion de codificacion
117  * Pre: opt->input posee el stream de entrada
118  *      opt->output posee el stream de salida
119  *      opt->encode_opt posee la opcion de codificacion
120  * Post: Datos procesados y escritos en el stream, si error devuelve 0, sino 1.
121  */
122 char _CommandEncodeDecode(CommandOptions *opt);
123 
124 /**
125  * Construye el TDA.
126  * Post: TDA construido
127  */
128 void FileCreate(File *f);
129 
130 /**
131  * Abre un File, devuelve 0 (NULL) si falla
132  * Pre: Ptr a File Inicializado ,
133  *      Ruta a archivo, si es 0 (NULL) utiliza stdin
134  */
135 char FileOpenForRead(File* file, const char* route);
136 
137 /**
138  * Abre un File, devuelve 0 (NULL) si falla
139  * Pre: Ptr a File Inicializado ,
140  *      Ruta a archivo, si es 0 (NULL) utiliza stdout
141  */
142 char FileOpenForWrite(File* file, const char* route);
143 
144 /*
145  * Cierra archivo abierto
146  * Pre: Archivo previamente abierto
147  */
148 int FileClose(File* file);
149 
150 /*********************************
151  * FIN: DECLARACION DE FUNCIONES *
152  *********************************/
153 
154 /*************************
155  * DEFINICION DE FUNCIONES *
156  *************************/
157 
158 void CommandHelp(){
159     printf("Options:\n");
160     printf("  -V, --version    Print version and quit.\n");
161     printf("  -h, --help       Print this information.\n");
162     printf("  -i, --input      Location of the input file.\n");
163     printf("  -o, --output     Location of the output file.\n");
164     printf("  -a, --action     Program action: encode (default) or decode.\n");
165     printf("Examples:\n");
166     printf("  tp0 -a encode -i ~/input -o ~/output\n");
167     printf("  tp0 -a decode\n");
168 }
169 
```

```
170 void CommandVersion() {
171     printf("Version: 0.2\n");
172 }
173
174 void CommandCreate(CommandOptions *opt) {
175     FileCreate(&opt->input);
176     FileCreate(&opt->output);
177     opt->error = FALSE;
178     opt->encode_opt = CMD_ENCODE;
179     opt->input_route = 0;
180     opt->output_route = 0;
181 }
182
183 void CommandSetInput(CommandOptions *opt, const char *input) {
184     opt->input_route = input;
185 }
186
187 void CommandSetOutput(CommandOptions *opt, const char *output) {
188     opt->output_route = output;
189 }
190
191 void CommandSetEncodeOpt(CommandOptions *opt, const char *encode_opt) {
192         if(strcmp(encode_opt,"decode") == 0) {
193             opt->encode_opt = CMD_DECODE;
194         } else {
195             opt->encode_opt = CMD_ENCODE;
196         }
197 }
198
199 char CommandHasError(CommandOptions *opt) {
200     return opt->error || opt->encode_opt == CMD_NOENCODE;
201 }
202
203 void CommandSetError(CommandOptions *opt) {
204     opt->error = TRUE;
205 }
206
207 char CommandProcess(CommandOptions *opt) {
208     opt->error = FileOpenForRead(&opt->input, opt->input_route);
209
210     if(opt->error != ERROR){
211         opt->error = FileOpenForWrite(&opt->output, opt->output_route);
212
213         if(opt->error != ERROR){
214                 opt->error = _CommandEncodeDecode(opt);
215                 FileClose(&opt->input);
216                 FileClose(&opt->output);
217         } else {
218                 FileClose(&opt->input);
219         }
220     }
221     return opt->error;
222 }
223
224 char _CommandEncodeDecode(CommandOptions *opt) {
225     if(opt->encode_opt == CMD_ENCODE){
226         int filein = fileno((opt->input).file);
227         int fileout = fileno((opt->output).file);
```

```
228         int res = base64_encode(filein, fileout);
229         if(res != 0)
230                 fprintf(stderr, "%s\n",errmsg[res]);
231 
232     }
233 
234    if (opt->encode_opt == CMD_DECODE) {
235         int filein = fileno((opt->input).file);
236         int fileout = fileno((opt->output).file);
237         int res = base64_decode(filein, fileout);
238         if(res != 0)
239                 fprintf(stderr, "%s\n",errmsg[res]);
240    }
241 
242     return opt->error;
243 }
244 
245 void CommandErrArg() {
246     fprintf(stderr, "Invalid Arguments\n");
247     fprintf(stderr,"Options:\n");
248     fprintf(stderr,"  -V, --version    Print version and quit.\n");
249     fprintf(stderr,"  -h, --help       Print this information.\n");
250     fprintf(stderr,"  -i, --input      Location of the input file.\n");
251     fprintf(stderr,"  -o, --output     Location of the output file.\n");
252     fprintf(stderr,"  -a, --action     Program action: encode (default) or decode.\n"
    );
253     fprintf(stderr,"Examples:\n");
254     fprintf(stderr,"  tp0 -a encode -i ~/input -o ~/output\n");
255     fprintf(stderr,"  tp0 -a decode\n");
256 }
257 
258 void FileCreate(File *file){
259     file->file = 0;
260     file->eof = 0;
261 }
262 
263 char FileOpenForRead(File* file, const char *route ){
264     if(route == NULL) {
265         file->file = stdin;
266     } else {
267         file->file = fopen(route, "rb");
268         if (file->file == NULL) {
269             int err = errno;
270             fprintf(stderr, "File Open Error; %s\n", strerror(err));
271             return ERROR;
272         }
273     }
274     return OK;
275 }
276 
277 char FileOpenForWrite(File* file, const char *route ) {
278     if(route == NULL) {
279         file->file = stdout;
280     } else {
281         file->file = fopen(route, "wb");
282         if (file->file == NULL) {
283             int err = errno;
284             fprintf(stderr, "File Open Error; %s\n", strerror(err));
```

```
285             return ERROR;
286         }
287     }
288     return OK;
289 }
290 
291 int FileClose(File* file){
292     if(file->file == stdin || file->file == stdout)
293         return OK;
294 
295     int result = fclose(file->file);
296     if (result == EOF){
297         int err = errno;
298         fprintf(stderr, "File Close Error; %s\n", strerror(err));
299         return ERROR;
300     }
301     return OK;
302 }
303 
304 /*******************************
305  * FIN: DEFINICION DE FUNCIONES *
306  *******************************/
307 
308 int main(int argc, char** argv) {
309     struct option arg_long[] = {
310             {"input",   required_argument,  NULL,  'i'},
311             {"output",  required_argument,  NULL,  'o'},
312             {"action",  required_argument,  NULL,  'a'},
313             {"help",    no_argument,        NULL,  'h'},
314             {"version", no_argument,        NULL,  'V'},
315     };
316     char arg_opt_str[] = "i:o:a:hV";
317     int arg_opt;
318     int arg_opt_idx = 0;
319     char should_finish = FALSE;
320 
321     CommandOptions cmd_opt;
322     CommandCreate(&cmd_opt);
323 
324     if(argc == 1)
325         CommandSetError(&cmd_opt);
326 
327     while((arg_opt =
328                     getopt_long(argc, argv, arg_opt_str, arg_long, &arg_opt_idx)) !=
    -1 && !should_finish) {
329         switch(arg_opt){
330                 case 'i':
331                         CommandSetInput(&cmd_opt, optarg);
332                         break;
333                 case 'o':
334                 CommandSetOutput(&cmd_opt, optarg);
335                 break;
336                 case 'h':
337                         CommandHelp();
338                 should_finish = TRUE;
339                 break;
340                 case 'V':
341                         CommandVersion();
```

```
342                 should_finish = TRUE;
343                 break;
344                 case 'a':
345                     CommandSetEncodeOpt(&cmd_opt, optarg);
346                                 break;
347                 default:
348                         CommandSetError(&cmd_opt);
349                         break;
350         }
351     }
352
353     if(should_finish)
354         return 0;
355
356     if(!CommandHasError(&cmd_opt)) {
357         CommandProcess(&cmd_opt);
358     } else {
359         CommandErrArg();
360         return 1;
361     }
362     return 0;
363 }
```

A.0.2. Header file base64.h

```
#ifndef TP1_BASE64_H
#define TP1_BASE64_H

extern const char* errmsg[];

int base64_encode(int infd, int outfd);
int base64_decode(int infd, int outfd);

#endif
```

A.0.3. Assembly base64.S

```
#include <mips/regdef.h>
#include <sys/syscall.h>
#define STACK_FRAME_ENCODE 16

#define OFFSET_OUTPUT_ENCODE 24
#define OFFSET_LENGTH_ENCODE 20
#define OFFSET_BUFFER_ENCODE 16
#define OFFSET_FP_ENCODE 12
#define OFFSET_GP_ENCODE 8

#define OFFSET_B4_AUX 7
#define OFFSET_B3_AUX_2 6
#define OFFSET_B3_AUX 5
#define OFFSET_B2_AUX 4
#define OFFSET_B1_AUX 3
#define OFFSET_B3 2
#define OFFSET_B2 1
#define OFFSET_B1 0
#define EQUAL_CHAR 61

#define RETURNO_OK 1
#define DECODE_ERROR   100
#define SIZE_DECODE_CHAR 4

#define SHIFT_2 2
#define SHIFT_4 4
#define SHIFT_6 6

#define EQUAL_CHAR 61

#define STACK_FRAME_DECODECHAR 32

#define OFFSET_FP_DECODECHAR 32
#define OFFSET_GP_DECODECHAR 28
#define OFFSET_CHARACTER_DECODECHAR 16
#define OFFSET_I_DECODECHAR 20
#define OFFSET_RETURN_DECODECHAR 24

#define STACK_FRAME_DECODE 64

#define OFFSET_BUFFER_OUTPUT_ENCODE 68
#define OFFSET_BUFFER_INPUT_ENCODE 64
#define OFFSET_RA_DECODE 60
#define OFFSET_FP_DECODE 56
#define OFFSET_GP_DECODE 52
#define OFFSET_S0_DECODE 48
#define OFFSET_CHAR1_AUX_ENCODE 37
#define OFFSET_CHAR0_AUX_ENCODE 36
#define OFFSET_CHARS3_ENCODE 27
#define OFFSET_CHARS2_ENCODE 26
#define OFFSET_CHARS1_ENCODE 25
#define OFFSET_CHARS0_ENCODE 24
#define OFFSET_RETURN_ENCODE 20
#define OFFSET_I_DECODE 32

        .data
```

```
        .align  2
sep:    .ascii  "\n"
pad:    .ascii  "="
        .globl errmsg
errmsg:
        .word base64_ok, base64_err1, base64_err2, base64_err3
        .size errmsg, 16
base64_ok:
        .asciiz "OK"
base64_err1:
        .asciiz "I/O Error"
base64_err2:
        .asciiz "File no es multiplo de 4"
base64_err3:
        .asciiz "File contiene caracteres invalidos"
        .text
        .align  2
        .globl  base64_encode
        .ent    base64_encode
base64_encode:
        // debugging info: descripcion del stack frame
        .frame  $fp, 40, ra     // $fp: registro usado como frame pointer
                                // 32: tamañodel stack frame
                                // ra: registro que almacena el return address
        // bloque para codigo PIC
        .set    noreorder       // apaga reordenamiento de instrucciones
        .cpload t9              // directiva usada para codigo PIC
        .set    reorder         // enciende reordenamiento de instrucciones
        // creo stack frame
        subu    sp, sp, 40      // 4 (SRA) + 2 (LTA) + 4 (ABA)
        // directiva para codigo PIC
        .cprestore 24           // inserta aqui "sw gp, 24(sp)",
                                // mas "lw gp, 24(sp)" luego de cada jal.
        // salvado de callee-saved regs en SRA
        sw      $fp, 28(sp)
        sw      ra, 32(sp)
        // de aqui al fin de la funcion uso $fp en lugar de sp.
        move    $fp, sp
        // salvo 1er arg (siempre)
        sw      a0, 40($fp)     // a0 contiene file input
        sw      a1, 44($fp)     // a1 contiene file output
        li      s1, 0           // count = 0

        //Limpio input para read
base64_encode_loop:
        sw      zero, 20($fp)   //input = 0

        //Leo archivo
        lw      a0, 40($fp)
        addi    a1, $fp, 20
        li      a2, 3
        li      v0, SYS_read
        syscall
        beqz    v0, base64_encode_return_ok     //Si no lei nada finalizo
        blt     v0, 0, base64_encode_io_error
        //Paso parametros y llamo a Encode
        addi    a0, $fp, 20
        move    a1, v0
```

```
        addi    a2, $fp, 16
        la      t9, Encode
        jal     ra, t9

        //Grabo en file
        lw      a0, 44($fp)     // File descriptor out
        addi    a1, $fp, 16     // Apunto a buffer out
        li      a2, 4           // length = 4
        li      v0, SYS_write
        syscall
        addi s1, s1, 1          // count++
        bne s1, 18, base64_encode_loop // Si count = 18 agrego un salto
        lw      a0, 44($fp)     // file out
        la      a1, sep         // sep = '\n'
        li      a2, 1           // length = 4
        li      v0, SYS_write
        syscall
        li      s1, 0
        j base64_encode_loop

base64_encode_return_ok:        // return;
        li      v0, 0
        j base64_encode_return
base64_encode_io_error:
        li      v0, 1
        // restauro callee-saved regs
base64_encode_return:
        lw      gp, 24(sp)
        lw      $fp, 28(sp)
        lw      ra, 32(sp)
        // destruyo stack frame
        addu    sp, sp, 40
        // vuelvo a funcion llamante
        jr      ra
        .end    base64_encode
        .size   base64_encode, .-base64_encode

        .globl  base64_decode
        .ent    base64_decode
base64_decode:
        // debugging info: descripcion del stack frame
        .frame  $fp, 40, ra     // $fp: registro usado como frame pointer
                                // 32: tamañodel stack frame
                                // ra: registro que almacena el return address
        // bloque para codigo PIC
        .set    noreorder       // apaga reordenamiento de instrucciones
        .cpload t9              // directiva usada para codigo PIC
        .set    reorder         // enciende reordenamiento de instrucciones
        // creo stack frame
        subu    sp, sp, 40      // 4 (SRA) + 2 (LTA) + 4 (ABA)
        // directiva para codigo PIC
        .cprestore 24           // inserta aqui "sw gp, 24(sp)",
                                // mas "lw gp, 24(sp)" luego de cada jal.
        // salvado de callee-saved regs en SRA
        sw      $fp, 28(sp)
        sw      ra, 32(sp)
        // de aqui al fin de la funcion uso $fp en lugar de sp.
        move    $fp, sp
```

```
        // salvo 1er arg (siempre)
        sw      a0, 40($fp)     // a0 contiene file input
        sw      a1, 44($fp)     // a1 contiene file output
        li      s1, 0           // count = 0
        la      s5, pad

        //Limpio input para read
base64_decode_loop:
        sw      zero, 20($fp)   //input = 0

        //Leo archivo
        lw      a0, 40($fp)
        addi    a1, $fp, 20
        li      a2, 4
        li      v0, SYS_read
        syscall
        beqz    v0, base64_decode_return_ok     //Si no lei nada finalizo
        blt     v0, 0, base64_decode_ioerror
        blt     v0, 4, base64_decode_nomult
        //Controlo si hay padding
        li      s3, 0                           //s3 = cant de padding a borrar
        lbu     s2, 43($fp)                     //s2 aux control padding
        bne     s2, s5, ctl1
        addi    s3, s3, 1
ctl1:
        lbu     s2, 42($fp)                     //s2 aux control padding
        bne     s2, s5, ctl2
        addi    s3, s3, 1
ctl2:
        //Controlo salto de linea
        addi s1, s1, 1          // count++
        bne s1, 18, not_sep     // Si count = 18 elimino un caracter
        lw      a0, 40($fp)     // file in
        addi    a1, $fp, 16     // grabo en out buffer, luego se pisa
        li      a2, 1           // length = 1
        li      v0, SYS_read
        syscall
        li      s1, 0
        //Paso parametros y llamo a Decode
not_sep:
        addi    a0, $fp, 20
        addi    a1, $fp, 16
        la      t9, Decode
        jal     ra, t9

        //Chequeo error
        beq     v0, DECODE_ERROR, base64_decode_decode_err

        //Grabo en file
        lw      a0, 44($fp)     // File descriptor out
        addi    a1, $fp, 16     // Apunto a buffer out
        li      s4, 3
        subu    a2, s4, s3      // a2 = 3 - cant de padding
        li      v0, SYS_write
        syscall
        j base64_decode_loop
base64_decode_return_ok:
        li v0, 0
```

```
231         j base64_decode_return
232 base64_decode_ioerror:
233         li v0, 1
234         j base64_decode_return
235 base64_decode_nomult:
236         li v0, 2
237         j base64_decode_return
238 base64_decode_decode_err:
239         li v0, 3
240 base64_decode_return:   // return;
241         // restauro callee-saved regs
242         lw      gp, 24(sp)
243         lw      $fp, 28(sp)
244         lw      ra, 32(sp)
245         // destruyo stack frame
246         addu    sp, sp, 40
247         // vuelvo a funcion llamante
248         jr      ra
249         .end    base64_decode
250         .size   base64_decode, .-base64_decode
251
252         //.file 1 "encode.c"
253         //.section .mdebug.abi32
254         //.previous
255         //.abicalls
256         .data
257         .align  2
258         .type   encoding_table, @object
259         .size   encoding_table, 64
260 encoding_table:
261         .byte   65
262         .byte   66
263         .byte   67
264         .byte   68
265         .byte   69
266         .byte   70
267         .byte   71
268         .byte   72
269         .byte   73
270         .byte   74
271         .byte   75
272         .byte   76
273         .byte   77
274         .byte   78
275         .byte   79
276         .byte   80
277         .byte   81
278         .byte   82
279         .byte   83
280         .byte   84
281         .byte   85
282         .byte   86
283         .byte   87
284         .byte   88
285         .byte   89
286         .byte   90
287         .byte   97
288         .byte   98
```

```
289         .byte   99
290         .byte   100
291         .byte   101
292         .byte   102
293         .byte   103
294         .byte   104
295         .byte   105
296         .byte   106
297         .byte   107
298         .byte   108
299         .byte   109
300         .byte   110
301         .byte   111
302         .byte   112
303         .byte   113
304         .byte   114
305         .byte   115
306         .byte   116
307         .byte   117
308         .byte   118
309         .byte   119
310         .byte   120
311         .byte   121
312         .byte   122
313         .byte   48
314         .byte   49
315         .byte   50
316         .byte   51
317         .byte   52
318         .byte   53
319         .byte   54
320         .byte   55
321         .byte   56
322         .byte   57
323         .byte   43
324         .byte   47
325
326         .type   encoding_table_size, @object
327         .size   encoding_table_size, 4
328 encoding_table_size:
329         .word   64
330
331         .text
332         .align  2
333         .globl  Encode
334         .ent    Encode
335
336         //////////////  Función Encode //////////////
337
338 Encode:
339         .frame  $fp,STACK_FRAME_ENCODE,ra               // vars= 8, regs= 2/0, args=
        0, extra= 8
340         //.mask 0x50000000,-4
341         //.fmask        0x00000000,0
342         .set    noreorder
343         .cpload t9
344         .set    reorder
345
```

```
346         // Creación del stack frame
347         subu    sp,sp,STACK_FRAME_ENCODE
348
349         .cprestore 0
350         sw      $fp,OFFSET_FP_ENCODE(sp)
351         sw      gp,OFFSET_GP_ENCODE(sp)
352
353         // De aquí al final de la función uso $fp en lugar de sp.
354         move    $fp,sp
355
356         // Guardo el primer parámetro *buffer
357         sw      a0,OFFSET_BUFFER_ENCODE($fp)
358         // Guardo el segundo parámetro 'length'(cantidad de caracteres)
359         sw      a1,OFFSET_LENGTH_ENCODE($fp)
360         // Guardo el puntero al array de salida(output)
361         sw      a2,OFFSET_OUTPUT_ENCODE($fp)
362
363         // Cargo en v0 el puntero al buffer.
364         lw      v0,OFFSET_BUFFER_ENCODE($fp)
365         // Cargo en v0 el 1er byte del buffer.
366         lbu     v0,0(v0)
367         // Guardo el 1er byte en el stack frame
368         sb      v0,OFFSET_B1($fp)
369         // Cargo nuevamente la dirección del buffer.
370         lw      v0,OFFSET_BUFFER_ENCODE($fp)
371         // Aumento en 1(1 byte) la dirección del buffer.
372         // Me muevo por el array del buffer.
373         addu    v0,v0,1
374         // Cargo el 2do byte del buffer.
375         lbu     v0,0(v0)
376         // Guardo el 2do byte en el stack frame.
377         sb      v0,OFFSET_B2($fp)
378         // Cargo nuevamente la dirección del buffer.
379         lw      v0,OFFSET_BUFFER_ENCODE($fp)
380         // Aumento en 2(2 byte) la dirección del buffer.
381         // Me muevo por el array del buffer.
382         addu    v0,v0,2
383         // Cargo el 2do byte del buffer.
384         lbu     v0,0(v0)
385         // Guardo el 3er byte en stack frame.
386         sb      v0,OFFSET_B3($fp)
387         // Cargo en v0 el 1er byte.
388         lbu     v0,OFFSET_B1($fp)
389         // Muevo 2 'posiciones' hacia la derecha(shift 2).
390         srl     v0,v0,2
391         // Guardo el nuevo byte en una variable auxiliar.
392         sb      v0,OFFSET_B1_AUX($fp)
393         // Cargo en v1 el puntero al output.
394         lw      v1,OFFSET_OUTPUT_ENCODE($fp)
395         // Cargo en v0 el byte shifteado.
396         lbu     v0,OFFSET_B1_AUX($fp)
397         // Cargo en v0 el caracter(byte) de la tabla encoding(encoding_table)
398         lbu     v0,encoding_table(v0)
399         // Cargo en v0 el 1er byte de la dirección del output.
400         sb      v0,0(v1)
401         // Cargo en v0 el 1er byte del buffer nuevamente.
402         lbu     v0,OFFSET_B1($fp)
403         // Muevo 6 'posiciones' hacia la izquierda(shift 6).
```

```
404         sll     v0,v0,6
405         // Guardo el resultado del shift en el Stack Frame.
406         sb      v0,OFFSET_B2_AUX($fp)
407         // Cargo el byte sin signo shifteado.
408         lbu     v0,OFFSET_B2_AUX($fp)
409         // Muevo 2 'posiciones' hacia la derecha(shift 2).
410         srl     v0,v0,2
411         // Guardo el nuevo resultado del shift en el Stack Frame.
412         sb      v0,OFFSET_B2_AUX($fp)
413         // Cargo el 2do byte del buffer en v0.
414         lbu     v0,OFFSET_B2($fp)
415         // Hago un shift left de 4 posiciones.
416         srl     v0,v0,4
417         // Cargo en v1 el resultado(byte) del shift right 2.
418         lbu     v1,OFFSET_B2_AUX($fp)
419         // Hago un 'or' entre v1 y v0 para obtener el 2 indice de la tabla.
420         or      v0,v1,v0
421         //(*) Guardo en stack frame(12) el resultado del 'or' anterior.
422         sb      v0,OFFSET_B2_AUX($fp)
423         // Cargo en v0 el puntero al output.
424         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
425         // Cargo en v1 la dirección del output + 1(1byte).
426         addu    v1,v0,1
427         // Cargo en v0 el ultimo resultado del shift(*)
428         lbu     v0,OFFSET_B2_AUX($fp)
429         // Cargo en v0 el caracter(byte) de la tabla encoding(encoding_table)
430         lbu     v0,encoding_table(v0)
431         // Salvo en el output array(output[1]) el valor del encoding_table
432         sb      v0,0(v1)
433         // Cargo en v0 el puntero al output.
434         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
435         // Sumo 2 a la dirección del output(output[2]).
436         // Me desplazo dentro del output array.
437         addu    v1,v0,2
438         // Cargo en v0 el caracter ascii 61('=').
439         li      v0,EQUAL_CHAR                   // 0x3d
440         // Salvo en el output array(output[2]) el valor '='.
441         sb      v0,0(v1)
442         // Cargo en v0 el puntero al output.
443         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
444         // Sumo 3 a la dirección del output(output[3]).
445         // Me desplazo dentro del output array.
446         addu    v1,v0,3
447         // Cargo en v0 el caracter ascii 61('=').
448         li      v0,EQUAL_CHAR                   // 0x3d
449         // Salvo en el output array(output[3]) el valor '='.
450         sb      v0,0(v1)
451         // Cargo en v1 el parametro length.
452         lw      v1,OFFSET_LENGTH_ENCODE($fp)
453         // Cargo en v0 el valor 3.
454         li      v0,3                    // 0x3
455         // Si el length == 3 salto a buffer_size_2.
456         bne     v1,v0,buffer_size_2
457         // Si el tamanio del buffer es 3 continuo NO salto.
458         // Cargo en v0 el 3er byte del buffer.
459         lbu     v0,OFFSET_B3($fp)
460         // Hago un shift right de 6.
461         srl     v0,v0,6
```

```
462         // Guardo el nuevo byte en el stack frame.
463         sb      v0,OFFSET_B3_AUX($fp)
464         // Cargo el 2do byte del buffer en v0.
465         lbu     v0,OFFSET_B2($fp)
466         // Hago un shift left de 4.
467         sll     v0,v0,4
468         // Guardo en el stack frame(14) el nuevo valor.
469         sb      v0,OFFSET_B3_AUX_2($fp)
470         // Cargo en v0 el byte shifteado sin signo.
471         lbu     v0,OFFSET_B3_AUX_2($fp)
472         // Hago un shift rigth de 2.
473         srl     v0,v0,2
474         // Guardo en el stack frame(14) el valor shifteado.
475         sb      v0,OFFSET_B3_AUX_2($fp)
476         // Cargo en v1 el valor del SF(13)
477         lbu     v1,OFFSET_B3_AUX($fp)
478         // Idem en v0(13).
479         lbu     v0,OFFSET_B3_AUX_2($fp)
480         // Hago un 'or' y almaceno en v0.
481         or      v0,v1,v0
482         // Guardo en el stack frame(13) el resultado del 'or'.
483         sb      v0,OFFSET_B3_AUX($fp)
484         // Cargo en v0 el puntero al output.
485         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
486         // Me desplazo por el vector 'output' en 2 posiciones(output[2]).
487         addu    v1,v0,2
488         // Cargo en v0 el resultado del 'or' anterior.
489         lbu     v0,OFFSET_B3_AUX($fp)
490         // Busco en la tabla de encoding el caracter que corresponde.
491         // Luego cargo el byte en v0.
492         lbu     v0,encoding_table(v0)
493         // Guardo el valor recuperado de la tabla encoding_table en el output[2].
494         sb      v0,0(v1)
495         // Cargo en v0 el 3er byte del buffer.
496         lbu     v0,OFFSET_B3($fp)
497         // Hago un shift left de 2.
498         sll     v0,v0,2
499         // Guardo en el stack frame el valor shifteado.
500         sb      v0,OFFSET_B4_AUX($fp)
501         // Cargo el byte sin signo shifteado.
502         lbu     v0,OFFSET_B4_AUX($fp)
503         // Hago un shift rigth de 2.
504         srl     v0,v0,2
505         // Guardo en el stack frame el valor shifteado.
506         sb      v0,OFFSET_B4_AUX($fp)
507         // Cargo en v0 el puntero al output.
508         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
509         // Sumo 3 a la dirección del output(output[3]).
510         // Me desplazo dentro del output array.
511         addu    v1,v0,3
512         // Cargo en v0 el ultimo valor shifteado guardado.
513         lbu     v0,OFFSET_B4_AUX($fp)
514         // Busco en la tabla de encoding el caracter que corresponde.
515         // Luego cargo el byte en v0.
516         lbu     v0,encoding_table(v0)
517         // Guardo el valor recuperado de la tabla encoding_table en el output[3].
518         sb      v0,0(v1)
519         // Salto a return_encode
```

```
520         b       return_encode
521 buffer_size_2:
522         // Cargo en v1 el valor del parámetro length.
523         lw      v1,OFFSET_LENGTH_ENCODE($fp)
524         // Cargo en v0 el valor 2.
525         li      v0,2                    // 0x2
526         // Si length != 2 salgo de la función.
527         bne     v1,v0,return_encode
528         // Cargo en v0 el 3er byte del buffer.
529         lbu     v0,OFFSET_B3($fp)
530         // Hago un shift right de 6.
531         srl     v0,v0,6
532         // Guardo en el stack frame el ultimo valor shifteado.
533         sb      v0,OFFSET_B4_AUX($fp)
534         // Cargo el 2do byte del buffer en v0.
535         lbu     v0,OFFSET_B2($fp)
536         // Hago un shift left de 4 posiciones.
537         sll     v0,v0,4
538         // Guardo en el stack frame nuevo valor shifteado.
539         sb      v0,OFFSET_B3_AUX_2($fp)
540         // Cargo en v0 el byte shifteado sin signo.
541         lbu     v0,OFFSET_B3_AUX_2($fp)
542         // Hago un shift right de 2 posiciones.
543         srl     v0,v0,2
544         // Guardo en el stack frame el valor shifteado.
545         sb      v0,OFFSET_B3_AUX_2($fp)
546         // Cargo en v1 uno de los valores shiftedos(b3aux).
547         lbu     v1,OFFSET_B4_AUX($fp)
548         // Cargo en v0 uno de los valores shiftedos(b3aux2).
549         lbu     v0,OFFSET_B3_AUX_2($fp)
550         // Hago un 'or' entre b3aux y b3aux2.
551         or      v0,v1,v0
552         // Guardo en el stack frame el resutado del 'or'.
553         sb      v0,OFFSET_B4_AUX($fp)
554         // Cargo en v0 el puntero al output.
555         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
556         // Me desplazo dentro del output array y lo guardo en v1.
557         addu    v1,v0,2
558         // Cargo en v0 ultimo resultado del 'or'
559         lbu     v0,OFFSET_B4_AUX($fp)
560         // Busco en la tabla de encoding el caracter que corresponde.
561         // Luego cargo el byte en v0.
562         lbu     v0,encoding_table(v0)
563         // Guardo el valor recuperado de la tabla encoding_table en el output[2].
564         sb      v0,0(v1)
565 return_encode:
566         move    sp,$fp
567         lw      $fp,OFFSET_FP_ENCODE(sp)
568         // destruyo stack frame
569         addu    sp,sp,STACK_FRAME_ENCODE
570         j       ra
571         .end    Encode
572         //.size Encode, .-Encode
573
574         .globl  DecodeChar
575         .ent    DecodeChar
576
577         ////////////// Begin Función DecodeChar //////////////
```

```
578
579 DecodeChar:
580         // Reservo espacio para el stack frame de STACK_FRAME_DECODECHAR bytes
581         .frame  $fp,STACK_FRAME_DECODECHAR,ra           // vars= 8, regs= 2/0, args=
    0, extra= 8
582         //.mask 0x50000000,-4
583         //.fmask        0x00000000,0
584         .set    noreorder
585         .cpload t9
586         .set    reorder
587
588         // Creación del stack frame STACK_FRAME_DECODECHAR
589         subu    sp,sp,STACK_FRAME_DECODECHAR
590         .cprestore 0
591
592         // Guardo fp y gp en el stack frame
593         sw      $fp,OFFSET_FP_DECODECHAR(sp)
594         sw      gp,OFFSET_GP_DECODECHAR(sp)
595         // De aquí al final de la función uso $fp en lugar de sp.
596         move    $fp,sp
597
598         // Guardo en v0 el parámetro recibido: 'character'.
599         move    v0,a0
600         // Guardo en el stack frame 'character'.
601         sb      v0,OFFSET_CHARACTER_DECODECHAR($fp)
602         // Guardo en un '0' en el stack frame.
603         // Inicializo la variable 'i'.
604         sb      zero,OFFSET_I_DECODECHAR($fp)
605 condition_loop:
606         // Cargo en v0 el byte guardado anteriormente(0 o el nuevo valor de 'i').
607         lbu     v0,OFFSET_I_DECODECHAR($fp)
608         // Cargo en v1 el size del encoding_table(64).
609         lw      v1,encoding_table_size
610         // Si (i < encoding_table_size), guardo TRUE en v0, sino FALSE.
611         slt     v0,v0,v1
612         // Salto a condition_if si v0 != 0.
613         bne     v0,zero,condition_if
614         // Brancheo a condition_if_equal
615         b       condition_if_equal
616 condition_if:
617         // Cargo en v0 el valor de 'i'.
618         lbu     v0,OFFSET_I_DECODECHAR($fp)
619         // Cargo en v1 el byte contenido en encoding_table según el valor de 'i'.
620         // encoding_table[i]
621         lbu     v1,encoding_table(v0)
622         // Cargo en v0 'character'.
623         lb      v0,OFFSET_CHARACTER_DECODECHAR($fp)
624         // Salto a increase_index si el valor recuperado del vector encoding_table
625         // es distinto al valor pasado por parámetro(character).
626         bne     v1,v0,increase_index
627         // Cargo en v0 nuevamente el valor de 'i'.
628         lbu     v0,OFFSET_I_DECODECHAR($fp)
629
630         // Guardo en el stack frame(12) el valor de 'i'
631         //sw    v0,12($fp) //VER
632         sw v0,OFFSET_RETURN_DECODECHAR($fp)
633
634         // Brancheo a return_decode_index_or_zero
```

```
635         b       return_decode_index_or_zero
636 increase_index:
637         // Cargo en v0 nuevamente el valor de 'i'.
638         lbu     v0,OFFSET_I_DECODECHAR($fp)
639         // Sumo en 1 el valor de 'i'(i++).
640         addu    v0,v0,1
641         // Guardo el valor modificado en el stack frame.
642         sb      v0,OFFSET_I_DECODECHAR($fp)
643         // Salto a condition_loop
644         b       condition_loop
645 condition_if_equal:
646         // Cargo en v1 el byte(char) recibido como parámetro.
647         // parametro: character.
648         lb      v1,OFFSET_CHARACTER_DECODECHAR($fp)
649         // Cargo en v0 el inmediato EQUAL_CHAR=61(corresponde a el char '=').
650         li      v0,EQUAL_CHAR                   // 0x3d
651         // Salto a return_decode_error si el char recibido por parámetro no es igual
        a '='.
652         bne     v1,v0,return_decode_error
653         // Guardo un 0(DECODE_EQUAL) en el stack frame(12).
654         sw      zero,OFFSET_RETURN_DECODECHAR($fp)
655         // Salto a return_decode_index_or_zero.
656         b       return_decode_index_or_zero
657 return_decode_error:
658         // Cargo en v0 el inmediato DECODE_ERROR=100
659         li      v0,DECODE_ERROR                 // 0x64
660         // Guardo el DECODE_ERROR en el stack frame.
661         sw      v0,OFFSET_RETURN_DECODECHAR($fp)
662 return_decode_index_or_zero:
663         // Cargo en v0 el valor retornado por DecodeChar
664         lw      v0,OFFSET_RETURN_DECODECHAR($fp)
665
666         move    sp,$fp
667         // Restauro fp
668         lw      $fp,OFFSET_FP_DECODECHAR(sp)
669         // Destruyo el stack frame
670         addu    sp,sp,STACK_FRAME_DECODECHAR
671         // Regreso el control a la función llamante.
672         j       ra
673         .end    DecodeChar
674         //.size DecodeChar, .-DecodeChar
675
676         ///////////////  End Función DecodeChar ///////////////
677
678         ///////////////  Begin Función Decode ///////////////
679
680         .align  2
681         .globl  Decode
682         .ent    Decode
683 Decode:
684         .frame  $fp,STACK_FRAME_DECODE,ra               // vars= 24, regs= 4/0, args=
        16, extra= 8
685         //.mask 0xd0010000,-4
686         //.fmask        0x00000000,0
687         .set    noreorder
688         .cpload t9
689         .set    reorder
690
```

```
691         // Creación del stack frame
692         subu    sp,sp,STACK_FRAME_DECODE
693         .cprestore 16
694
695         sw      ra,OFFSET_RA_DECODE(sp)
696         sw      $fp,OFFSET_FP_DECODE(sp)
697         sw      gp,OFFSET_GP_DECODE(sp)
698         sw      s0,OFFSET_S0_DECODE(sp)
699
700         // De aquí al final de la función uso $fp en lugar de sp.
701         move    $fp,sp
702
703         // Guardo en el stack frame los parámetros recibidos.
704         // a0=puntero a buffer_input
705         sw      a0,OFFSET_BUFFER_INPUT_ENCODE($fp)
706         // Guardo en el stack frame los parámetros recibidos.
707         // a1=puntero a buffer_output
708         sw      a1,OFFSET_BUFFER_OUTPUT_ENCODE($fp)
709         // Guardo un 0 en el stack frame(OFFSET_I_DECODE). Inicializo 'i'.
710         sw      zero,OFFSET_I_DECODE($fp)
711 loop_decode_char:
712         // Cargo en v0 el valor de 'i' guardado en el stack frame.
713         lw      v0,OFFSET_I_DECODE($fp)
714         // Si (i < SIZE_DECODE_CHAR), guardo TRUE en v0, sino FALSE.
715         sltu    v0,v0,SIZE_DECODE_CHAR
716         // Salto a if_decode_char si sigo dentro del bucle.
717         bne     v0,zero,if_decode_char
718         // Salto a main_shift
719         b       main_shift
720 if_decode_char:
721         // Cargo en v1 el valor de 'i'.
722         lw      v1,OFFSET_I_DECODE($fp)
723         // Cargo en v0 el valor de fp + OFFSET_CHARS0_ENCODE ???
724         addu    v0,$fp,OFFSET_CHARS0_ENCODE
725         // Cargo en s0 el valor de buf_input[i]
726         addu    s0,v0,v1
727         // Cargo en v1 el puntero a buf_input
728         lw      v1,OFFSET_BUFFER_INPUT_ENCODE($fp)
729         // Cargo en v0 el valor de 'i'.
730         lw      v0,OFFSET_I_DECODE($fp)
731         // Me desplazo por el vector(buf_input[i])
732         addu    v0,v1,v0
733         // Cargo en v0 el valor del buf_input[i](1 byte).
734         lb      v0,0(v0)
735         // Asigna el valor del byte a a0 antes de llamar a la función.
736         move    a0,v0
737         // Carga en t9 la direccion de la funcion DecodeChar.
738         la      t9,DecodeChar
739         // Hace el llamado a la función.
740         jal     ra,t9
741         // Guardo en s0 el resultado de la función.
742         // El valor regresa en el registro v0
743         sb      v0,0(s0)
744         // Cargo en v1 el valor de 'i'.
745         lw      v1,OFFSET_I_DECODE($fp)
746         // Cargo en v0 el valor de fp + OFFSET_CHARS_ENCODE ???
747         addu    v0,$fp,OFFSET_CHARS0_ENCODE
748         // Cargo en v0 el valor de chars[i](direccion).
```

```
749         addu    v0,v0,v1
750         // Cargo en v1 el byte apuntado.
751         lbu     v1,0(v0)
752         // Cargo en v0 el DECODE_ERROR
753         li      v0,DECODE_ERROR                 // 0x64
754         // Si chars[i] != DECODE_ERROR salto a increase_index_decode
755         bne     v1,v0,increase_index_decode
756         // Guarda en el stack frame un 0.
757         sw      zero,OFFSET_RETURN_ENCODE($fp)
758         // Si chars[i] == DECODE_ERROR retorno un 0.
759         b       return_zero
760 increase_index_decode:
761         // Cargo en v0 el valor de 'i'.
762         lw      v0,OFFSET_I_DECODE($fp)
763         // Sumo en 1 el valor de 'i'(i++).
764         addu    v0,v0,1
765         // Guardo el valor modificado en el stack frame.
766         sw      v0,OFFSET_I_DECODE($fp)
767         // Salto a loop_decode_char
768         b       loop_decode_char
769 main_shift:
770         // Cargo en v0 la dirección de chars[0]
771         lbu     v0,OFFSET_CHARS0_ENCODE($fp)
772         // Hago un shift left logical de SHIFT_2 y lo asigno a v0.
773         sll     v0,v0,SHIFT_2
774         // Guardo el valor en el stack frame.
775         sb      v0,OFFSET_CHAR0_AUX_ENCODE($fp)
776         // Cargo el valor de chars[1] en v0.
777         lbu     v0,OFFSET_CHARS1_ENCODE($fp)
778         // Hago un shift left logical de SHIFT_2 y lo asigno a v0.
779         srl     v0,v0,SHIFT_4
780         // Guardo en el stack frame el valor shifteado.
781         sb      v0,OFFSET_CHAR1_AUX_ENCODE($fp)
782         // Cargo en v1 char1_aux(chars[0] luego de ser shifteado).
783         lbu     v1,OFFSET_CHAR0_AUX_ENCODE($fp)
784         // Cargo en v0 char2_aux(chars[1] luego de ser shifteado).
785         lbu     v0,OFFSET_CHAR1_AUX_ENCODE($fp)
786         // Hago un or de v1 y v0 y lo asigno a v0.
787         or      v0,v1,v0
788         // Guardo en valor en el stack frame.
789         sb      v0,OFFSET_CHAR0_AUX_ENCODE($fp)
790         // Cargo en v1 el puntero al buffer_output.
791         lw      v1,OFFSET_BUFFER_OUTPUT_ENCODE($fp)
792         // Cargo en v0 char1_aux(chars[0] luego de ser shifteado).
793         lbu     v0,OFFSET_CHAR0_AUX_ENCODE($fp)
794         // Guardo en el vector buffer_output el valor de char1_aux.
795         sb      v0,0(v1)
796         // Cargo el valor de chars[1] en v0.
797         lbu     v0,OFFSET_CHARS1_ENCODE($fp)
798         // Hago un shift left de 4 posiciones y lo guardo en v0.
799         sll     v0,v0,SHIFT_4
800         // Guardo en el stack frame el valor shifteado.
801         sb      v0,OFFSET_CHAR0_AUX_ENCODE($fp)
802         // Cargo en v0 chars[2].
803         lbu     v0,OFFSET_CHARS2_ENCODE($fp)
804         // Hago un shift rigth de 2 de chars[2] y lo guardo en v0.
805         srl     v0,v0,SHIFT_2
806         // Guardo en stack frame el valor shifteado.
```

```
807         sb      v0,OFFSET_CHAR1_AUX_ENCODE($fp)
808         // Cargo en v1 y v0 los valores shifteados anteriormente.
809         lbu     v1,OFFSET_CHAR1_AUX_ENCODE($fp)
810         lbu     v0,OFFSET_CHAR0_AUX_ENCODE($fp)
811         // Hago un or de v1 y v0 y lo asigno a v0.
812         or      v0,v1,v0
813         // Vuelvo a guardar en el stack frame el resultado del or.
814         // (**)
815         sb      v0,OFFSET_CHAR1_AUX_ENCODE($fp)
816         // Cargo en v0 el puntero al buffer_output.
817         lw      v0,OFFSET_BUFFER_OUTPUT_ENCODE($fp)
818         // Sumo 1 al puntero para desplazarme dentro del vector.
819         // Luego asigno el resultado a v1.
820         addu    v1,v0,1
821         // Cargo en v0 el resultado de (**).
822         lbu     v0,OFFSET_CHAR1_AUX_ENCODE($fp)
823         // Guardo en el vector buffer_output el valor (**).
824         sb      v0,0(v1)
825         // Cargo en v0 chars[2]
826         lbu     v0,OFFSET_CHARS2_ENCODE($fp)
827         // Hago un shift left de 6.
828         sll     v0,v0,SHIFT_6
829         // Guardo en el stack frame el valor shifteado.
830         // (***)
831         sb      v0,OFFSET_CHAR0_AUX_ENCODE($fp)
832         // Cargo en v0 el puntero al buffer_output.
833         lw      v0,OFFSET_BUFFER_OUTPUT_ENCODE($fp)
834         // Sumo 2 al puntero para desplazarme dentro del vector buffer_output.
835         // Luego asigno el resultado a a0.
836         addu    a0,v0,2
837         // Cargo en v1 el ultimo valor shifteado (***).
838         lbu     v1,OFFSET_CHAR0_AUX_ENCODE($fp)
839         // Cargo en v0 chars[3]
840         lbu     v0,OFFSET_CHARS3_ENCODE($fp)
841         // Hago un or de v1 y v0 y lo asigno a v0.
842         or      v0,v1,v0
843         // Guardo en el vector buffer_output el resultado del or.
844         sb      v0,0(a0)
845         // Cargo en v0 el inmediato 1(RETURNO_OK).
846         li      v0,RETURNO_OK                   // 0x1
847         // Guardo en el stack frame el valor de retorno.
848         sw      v0,OFFSET_RETURN_ENCODE($fp)
849 return_zero:
850         // Cargo en v0 el valor salvado en el stack frame(0).
851         lw      v0,OFFSET_RETURN_ENCODE($fp)
852         move    sp,$fp
853
854         // Restauro ra,fp y gp.
855         lw      ra,OFFSET_RA_DECODE(sp)
856         lw      $fp,OFFSET_FP_DECODE(sp)
857         lw      s0,OFFSET_S0_DECODE(sp)
858
859         // Destruyo el stack frame.
860         addu    sp,sp,STACK_FRAME_DECODE
861         // Devuelvo el control a la función llamante.
862         j       ra
863
864         .end    Decode
```

```
865         .size   Decode, .-Decode
```

A.0.4. Assembly encode.S

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

#define STACK_FRAME_ENCODE 16

#define OFFSET_OUTPUT_ENCODE 24
#define OFFSET_LENGTH_ENCODE 20
#define OFFSET_BUFFER_ENCODE 16
#define OFFSET_FP_ENCODE 12
#define OFFSET_GP_ENCODE 8

#define OFFSET_B4_AUX 7
#define OFFSET_B3_AUX_2 6
#define OFFSET_B3_AUX 5
#define OFFSET_B2_AUX 4
#define OFFSET_B1_AUX 3
#define OFFSET_B3 2
#define OFFSET_B2 1
#define OFFSET_B1 0

#define EQUAL_CHAR 61

        .file   1 "encode.c"
        #.section .mdebug.abi32
        #.previous
        #.abicalls
        .data
        .align  2
        .type   encoding_table, @object
        .size   encoding_table, 64
encoding_table:
        .byte   65
        .byte   66
        .byte   67
        .byte   68
        .byte   69
        .byte   70
        .byte   71
        .byte   72
        .byte   73
        .byte   74
        .byte   75
        .byte   76
        .byte   77
        .byte   78
        .byte   79
        .byte   80
        .byte   81
        .byte   82
        .byte   83
        .byte   84
        .byte   85
        .byte   86
        .byte   87
        .byte   88
        .byte   89
```

```
57          .byte   90
58          .byte   97
59          .byte   98
60          .byte   99
61          .byte   100
62          .byte   101
63          .byte   102
64          .byte   103
65          .byte   104
66          .byte   105
67          .byte   106
68          .byte   107
69          .byte   108
70          .byte   109
71          .byte   110
72          .byte   111
73          .byte   112
74          .byte   113
75          .byte   114
76          .byte   115
77          .byte   116
78          .byte   117
79          .byte   118
80          .byte   119
81          .byte   120
82          .byte   121
83          .byte   122
84          .byte   48
85          .byte   49
86          .byte   50
87          .byte   51
88          .byte   52
89          .byte   53
90          .byte   54
91          .byte   55
92          .byte   56
93          .byte   57
94          .byte   43
95          .byte   47
96          .text
97          .align  2
98          .globl  Encode
99          .ent    Encode
100
101         ####### Función Encode #######
102
103 Encode:
104         .frame  $fp,STACK_FRAME_ENCODE,ra               # vars= 8, regs= 2/0, args=
        0, extra= 8
105         #.mask  0x50000000,-4
106         #.fmask 0x00000000,0
107         .set    noreorder
108         .cpload t9
109         .set    reorder
110
111         # Creación del stack frame
112         subu    sp,sp,STACK_FRAME_ENCODE
113
```

```
114         .cprestore 0
115         sw      $fp,OFFSET_FP_ENCODE(sp)
116         sw      gp,OFFSET_GP_ENCODE(sp)
117
118         # De aquí al final de la función uso $fp en lugar de sp.
119         move    $fp,sp
120
121         # Guardo el primer parámetro *buffer
122         sw      a0,OFFSET_BUFFER_ENCODE($fp)
123         # Guardo el segundo parámetro 'length'(cantidad de caracteres)
124         sw      a1,OFFSET_LENGTH_ENCODE($fp)
125         # Guardo el puntero al array de salida(output)
126         sw      a2,OFFSET_OUTPUT_ENCODE($fp)
127
128         # Cargo en v0 el puntero al buffer.
129         lw      v0,OFFSET_BUFFER_ENCODE($fp)
130         # Cargo en v0 el 1er byte del buffer.
131         lbu     v0,0(v0)
132         # Guardo el 1er byte en el stack frame
133         sb      v0,OFFSET_B1($fp)
134         # Cargo nuevamente la dirección del buffer.
135         lw      v0,OFFSET_BUFFER_ENCODE($fp)
136         # Aumento en 1(1 byte) la dirección del buffer.
137         # Me muevo por el array del buffer.
138         addu    v0,v0,1
139         # Cargo el 2do byte del buffer.
140         lbu     v0,0(v0)
141         # Guardo el 2do byte en el stack frame.
142         sb      v0,OFFSET_B2($fp)
143         # Cargo nuevamente la dirección del buffer.
144         lw      v0,OFFSET_BUFFER_ENCODE($fp)
145         # Aumento en 2(2 byte) la dirección del buffer.
146         # Me muevo por el array del buffer.
147         addu    v0,v0,2
148         # Cargo el 2do byte del buffer.
149         lbu     v0,0(v0)
150         # Guardo el 3er byte en stack frame.
151         sb      v0,OFFSET_B3($fp)
152         # Cargo en v0 el 1er byte.
153         lbu     v0,OFFSET_B1($fp)
154         # Muevo 2 'posiciones' hacia la derecha(shift 2).
155         srl     v0,v0,2
156         # Guardo el nuevo byte en una variable auxiliar.
157         sb      v0,OFFSET_B1_AUX($fp)
158         # Cargo en v1 el puntero al output.
159         lw      v1,OFFSET_OUTPUT_ENCODE($fp)
160         # Cargo en v0 el byte shifteado.
161         lbu     v0,OFFSET_B1_AUX($fp)
162         # Cargo en v0 el caracter(byte) de la tabla encoding(encoding_table)
163         lbu     v0,encoding_table(v0)
164         # Cargo en v0 el 1er byte de la dirección del output.
165         sb      v0,0(v1)
166         # Cargo en v0 el 1er byte del buffer nuevamente.
167         lbu     v0,OFFSET_B1($fp)
168         # Muevo 6 'posiciones' hacia la izquierda(shift 6).
169         sll     v0,v0,6
170         # Guardo el resultado del shift en el Stack Frame.
171         sb      v0,OFFSET_B2_AUX($fp)
```

```
172         # Cargo el byte sin signo shifteado.
173         lbu     v0,OFFSET_B2_AUX($fp)
174         # Muevo 2 'posiciones' hacia la derecha(shift 2).
175         srl     v0,v0,2
176         # Guardo el nuevo resultado del shift en el Stack Frame.
177         sb      v0,OFFSET_B2_AUX($fp)
178         # Cargo el 2do byte del buffer en v0.
179         lbu     v0,OFFSET_B2($fp)
180         # Hago un shift left de 4 posiciones.
181         srl     v0,v0,4
182         # Cargo en v1 el resultado(byte) del shift right 2.
183         lbu     v1,OFFSET_B2_AUX($fp)
184         # Hago un 'or' entre v1 y v0 para obtener el 2 indice de la tabla.
185         or      v0,v1,v0
186         #(*) Guardo en stack frame(12) el resultado del 'or' anterior.
187         sb      v0,OFFSET_B2_AUX($fp)
188         # Cargo en v0 el puntero al output.
189         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
190         # Cargo en v1 la dirección del output + 1(1byte).
191         addu    v1,v0,1
192         # Cargo en v0 el ultimo resultado del shift(*)
193         lbu     v0,OFFSET_B2_AUX($fp)
194         # Cargo en v0 el caracter(byte) de la tabla encoding(encoding_table)
195         lbu     v0,encoding_table(v0)
196         # Salvo en el output array(output[1]) el valor del encoding_table
197         sb      v0,0(v1)
198         # Cargo en v0 el puntero al output.
199         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
200         # Sumo 2 a la dirección del output(output[2]).
201         # Me desplazo dentro del output array.
202         addu    v1,v0,2
203         # Cargo en v0 el caracter ascii 61('=').
204         li      v0,EQUAL_CHAR                   # 0x3d
205         # Salvo en el output array(output[2]) el valor '='.
206         sb      v0,0(v1)
207         # Cargo en v0 el puntero al output.
208         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
209         # Sumo 3 a la dirección del output(output[3]).
210         # Me desplazo dentro del output array.
211         addu    v1,v0,3
212         # Cargo en v0 el caracter ascii 61('=').
213         li      v0,EQUAL_CHAR                   # 0x3d
214         # Salvo en el output array(output[3]) el valor '='.
215         sb      v0,0(v1)
216         # Cargo en v1 el parametro length.
217         lw      v1,OFFSET_LENGTH_ENCODE($fp)
218         # Cargo en v0 el valor 3.
219         li      v0,3                    # 0x3
220         # Si el length == 3 salto a buffer_size_2.
221         bne     v1,v0,buffer_size_2
222         # Si el tamanio del buffer es 3 continuo NO salto.
223         # Cargo en v0 el 3er byte del buffer.
224         lbu     v0,OFFSET_B3($fp)
225         # Hago un shift right de 6.
226         srl     v0,v0,6
227         # Guardo el nuevo byte en el stack frame.
228         sb      v0,OFFSET_B3_AUX($fp)
229         # Cargo el 2do byte del buffer en v0.
```

```
230         lbu     v0,OFFSET_B2($fp)
231         # Hago un shift left de 4.
232         sll     v0,v0,4
233         # Guardo en el stack frame(14) el nuevo valor.
234         sb      v0,OFFSET_B3_AUX_2($fp)
235         # Cargo en v0 el byte shifteado sin signo.
236         lbu     v0,OFFSET_B3_AUX_2($fp)
237         # Hago un shift rigth de 2.
238         srl     v0,v0,2
239         # Guardo en el stack frame(14) el valor shifteado.
240         sb      v0,OFFSET_B3_AUX_2($fp)
241         # Cargo en v1 el valor del SF(13)
242         lbu     v1,OFFSET_B3_AUX($fp)
243         # Idem en v0(13).
244         lbu     v0,OFFSET_B3_AUX_2($fp)
245         # Hago un 'or' y almaceno en v0.
246         or      v0,v1,v0
247         # Guardo en el stack frame(13) el resultado del 'or'.
248         sb      v0,OFFSET_B3_AUX($fp)
249         # Cargo en v0 el puntero al output.
250         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
251         # Me desplazo por el vector 'output' en 2 posiciones(output[2]).
252         addu    v1,v0,2
253         # Cargo en v0 el resultado del 'or' anterior.
254         lbu     v0,OFFSET_B3_AUX($fp)
255         # Busco en la tabla de encoding el caracter que corresponde.
256         # Luego cargo el byte en v0.
257         lbu     v0,encoding_table(v0)
258         # Guardo el valor recuperado de la tabla encoding_table en el output[2].
259         sb      v0,0(v1)
260         # Cargo en v0 el 3er byte del buffer.
261         lbu     v0,OFFSET_B3($fp)
262         # Hago un shift left de 2.
263         sll     v0,v0,2
264         # Guardo en el stack frame el valor shifteado.
265         sb      v0,OFFSET_B4_AUX($fp)
266         # Cargo el byte sin signo shifteado.
267         lbu     v0,OFFSET_B4_AUX($fp)
268         # Hago un shift rigth de 2.
269         srl     v0,v0,2
270         # Guardo en el stack frame el valor shifteado.
271         sb      v0,OFFSET_B4_AUX($fp)
272         # Cargo en v0 el puntero al output.
273         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
274         # Sumo 3 a la dirección del output(output[3]).
275         # Me desplazo dentro del output array.
276         addu    v1,v0,3
277         # Cargo en v0 el ultimo valor shifteado guardado.
278         lbu     v0,OFFSET_B4_AUX($fp)
279         # Busco en la tabla de encoding el caracter que corresponde.
280         # Luego cargo el byte en v0.
281         lbu     v0,encoding_table(v0)
282         # Guardo el valor recuperado de la tabla encoding_table en el output[3].
283         sb      v0,0(v1)
284         # Salto a return_encode
285         b       return_encode
286 buffer_size_2:
287         # Cargo en v1 el valor del parámetro length.
```

```
288         lw      v1,OFFSET_LENGTH_ENCODE($fp)
289         # Cargo en v0 el valor 2.
290         li      v0,2                    # 0x2
291         # Si length != 2 salgo de la función.
292         bne     v1,v0,return_encode
293         # Cargo en v0 el 3er byte del buffer.
294         lbu     v0,OFFSET_B3($fp)
295         # Hago un shift right de 6.
296         srl     v0,v0,6
297         # Guardo en el stack frame el ultimo valor shifteado.
298         sb      v0,OFFSET_B4_AUX($fp)
299         # Cargo el 2do byte del buffer en v0.
300         lbu     v0,OFFSET_B2($fp)
301         # Hago un shift left de 4 posiciones.
302         sll     v0,v0,4
303         # Guardo en el stack frame nuevo valor shifteado.
304         sb      v0,OFFSET_B3_AUX_2($fp)
305         # Cargo en v0 el byte shifteado sin signo.
306         lbu     v0,OFFSET_B3_AUX_2($fp)
307         # Hago un shift right de 2 posiciones.
308         srl     v0,v0,2
309         # Guardo en el stack frame el valor shifteado.
310         sb      v0,OFFSET_B3_AUX_2($fp)
311         # Cargo en v1 uno de los valores shiftedos(b3aux).
312         lbu     v1,OFFSET_B4_AUX($fp)
313         # Cargo en v0 uno de los valores shiftedos(b3aux2).
314         lbu     v0,OFFSET_B3_AUX_2($fp)
315         # Hago un 'or' entre b3aux y b3aux2.
316         or      v0,v1,v0
317         # Guardo en el stack frame el resutado del 'or'.
318         sb      v0,OFFSET_B4_AUX($fp)
319         # Cargo en v0 el puntero al output.
320         lw      v0,OFFSET_OUTPUT_ENCODE($fp)
321         # Me desplazo dentro del output array y lo guardo en v1.
322         addu    v1,v0,2
323         # Cargo en v0 ultimo resultado del 'or'
324         lbu     v0,OFFSET_B4_AUX($fp)
325         # Busco en la tabla de encoding el caracter que corresponde.
326         # Luego cargo el byte en v0.
327         lbu     v0,encoding_table(v0)
328         # Guardo el valor recuperado de la tabla encoding_table en el output[2].
329         sb      v0,0(v1)
330 return_encode:
331         move    sp,$fp
332         lw      $fp,OFFSET_FP_ENCODE(sp)
333         # destruyo stack frame
334         addu    sp,sp,STACK_FRAME_ENCODE
335         j       ra
336         .end    Encode
337         #.size  Encode, .-Encode
338         #.ident "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

A.0.5. Assembly decode.S

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

#define RETURNO_OK 1
#define DECODE_ERROR   100
#define SIZE_DECODE_CHAR 4

#define SHIFT_2 2
#define SHIFT_4 4
#define SHIFT_6 6

#define EQUAL_CHAR 61

#define STACK_FRAME_DECODECHAR 32

#define OFFSET_FP_DECODECHAR 32
#define OFFSET_GP_DECODECHAR 28
#define OFFSET_CHARACTER_DECODECHAR 16
#define OFFSET_I_DECODECHAR 20
#define OFFSET_RETURN_DECODECHAR 24

#define STACK_FRAME_DECODE 64

#define OFFSET_BUFFER_OUTPUT_ENCODE 68
#define OFFSET_BUFFER_INPUT_ENCODE 64
#define OFFSET_RA_DECODE 60
#define OFFSET_FP_DECODE 56
#define OFFSET_GP_DECODE 52
#define OFFSET_S0_DECODE 48
#define OFFSET_CHAR1_AUX_ENCODE 37
#define OFFSET_CHAR0_AUX_ENCODE 36
#define OFFSET_CHARS3_ENCODE 27
#define OFFSET_CHARS2_ENCODE 26
#define OFFSET_CHARS1_ENCODE 25
#define OFFSET_CHARS0_ENCODE 24
#define OFFSET_RETURN_ENCODE 20
#define OFFSET_I_DECODE 32

        .file   1 "decode.c"
        #.section .mdebug.abi32
        #.previous
        #.abicalls
        .data
        .align  2
        .type   encoding_table, @object
        .size   encoding_table, 64
encoding_table:
        .byte   65
        .byte   66
        .byte   67
        .byte   68
        .byte   69
        .byte   70
        .byte   71
        .byte   72
        .byte   73
```

```
57          .byte   74
58          .byte   75
59          .byte   76
60          .byte   77
61          .byte   78
62          .byte   79
63          .byte   80
64          .byte   81
65          .byte   82
66          .byte   83
67          .byte   84
68          .byte   85
69          .byte   86
70          .byte   87
71          .byte   88
72          .byte   89
73          .byte   90
74          .byte   97
75          .byte   98
76          .byte   99
77          .byte   100
78          .byte   101
79          .byte   102
80          .byte   103
81          .byte   104
82          .byte   105
83          .byte   106
84          .byte   107
85          .byte   108
86          .byte   109
87          .byte   110
88          .byte   111
89          .byte   112
90          .byte   113
91          .byte   114
92          .byte   115
93          .byte   116
94          .byte   117
95          .byte   118
96          .byte   119
97          .byte   120
98          .byte   121
99          .byte   122
100         .byte   48
101         .byte   49
102         .byte   50
103         .byte   51
104         .byte   52
105         .byte   53
106         .byte   54
107         .byte   55
108         .byte   56
109         .byte   57
110         .byte   43
111         .byte   47
112         .align  2
113         .type   encoding_table_size, @object
114         .size   encoding_table_size, 4
```

```
115
116 encoding_table_size:
117         .word   64
118         .text
119         .align  2
120         .globl  DecodeChar
121         .ent    DecodeChar
122
123         ####### Begin Función DecodeChar #######
124
125 DecodeChar:
126         # Reservo espacio para el stack frame de STACK_FRAME_DECODECHAR bytes
127         .frame  $fp,STACK_FRAME_DECODECHAR,ra           # vars= 8, regs= 2/0, args=
    0, extra= 8
128         #.mask  0x50000000,-4
129         #.fmask 0x00000000,0
130         .set    noreorder
131         .cpload t9
132         .set    reorder
133
134         # Creación del stack frame STACK_FRAME_DECODECHAR
135         subu    sp,sp,STACK_FRAME_DECODECHAR
136         .cprestore 0
137
138         # Guardo fp y gp en el stack frame
139         sw      $fp,OFFSET_FP_DECODECHAR(sp)
140         sw      gp,OFFSET_GP_DECODECHAR(sp)
141         # De aquí al final de la función uso $fp en lugar de sp.
142         move    $fp,sp
143
144         # Guardo en v0 el parámetro recibido: 'character'.
145         move    v0,a0
146         # Guardo en el stack frame 'character'.
147         sb      v0,OFFSET_CHARACTER_DECODECHAR($fp)
148         # Guardo en un '0' en el stack frame.
149         # Inicializo la variable 'i'.
150         sb      zero,OFFSET_I_DECODECHAR($fp)
151 condition_loop:
152         # Cargo en v0 el byte guardado anteriormente(0 o el nuevo valor de 'i').
153         lbu     v0,OFFSET_I_DECODECHAR($fp)
154         # Cargo en v1 el size del encoding_table(64).
155         lw      v1,encoding_table_size
156         # Si (i < encoding_table_size), guardo TRUE en v0, sino FALSE.
157         slt     v0,v0,v1
158         # Salto a condition_if si v0 != 0.
159         bne     v0,zero,condition_if
160         # Brancheo a condition_if_equal
161         b       condition_if_equal
162 condition_if:
163         # Cargo en v0 el valor de 'i'.
164         lbu     v0,OFFSET_I_DECODECHAR($fp)
165         # Cargo en v1 el byte contenido en encoding_table según el valor de 'i'.
166         # encoding_table[i]
167         lbu     v1,encoding_table(v0)
168         # Cargo en v0 'character'.
169         lb      v0,OFFSET_CHARACTER_DECODECHAR($fp)
170         # Salto a increase_index si el valor recuperado del vector encoding_table
171         # es distinto al valor pasado por parámetro(character).
```

```
172         bne     v1,v0,increase_index
173         # Cargo en v0 nuevamente el valor de 'i'.
174         lbu     v0,OFFSET_I_DECODECHAR($fp)
175
176         # Guardo en el stack frame(12) el valor de 'i'
177         #sw     v0,12($fp) #VER
178         sw v0,OFFSET_RETURN_DECODECHAR($fp)
179
180         # Brancheo a return_decode_index_or_zero
181         b       return_decode_index_or_zero
182 increase_index:
183         # Cargo en v0 nuevamente el valor de 'i'.
184         lbu     v0,OFFSET_I_DECODECHAR($fp)
185         # Sumo en 1 el valor de 'i'(i++).
186         addu    v0,v0,1
187         # Guardo el valor modificado en el stack frame.
188         sb      v0,OFFSET_I_DECODECHAR($fp)
189         # Salto a condition_loop
190         b       condition_loop
191 condition_if_equal:
192         # Cargo en v1 el byte(char) recibido como parámetro.
193         # parametro: character.
194         lb      v1,OFFSET_CHARACTER_DECODECHAR($fp)
195         # Cargo en v0 el inmediato EQUAL_CHAR=61(corresponde a el char '=').
196         li      v0,EQUAL_CHAR                   # 0x3d
197         # Salto a return_decode_error si el char recibido por parámetro no es igual a
        '='.
198         bne     v1,v0,return_decode_error
199         # Guardo un 0(DECODE_EQUAL) en el stack frame(12).
200         sw      zero,OFFSET_RETURN_DECODECHAR($fp)
201         # Salto a return_decode_index_or_zero.
202         b       return_decode_index_or_zero
203 return_decode_error:
204         # Cargo en v0 el inmediato DECODE_ERROR=100
205         li      v0,DECODE_ERROR                 # 0x64
206         # Guardo el DECODE_ERROR en el stack frame.
207         sw      v0,OFFSET_RETURN_DECODECHAR($fp)
208 return_decode_index_or_zero:
209         # Cargo en v0 el valor retornado por DecodeChar
210         lw      v0,OFFSET_RETURN_DECODECHAR($fp)
211
212         move    sp,$fp
213         # Restauro fp
214         lw      $fp,OFFSET_FP_DECODECHAR(sp)
215         # Destruyo el stack frame
216         addu    sp,sp,STACK_FRAME_DECODECHAR
217         # Regreso el control a la función llamante.
218         j       ra
219         .end    DecodeChar
220         #.size  DecodeChar, .-DecodeChar
221
222         ####### End Función DecodeChar #######
223
224         ####### Begin Función Decode #######
225
226         .align  2
227         .globl  Decode
228         .ent    Decode
```

```
Decode:
        .frame  $fp,STACK_FRAME_DECODE,ra               # vars= 24, regs= 4/0, args=
    16, extra= 8
        #.mask  0xd0010000,-4
        #.fmask 0x00000000,0
        .set    noreorder
        .cpload t9
        .set    reorder

        # Creación del stack frame
        subu    sp,sp,STACK_FRAME_DECODE
        .cprestore 16

        sw      ra,OFFSET_RA_DECODE(sp)
        sw      $fp,OFFSET_FP_DECODE(sp)
        sw      gp,OFFSET_GP_DECODE(sp)
        sw      s0,OFFSET_S0_DECODE(sp)

        # De aquí al final de la función uso $fp en lugar de sp.
        move    $fp,sp

        # Guardo en el stack frame los parámetros recibidos.
        # a0=puntero a buffer_input
        sw      a0,OFFSET_BUFFER_INPUT_ENCODE($fp)
        # Guardo en el stack frame los parámetros recibidos.
        # a1=puntero a buffer_output
        sw      a1,OFFSET_BUFFER_OUTPUT_ENCODE($fp)
        # Guardo un 0 en el stack frame(OFFSET_I_DECODE). Inicializo 'i'.
        sw      zero,OFFSET_I_DECODE($fp)
loop_decode_char:
        # Cargo en v0 el valor de 'i' guardado en el stack frame.
        lw      v0,OFFSET_I_DECODE($fp)
        # Si (i < SIZE_DECODE_CHAR), guardo TRUE en v0, sino FALSE.
        sltu    v0,v0,SIZE_DECODE_CHAR
        # Salto a if_decode_char si sigo dentro del bucle.
        bne     v0,zero,if_decode_char
        # Salto a main_shift
        b       main_shift
if_decode_char:
        # Cargo en v1 el valor de 'i'.
        lw      v1,OFFSET_I_DECODE($fp)
        # Cargo en v0 el valor de fp + OFFSET_CHARS0_ENCODE ???
        addu    v0,$fp,OFFSET_CHARS0_ENCODE
        # Cargo en s0 el valor de buf_input[i]
        addu    s0,v0,v1
        # Cargo en v1 el puntero a buf_input
        lw      v1,OFFSET_BUFFER_INPUT_ENCODE($fp)
        # Cargo en v0 el valor de 'i'.
        lw      v0,OFFSET_I_DECODE($fp)
        # Me desplazo por el vector(buf_input[i])
        addu    v0,v1,v0
        # Cargo en v0 el valor del buf_input[i](1 byte).
        lb      v0,0(v0)
        # Asigna el valor del byte a a0 antes de llamar a la función.
        move    a0,v0
        # Carga en t9 la direccion de la funcion DecodeChar.
        la      t9,DecodeChar
        # Hace el llamado a la función.
```

```
286         jal     ra,t9
287         # Guardo en s0 el resultado de la función.
288         # El valor regresa en el registro v0
289         sb      v0,0(s0)
290         # Cargo en v1 el valor de 'i'.
291         lw      v1,OFFSET_I_DECODE($fp)
292         # Cargo en v0 el valor de fp + OFFSET_CHARS_ENCODE ???
293         addu    v0,$fp,OFFSET_CHARS0_ENCODE
294         # Cargo en v0 el valor de chars[i](direccion).
295         addu    v0,v0,v1
296         # Cargo en v1 el byte apuntado.
297         lbu     v1,0(v0)
298         # Cargo en v0 el DECODE_ERROR
299         li      v0,DECODE_ERROR                 # 0x64
300         # Si chars[i] != DECODE_ERROR salto a increase_index_decode
301         bne     v1,v0,increase_index_decode
302         # Guarda en el stack frame un 0.
303         sw      zero,OFFSET_RETURN_ENCODE($fp)
304         # Si chars[i] == DECODE_ERROR retorno un 0.
305         b       return_zero
306 increase_index_decode:
307         # Cargo en v0 el valor de 'i'.
308         lw      v0,OFFSET_I_DECODE($fp)
309         # Sumo en 1 el valor de 'i'(i++).
310         addu    v0,v0,1
311         # Guardo el valor modificado en el stack frame.
312         sw      v0,OFFSET_I_DECODE($fp)
313         # Salto a loop_decode_char
314         b       loop_decode_char
315 main_shift:
316         # Cargo en v0 la dirección de chars[0]
317         lbu     v0,OFFSET_CHARS0_ENCODE($fp)
318         # Hago un shift left logical de SHIFT_2 y lo asigno a v0.
319         sll     v0,v0,SHIFT_2
320         # Guardo el valor en el stack frame.
321         sb      v0,OFFSET_CHAR0_AUX_ENCODE($fp)
322         # Cargo el valor de chars[1] en v0.
323         lbu     v0,OFFSET_CHARS1_ENCODE($fp)
324         # Hago un shift left logical de SHIFT_2 y lo asigno a v0.
325         srl     v0,v0,SHIFT_4
326         # Guardo en el stack frame el valor shifteado.
327         sb      v0,OFFSET_CHAR1_AUX_ENCODE($fp)
328         # Cargo en v1 char1_aux(chars[0] luego de ser shifteado).
329         lbu     v1,OFFSET_CHAR0_AUX_ENCODE($fp)
330         # Cargo en v0 char2_aux(chars[1] luego de ser shifteado).
331         lbu     v0,OFFSET_CHAR1_AUX_ENCODE($fp)
332         # Hago un or de v1 y v0 y lo asigno a v0.
333         or      v0,v1,v0
334         # Guardo en valor en el stack frame.
335         sb      v0,OFFSET_CHAR0_AUX_ENCODE($fp)
336         # Cargo en v1 el puntero al buffer_output.
337         lw      v1,OFFSET_BUFFER_OUTPUT_ENCODE($fp)
338         # Cargo en v0 char1_aux(chars[0] luego de ser shifteado).
339         lbu     v0,OFFSET_CHAR0_AUX_ENCODE($fp)
340         # Guardo en el vector buffer_output el valor de char1_aux.
341         sb      v0,0(v1)
342         # Cargo el valor de chars[1] en v0.
343         lbu     v0,OFFSET_CHARS1_ENCODE($fp)
```

```
344         # Hago un shift left de 4 posiciones y lo guardo en v0.
345         sll     v0,v0,SHIFT_4
346         # Guardo en el stack frame el valor shifteado.
347         sb      v0,OFFSET_CHAR0_AUX_ENCODE($fp)
348         # Cargo en v0 chars[2].
349         lbu     v0,OFFSET_CHARS2_ENCODE($fp)
350         # Hago un shift rigth de 2 de chars[2] y lo guardo en v0.
351         srl     v0,v0,SHIFT_2
352         # Guardo en stack frame el valor shifteado.
353         sb      v0,OFFSET_CHAR1_AUX_ENCODE($fp)
354         # Cargo en v1 y v0 los valores shifteados anteriormente.
355         lbu     v1,OFFSET_CHAR1_AUX_ENCODE($fp)
356         lbu     v0,OFFSET_CHAR0_AUX_ENCODE($fp)
357         # Hago un or de v1 y v0 y lo asigno a v0.
358         or      v0,v1,v0
359         # Vuelvo a guardar en el stack frame el resultado del or.
360         # (**)
361         sb      v0,OFFSET_CHAR1_AUX_ENCODE($fp)
362         # Cargo en v0 el puntero al buffer_output.
363         lw      v0,OFFSET_BUFFER_OUTPUT_ENCODE($fp)
364         # Sumo 1 al puntero para desplazarme dentro del vector.
365         # Luego asigno el resultado a v1.
366         addu    v1,v0,1
367         # Cargo en v0 el resultado de (**).
368         lbu     v0,OFFSET_CHAR1_AUX_ENCODE($fp)
369         # Guardo en el vector buffer_output el valor (**).
370         sb      v0,0(v1)
371         # Cargo en v0 chars[2]
372         lbu     v0,OFFSET_CHARS2_ENCODE($fp)
373         # Hago un shift left de 6.
374         sll     v0,v0,SHIFT_6
375         # Guardo en el stack frame el valor shifteado.
376         # (***)
377         sb      v0,OFFSET_CHAR0_AUX_ENCODE($fp)
378         # Cargo en v0 el puntero al buffer_output.
379         lw      v0,OFFSET_BUFFER_OUTPUT_ENCODE($fp)
380         # Sumo 2 al puntero para desplazarme dentro del vector buffer_output.
381         # Luego asigno el resultado a a0.
382         addu    a0,v0,2
383         # Cargo en v1 el ultimo valor shifteado (***).
384         lbu     v1,OFFSET_CHAR0_AUX_ENCODE($fp)
385         # Cargo en v0 chars[3]
386         lbu     v0,OFFSET_CHARS3_ENCODE($fp)
387         # Hago un or de v1 y v0 y lo asigno a v0.
388         or      v0,v1,v0
389         # Guardo en el vector buffer_output el resultado del or.
390         sb      v0,0(a0)
391         # Cargo en v0 el inmediato 1(RETURNO_OK).
392         li      v0,RETURNO_OK                   # 0x1
393         # Guardo en el stack frame el valor de retorno.
394         sw      v0,OFFSET_RETURN_ENCODE($fp)
395 return_zero:
396         # Cargo en v0 el valor salvado en el stack frame(0).
397         lw      v0,OFFSET_RETURN_ENCODE($fp)
398         move    sp,$fp
399
400         # Restauro ra,fp y gp.
401         lw      ra,OFFSET_RA_DECODE(sp)
```

```
402         lw      $fp,OFFSET_FP_DECODE(sp)
403         lw      s0,OFFSET_S0_DECODE(sp)
404
405         # Destruyo el stack frame.
406         addu    sp,sp,STACK_FRAME_DECODE
407         # Devuelvo el control a la función llamante.
408         j       ra
409
410         .end    Decode
411         #.size  Decode, .-Decode
412         #.ident "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

A.0.6. Assembly command.s

```
        .file   1 "command.c"
        .section .mdebug.abi32
        .previous
        .abicalls
        .rdata
        .align  2
$LC0:
        .ascii  "Options:\n\000"
        .align  2
$LC1:
        .ascii  "  -V, --version    Print version and quit.\n\000"
        .align  2
$LC2:
        .ascii  "  -h, --help       Print this information.\n\000"
        .align  2
$LC3:
        .ascii  "  -i, --input      Location of the input file.\n\000"
        .align  2
$LC4:
        .ascii  "  -o, --output     Location of the output file.\n\000"
        .align  2
$LC5:
        .ascii  "  -a, --action     Program action: encode (default) or d"
        .ascii  "ecode.\n\000"
        .align  2
$LC6:
        .ascii  "Examples:\n\000"
        .align  2
$LC7:
        .ascii  "  tp0 -a encode -i ~/input -o ~/output\n\000"
        .align  2
$LC8:
        .ascii  "  tp0 -a decode\n\000"
        .text
        .align  2
        .globl  CommandHelp
        .ent    CommandHelp
CommandHelp:
        .frame  $fp,40,$ra              # vars= 0, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,40
        .cprestore 16
        sw      $ra,32($sp)
        sw      $fp,28($sp)
        sw      $gp,24($sp)
        move    $fp,$sp
        la      $a0,$LC0
        la      $t9,printf
        jal     $ra,$t9
        la      $a0,$LC1
        la      $t9,printf
        jal     $ra,$t9
```

```
57          la      $a0,$LC2
58          la      $t9,printf
59          jal     $ra,$t9
60          la      $a0,$LC3
61          la      $t9,printf
62          jal     $ra,$t9
63          la      $a0,$LC4
64          la      $t9,printf
65          jal     $ra,$t9
66          la      $a0,$LC5
67          la      $t9,printf
68          jal     $ra,$t9
69          la      $a0,$LC6
70          la      $t9,printf
71          jal     $ra,$t9
72          la      $a0,$LC7
73          la      $t9,printf
74          jal     $ra,$t9
75          la      $a0,$LC8
76          la      $t9,printf
77          jal     $ra,$t9
78          move    $sp,$fp
79          lw      $ra,32($sp)
80          lw      $fp,28($sp)
81          addu    $sp,$sp,40
82          j       $ra
83          .end    CommandHelp
84          .size   CommandHelp, .-CommandHelp
85          .rdata
86          .align  2
87  $LC9:
88          .ascii  "Version: 0.1\n\000"
89          .text
90          .align  2
91          .globl  CommandVersion
92          .ent    CommandVersion
93  CommandVersion:
94          .frame  $fp,40,$ra              # vars= 0, regs= 3/0, args= 16, extra= 8
95          .mask   0xd0000000,-8
96          .fmask  0x00000000,0
97          .set    noreorder
98          .cpload $t9
99          .set    reorder
100         subu    $sp,$sp,40
101         .cprestore 16
102         sw      $ra,32($sp)
103         sw      $fp,28($sp)
104         sw      $gp,24($sp)
105         move    $fp,$sp
106         la      $a0,$LC9
107         la      $t9,printf
108         jal     $ra,$t9
109         move    $sp,$fp
110         lw      $ra,32($sp)
111         lw      $fp,28($sp)
112         addu    $sp,$sp,40
113         j       $ra
114         .end    CommandVersion
```

```
115         .size   CommandVersion, .-CommandVersion
116         .align  2
117         .globl  CommandCreate
118         .ent    CommandCreate
119 CommandCreate:
120         .frame  $fp,40,$ra              # vars= 0, regs= 3/0, args= 16, extra= 8
121         .mask   0xd0000000,-8
122         .fmask  0x00000000,0
123         .set    noreorder
124         .cpload $t9
125         .set    reorder
126         subu    $sp,$sp,40
127         .cprestore 16
128         sw      $ra,32($sp)
129         sw      $fp,28($sp)
130         sw      $gp,24($sp)
131         move    $fp,$sp
132         sw      $a0,40($fp)
133         lw      $a0,40($fp)
134         la      $t9,FileCreate
135         jal     $ra,$t9
136         lw      $v0,40($fp)
137         addu    $v0,$v0,8
138         move    $a0,$v0
139         la      $t9,FileCreate
140         jal     $ra,$t9
141         lw      $v0,40($fp)
142         sb      $zero,24($v0)
143         lw      $v1,40($fp)
144         li      $v0,2                   # 0x2
145         sb      $v0,25($v1)
146         lw      $v0,40($fp)
147         sw      $zero,16($v0)
148         lw      $v0,40($fp)
149         sw      $zero,20($v0)
150         move    $sp,$fp
151         lw      $ra,32($sp)
152         lw      $fp,28($sp)
153         addu    $sp,$sp,40
154         j       $ra
155         .end    CommandCreate
156         .size   CommandCreate, .-CommandCreate
157         .align  2
158         .globl  CommandSetInput
159         .ent    CommandSetInput
160 CommandSetInput:
161         .frame  $fp,16,$ra              # vars= 0, regs= 2/0, args= 0, extra= 8
162         .mask   0x50000000,-4
163         .fmask  0x00000000,0
164         .set    noreorder
165         .cpload $t9
166         .set    reorder
167         subu    $sp,$sp,16
168         .cprestore 0
169         sw      $fp,12($sp)
170         sw      $gp,8($sp)
171         move    $fp,$sp
172         sw      $a0,16($fp)
```

```
173         sw      $a1,20($fp)
174         lw      $v1,16($fp)
175         lw      $v0,20($fp)
176         sw      $v0,16($v1)
177         move    $sp,$fp
178         lw      $fp,12($sp)
179         addu    $sp,$sp,16
180         j       $ra
181         .end    CommandSetInput
182         .size   CommandSetInput, .-CommandSetInput
183         .align  2
184         .globl  CommandSetOutput
185         .ent    CommandSetOutput
186 CommandSetOutput:
187         .frame  $fp,16,$ra              # vars= 0, regs= 2/0, args= 0, extra= 8
188         .mask   0x50000000,-4
189         .fmask  0x00000000,0
190         .set    noreorder
191         .cpload $t9
192         .set    reorder
193         subu    $sp,$sp,16
194         .cprestore 0
195         sw      $fp,12($sp)
196         sw      $gp,8($sp)
197         move    $fp,$sp
198         sw      $a0,16($fp)
199         sw      $a1,20($fp)
200         lw      $v1,16($fp)
201         lw      $v0,20($fp)
202         sw      $v0,20($v1)
203         move    $sp,$fp
204         lw      $fp,12($sp)
205         addu    $sp,$sp,16
206         j       $ra
207         .end    CommandSetOutput
208         .size   CommandSetOutput, .-CommandSetOutput
209         .rdata
210         .align  2
211 $LC10:
212         .ascii  "encode\000"
213         .align  2
214 $LC11:
215         .ascii  "decode\000"
216         .align  2
217 $LC12:
218         .ascii  "Encoding option should be encode/decode\000"
219         .text
220         .align  2
221         .globl  CommandSetEncodeOpt
222         .ent    CommandSetEncodeOpt
223 CommandSetEncodeOpt:
224         .frame  $fp,40,$ra              # vars= 0, regs= 3/0, args= 16, extra= 8
225         .mask   0xd0000000,-8
226         .fmask  0x00000000,0
227         .set    noreorder
228         .cpload $t9
229         .set    reorder
230         subu    $sp,$sp,40
```

```
231         .cprestore 16
232         sw      $ra,32($sp)
233         sw      $fp,28($sp)
234         sw      $gp,24($sp)
235         move    $fp,$sp
236         sw      $a0,40($fp)
237         sw      $a1,44($fp)
238         lw      $a0,44($fp)
239         la      $a1,$LC10
240         la      $t9,strcmp
241         jal     $ra,$t9
242         bne     $v0,$zero,$L11
243         lw      $v1,40($fp)
244         li      $v0,1                   # 0x1
245         sb      $v0,25($v1)
246         b       $L10
247 $L11:
248         lw      $a0,44($fp)
249         la      $a1,$LC11
250         la      $t9,strcmp
251         jal     $ra,$t9
252         bne     $v0,$zero,$L13
253         lw      $v0,40($fp)
254         sb      $zero,25($v0)
255         b       $L10
256 $L13:
257         la      $a0,__sF+176
258         la      $a1,$LC12
259         la      $t9,fprintf
260         jal     $ra,$t9
261         lw      $v1,40($fp)
262         li      $v0,1                   # 0x1
263         sb      $v0,24($v1)
264 $L10:
265         move    $sp,$fp
266         lw      $ra,32($sp)
267         lw      $fp,28($sp)
268         addu    $sp,$sp,40
269         j       $ra
270         .end    CommandSetEncodeOpt
271         .size   CommandSetEncodeOpt, .-CommandSetEncodeOpt
272         .align  2
273         .globl  CommandHasError
274         .ent    CommandHasError
275 CommandHasError:
276         .frame  $fp,24,$ra              # vars= 8, regs= 2/0, args= 0, extra= 8
277         .mask   0x50000000,-4
278         .fmask  0x00000000,0
279         .set    noreorder
280         .cpload $t9
281         .set    reorder
282         subu    $sp,$sp,24
283         .cprestore 0
284         sw      $fp,20($sp)
285         sw      $gp,16($sp)
286         move    $fp,$sp
287         sw      $a0,24($fp)
288         sw      $zero,8($fp)
```

```
289         lw      $v0,24($fp)
290         lb      $v0,24($v0)
291         bne     $v0,$zero,$L17
292         lw      $v0,24($fp)
293         lb      $v1,25($v0)
294         li      $v0,2                   # 0x2
295         beq     $v1,$v0,$L17
296         b       $L16
297 $L17:
298         li      $v0,1                   # 0x1
299         sw      $v0,8($fp)
300 $L16:
301         lw      $v0,8($fp)
302         move    $sp,$fp
303         lw      $fp,20($sp)
304         addu    $sp,$sp,24
305         j       $ra
306         .end    CommandHasError
307         .size   CommandHasError, .-CommandHasError
308         .align  2
309         .globl  CommandSetError
310         .ent    CommandSetError
311 CommandSetError:
312         .frame  $fp,16,$ra              # vars= 0, regs= 2/0, args= 0, extra= 8
313         .mask   0x50000000,-4
314         .fmask  0x00000000,0
315         .set    noreorder
316         .cpload $t9
317         .set    reorder
318         subu    $sp,$sp,16
319         .cprestore 0
320         sw      $fp,12($sp)
321         sw      $gp,8($sp)
322         move    $fp,$sp
323         sw      $a0,16($fp)
324         lw      $v1,16($fp)
325         li      $v0,1                   # 0x1
326         sb      $v0,24($v1)
327         move    $sp,$fp
328         lw      $fp,12($sp)
329         addu    $sp,$sp,16
330         j       $ra
331         .end    CommandSetError
332         .size   CommandSetError, .-CommandSetError
333         .align  2
334         .globl  CommandProcess
335         .ent    CommandProcess
336 CommandProcess:
337         .frame  $fp,40,$ra              # vars= 0, regs= 4/0, args= 16, extra= 8
338         .mask   0xd0010000,-4
339         .fmask  0x00000000,0
340         .set    noreorder
341         .cpload $t9
342         .set    reorder
343         subu    $sp,$sp,40
344         .cprestore 16
345         sw      $ra,36($sp)
346         sw      $fp,32($sp)
```

```
347         sw      $gp,28($sp)
348         sw      $s0,24($sp)
349         move    $fp,$sp
350         sw      $a0,40($fp)
351         lw      $s0,40($fp)
352         lw      $v0,40($fp)
353         lw      $a0,40($fp)
354         lw      $a1,16($v0)
355         la      $t9,FileOpenForRead
356         jal     $ra,$t9
357         sb      $v0,24($s0)
358         lw      $v0,40($fp)
359         lb      $v0,24($v0)
360         bne     $v0,$zero,$L20
361         lw      $s0,40($fp)
362         lw      $v0,40($fp)
363         addu    $v0,$v0,8
364         lw      $v1,40($fp)
365         move    $a0,$v0
366         lw      $a1,20($v1)
367         la      $t9,FileOpenForWrite
368         jal     $ra,$t9
369         sb      $v0,24($s0)
370 $L20:
371         lw      $v0,40($fp)
372         lb      $v0,24($v0)
373         bne     $v0,$zero,$L21
374         lw      $s0,40($fp)
375         lw      $a0,40($fp)
376         la      $t9,_CommandEncodeDecode
377         jal     $ra,$t9
378         sb      $v0,24($s0)
379         lw      $a0,40($fp)
380         la      $t9,FileClose
381         jal     $ra,$t9
382         lw      $v0,40($fp)
383         addu    $v0,$v0,8
384         move    $a0,$v0
385         la      $t9,FileClose
386         jal     $ra,$t9
387         b       $L22
388 $L21:
389         lw      $a0,40($fp)
390         la      $t9,FileClose
391         jal     $ra,$t9
392 $L22:
393         lw      $v0,40($fp)
394         lb      $v0,24($v0)
395         move    $sp,$fp
396         lw      $ra,36($sp)
397         lw      $fp,32($sp)
398         lw      $s0,24($sp)
399         addu    $sp,$sp,40
400         j       $ra
401         .end    CommandProcess
402         .size   CommandProcess, .-CommandProcess
403         .rdata
404         .align  2
```

```
$LC13:
        .ascii  "\n\000"
        .align  2
$LC14:
        .ascii  "Longitud de archivo no es multiplo de 4\n\000"
        .align  2
$LC15:
        .ascii  "Caracteres invalidos en archivo codificado: \000"
        .align  2
$LC16:
        .ascii  "%c\000"
        .text
        .align  2
        .globl  _CommandEncodeDecode
        .ent    _CommandEncodeDecode
_CommandEncodeDecode:
        .frame  $fp,72,$ra              # vars= 32, regs= 3/0, args= 16, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,72
        .cprestore 16
        sw      $ra,64($sp)
        sw      $fp,60($sp)
        sw      $gp,56($sp)
        move    $fp,$sp
        sw      $a0,72($fp)
        sb      $zero,40($fp)
        lw      $v0,72($fp)
        lb      $v1,25($v0)
        li      $v0,1                   # 0x1
        bne     $v1,$v0,$L24
$L25:
        lw      $a0,72($fp)
        la      $t9,FileEofReached
        jal     $ra,$t9
        beq     $v0,$zero,$L27
        b       $L24
$L27:
        addu    $a0,$fp,24
        move    $a1,$zero
        li      $a2,3                   # 0x3
        la      $t9,memset
        jal     $ra,$t9
        lw      $a0,72($fp)
        addu    $a1,$fp,24
        li      $a2,3                   # 0x3
        la      $t9,FileRead
        jal     $ra,$t9
        sw      $v0,44($fp)
        lw      $v0,44($fp)
        beq     $v0,$zero,$L25
        addu    $v0,$fp,32
        addu    $a0,$fp,24
        lw      $a1,44($fp)
        move    $a2,$v0
```

```
463         la      $t9,Encode
464         jal     $ra,$t9
465         lw      $v0,72($fp)
466         addu    $v0,$v0,8
467         addu    $v1,$fp,32
468         move    $a0,$v0
469         move    $a1,$v1
470         li      $a2,4                   # 0x4
471         la      $t9,FileWrite
472         jal     $ra,$t9
473         lbu     $v0,40($fp)
474         addu    $v0,$v0,1
475         sb      $v0,40($fp)
476         lbu     $v1,40($fp)
477         li      $v0,18                  # 0x12
478         bne     $v1,$v0,$L25
479         lw      $v0,72($fp)
480         addu    $v0,$v0,8
481         move    $a0,$v0
482         la      $a1,$LC13
483         li      $a2,1                   # 0x1
484         la      $t9,FileWrite
485         jal     $ra,$t9
486         sb      $zero,40($fp)
487         b       $L25
488 $L24:
489         lw      $v0,72($fp)
490         lb      $v0,25($v0)
491         bne     $v0,$zero,$L30
492 $L31:
493         lw      $a0,72($fp)
494         la      $t9,FileEofReached
495         jal     $ra,$t9
496         bne     $v0,$zero,$L30
497         lw      $a0,72($fp)
498         la      $t9,CommandHasError
499         jal     $ra,$t9
500         bne     $v0,$zero,$L30
501         addu    $v0,$fp,32
502         lw      $a0,72($fp)
503         move    $a1,$v0
504         li      $a2,4                   # 0x4
505         la      $t9,FileRead
506         jal     $ra,$t9
507         sw      $v0,44($fp)
508         lw      $v0,44($fp)
509         beq     $v0,$zero,$L31
510         lw      $v1,44($fp)
511         li      $v0,4                   # 0x4
512         beq     $v1,$v0,$L36
513         la      $a0,__sF+176
514         la      $a1,$LC14
515         la      $t9,fprintf
516         jal     $ra,$t9
517         lw      $a0,72($fp)
518         la      $t9,CommandSetError
519         jal     $ra,$t9
520         b       $L31
```

```
$L36:
        lbu     $v0,40($fp)
        addu    $v0,$v0,1
        sb      $v0,40($fp)
        lbu     $v1,40($fp)
        li      $v0,18                  # 0x12
        bne     $v1,$v0,$L38
        addu    $v0,$fp,48
        lw      $a0,72($fp)
        move    $a1,$v0
        li      $a2,1                   # 0x1
        la      $t9,FileRead
        jal     $ra,$t9
        sb      $zero,40($fp)
$L38:
        addu    $v0,$fp,32
        move    $a0,$v0
        addu    $a1,$fp,24
        la      $t9,Decode
        jal     $ra,$t9
        beq     $v0,$zero,$L39
        sb      $zero,49($fp)
        lbu     $v1,34($fp)
        li      $v0,61                  # 0x3d
        bne     $v1,$v0,$L40
        lbu     $v0,49($fp)
        addu    $v0,$v0,1
        sb      $v0,49($fp)
$L40:
        lbu     $v1,35($fp)
        li      $v0,61                  # 0x3d
        bne     $v1,$v0,$L41
        lbu     $v0,49($fp)
        addu    $v0,$v0,1
        sb      $v0,49($fp)
$L41:
        lw      $v0,72($fp)
        addu    $a0,$v0,8
        lb      $v1,49($fp)
        li      $v0,3                   # 0x3
        subu    $v0,$v0,$v1
        addu    $a1,$fp,24
        move    $a2,$v0
        la      $t9,FileWrite
        jal     $ra,$t9
        b       $L31
$L39:
        la      $a0,__sF+176
        la      $a1,$LC15
        la      $t9,fprintf
        jal     $ra,$t9
        sw      $zero,52($fp)
$L43:
        lw      $v0,52($fp)
        sltu    $v0,$v0,4
        bne     $v0,$zero,$L46
        b       $L44
$L46:
```

```
579         addu    $v1,$fp,32
580         lw      $v0,52($fp)
581         addu    $v0,$v1,$v0
582         lbu     $v0,0($v0)
583         la      $a0,__sF+176
584         la      $a1,$LC16
585         move    $a2,$v0
586         la      $t9,fprintf
587         jal     $ra,$t9
588         lw      $v0,52($fp)
589         addu    $v0,$v0,1
590         sw      $v0,52($fp)
591         b       $L43
592 $L44:
593         lw      $a0,72($fp)
594         la      $t9,CommandSetError
595         jal     $ra,$t9
596         b       $L31
597 $L30:
598         lw      $v0,72($fp)
599         lb      $v0,24($v0)
600         move    $sp,$fp
601         lw      $ra,64($sp)
602         lw      $fp,60($sp)
603         addu    $sp,$sp,72
604         j       $ra
605         .end    _CommandEncodeDecode
606         .size   _CommandEncodeDecode, .-_CommandEncodeDecode
607         .rdata
608         .align  2
609 $LC17:
610         .ascii  "Invalid Arguments\n\000"
611         .text
612         .align  2
613         .globl  CommandErrArg
614         .ent    CommandErrArg
615 CommandErrArg:
616         .frame  $fp,40,$ra              # vars= 0, regs= 3/0, args= 16, extra= 8
617         .mask   0xd0000000,-8
618         .fmask  0x00000000,0
619         .set    noreorder
620         .cpload $t9
621         .set    reorder
622         subu    $sp,$sp,40
623         .cprestore 16
624         sw      $ra,32($sp)
625         sw      $fp,28($sp)
626         sw      $gp,24($sp)
627         move    $fp,$sp
628         la      $a0,__sF+176
629         la      $a1,$LC17
630         la      $t9,fprintf
631         jal     $ra,$t9
632         la      $a0,__sF+176
633         la      $a1,$LC0
634         la      $t9,fprintf
635         jal     $ra,$t9
636         la      $a0,__sF+176
```

```
637         la      $a1,$LC1
638         la      $t9,fprintf
639         jal     $ra,$t9
640         la      $a0,__sF+176
641         la      $a1,$LC2
642         la      $t9,fprintf
643         jal     $ra,$t9
644         la      $a0,__sF+176
645         la      $a1,$LC3
646         la      $t9,fprintf
647         jal     $ra,$t9
648         la      $a0,__sF+176
649         la      $a1,$LC4
650         la      $t9,fprintf
651         jal     $ra,$t9
652         la      $a0,__sF+176
653         la      $a1,$LC5
654         la      $t9,fprintf
655         jal     $ra,$t9
656         la      $a0,__sF+176
657         la      $a1,$LC6
658         la      $t9,fprintf
659         jal     $ra,$t9
660         la      $a0,__sF+176
661         la      $a1,$LC7
662         la      $t9,fprintf
663         jal     $ra,$t9
664         la      $a0,__sF+176
665         la      $a1,$LC8
666         la      $t9,fprintf
667         jal     $ra,$t9
668         move    $sp,$fp
669         lw      $ra,32($sp)
670         lw      $fp,28($sp)
671         addu    $sp,$sp,40
672         j       $ra
673         .end    CommandErrArg
674         .size   CommandErrArg, .-CommandErrArg
675         .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

A.0.7. Assembly decode.s

```
        .file   1 "decode.c"
        .section .mdebug.abi32
        .previous
        .abicalls
        .data
        .align  2
        .type   encoding_table, @object
        .size   encoding_table, 64
encoding_table:
        .byte   65
        .byte   66
        .byte   67
        .byte   68
        .byte   69
        .byte   70
        .byte   71
        .byte   72
        .byte   73
        .byte   74
        .byte   75
        .byte   76
        .byte   77
        .byte   78
        .byte   79
        .byte   80
        .byte   81
        .byte   82
        .byte   83
        .byte   84
        .byte   85
        .byte   86
        .byte   87
        .byte   88
        .byte   89
        .byte   90
        .byte   97
        .byte   98
        .byte   99
        .byte   100
        .byte   101
        .byte   102
        .byte   103
        .byte   104
        .byte   105
        .byte   106
        .byte   107
        .byte   108
        .byte   109
        .byte   110
        .byte   111
        .byte   112
        .byte   113
        .byte   114
        .byte   115
        .byte   116
        .byte   117
```

```
57          .byte   118
58          .byte   119
59          .byte   120
60          .byte   121
61          .byte   122
62          .byte   48
63          .byte   49
64          .byte   50
65          .byte   51
66          .byte   52
67          .byte   53
68          .byte   54
69          .byte   55
70          .byte   56
71          .byte   57
72          .byte   43
73          .byte   47
74          .align  2
75          .type   encoding_table_size, @object
76          .size   encoding_table_size, 4
77  encoding_table_size:
78          .word   64
79          .text
80          .align  2
81          .globl  DecodeChar
82          .ent    DecodeChar
83  DecodeChar:
84          .frame  $fp,24,$ra              # vars= 8, regs= 2/0, args= 0, extra= 8
85          .mask   0x50000000,-4
86          .fmask  0x00000000,0
87          .set    noreorder
88          .cpload $t9
89          .set    reorder
90          subu    $sp,$sp,24
91          .cprestore 0
92          sw      $fp,20($sp)
93          sw      $gp,16($sp)
94          move    $fp,$sp
95          move    $v0,$a0
96          sb      $v0,8($fp)
97          sb      $zero,9($fp)
98  $L2:
99          lbu     $v0,9($fp)
100         lw      $v1,encoding_table_size
101         slt     $v0,$v0,$v1
102         bne     $v0,$zero,$L5
103         b       $L3
104 $L5:
105         lbu     $v0,9($fp)
106         lbu     $v1,encoding_table($v0)
107         lb      $v0,8($fp)
108         bne     $v1,$v0,$L4
109         lbu     $v0,9($fp)
110         sw      $v0,12($fp)
111         b       $L1
112 $L4:
113         lbu     $v0,9($fp)
114         addu    $v0,$v0,1
```

```
115         sb      $v0,9($fp)
116         b       $L2
117 $L3:
118         lb      $v1,8($fp)
119         li      $v0,61                  # 0x3d
120         bne     $v1,$v0,$L7
121         sw      $zero,12($fp)
122         b       $L1
123 $L7:
124         li      $v0,100                 # 0x64
125         sw      $v0,12($fp)
126 $L1:
127         lw      $v0,12($fp)
128         move    $sp,$fp
129         lw      $fp,20($sp)
130         addu    $sp,$sp,24
131         j       $ra
132         .end    DecodeChar
133         .size   DecodeChar, .-DecodeChar
134         .align  2
135         .globl  Decode
136         .ent    Decode
137 Decode:
138         .frame  $fp,64,$ra              # vars= 24, regs= 4/0, args= 16, extra= 8
139         .mask   0xd0010000,-4
140         .fmask  0x00000000,0
141         .set    noreorder
142         .cpload $t9
143         .set    reorder
144         subu    $sp,$sp,64
145         .cprestore 16
146         sw      $ra,60($sp)
147         sw      $fp,56($sp)
148         sw      $gp,52($sp)
149         sw      $s0,48($sp)
150         move    $fp,$sp
151         sw      $a0,64($fp)
152         sw      $a1,68($fp)
153         sw      $zero,32($fp)
154 $L9:
155         lw      $v0,32($fp)
156         sltu    $v0,$v0,4
157         bne     $v0,$zero,$L12
158         b       $L10
159 $L12:
160         lw      $v1,32($fp)
161         addu    $v0,$fp,24
162         addu    $s0,$v0,$v1
163         lw      $v1,64($fp)
164         lw      $v0,32($fp)
165         addu    $v0,$v1,$v0
166         lb      $v0,0($v0)
167         move    $a0,$v0
168         la      $t9,DecodeChar
169         jal     $ra,$t9
170         sb      $v0,0($s0)
171         lw      $v1,32($fp)
172         addu    $v0,$fp,24
```

```
173         addu    $v0,$v0,$v1
174         lbu     $v1,0($v0)
175         li      $v0,100                 # 0x64
176         bne     $v1,$v0,$L11
177         sw      $zero,40($fp)
178         b       $L8
179 $L11:
180         lw      $v0,32($fp)
181         addu    $v0,$v0,1
182         sw      $v0,32($fp)
183         b       $L9
184 $L10:
185         lbu     $v0,24($fp)
186         sll     $v0,$v0,2
187         sb      $v0,36($fp)
188         lbu     $v0,25($fp)
189         srl     $v0,$v0,4
190         sb      $v0,37($fp)
191         lbu     $v1,36($fp)
192         lbu     $v0,37($fp)
193         or      $v0,$v1,$v0
194         sb      $v0,36($fp)
195         lw      $v1,68($fp)
196         lbu     $v0,36($fp)
197         sb      $v0,0($v1)
198         lbu     $v0,25($fp)
199         sll     $v0,$v0,4
200         sb      $v0,36($fp)
201         lbu     $v0,26($fp)
202         srl     $v0,$v0,2
203         sb      $v0,37($fp)
204         lbu     $v1,37($fp)
205         lbu     $v0,36($fp)
206         or      $v0,$v1,$v0
207         sb      $v0,37($fp)
208         lw      $v0,68($fp)
209         addu    $v1,$v0,1
210         lbu     $v0,37($fp)
211         sb      $v0,0($v1)
212         lbu     $v0,26($fp)
213         sll     $v0,$v0,6
214         sb      $v0,36($fp)
215         lw      $v0,68($fp)
216         addu    $a0,$v0,2
217         lbu     $v1,36($fp)
218         lbu     $v0,27($fp)
219         or      $v0,$v1,$v0
220         sb      $v0,0($a0)
221         li      $v0,1                   # 0x1
222         sw      $v0,40($fp)
223 $L8:
224         lw      $v0,40($fp)
225         move    $sp,$fp
226         lw      $ra,60($sp)
227         lw      $fp,56($sp)
228         lw      $s0,48($sp)
229         addu    $sp,$sp,64
230         j       $ra
```

```
231         .end    Decode
232         .size   Decode, .-Decode
233         .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

A.0.8. Assembly file.s

```
        .file   1 "file.c"
        .section .mdebug.abi32
        .previous
        .abicalls
        .text
        .align  2
        .globl  FileCreate
        .ent    FileCreate
FileCreate:
        .frame  $fp,16,$ra              # vars= 0, regs= 2/0, args= 0, extra= 8
        .mask   0x50000000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,16
        .cprestore 0
        sw      $fp,12($sp)
        sw      $gp,8($sp)
        move    $fp,$sp
        sw      $a0,16($fp)
        lw      $v0,16($fp)
        sw      $zero,0($v0)
        lw      $v0,16($fp)
        sb      $zero,4($v0)
        move    $sp,$fp
        lw      $fp,12($sp)
        addu    $sp,$sp,16
        j       $ra
        .end    FileCreate
        .size   FileCreate, .-FileCreate
        .rdata
        .align  2
$LC0:
        .ascii  "rb\000"
        .align  2
$LC1:
        .ascii  "File Open Error; %s\n\000"
        .text
        .align  2
        .globl  FileOpenForRead
        .ent    FileOpenForRead
FileOpenForRead:
        .frame  $fp,48,$ra              # vars= 8, regs= 4/0, args= 16, extra= 8
        .mask   0xd0010000,-4
        .fmask  0x00000000,0
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,48
        .cprestore 16
        sw      $ra,44($sp)
        sw      $fp,40($sp)
        sw      $gp,36($sp)
        sw      $s0,32($sp)
        move    $fp,$sp
```

```
        sw      $a0,48($fp)
        sw      $a1,52($fp)
        lw      $v0,52($fp)
        bne     $v0,$zero,$L19
        lw      $v1,48($fp)
        la      $v0,__sF
        sw      $v0,0($v1)
        b       $L20
$L19:
        lw      $s0,48($fp)
        lw      $a0,52($fp)
        la      $a1,$LC0
        la      $t9,fopen
        jal     $ra,$t9
        sw      $v0,0($s0)
        lw      $v0,48($fp)
        lw      $v0,0($v0)
        bne     $v0,$zero,$L20
        la      $t9,__errno
        jal     $ra,$t9
        lw      $v0,0($v0)
        sw      $v0,24($fp)
        lw      $a0,24($fp)
        la      $t9,strerror
        jal     $ra,$t9
        la      $a0,__sF+176
        la      $a1,$LC1
        move    $a2,$v0
        la      $t9,fprintf
        jal     $ra,$t9
        li      $v0,1                   # 0x1
        sw      $v0,28($fp)
        b       $L18
$L20:
        sw      $zero,28($fp)
$L18:
        lw      $v0,28($fp)
        move    $sp,$fp
        lw      $ra,44($sp)
        lw      $fp,40($sp)
        lw      $s0,32($sp)
        addu    $sp,$sp,48
        j       $ra
        .end    FileOpenForRead
        .size   FileOpenForRead, .-FileOpenForRead
        .rdata
        .align  2
$LC2:
        .ascii  "wb\000"
        .text
        .align  2
        .globl  FileOpenForWrite
        .ent    FileOpenForWrite
FileOpenForWrite:
        .frame  $fp,48,$ra              # vars= 8, regs= 4/0, args= 16, extra= 8
        .mask   0xd0010000,-4
        .fmask  0x00000000,0
        .set    noreorder
```

```
115         .cpload $t9
116         .set    reorder
117         subu    $sp,$sp,48
118         .cprestore 16
119         sw      $ra,44($sp)
120         sw      $fp,40($sp)
121         sw      $gp,36($sp)
122         sw      $s0,32($sp)
123         move    $fp,$sp
124         sw      $a0,48($fp)
125         sw      $a1,52($fp)
126         lw      $v0,52($fp)
127         bne     $v0,$zero,$L23
128         lw      $v1,48($fp)
129         la      $v0,__sF+88
130         sw      $v0,0($v1)
131         b       $L24
132 $L23:
133         lw      $s0,48($fp)
134         lw      $a0,52($fp)
135         la      $a1,$LC2
136         la      $t9,fopen
137         jal     $ra,$t9
138         sw      $v0,0($s0)
139         lw      $v0,48($fp)
140         lw      $v0,0($v0)
141         bne     $v0,$zero,$L24
142         la      $t9,__errno
143         jal     $ra,$t9
144         lw      $v0,0($v0)
145         sw      $v0,24($fp)
146         lw      $a0,24($fp)
147         la      $t9,strerror
148         jal     $ra,$t9
149         la      $a0,__sF+176
150         la      $a1,$LC1
151         move    $a2,$v0
152         la      $t9,fprintf
153         jal     $ra,$t9
154         li      $v0,1                   # 0x1
155         sw      $v0,28($fp)
156         b       $L22
157 $L24:
158         sw      $zero,28($fp)
159 $L22:
160         lw      $v0,28($fp)
161         move    $sp,$fp
162         lw      $ra,44($sp)
163         lw      $fp,40($sp)
164         lw      $s0,32($sp)
165         addu    $sp,$sp,48
166         j       $ra
167         .end    FileOpenForWrite
168         .size   FileOpenForWrite, .-FileOpenForWrite
169         .rdata
170         .align  2
171 $LC3:
172         .ascii  "File Close Error; %s\n\000"
```

```
173         .text
174         .align  2
175         .globl  FileClose
176         .ent    FileClose
177 FileClose:
178         .frame  $fp,56,$ra              # vars= 16, regs= 3/0, args= 16, extra= 8
179         .mask   0xd0000000,-8
180         .fmask  0x00000000,0
181         .set    noreorder
182         .cpload $t9
183         .set    reorder
184         subu    $sp,$sp,56
185         .cprestore 16
186         sw      $ra,48($sp)
187         sw      $fp,44($sp)
188         sw      $gp,40($sp)
189         move    $fp,$sp
190         sw      $a0,56($fp)
191         lw      $v0,56($fp)
192         lw      $v1,0($v0)
193         la      $v0,__sF
194         beq     $v1,$v0,$L28
195         lw      $v0,56($fp)
196         lw      $v1,0($v0)
197         la      $v0,__sF+88
198         beq     $v1,$v0,$L28
199         b       $L27
200 $L28:
201         sw      $zero,32($fp)
202         b       $L26
203 $L27:
204         lw      $v0,56($fp)
205         lw      $a0,0($v0)
206         la      $t9,fclose
207         jal     $ra,$t9
208         sw      $v0,24($fp)
209         lw      $v1,24($fp)
210         li      $v0,-1                  # 0xffffffffffffffff
211         bne     $v1,$v0,$L29
212         la      $t9,__errno
213         jal     $ra,$t9
214         lw      $v0,0($v0)
215         sw      $v0,28($fp)
216         lw      $a0,28($fp)
217         la      $t9,strerror
218         jal     $ra,$t9
219         la      $a0,__sF+176
220         la      $a1,$LC3
221         move    $a2,$v0
222         la      $t9,fprintf
223         jal     $ra,$t9
224         li      $v0,1                   # 0x1
225         sw      $v0,32($fp)
226         b       $L26
227 $L29:
228         sw      $zero,32($fp)
229 $L26:
230         lw      $v0,32($fp)
```

```
231         move    $sp,$fp
232         lw      $ra,48($sp)
233         lw      $fp,44($sp)
234         addu    $sp,$sp,56
235         j       $ra
236         .end    FileClose
237         .size   FileClose, .-FileClose
238         .align  2
239         .globl  FileRead
240         .ent    FileRead
241 FileRead:
242         .frame  $fp,48,$ra              # vars= 8, regs= 3/0, args= 16, extra= 8
243         .mask   0xd0000000,-8
244         .fmask  0x00000000,0
245         .set    noreorder
246         .cpload $t9
247         .set    reorder
248         subu    $sp,$sp,48
249         .cprestore 16
250         sw      $ra,40($sp)
251         sw      $fp,36($sp)
252         sw      $gp,32($sp)
253         move    $fp,$sp
254         sw      $a0,48($fp)
255         sw      $a1,52($fp)
256         sw      $a2,56($fp)
257         sw      $zero,24($fp)
258         lw      $a0,48($fp)
259         la      $t9,FileEofReached
260         jal     $ra,$t9
261         bne     $v0,$zero,$L31
262         lw      $v0,48($fp)
263         lw      $a0,52($fp)
264         li      $a1,1                   # 0x1
265         lw      $a2,56($fp)
266         lw      $a3,0($v0)
267         la      $t9,fread
268         jal     $ra,$t9
269         sw      $v0,24($fp)
270         lw      $v0,48($fp)
271         lw      $v0,0($v0)
272         lhu     $v0,12($v0)
273         srl     $v0,$v0,5
274         andi    $v0,$v0,0x1
275         beq     $v0,$zero,$L31
276         lw      $v1,48($fp)
277         li      $v0,1                   # 0x1
278         sb      $v0,4($v1)
279 $L31:
280         lw      $v0,24($fp)
281         move    $sp,$fp
282         lw      $ra,40($sp)
283         lw      $fp,36($sp)
284         addu    $sp,$sp,48
285         j       $ra
286         .end    FileRead
287         .size   FileRead, .-FileRead
288         .align  2
```

```
289         .globl  FileEofReached
290         .ent    FileEofReached
291 FileEofReached:
292         .frame  $fp,16,$ra              # vars= 0, regs= 2/0, args= 0, extra= 8
293         .mask   0x50000000,-4
294         .fmask  0x00000000,0
295         .set    noreorder
296         .cpload $t9
297         .set    reorder
298         subu    $sp,$sp,16
299         .cprestore 0
300         sw      $fp,12($sp)
301         sw      $gp,8($sp)
302         move    $fp,$sp
303         sw      $a0,16($fp)
304         lw      $v0,16($fp)
305         lb      $v0,4($v0)
306         move    $sp,$fp
307         lw      $fp,12($sp)
308         addu    $sp,$sp,16
309         j       $ra
310         .end    FileEofReached
311         .size   FileEofReached, .-FileEofReached
312         .align  2
313         .globl  FileWrite
314         .ent    FileWrite
315 FileWrite:
316         .frame  $fp,40,$ra              # vars= 0, regs= 3/0, args= 16, extra= 8
317         .mask   0xd0000000,-8
318         .fmask  0x00000000,0
319         .set    noreorder
320         .cpload $t9
321         .set    reorder
322         subu    $sp,$sp,40
323         .cprestore 16
324         sw      $ra,32($sp)
325         sw      $fp,28($sp)
326         sw      $gp,24($sp)
327         move    $fp,$sp
328         sw      $a0,40($fp)
329         sw      $a1,44($fp)
330         sw      $a2,48($fp)
331         lw      $v0,40($fp)
332         lw      $a0,44($fp)
333         li      $a1,1                   # 0x1
334         lw      $a2,48($fp)
335         lw      $a3,0($v0)
336         la      $t9,fwrite
337         jal     $ra,$t9
338         move    $sp,$fp
339         lw      $ra,32($sp)
340         lw      $fp,28($sp)
341         addu    $sp,$sp,40
342         j       $ra
343         .end    FileWrite
344         .size   FileWrite, .-FileWrite
345         .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

A.0.9. Assembly main.s

```
        .file   1 "main.c"
        .section .mdebug.abi32
        .previous
        .abicalls
        .rdata
        .align  2
$LC0:
        .ascii  "input\000"
        .align  2
$LC1:
        .ascii  "output\000"
        .align  2
$LC2:
        .ascii  "action\000"
        .align  2
$LC3:
        .ascii  "help\000"
        .align  2
$LC4:
        .ascii  "version\000"
        .data
        .align  2
$LC5:
        .word   $LC0
        .word   1
        .word   0
        .word   105
        .word   $LC1
        .word   1
        .word   0
        .word   111
        .word   $LC2
        .word   1
        .word   0
        .word   97
        .word   $LC3
        .word   0
        .word   0
        .word   104
        .word   $LC4
        .word   0
        .word   0
        .word   86
        .globl  memcpy
        .rdata
        .align  2
$LC6:
        .ascii  "i:o:a:hV\000"
        .text
        .align  2
        .globl  main
        .ent    main
main:
        .frame  $fp,200,$ra             # vars= 152, regs= 3/0, args= 24, extra= 8
        .mask   0xd0000000,-8
        .fmask  0x00000000,0
```

```
        .set    noreorder
        .cpload $t9
        .set    reorder
        subu    $sp,$sp,200
        .cprestore 24
        sw      $ra,192($sp)
        sw      $fp,188($sp)
        sw      $gp,184($sp)
        move    $fp,$sp
        sw      $a0,200($fp)
        sw      $a1,204($fp)
        addu    $v0,$fp,32
        la      $v1,$LC5
        move    $a0,$v0
        move    $a1,$v1
        li      $a2,80                  # 0x50
        la      $t9,memcpy
        jal     $ra,$t9
        lw      $v0,$LC6
        sw      $v0,112($fp)
        lw      $v0,$LC6+4
        sw      $v0,116($fp)
        lbu     $v0,$LC6+8
        sb      $v0,120($fp)
        sw      $zero,132($fp)
        sb      $zero,136($fp)
        addu    $v0,$fp,144
        move    $a0,$v0
        la      $t9,CommandCreate
        jal     $ra,$t9
        lw      $v1,200($fp)
        li      $v0,1                   # 0x1
        bne     $v1,$v0,$L18
        addu    $v0,$fp,144
        move    $a0,$v0
        la      $t9,CommandSetError
        jal     $ra,$t9
$L18:
        .set    noreorder
        nop
        .set    reorder
$L19:
        addu    $v1,$fp,112
        addu    $v0,$fp,132
        sw      $v0,16($sp)
        lw      $a0,200($fp)
        lw      $a1,204($fp)
        move    $a2,$v1
        addu    $a3,$fp,32
        la      $t9,getopt_long
        jal     $ra,$t9
        sw      $v0,128($fp)
        lw      $v1,128($fp)
        li      $v0,-1                  # 0xffffffffffffffff
        beq     $v1,$v0,$L20
        lb      $v0,136($fp)
        bne     $v0,$zero,$L20
        lw      $v0,128($fp)
```

```
115         addu    $v0,$v0,-86
116         sw      $v0,180($fp)
117         lw      $v1,180($fp)
118         sltu    $v0,$v1,26
119         beq     $v0,$zero,$L29
120         lw      $v0,180($fp)
121         sll     $v1,$v0,2
122         la      $v0,$L30
123         addu    $v0,$v1,$v0
124         lw      $v0,0($v0)
125         .cpadd  $v0
126         j       $v0
127         .rdata
128         .align  2
129 $L30:
130         .gpword $L27
131         .gpword $L29
132         .gpword $L29
133         .gpword $L29
134         .gpword $L29
135         .gpword $L29
136         .gpword $L29
137         .gpword $L29
138         .gpword $L29
139         .gpword $L29
140         .gpword $L29
141         .gpword $L28
142         .gpword $L29
143         .gpword $L29
144         .gpword $L29
145         .gpword $L29
146         .gpword $L29
147         .gpword $L29
148         .gpword $L26
149         .gpword $L24
150         .gpword $L29
151         .gpword $L29
152         .gpword $L29
153         .gpword $L29
154         .gpword $L29
155         .gpword $L25
156         .text
157 $L24:
158         addu    $v0,$fp,144
159         move    $a0,$v0
160         lw      $a1,optarg
161         la      $t9,CommandSetInput
162         jal     $ra,$t9
163         b       $L19
164 $L25:
165         addu    $v0,$fp,144
166         move    $a0,$v0
167         lw      $a1,optarg
168         la      $t9,CommandSetOutput
169         jal     $ra,$t9
170         b       $L19
171 $L26:
172         la      $t9,CommandHelp
```

```
173         jal     $ra,$t9
174         li      $v0,1                   # 0x1
175         sb      $v0,136($fp)
176         b       $L19
177 $L27:
178         la      $t9,CommandVersion
179         jal     $ra,$t9
180         li      $v0,1                   # 0x1
181         sb      $v0,136($fp)
182         b       $L19
183 $L28:
184         addu    $v0,$fp,144
185         move    $a0,$v0
186         lw      $a1,optarg
187         la      $t9,CommandSetEncodeOpt
188         jal     $ra,$t9
189         b       $L19
190 $L29:
191         addu    $v0,$fp,144
192         move    $a0,$v0
193         la      $t9,CommandSetError
194         jal     $ra,$t9
195         b       $L19
196 $L20:
197         lb      $v0,136($fp)
198         beq     $v0,$zero,$L31
199         sw      $zero,176($fp)
200         b       $L17
201 $L31:
202         addu    $v0,$fp,144
203         move    $a0,$v0
204         la      $t9,CommandHasError
205         jal     $ra,$t9
206         bne     $v0,$zero,$L32
207         addu    $v0,$fp,144
208         move    $a0,$v0
209         la      $t9,CommandProcess
210         jal     $ra,$t9
211         b       $L33
212 $L32:
213         la      $t9,CommandErrArg
214         jal     $ra,$t9
215         li      $v0,1                   # 0x1
216         sw      $v0,176($fp)
217         b       $L17
218 $L33:
219         sw      $zero,176($fp)
220 $L17:
221         lw      $v0,176($fp)
222         move    $sp,$fp
223         lw      $ra,192($sp)
224         lw      $fp,188($sp)
225         addu    $sp,$sp,200
226         j       $ra
227         .end    main
228         .size   main, .-main
229         .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

B. Stack frame

B.1. Stack frame base_64decode

int base64_decode(int infd, int outfd)		
Offset	**Contents**	**Type reserved area**
68	outfd	**ABA (caller)**
64	infd	
60	//////////////////////////	**SRA**
56	ra	
52	fp	
48	gp	
39	//////////////////////////	**LTA**
38	OUT_BUFFER	
37		
36		
32	IN_BUFFER	
28		
27		
26		
12		**ABA (callee)**
8		
4		
0		

Figura 1: Stack frame base64decode

B.2. Stack frame base_64encode

int base64_encode(int infd, int outfd)		
Offset	**Contents**	**Type reserved area**
68	outfd	**ABA (caller)**
64	infd	
60	//////////////////////////	**SRA**
56	ra	
52	fp	
48	gp	
39	//////////////////////////	**LTA**
38	IN_BUFFER	
37		
36		
32	OUT_BUFFER	
28		
27		
26		
12		**ABA (callee)**
8		
4		
0		

Figura 2: Stack frame base64encode

B.3. Stack frame decodeChar

unsigned char DecodeChar(char character)		
Offset	**Contents**	**Type reserved area**
36	character	**ABA (caller)**
32	fp	**SRA**
28	gp	
24	return	**LTA**
20	i	
16	character	
12	a3	**ABA(callee)**
8	a2	
4	a1	
0	a0	

Figura 3: Stack frame decodeChar

B.4. Stack frame decode

unsigned char Decode(unsigned char *buf_input, unsigned char *buf_output)		
Offset	**Contents**	**Type reserved area**
68	*buffer_output	**ABA (caller)**
64	*buffer_input	
60	ra	**SRA**
56	fp	
52	gp	
48	s0	
39	////////////////////////	**LTA**
38	////////////////////////	
37	char1_aux	
36	char0_aux	
32	i	
28	////////////////////////	
27	chars3	
26	chars2	
25	chars1	
24	chars0	
20	return	
16	////////////////////////	
12	a3	**ABA (callee)**
8	a2	
4	a1	
0	a0	

Figura 4: Stack frame decode

B.5. Stack frame encode

void Encode(const unsigned char* buffer, unsigned int length, unsigned char* output)		
Offset	**Contents**	**Type reserved area**
24	*output	**ABA (caller)**
20	length	
16	*buffer	
12	fp	**SRA**
8	gp	
7	OFFSET_B4_AUX	**LTA**
6	OFFSET_B3_AUX_2	
5	OFFSET_B3_AUX	
4	OFFSET_B2_AUX	
3	OFFSET_B1_AUX	
2	OFFSET_B3	
1	OFFSET_B2	
0	OFFSET_B1	

Figura 5: Stack frame encode