

# Contenido

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>Documentación del diseño e implementación</b>	<b>2</b>
2.1	Desarrollo de actividades . . . . .	2
2.2	Menú (main.c) . . . . .	2
2.3	Codificación . . . . .	3
2.4	Decodificación . . . . .	5
2.5	Compilación . . . . .	7
<b>3</b>	<b>Pruebas</b>	<b>7</b>
3.1	Código de pruebas.sh . . . . .	7
3.2	Salida de pruebas.sh . . . . .	8
<b>4</b>	<b>El código fuente, en lenguaje C</b>	<b>10</b>
4.1	main.c . . . . .	10
4.2	base64.h . . . . .	11
<b>5</b>	<b>Funciones de ENCODE/DECODE en código MIPS32</b>	<b>11</b>
<b>6</b>	<b>Enunciado del trabajo práctico</b>	<b>24</b>
<b>7</b>	<b>Conclusión</b>	<b>28</b>

# 1 Introducción

El presente informe corresponde al trabajo práctico grupal **TP1** de la materia organización de computadoras.

## 2 Documentación del diseño e implementación

En el archivo `main.c` se usa a `getopt` para analizar los argumentos dados en la línea de comando y en `base64.S` se encuentra el código de codificación y decodificación en lenguaje MIPS32.

### 2.1 Desarrollo de actividades

1. Se instala en un linux el emulador `gxemul` con `NetBSD` para realizar las pruebas.
2. Para el caso de la funcion de codificación fue necesario realizar un refactor para dividir mejor las tareas y facilitar la posterior traducción a MIPS32 assembly.
3. Se propuso antes de iniciar el desarrollo de cada función en MIPS32 assembly, estimar el tamaño de los stacks frames de las funciones que utilizan el `base64_encode` y `base64_decode`, respetando la convención de la ABI.
4. Una vez que se programo ambas funciones en MIPS32 assembly, ambas funciones fueron incluidas en el main y probadas en el ambiente de NetBSD instalado.
5. Para el debugging y corrección de errores se requirio utilizar el comando GDB

```
$ gdb --args tp1 -a encode -i /root/envio/test.txt
```

6. Para crear el presente informe se debe utilizar el comando `pdflatex` en el directorio informe.

```
$ pdflatex informe_final.tex
```

### 2.2 Menú (`main.c`)

En `main.c` se puede ver la implementación de un menú de ayuda, que posee una opción que ejecuta el programa en modo información "`-h`" que detalla las variantes de ejecución del programa, estas son `-v` que imprime la versión del programa y sale del mismo y otra es `[options]` que muestra las opciones disponibles; estas son `-V`, `-h`, `-i` que está disponible para indicar en donde se encuentra la dirección del archivo de entrada, `-o` lo mismo pero para el archivo de salida, `-a` que indica la acción que queremos que ejecute el programa, por default será `encode` y si quisiéramos que decodifique solo hace falta agregarle un `decode`.

Si se especifica la opción de codificar o decodificar entonces se podrá también especificar el archivo de entrada y de salida, se le podrá pasar un archivo específico llamando a la función o también podrá pasárselo desde consola (`Stdin` y `Stdout`).

## 2.3 Codificación

La codificación se hace posible con la agrupación de casos que conforman la solución, para esto se dividen los bytes en bits y se agrupan dependiendo de la llegada de los mismos; a cada caso le corresponde una respectiva máscara, un respectivo delta de desplazamiento, y los bits faltantes para completar con el signo "=". Los casos están ordenados, entonces con un bucle se irá codificando byte a byte. Esta codificación, agarra el caso y sus parámetros, busca en la tabla de base 64 y traduce los bits en el carácter correspondiente. Finalmente, de ser necesario, completa la salida con caracteres de relleno.

Para la traducción a MIPS32 assembly fue necesario dividirlo en las siguientes funciones

1. `base64_encode`: Función principal para la codificación.
2. `calc_indice`: Calcula el índice para tabla de base 64.
3. `escrib_faltantes`: Escribe los caracteres faltantes y los signos para completar.
4. `escribir_byte`: Escribe un byte en el file descriptor indicado.

A continuación se muestra el stack frame de estas funciones:

		ENCODE_FDOUT / a1	60
	ABA caller	ENCODE_FDIN / a0	56
STACK base64_encode	SRA	////////////////////////////////	52
		ra	48
		fp	44
		gp	40
	LTA	ENCODE_BYTE_LEIDO	36
		ENCODE_CODRETORNO	32
		ENCODE_INDICE	28
		ENCODE_ACTUAL	24
		ENCODE_PREVIO	20
		ENCODE_CASO	16
	ABA	a3	12
		a2	8
		a1	4
		a0	0
	ABA caller	CAL_IDX_CASO / a2	16
		CAL_IDX_ACTUAL / a1	12
		CAL_IDX_PREVIO / a0	8
STACK calc_indice	SRA	fp	4
		gp	0
	ABA caller	ENCODE_FDOUT / a2	48
		FALTANTES_CASO / a1	44
		FALTANTES_PREVIO / a0	40
STACK escrib_faltantes	SRA	////////////////////////////////	36
		ra	32
		fp	28
		gp	24
	LTA	FALTANTES_FALTAN	20
		FALTANTES_K	16
	ABA	a3	12
		a2	8
		a1	4
		a0	0
	ABA caller	ESCRIBIR_FDOUT / a1	36
		ESCRIBIR_VALOR / a0	32
STACK escribir_byte	SRA	////////////////////////////////	28
		ra	24
		fp	20
		gp	16
	ABA	a3	12
		a2	8
		a1	4
		a0	0

## 2.4 Decodificación

Se incluye una tabla de decodificación generada con la función `crear_tabla_de_decodificacion` (incluida en el código), con ésta se podrán traducir los caracteres a decodificar a su índice dentro de la tabla de codificación, para de este modo poder concatenarlos y obtener la salida. Se tendrá en cuenta los caracteres inválidos, como por ejemplo '.', '!', etc o un '=' que no se encuentre en el lugar de caracter de relleno. La decodificación se hará de a bloques de 4 bytes, aunque en principio se leen de a 5 bytes para estar seguros de que en los primeros 4 no habrá caracteres de relleno. El programa finaliza decodificando el final de la entrada, verificando que el número de caracteres de relleno sea el correcto.

Para la traducción a MIPS32 assembly fue necesario dividirlo en las siguientes funciones

1. `base64_decode`: Función principal para la decodificación.
2. `deco_leer`: Garantiza leer la cantidad de caracteres requeridos.
3. `deco_escribir_char`: Escribe un caracter a la salida.
4. `deco_resolver`: Busca el caracter correspondiente en la tabla.
5. `deco_finales`: Decodifica el final de la entrada y verifica el número de caracteres de relleno.

A continuacion se muestra el stack frame de estas funciones:

		DECODE_FDOUT / a1	60
	ABA caller	DECODE_FDIN / a0	56
STACK base64_decode	SRA	////////////////////////////////	52
		ra	48
		fp	44
		gp	40
	LTA	////////////////////////////////	36
			32
		DECODE_BUFFER	28
		DECODE_FALTANTES	24
		DECODE_FINALES	20
		DECODE_LEIDOS	16
	ABA	a3	12
		a2	8
		a1	4
		a0	0
		DECODE_LEER_A2(COUNT)	48
		DECODE_LEER_A1(BUFFER)	44
	ABA caller	DECODE_LEER_A0(FDIN)	40
STACK deco_leer	SRA	////////////////////////////////	36
		ra	32
		fp	28
		gp	24
	LTA	////////////////////////////////	20
		DECODE_LEER_LEIDOS	16
	ABA	a3	12
		a2	8
		a1	4
		a0	0
		DECODE_ESCRIBIR_A1(FDOUT)	36
	ABA caller	DECODE_ESCRIBIR_A0(BUFFER)	32
STACK deco_escribir_char	SRA	////////////////////////////////	28
		ra	24
		fp	20
		gp	16
	ABA	a3	12
		a2	8
		a1	4
		a0	0
		DECODE_RESOLVER_A1 (LARGO)	28
	ABA caller	DECODE_RESOLVER_A0(BUFFER)	24
STACK deco_resolver	SRA	fp	20
		gp	16
	LTA	////////	12
		DECODE_RESOLVER_I	8
		DECODE_RESOLVER_TABLA	4
		DECODE_RESOLVER_C	0
		DECODE_FINALES_A2(FDOUT)	36
		DECODE_FINALES_A1(FINALES)	36
	ABA caller	DECODE_FINALES_A0(BUFFER)	32
STACK deco_finales	SRA	////////////////////////////////	28
		ra	24
		fp	20
		gp	16
	ABA	a3	12
		a2	8
		a1	4
		a0	0

## 2.5 Compilación

Junto al código fuente se encuentra el Makefile de proyecto. La compilación debe realizarse dentro de la VM de NetBSD y como resultado final generara un binario con el nombre tp1.

```
$ make
```

## 3 Pruebas

### 3.1 Código de pruebas.sh

```
#!/bin/sh

#Codificamos un archivo vacio (cantidad de bytes nula)
touch /tmp/zero.txt #creamos un archivo de texto vacio
./tp0 -a encode -i /tmp/zero.txt -o /tmp/zero.txt.b64
ls -l /tmp/zero.txt.b64
#-rw-r--r-- 1 user group 0 2018-09-08 16:21 /tmp/zero.txt.b64


#codificamos caracter ASCII M
echo -n M | ./tp0
#TQ==
echo

#codificamos caracter ASCII M y a
echo -n Ma | ./tp0
#TWE=
echo

echo "Test Codifico y decodifico una imagen. Prueba de
    binarios"
./tp0 -a encode -i recursos/linux-icon.png | ./tp0 -a decode -
    o recursos/linux-icon.png.b64 &&
    diff -s recursos/linux-icon.png recursos/linux-icon.png.
    b64

#codificamos Man
echo -n "Man" | ./tp0
#TWFu
echo

#codificamos y decodificamos
echo Man | ./tp0 | ./tp0 -a decode
#Man
echo

#verificamos bit a bit
echo xyz | ./tp0 | ./tp0 -a decode | od -t c
#0000000 x y z \n
#00000004
```

```

echo
yes | head -c 1024 | ./tp0 -a encode #codificamos 1024 bytes,
    comprobamos longitud
#
    eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
#...
#
    eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5Cg
    ==
echo

#verificamos que los bytes sean 1024
yes | head -c 1024 | ./tp0 -a encode | ./tp0 -a decode | wc -c
#1024
echo

#Generamos archivos de largo creciente, y verificamos que el
    procesamiento
de nuestro programa no altere los datos

n=1;
while ;; do

head -c $n </dev/urandom >/tmp/in.bin;

./tp0 -a encode -i /tmp/in.bin -o /tmp/out.b64;

./tp0 -a decode -i /tmp/out.b64 -o /tmp/out.bin;

if diff /tmp/in.bin /tmp/out.bin; then ;; else

echo ERROR: \$n;

break;

fi

echo ok: \$n;

n=\$((\$n+1));

rm -f /tmp/in.bin /tmp/out.b64 /tmp/out.bin
done

```

### 3.2 Salida de pruebas.sh

```

-rw-r--r--  1 root  wheel  0 Oct 30 03:51 /tmp/zero.txt.b64
TQ==
TWE=
Test Codifico y decodifico una imagen. Prueba de binarios

```



Files recursos/linux-icon.png and recursos/linux-icon.png.b64  
are identical  
TWFu  
Man

0000000 x y z \n  
0000004

eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK  
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5Cg  
==  
1024  
ok: 1

ok: 2  
ok: 3  
ok: 4  
ok: 5  
[...]

## 4 El código fuente, en lenguaje C

### 4.1 main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include "codec.h"
#include "base64.h"

#define ERROR_CODIFICANDO 2
#define ERROR_DECODIFICANDO 3

static char* ENCODE = "encode";
static char* DECODE = "decode";

typedef struct {
    char* accion;
    char* entrada;
    char* salida;
} Parametro;

void imprimirAyuda()
{
    printf("Usage:\n");
    printf("\t\t -h\n");
    printf("\t\t -V\n");
    printf("\t\t [ options ]\n");

    printf("Options:\n");
    printf("\t-V, --version          Print version and quit.\n");
    printf("\t-h, --help             Print this information.\n");
    printf("\t-i, --input             Location of the input file.\n");
    printf("\t-o, --output            Location of the output file.\n");
    printf("\t-a, --action           Program action: encode (default) or decode.\n");

    printf("Examples:\n");
    printf("\t\t -a encode -i ~/input -o ~/output\n");
    printf("\t\t -a encode\n");
}

Parametro manejarArgumentosEntrada(int argc, char** argv)
{
    int siguiente_opcion;
    int option_index;

    /* Una cadena que lista las opciones cortas validas */
    const char* const op_cortas = "hva:i:o:"; /* "hva::i:o:" */

    /* Una estructura de varios arrays describiendo los valores largos */
    const struct option op_largas[] = {
        { "help",          no_argument,          0, 'h' },
        { "version",       no_argument,          0, 'V' },
        { "action",        required_argument,    0, 'a' }, /* optional_argument */
        { "input",         required_argument,    0, 'i' },
        { "output",        required_argument,    0, 'o' },
        { 0, 0, 0, 0 }
    };

    Parametro parametro;
    parametro.accion = ENCODE;
    parametro.entrada = "";
    parametro.salida = "";
    while (1) {
        siguiente_opcion = getopt_long (argc, argv, op_cortas, op_largas, &option_index);
        if (siguiente_opcion == -1) {
            break;
        }

        switch (siguiente_opcion) {
            case 'h':
                imprimirAyuda();
                exit(0);
                break;
            case 'V':
                printf("Tp0:Version_0.1:Grupo: B rbara Mesones Miret, Nestor Huallpa, Sebasti n D'Alessandro Szymanowski\n");
                exit(0);
                break;
            case 'a':
                if ( optarg )
                    parametro.accion = optarg;
                break;
        }
    }
}
```

```

        case 'i' :
            if ( optarg )
                parametro.entrada = optarg;
            break;
        case 'o' :
            if ( optarg )
                parametro.salida = optarg;
            break;
        case '?':
            /* getopt_long already printed an error message. */
            imprimirAyuda();
            exit(0);
            break;
    }
}
return parametro;
}

int main (int argc, char** argv) {
    int returnCode = 0;
    Parametro p = manejarArgumentosEntrada(argc, argv);
    int isEntradaArchivo = strcmp(p.entrada, "");
    int isSalidaArchivo = strcmp(p.salida, "");
    int isEntradaEstandar = strcmp(p.entrada, "-");
    int isSalidaEstandar = strcmp(p.salida, "-");
    if (!isEntradaEstandar) {
        isEntradaArchivo = 0;
    }
    if (!isSalidaEstandar) {
        isSalidaArchivo = 0;
    }
    //Si la entrada esta vacia lee stdin (teclado)
    FILE* archivoEntrada = (isEntradaArchivo!=0)?fopen(p.entrada, "rb"):stdin;
    //Si la salida esta vacia escribe stdout (pantalla)
    FILE* archivoSalida = (isSalidaArchivo!=0) ? fopen(p.salida, "w") : stdout;

    if (!archivoEntrada) {
        fprintf(stderr, "ERROR: EL ARCHIVO DE ENTRADA NO SE ENCUENTRA\n");
        exit(1);
    } else if (!archivoSalida) {
        fprintf(stderr, "ERROR: EL ARCHIVO DE SALIDA NO SE ENCUENTRA\n");
        exit(1);
    }
    if (strcmp(p.accion, ENCODE) == 0) {
        int fileDescriptorEntrada = fileno(archivoEntrada);
        int fileDescriptorSalida = fileno(archivoSalida);
        returnCode = base64_encode(fileDescriptorEntrada, fileDescriptorSalida);
        if (returnCode!=0) fprintf(stderr, "Error: %s \n", errmsg[returnCode]);
    } else if (strcmp(p.accion, DECODE) == 0) {
        int infd = fileno(archivoEntrada);
        int outfd = fileno(archivoSalida);
        returnCode = base64_decode(infd, outfd);
        if (returnCode!=0) fprintf(stderr, "Error: %s \n", errmsg[returnCode]);
    } else {
        fprintf(stderr, "ERROR: SE DEBE INGRESAR UN ARGUMENTO CORRECTO PARA LA OPCION\n");
    }

    if (isEntradaArchivo!=0) {
        fclose(archivoEntrada);
    }
    if (isSalidaArchivo!=0) {
        fclose(archivoSalida);
    }
    if (returnCode!=0){
        exit(1);
    }
    return returnCode;
}

```

## 4.2 base64.h

```

#ifndef _BASE64_H
#define _BASE64_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

extern const char* errmsg[];
extern int base64_encode(int fileDescriptorEntrada, int fileDescriptorSalida);
extern int base64_decode(int fileDescriptorEntrada, int fileDescriptorSalida);

#endif

```

## 5 Funciones de ENCODE/DECODE en codigo MIPS32

```

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

```

```

#define ERROR_Lectura_DECODIF 1

```

```

#define ERROR_ESCRITURA_DECODIF 2
#define ERROR_LECTURA_CODIF 3
#define ERROR_ESCRITURA_CODIF 4
#define ERROR_CARACTER_DECODIF 5
#define ERROR_ARGUMENTO_FINALES 6
#define ERROR_FALTANTES_A 7
#define ERROR_FALTANTES_B 8
#define ERROR_FALTANTES_C 9

# STACKFRAME: ABA(16) + LA(24) + SRA(16)

#define ENCODE_FDOUT 60
#define ENCODE_FDIN 56
#define ENCODE_STACK_SIZE 56
#define ENCODE_VAR_RA 48
#define ENCODE_VAR_FP 44
#define ENCODE_VAR_GP 40
#define ENCODE_BYTE_LEIDO 36
#define ENCODE_CODRETORNO 32
#define ENCODE_INDICE 28
#define ENCODE_ACTUAL 24
#define ENCODE_PREVIO 20
#define ENCODE_CASO 16

.text
.abicalls
.align 2
.globl base64_encode
.ent base64_encode
base64_encode:
    .frame $fp, ENCODE_STACK_SIZE, ra
    .set noreorder
    .cload t9
    .set reorder
    subu $sp, $sp, ENCODE_STACK_SIZE
    .cprestore ENCODE_VAR_GP
    sw ra, ENCODE_VAR_RA($sp) #save ra
    sw $fp, ENCODE_VAR_FP($sp) #save $fp
    move $fp, $sp
    sw zero, ENCODE_ACTUAL($fp)
    sw zero, ENCODE_PREVIO($fp)
    sw zero, ENCODE_CASO($fp)
    sw zero, ENCODE_CODRETORNO($fp)
    sw zero, ENCODE_BYTE_LEIDO($fp)
    sw a0, ENCODE_FDIN($fp)
    sw a1, ENCODE_FDOUT($fp)

leer_siguiente:
    lw a0, ENCODE_FDIN($fp) # fd para el read
    la a1, ENCODE_BYTE_LEIDO($fp) # dir buffer lectura
    li a2, 1 # read size en 1
    li v0, SYS_read # leeo un char
    syscall
    bnez a3, error_lectura_encode # si hay error retorno
    beqz v0, ultimo_paso # es EOF, vamos al ultimo paso

# calculamos indice
    lb t0, ENCODE_BYTE_LEIDO($fp) # cargo el valor byte leido
    sb t0, ENCODE_ACTUAL($fp) # guardo el byte leido en ACTUAL

    lb a0, ENCODE_PREVIO($fp) # cargo byte previo
    lb a1, ENCODE_ACTUAL($fp) # cargo byte actual
    lw a2, ENCODE_CASO($fp) # cargo caso
    jal calc_indice # calculamos indice de tabla
    sw v0, ENCODE_INDICE($fp) # guardo indice calculado

    lw t0, ENCODE_INDICE($fp) # busco indice para ir a tabla
    la t1, tabla # t1 con direccion de tabla
    addu t1, t1, t0 # posiciono en tabla[indice]
    lb t2, 0(t1) # tengo el valor de tabla[indice]

# escribimos
    move a0, t2 # cargamos valor a escribir
    lw a1, ENCODE_FDOUT($fp) # cargamos file descriptor out
    jal escribir_byte
    bnez v0, error_escritura_encode # si no es cero, hay error me voy

    lw t0, ENCODE_CASO($fp) # Actualizo el numero de caso
    addui t0, t0, 1 # caso++
    sw t0, ENCODE_CASO($fp) # guardo caso

    lw t0, ENCODE_CASO($fp) # leo caso para el caso 3
    li t1, 3
    bne t0, t1, preparar_leer # Evaluo si no estoy en CASO:3

    lb a0, ENCODE_PREVIO($fp) # EN CASO 3: calculamos indice
    lb a1, ENCODE_ACTUAL($fp)
    lw a2, ENCODE_CASO($fp)
    jal calc_indice
    sw v0, ENCODE_INDICE($fp) # guardo indice calculado

    lw t0, ENCODE_INDICE($fp) # EN CASO 3: busco indice para ir a tabla
    la t1, tabla # t1 con direccion de tabla
    addu t1, t1, t0 # posiciono en tabla[indice]
    lb t2, 0(t1) # tengo el valor de tabla[indice]

```

```

        move        a0, t2                # EN CASO 3: Escribimos
        lw          a1, ENCODE_FDOUT($fp)
        jal         escribir_byte
        bnez        v0, error_escritura_encode

        sw          zero, ENCODE_CASO($fp) # EN CASO 3: reiniciamos el contador de caso

preparar_leer:
        lb          t0, ENCODE_ACTUAL($fp) # leo el byte actual
        sb          t0, ENCODE_PREVIO($fp) # y guardo el actual en previo
        j           leer_siguiente

error_escritura_encode:
        li          t0, ERROR_ESCRITURA_CODIF
        sw          t0, ENCODE_CODRETORNO($fp)
        j           retornar_encode
error_lectura_encode:
        li          t0, ERROR_LECTURA_CODIF
        sw          t0, ENCODE_CODRETORNO($fp)
        j           retornar_encode

ultimo_paso:
        lb          a0, ENCODE_PREVIO($fp)
        lw          a1, ENCODE_CASO($fp)
        lw          a2, ENCODE_FDOUT($fp)
        jal         escrib_faltantes     ## llamar a escrib_faltantes
        bnez        v0, error_escritura_encode

retornar_encode:
        lw          v0, ENCODE_CODRETORNO($fp)
        move        sp, $fp
        lw          gp, ENCODE_VAR_GP(sp)
        lw          $fp, ENCODE_VAR_FP(sp)
        lw          ra, ENCODE_VAR_RA(sp)
        addu        sp, sp, ENCODE_STACK_SIZE
        jr          ra
        .end        base64_encode

# funcion calc_indice es una funcion leaf,
# o sea que no requiere seccion ABA,
# No requiere guardar RA

# STACKFRAME: SRA(8)

#define CAL_IDX_CASO          16
#define CAL_IDX_ACTUAL        12
#define CAL_IDX_PREVIO        8
#define CAL_IDX_FRAME_SIZE    8
#define CAL_IDX_VAR_SP        4
#define CAL_IDX_VAR_GP        0

        .text
        .align 2
        .globl calc_indice
        .ent calc_indice
calc_indice:
        .frame      $fp, CAL_IDX_FRAME_SIZE, ra
        .set        noreorder
        .cplload    t9
        .set        reorder
        subu        sp, sp, CAL_IDX_FRAME_SIZE
        .cprestore   CAL_IDX_VAR_GP
        sw          $fp, 4(sp)
        move        $fp, sp
        sw          a0, CAL_IDX_PREVIO($fp) # byteleido_previo
        sw          a1, CAL_IDX_ACTUAL($fp) # byteleido_actual
        sw          a2, CAL_IDX_CASO($fp)   # caso
        li          t0, 3
        bne         a2, t0, caso012        # if caso != 3 then caso012
        la          t1, mascaraactual
        lb          t2, 3(t1)                # leo mascaraactual[3]
        move        t3, a1
        and         v0, t3, t2
        j           retornar_indice         # jump to retornar_indice
caso012:
        la          t1, mascaraactual
        addu        t1, t1, a2
        lb          t2, 0(t1)                # t2 := mascaraactual[caso]
        la          t1, mascaraprevio
        addu        t1, t1, a2
        lb          t3, 0(t1)                # t3 := mascaraprevio[caso]
        la          t1, deltaactual
        addu        t1, t1, a2
        lb          t4, 0(t1)                # t4 := deltaactual[caso]
        la          t1, deltaprevio
        addu        t1, t1, a2

        lb          t5, 0(t1)                # t5 := deltaprevio[caso]
        and         t6, a0, t3
        sll         t6, t6, t5                # t6 := (byteleido_previo & mascaraprevio[caso]) << deltaprevio[caso]

        lb         t8, CAL_IDX_ACTUAL($fp)
        and         t7, t8, t2
        srl         t7, t7, t4                # t7 := (byteleido_actual & mascaraactual[caso]) >> deltaactual[caso]
        or          v0, t6, t7

retornar_indice:
        move        sp, $fp

```

```

        lw          gp, 0(sp)
        lw          $fp, 4(sp)
        addu        sp, sp, CAL_IDX_FRAME_SIZE
        jr          ra                                # jump to $ra
    .end          calc_indice

# escribir caracteres faltantes
# es una funcion no-leaf

#STACKFRAME: ABA(16) + LA(8) + SRA(16)

#define          FALTANTES_FDOUT          48
#define          FALTANTES_CASO           44
#define          FALTANTES_PREVIO         40
#define          FALTANTES_FRAME_SIZE     40
#define          FALTANTES_RA             32
#define          FALTANTES_FP             28
#define          FALTANTES_GP             24
#define          FALTANTES_FALTAN         20
#define          FALTANTES_K              16

    .text
    .align        2
    .abicalls
    .globl        escrib_faltantes
    .ent          escrib_faltantes
escrib_faltantes:
    .frame        $fp, FALTANTES_FRAME_SIZE, ra
    .set          noreorder
    .cload        t9
    .set          reorder
    subu          sp, sp, FALTANTES_FRAME_SIZE
    .cprestore    FALTANTES_GP
    sw            $fp, FALTANTES_FP(sp)
    sw            ra, FALTANTES_RA(sp)
    move          $fp, sp
    sw            a0, FALTANTES_PREVIO($fp)    # a0: byteleido_previo
    sw            a1, FALTANTES_CASO($fp)      # a1: caso
    sw            a2, FALTANTES_FDOUT($fp)     # a2: file descriptor output
    sw            zero, FALTANTES_K($fp)
    la            t1, faltantes
    lw            t2, FALTANTES_CASO($fp)
    sll           t2, t2, 2
    addu          t1, t1, t2
    lw            t1, 0(t1)                    # t1 : faltann = faltantes[caso]
    sw            t1, FALTANTES_FALTAN($fp)

    lw            t1, FALTANTES_FALTAN($fp)
    li            t2, 1
    bne           t1, t2, faltan2             # Verifico si falta 1 sino salto a caso 2
    andi          t3, a0, 0x0f                # CASO faltan = 1
    sll           t3, t3, 2                    # t3: indice = (byteleido_previo & 0x0f) << 2
    la            t4, tabla
    addu          t4, t4, t3                    # *tabla + indice apunto al byte que esta en la posicion indice

    lb            a0, 0(t4)                    # a0 := tabla[indice]
    lw            a1, FALTANTES_FDOUT($fp)
    jal           escribir_byte                # jump to target and save position to $ra
    bnez          v0, error_escritura_faltantes
    j             escribir_simbolo_cierre     #

faltan2:
    li            t2, 2
    lw            t1, FALTANTES_FALTAN($fp)
    bne           t1, t2, retornar_faltantes
    andi          t3, a0, 0x03                # CASO faltan = 2
    sll           t3, t3, 4                    # t3: indice = (byteleido_previo & 0x0f) << 2
    la            t4, tabla
    addu          t4, t4, t3                    # *tabla + indice apunto al byte que esta en la posicion indice
    lb            a0, 0(t4)                    # a0 := tabla[indice]
    lw            a1, FALTANTES_FDOUT($fp)
    jal           escribir_byte                # jump to target and save position to $ra
    bnez          v0, error_escritura_faltantes

escribir_simbolo_cierre:
    lw            t0, FALTANTES_K($fp)
    lw            t1, FALTANTES_FALTAN($fp)
    bge           t0, t1, retornar_faltantes;
    la            t0, CHAR_IGUAL
    lb            a0, 0(t0)
    lw            a1, FALTANTES_FDOUT($fp)
    jal           escribir_byte                # jump to target and save position to $ra
    bnez          v0, error_escritura_faltantes
    lw            t0, FALTANTES_K($fp)
    addiu         t0, t0, 1
    sw            t0, FALTANTES_K($fp)
    j             escribir_simbolo_cierre     # Escribo otro simbolo

error_escritura_faltantes:
    li            v0, 4
    j             retornar_faltantes
retornar_faltantes:
    move          sp, $fp
    lw            gp, FALTANTES_GP(sp)
    lw            $fp, FALTANTES_FP(sp)
    lw            ra, FALTANTES_RA(sp)
    addu          sp, sp, FALTANTES_FRAME_SIZE
    jr            ra

```

```

        .end                escrib_faltantes

#STACKFRAME: ABA(16) + SRA(16)

#define          ESCRIBIR_FDOUT          36
#define          ESCRIBIR_VALOR          32
#define          ESCRIBIR_FRAME_SIZE    32
#define          ESCRIBIR_RA            24
#define          ESCRIBIR_FP            20
#define          ESCRIBIR_GP            16

        .text
        .abicalls
        .align      2
        .globl      escribir_byte
        .ent        escribir_byte
escribir_byte:
        .frame       $fp, ESCRIBIR_FRAME_SIZE, ra
        .set         noreorder
        .cpload      t9
        .set         reorder
        subu          sp, sp, ESCRIBIR_FRAME_SIZE
        .cprestore    ESCRIBIR_GP
        sw            $fp, ESCRIBIR_FP(sp)
        sw            ra, ESCRIBIR_RA(sp)
        move          $fp, sp
        sw            a0, ESCRIBIR_VALOR($fp)
        sw            a1, ESCRIBIR_FDOUT($fp)
        lw            a0, ESCRIBIR_FDOUT($fp)           # cargamos fd de escritura
        la            a1, ESCRIBIR_VALOR($fp)           # cargamos dir de valor a escribir
        li            a2, 1                             # cargamos longitud
        li            v0, SYS_write
        syscall
        bnez          a3, error_escritura
        li            v0, 0
        j             retornar_escribir                 # jump to retornar_escribir
error_escritura:
        li            v0, 1
retornar_escribir:
        move          sp, $fp
        lw            gp, ESCRIBIR_GP(sp)
        lw            $fp, ESCRIBIR_FP(sp)
        lw            ra, ESCRIBIR_RA(sp)
        addu          sp, sp, ESCRIBIR_FRAME_SIZE
        jr            ra
        .end        escribir_byte

        .rdata
        .align      2
mascaraactual:      .byte  0xfc, 0xf0, 0xc0, 0x3f
mascaraprevio:      .byte  0x00, 0x03, 0x0f, 0x00
deltaprevio:         .byte  0, 4, 2, 0
deltaactual:         .byte  2, 4, 6, 0
faltantes:           .word  0, 2, 1
        .align      2
CHAR_IGUAL:         .ascii  "="
        .globl      errmsg
        .align      2
errmsg: .word msg_error_ninguno, \
        msg_error_lectura_decodif, \
        msg_error_escritura_decodif, \
        msg_error_lectura_codif, \
        msg_error_escritura_codif, \
        msg_error_caracter_decodif, \
        msg_error_argumento_finales, \
        msg_error_faltantes_a, \
        msg_error_faltantes_b, \
        msg_error_faltantes_c
        .align      2
msg_error_ninguno: .ascii  "No hubo errores."
msg_error_lectura_decodif: .ascii  "Hubo un error en la lectura del archivo a decodificar."
msg_error_escritura_decodif: .ascii  "Hubo un error en la escritura del archivo para la decodificacion."
msg_error_lectura_codif: .ascii  "Hubo un error en la lectura del archivo para codificar."
msg_error_escritura_codif: .ascii  "Hubo un error en la escritura del archivo cuando codificamos."
msg_error_caracter_decodif: .ascii  "Caracter ilegal en la entrada del decodificador."
msg_error_argumento_finales: .ascii  "Argumento invalido pasado a deco_finales."
msg_error_faltantes_a: .ascii  "Falta 1 caracter de relleno."
msg_error_faltantes_b: .ascii  "Faltan 2 caracteres de relleno."
msg_error_faltantes_c: .ascii  "Faltan 2 caracteres de relleno y 1 de informacion."

        .align      2
tabla: .ascii  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

#define  STDERR          2

#define  DECODE_FDOUT    60
#define  DECODE_FDIN     56
#define  DECODE_STACK_SIZE 56
#define  DECODE_VAR_RA   48
#define  DECODE_VAR_FP   44
#define  DECODE_VAR_GP   40
#define  DECODE_BUFFER   28
#define  DECODE_FALTANTES 24

```

```

#define DECODE_FINALES          20
#define DECODE_LEIDOS          16

        .text
        .abicalls
        .align 2
        .globl base64_decode
        .ent base64_decode
base64_decode:
        .frame $fp, DECODE_STACK_SIZE, ra
        .set noreorder
        .cload t9
        .set reorder
        subu $p, $p, DECODE_STACK_SIZE
        .cprestore DECODE_VAR_GP
        sw ra, DECODE_VAR_RA($p)
        sw $fp, DECODE_VAR_FP($p)
        move $fp, $p
        sw a0, DECODE_FDIN($fp)
        sw a1, DECODE_FDOUT($fp)
        sw zero, DECODE_FALTANTES($fp)

        la a1, DECODE_BUFFER($fp)
        li a2, 5
        jal deco_leer
        bnez a3, base64_decode_error_lectura

        sw v0, DECODE_LEIDOS($fp)
        bne v0, 5, leer_finales

loop:
        # resolver (b, 4);
        la a0, DECODE_BUFFER($fp)
        li a1, 4
        jal deco_resolver
        bnez v0, base64_decode_salir

        lb t0, DECODE_BUFFER($fp)
        lb t1, DECODE_BUFFER+1($fp)
        # escribir (((b[0] & 0x3f) << 2) | ((b[1] & 0x30) >> 4), salida);
        andi t5, t0, 0x3f
        sll t5, t5, 2
        andi t6, t1, 0x30
        srl t6, t6, 4
        or t7, t5, t6
        move a0, t7
        lw a1, DECODE_FDOUT($fp)
        jal deco_escribir_char
        bnez v0, base64_decode_salir

        lb t1, DECODE_BUFFER+1($fp)
        lb t2, DECODE_BUFFER+2($fp)
        # escribir (((b[1] & 0x0f) << 4) | ((b[2] & 0x3c) >> 2), salida);
        andi t5, t1, 0x0f
        sll t5, t5, 4
        andi t6, t2, 0x3c
        srl t6, t6, 2
        or t7, t5, t6
        move a0, t7
        lw a1, DECODE_FDOUT($fp)
        jal deco_escribir_char
        bnez v0, base64_decode_salir

        lb t2, DECODE_BUFFER+2($fp)
        lb t3, DECODE_BUFFER+3($fp)
        # escribir (((b[2] & 0x03) << 6) | (b[3] & 0x3f), salida);
        andi t5, t2, 0x03
        sll t5, t5, 6
        andi t6, t3, 0x3f
        or t7, t5, t6
        move a0, t7
        lw a1, DECODE_FDOUT($fp)
        jal deco_escribir_char
        bnez v0, base64_decode_salir

        # b[0] = b[4];
        lb t0, DECODE_BUFFER+4($fp)
        sb t0, DECODE_BUFFER($fp)

        # while ((leidos = fread (b+1, 1, 4, entrada)) == 4);
        lw a0, DECODE_FDIN($fp)
        la a1, DECODE_BUFFER+1($fp)
        li a2, 4
        jal deco_leer
        bnez a3, base64_decode_error_lectura
        sw v0, DECODE_LEIDOS($fp)
        beq v0, 4, loop

        # leidos += 1;
        addi v0, v0, 1
        sw v0, DECODE_LEIDOS($fp)

leer_finales:
        beqz v0, base64_decode_salir_ok

        # si (leidos == 4)
        bne v0, 4, no_es_4

```



```

        # finales = b[3] == '=' ? (b[2] == '=' ? 1 : 2) : 3;

        lb          t2, DECODE_BUFFER+2($fp)
        lb          t3, DECODE_BUFFER+3($fp)
        bne         t3, '=', else_e1
        bne         t2, '=', else_i1
        li          t4, 1
        b           fin_1
    else_i1:
        li          t4, 2
        b           fin_1
    else_e1:
        li          t4, 3
    fin_1:
        sw          t4, DECODE_FINALES($fp)
        b           finales

no_es_4:
        lw          t0, DECODE_LEIDOS($fp)
        bne         t0, 3, no_es_3
        # finales = b[2] == '=' ? 1 : 2;
        lb          t2, DECODE_BUFFER+2($fp)
        bne         t2, '=', else_2
        li          t4, 1
        b           fin_2
    else_2:
        li          t4, 2
    fin_2:
        sw          t4, DECODE_FINALES($fp)
        li          t0, ERROR_FALTANTES_A
        sw          t0, DECODE_FALTANTES($fp)
        b           finales

no_es_3:
        lw          t0, DECODE_LEIDOS($fp)
        bne         t0, 2, no_es_2
        li          t4, 1
        sw          t4, DECODE_FINALES($fp)
        li          t0, ERROR_FALTANTES_B
        sw          t0, DECODE_FALTANTES($fp)
        b           finales

no_es_2:
        li          v0, ERROR_FALTANTES_C
        b           base64_decode_salir

finales:
        la          a0, DECODE_BUFFER($fp)
        lw          a1, DECODE_FINALES($fp)
        lw          a2, DECODE_FDOUT($fp)
        jal         deco_finales
        bnez        v0, base64_decode_salir
        # veo si faltaban caracteres
        lw          t0, DECODE_FALTANTES($fp)
        beqz         t0, base64_decode_salir_ok
        move        v0, t0
        b           base64_decode_salir

base64_decode_error_lectura:
        li          v0, ERROR_Lectura_DECODIF
        b           base64_decode_salir

base64_decode_salir_ok:
        li          v0, 0

base64_decode_salir:
        move        sp, $fp
        lw          gp, DECODE_VAR_GP(sp)
        lw          $fp, DECODE_VAR_FP(sp)
        lw          ra, DECODE_VAR_RA(sp)
        addu        sp, sp, DECODE_STACK_SIZE
        jr          ra
        .end        base64_decode

#define DECODE_LEER_A2      48
#define DECODE_LEER_A1      44
#define DECODE_LEER_A0      40
#define DECODE_LEER_STACK_SIZE 40
#define DECODE_LEER_RA      32
#define DECODE_LEER_FP      28
#define DECODE_LEER_GP      24
#define DECODE_LEER_LEIDOS  16

        .globl     deco_leer
        .ent       deco_leer
    deco_leer:
        .frame     $fp, DECODE_LEER_STACK_SIZE, ra
        .set       noreorder
        .cpld      t9
        .set       reorder
        subu        sp, sp, DECODE_LEER_STACK_SIZE
        .cprestore  DECODE_LEER_GP
        sw          $fp, DECODE_LEER_FP(sp)
        sw          ra, DECODE_LEER_RA(sp)
        move        $fp, sp

```

```

        sw          a0, DECODE_LEER_A0($fp)
        sw          a1, DECODE_LEER_A1($fp)
        sw          a2, DECODE_LEER_A2($fp)

        li          t1, 0
        move        t2, a2
        sw          t1, DECODE_LEER_LEIDOS($fp)
loop_deco_leer:
        lw          a0, DECODE_LEER_A0($fp)
        lw          a1, DECODE_LEER_A1($fp)
        addu        a1, a1, t1
        move        a2, t2
        li          v0, SYS_read
        syscall

        bnez        a3, salir_deco_leer_error    # si fallo SYS_read

        lw          t1, DECODE_LEER_LEIDOS($fp)
        beqz        v0, salir_deco_leer          # si termino EOF
        addu        t1, t1, v0                    # leidos hasta ahora
        sw          t1, DECODE_LEER_LEIDOS($fp)
        lw          t2, DECODE_LEER_A2($fp)
        beq         t1, t2, salir_deco_leer      # si concluyo lectura
        subu        t2, t2, t1                    # restan leer
        b           loop_deco_leer

salir_deco_leer:
        move        v0, t1

salir_deco_leer_error:
        move        sp, $fp
        lw          gp, DECODE_LEER_GP(sp)
        lw          $fp, DECODE_LEER_FP(sp)
        lw          ra, DECODE_LEER_RA(sp)
        addu        sp, sp, DECODE_LEER_STACK_SIZE
        jr          ra
        .end        deco_leer

#define DECODE_ESCRIBIR_A1      36
#define DECODE_ESCRIBIR_A0      32
#define DECODE_ESCRIBIR_STACK_SIZE 32
#define DECODE_ESCRIBIR_RA      24
#define DECODE_ESCRIBIR_FP      20
#define DECODE_ESCRIBIR_GP      16

        .globl     deco_escribir_char
        .ent       deco_escribir_char
deco_escribir_char:
        .frame     $fp, DECODE_ESCRIBIR_STACK_SIZE, ra
        .set       noreorder
        .cpld      t9
        .set       reorder
        subu        sp, sp, DECODE_ESCRIBIR_STACK_SIZE
        .cprestore  DECODE_ESCRIBIR_GP
        sw          $fp, DECODE_ESCRIBIR_FP(sp)
        sw          ra, DECODE_ESCRIBIR_RA(sp)
        move        $fp, sp
        sw          a0, DECODE_ESCRIBIR_A0($fp)
        sw          a1, DECODE_ESCRIBIR_A1($fp)

        move        t0, a0
        sw          t0, 0($fp)

        lw          a0, DECODE_ESCRIBIR_A1($fp)
        la          a1, 0($fp)
        li          a2, 1
        li          v0, SYS_write
        syscall
        beqz        a3, deco_escribir_char_salir_ok

deco_escribir_char_error:
        li          v0, ERROR_ESCRITURA_DECODIF
        b           deco_escribir_char_salir

deco_escribir_char_salir_ok:
        li          v0, 0

deco_escribir_char_salir:
        move        sp, $fp
        lw          gp, DECODE_ESCRIBIR_GP(sp)
        lw          $fp, DECODE_ESCRIBIR_FP(sp)
        lw          ra, DECODE_ESCRIBIR_RA(sp)
        addu        sp, sp, DECODE_ESCRIBIR_STACK_SIZE
        jr          ra
        .end        deco_escribir_char

#define DECODE_RESOLVER_A1      28
#define DECODE_RESOLVER_A0      24
#define DECODE_RESOLVER_STACK_SIZE 24
#define DECODE_RESOLVER_FP      20
#define DECODE_RESOLVER_GP      16
#define DECODE_RESOLVER_I       8
#define DECODE_RESOLVER_TABLA    4
#define DECODE_RESOLVER_C       0

```

```

        .globl deco_resolver
        .ent    deco_resolver
deco_resolver:
        .frame $fp, DECODE_RESOLVER_STACK_SIZE, ra
        .set noreorder
        .cload t9
        .set reorder
        subu    $p, $p, DECODE_RESOLVER_STACK_SIZE
        .cprestore
        sw      $fp, DECODE_RESOLVER_FP($p)
        move    $fp, $p
        sw      a0, DECODE_RESOLVER_A0($fp)
        sw      a1, DECODE_RESOLVER_A1($fp)
        la      t0, deco_tabla
        sw      t0, DECODE_RESOLVER_TABLA($fp)

        move    t5, t0 # t5 = tabla
        li      t0, -1 # t0 = i
        move    t1, a1 # t1 = largo
        addu    t1, t1, -1 # t1 = largo - 1
        move    t2, a0 # t2 = b
        b       while_resolver

loop_resolver:
        addu    t0, t0, 1
        sw      t0, DECODE_RESOLVER_I($fp)

        addu    t3, t2, t0 # t3 = b + i
        lb      t4, 0(t3) # t4 = b[i]
        sb      t4, DECODE_RESOLVER_C($fp)

        addu    t6, t5, t4 # t6 = tabla + b[i]
        lb      t7, 0(t6) # tabla[c]
        sb      t7, 0(t3)

        beq     t7, -1, deco_resolver_error

while_resolver:
        blt     t0, t1, loop_resolver
        b       deco_resolver_salir_ok

deco_resolver_error:
        li      v0, ERROR_CHARACTER_DECODIF
        b       deco_resolver_salir

deco_resolver_salir_ok:
        li      v0, 0

deco_resolver_salir:
        move    $p, $fp
        lw      gp, DECODE_RESOLVER_GP($p)
        lw      $fp, DECODE_RESOLVER_FP($p)
        addu    $p, $p, DECODE_RESOLVER_STACK_SIZE
        jr      ra
        .end    deco_resolver

#define DECODE_FINALES_A2      40
#define DECODE_FINALES_A1      36
#define DECODE_FINALES_A0      32
#define DECODE_FINALES_STACK_SIZE 32
#define DECODE_FINALES_RA      24
#define DECODE_FINALES_FP      20
#define DECODE_FINALES_GP      16

        .globl deco_finales
        .ent    deco_finales
deco_finales:
        .frame $fp, DECODE_FINALES_STACK_SIZE, ra
        .set noreorder
        .cload t9
        .set reorder
        subu    $p, $p, DECODE_FINALES_STACK_SIZE
        .cprestore
        sw      $fp, DECODE_FINALES_FP($p)
        sw      ra, DECODE_FINALES_RA($p)
        move    $fp, $p
        sw      a0, DECODE_FINALES_A0($fp)
        sw      a1, DECODE_FINALES_A1($fp)
        sw      a2, DECODE_FINALES_A2($fp)

        bne     a1, 1, finales_no_es_1

# resolver (b, 2);
        lw      a0, DECODE_FINALES_A0($fp)
        li      a1, 2
        jal     deco_resolver
        bnez    v0, deco_finales_salir

        lw      t8, DECODE_FINALES_A0($fp)
        lb      t0, 0(t8)
        lb      t1, 1(t8)
        # escribir (((b[0] & 0x3f) << 2) | ((b[1] & 0x30) >> 4), salida);
        andi    t5, t0, 0x3f
        sll     t5, t5, 2
        andi    t6, t1, 0x30

```

```

        srl            t6, t6, 4
        or             t7, t5, t6
        move          a0, t7
        lw            a1, DECODE_FINALES_A2($fp)
        jal           deco_escribir_char
        bnez          v0, deco_finales_salir

        b             deco_finales_salir_ok

finales_no_es_1:
        lw            a1, DECODE_FINALES_A1($fp)
        bne           a1, 2, finales_no_es_2

        # resolver (b, 3);
        lw            a0, DECODE_FINALES_A0($fp)
        li            a1, 3
        jal           deco_resolver
        bnez          v0, deco_finales_salir

        lw            t8, DECODE_FINALES_A0($fp)
        lb            t0, 0(t8)
        lb            t1, 1(t8)

        # escribir (((b[0] & 0x3f) << 2) | ((b[1] & 0x30) >> 4), salida);
        andi          t5, t0, 0x3f
        sll           t5, t5, 2
        andi          t6, t1, 0x30
        srl           t6, t6, 4
        or            t7, t5, t6
        move          a0, t7
        lw            a1, DECODE_FINALES_A2($fp)
        jal           deco_escribir_char
        bnez          v0, deco_finales_salir

        lw            t8, DECODE_FINALES_A0($fp)
        lb            t1, 1(t8)
        lb            t2, 2(t8)

        # escribir (((b[1] & 0x0f) << 4) | ((b[2] & 0x3c) >> 2), salida);
        andi          t5, t1, 0x0f
        sll           t5, t5, 4
        andi          t6, t2, 0x3c
        srl           t6, t6, 2
        or            t7, t5, t6
        move          a0, t7
        lw            a1, DECODE_FINALES_A2($fp)
        jal           deco_escribir_char
        bnez          v0, deco_finales_salir

        b             deco_finales_salir_ok

finales_no_es_2:
        lw            a1, DECODE_FINALES_A1($fp)
        bne           a1, 3, deco_finales_error

        # resolver (b, 4);
        lw            a0, DECODE_FINALES_A0($fp)
        li            a1, 4
        jal           deco_resolver
        bnez          v0, deco_finales_salir

        lw            t8, DECODE_FINALES_A0($fp)
        lb            t0, 0(t8)
        lb            t1, 1(t8)

        # escribir (((b[0] & 0x3f) << 2) | ((b[1] & 0x30) >> 4), salida);
        andi          t5, t0, 0x3f
        sll           t5, t5, 2
        andi          t6, t1, 0x30
        srl           t6, t6, 4
        or            t7, t5, t6
        move          a0, t7
        lw            a1, DECODE_FINALES_A2($fp)
        jal           deco_escribir_char
        bnez          v0, deco_finales_salir

        lw            t8, DECODE_FINALES_A0($fp)
        lb            t1, 1(t8)
        lb            t2, 2(t8)

        # escribir (((b[1] & 0x0f) << 4) | ((b[2] & 0x3c) >> 2), salida);
        andi          t5, t1, 0x0f
        sll           t5, t5, 4
        andi          t6, t2, 0x3c
        srl           t6, t6, 2
        or            t7, t5, t6
        move          a0, t7
        lw            a1, DECODE_FINALES_A2($fp)
        jal           deco_escribir_char
        bnez          v0, deco_finales_salir

        lw            t8, DECODE_FINALES_A0($fp)
        lb            t2, 2(t8)
        lb            t3, 3(t8)

        # escribir (((b[2] & 0x03) << 6) | (b[3] & 0x3f), salida);
        andi          t5, t2, 0x03

```



[illegible]

[illegible]

## **6    Enunciado del trabajo practico**



# 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 descrito 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, corresponden 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 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.

## 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 completo, 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/>).

## 7 Conclusión

Tuvimos problemas con los casos de prueba que involucraron pipes. Tuvimos que entender el funcionamiento de la `SYS_read` con pipes para poder solucionarlo. La solución fue leer hasta completar un buffer del tamaño esperado.

Se respetó las convenciones de la ABI para el desarrollo de las funciones en MIPS32, entendiendo que función es “leaf” y “non leaf” a la hora de definir el Stack Frame de cada una.

En cuanto al trabajo grupal en sí mismo, no hubo inconvenientes de ningún tipo ya que al ser el grupo relativamente chico y tener conocimiento del manejo del versionado de un proyecto ante cambios ingresado por los integrantes (por medio del GIT), la introducción de modificaciones y correcciones fui fluida.