



UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

2DO CUATRIMESTRE DE 2020

[86.37 / 66.20] ORGANIZACIÓN DE COMPUTADORAS

CURSO 2

Trabajo práctico 0

Infraestructura básica

Padrón	Alumno	Email
103442	Lovera, Daniel	dlovera@fi.uba.ar
102914	More, Agustín	amore@fi.uba.ar
99846	Torresetti, Lisandro	ltorresetti@fi.uba.ar

Índice

1. Objetivos	2
2. Introducción	2
3. Detalles de implementación	2
3.1. Base64	2
3.1.1. Caso borde	3
3.2. Implementación función de codificación	4
3.3. Implementación función de decodificación	5
4. Compilación y ejecución	6
4.1. Compilación	6
4.2. Ejecución	6
4.2.1. Descripción de parámetros	6
4.2.2. Ejemplos de ejecución	6
5. Pruebas	7
5.1. Prueba 1	7
5.2. Prueba 2	7
5.3. Prueba 3	8
5.4. Prueba 4	8
5.5. Prueba 5	8
5.6. Prueba 6	9
5.7. Prueba 7	9
5.8. Prueba 8	10
5.9. Pruebas automatizadas	10
5.10. Pruebas de volumen	11
5.10.1. Caso 1: 1Kb	11
5.10.2. Caso 2: 1Mb	12
5.10.3. Caso 3: 1Gb	12
5.11. Pruebas con archivos binarios	12
6. Código	13
6.1. Código en C	13
6.2. Código assembly en MIPS32	13
7. Conclusiones	13
A. Código	14
A.1. Código en C	14
A.2. Código MIPS32	17
B. Enunciado del trabajo práctico	42
C. Código auxiliar	45
C.1. Pruebas automatizadas	45

1. Objetivos

Familiarización de las herramientas de software utilizadas por la cátedra:

- GCC: Compilador del código fuente.
- L^AT_EX: Herramienta para la generación de documentos.
- QEMU: Emulador de distintos procesadores sin necesidad de particiones de disco, en particular se trabaja con la arquitectura MIPS.

2. Introducción

En el presente informe se plantea la resolución de un programa en C que permite la codificación Base64, el programa es controlado por una serie de comandos que permiten seleccionar entre codificación o decodificación así como el origen y el destino de estos datos.

Base64 se creó con el fin de poder transmitir archivos binarios en medios que solo admitían texto, debe su nombre a que 64 es la mayor potencia de 2 que puede ser representada usando únicamente caracteres ASCII imprimibles, los primeros 62 dígitos de la tabla de símbolos corresponden a las letras del alfabeto latino (mayúsculas y minúsculas) y los últimos 2 suelen variar dependiendo del tipo de implementación que se realice, en este trabajo se consideraron los símbolos + y / como últimos dígitos.

3. Detalles de implementación

3.1. Base64

Es un sistema de numeración posicional cuya base es 64 [1]. Se utilizará para codificar una secuencia de caracteres a un subconjunto de caracteres `ascii`. El proceso de codificación consiste en tomar el texto que se quiere codificar, formar una secuencia de bytes con el equivalente de cada carácter con su valor en binario del código `ascii` [2]. Por ejemplo en el cuadro 1:

Texto	C	o	s
Equivalente ascii	67	111	115
Binario	01000011	01101111	01110011

Cuadro 1: Texto a binario mediante la codificación ascii

Una vez obtenida la secuencia de bits (cuya longitud siempre será un múltiplo de ocho), se tomaran cada seis bits (tomándolo de izquierda a derecha), obteniendo el cuadro 2:

Texto	C	o	s
Equivalente ascii	67	111	115
Binario	0 1 0 0 0 0 0 1 1	0 1 1 0 1 1 1 1	0 1 1 1 0 0 1 1
Índice	16	54	61

Cuadro 2: Reagrupación de bits y conversión a índices

Como se puede ver en el cuadro 2, cada tres caracteres en el texto original, se generan cuatro secuencias de seis bits completos, esto permite codificar desde secuencias fijas de tres bytes.

Dado que se toman de a seis bits, el mayor índice será 63 ($2^6 - 1 = 63$, obteniendo el rango [0, 63] índices posibles). Una vez obtenidos estos índices se mapean a un carácter `ascii` del cuadro 3.

Valor	Carácter	Valor	Carácter	Valor	Carácter	Valor	Carácter
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Cuadro 3: Tabla de equivalencias entre caracteres con su equivalente decimal codificado

Texto	C						o						s					
Equivalente ascii	67						111						115					
Binario	0	1	0	0	0	0	1	1	0	1	1	0	1	1	1	1	0	1
Índice	16						54						61					
Codificado	Q						2						9					

Cuadro 4: Mapeo de índices a caracteres

Pasando así del texto ‘Cos’ a ‘Q29z’.

3.1.1. Caso borde

Cuando la secuencia a codificar no es múltiplo de tres, no se logran reagrupar los bytes en conjuntos de seis bits, con lo cual, en este caso, se completan con ceros los bits restantes y se procede a codificar de la forma explicada previamente, salvo que los conjuntos de seis bits que queden en cero por la limpieza anterior serán codificadas por el carácter ‘=’. En cada conjunto de cuatro letras resultantes se pueden obtener 0, 1, 2 caracteres ‘=’ dependiendo de la longitud original del texto a codificar, siguiendo la fórmula $|txt_{codificado}|_{‘=’} = 3 - |txt_{codificar}|$ donde $|.$ es la longitud del texto. Ejemplo:

Texto	C						-						-					
Equivalente ascii	67						-						-					
Binario	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
Índice	16						48						-					
Codificado	Q						w						=					

Cuadro 5: Caso borde de codificación

En este caso, partiendo de ‘C’ se obtiene ‘Qw==’. Se puede ver que se cumple con la fórmula, dado que la longitud del texto original $|txt_{codificar}| = 1$ se cumple que la cantidad de iguales en el texto codificado es $|txt_{codificado}|_{‘=’} = 3 - 1 = 2$.

Conclusión, la longitud del resultado siempre será múltiplo de cuatro.

Como detalle de implementación, aprovechando las invariantes de la multiplicidad de las longitudes de los textos, se codifican de a bloques de tres caracteres, obteniendo un bloque constante de cuatro caracteres codificados. Además, se arranca de una cadena con cuatro caracteres '=' que serán sobrescritos a medida que se codifica, limitando la ejecución con la fórmula previamente mencionada.

3.2. Implementación función de codificación

En primer lugar se leen 3 bytes del archivo pasado por parámetro al programa y se los almacena en un *buffer*, luego este es pasado a la función *combineBytes* para obtener un código binario de los bytes leídos. A continuación se muestra el código de dicha función:

```

1 | int combineBytes(char* block, size_t readBytes, bool code){
2 |     int resultado = (0x00000000);
3 |     int SHIFTS = 24;
4 |     int mult = (code) ? 8 : 6;
5 |     for (int i = 0; i < readBytes; i++){
6 |         int shiftLeft = SHIFTS - (i + 1) * mult;
7 |         int toShift = (code) ? (int)block[i] : findPosition(block[i]);
8 |         if (toShift < 0){
9 |             fprintf(stderr, "Error: El caracter %c no se encuentra en base
           ↳ 64\n", block[i]);
10 |             return -1;
11 |         }
12 |         resultado |= toShift << shiftLeft;
13 |     }
14 |     return resultado;
15 | }
```

El parámetro *code* es un booleano que sirve para saber si tenemos que codificar o decodificar el buffer. Dado que en ambos casos se debe realizar un shift a izquierda primero de cierta cantidad de bits para alinearlos de manera correcta y que sea trivial la separación en la codificación. Se encontró una relación para hacerlo, la misma es la siguiente:

$$24 - (i + 1) * mult$$

En caso de estar codificando *mult* debe valer 8 y en caso contrario 6. Luego realizamos un *loop* en el cual la cantidad de iteraciones está dada por la cantidad de bytes leídos anteriormente. En cada iteración la cantidad de bits a desplazar hacia la izquierda se va adaptando hasta que no sea necesario un nuevo desplazamiento, así obtenemos el código binario de manera correcta.

La variable *toShift* contiene un entero que representa el código ASCII o el código en Base64, en caso de estar decodificando y que el carácter no se encuentre en la tabla de símbolos de Base64, *findPosition* devolverá -1 lo cual hará que la sentencia del *if* sea verdadera y se imprimirá el mensaje por *stderr*.

En caso de que no haya ningún error, la ejecución continua y se realiza el shift correspondiente.

Una vez que la función devuelve el código binario de los bytes leídos del archivo se pasa el resultado a la función *codification*:

```

1 | char* codification(int binaryCode, size_t readBytes){
2 |     int shiftsLeft[] = {8, 14, 20, 26};
3 |     int shiftRight = 26;
```

```

4 | char* code = malloc(5);
5 | code[0] = '=';
6 | code[1] = '=';
7 | code[2] = '=';
8 | code[3] = '=';
9 | code[4] = '\\0';
10 | size_t posCode = 0;
11 | for (size_t i = 0; i < readBytes + 1; i++){
12 |     int binaryCodeAux = binaryCode; //Para no modificar el codigo
    ↪ binario original
13 |     binaryCodeAux = binaryCodeAux << shiftsLeft[i];
14 |     binaryCodeAux = (int)((unsigned) binaryCodeAux >> shiftRight);
15 |     code[posCode++] = BASE64[binaryCodeAux];
16 | }
17 | return code;
18 | }

```

Primero se utiliza memoria dinámica para crear un arreglo de caracteres que contendrá el resultado de la codificación y se inicializa cada posición con el carácter '=' para solucionar los casos bordes en los cuales la cantidad de bytes leídos es menor a 3.

Posteriormente se realiza una cantidad de iteraciones igual al número de bytes leídos más 1 (se suma 1 ya que en caso de leer 3 bytes debemos tener un total de 4 caracteres como resultado de la codificación).

En cada iteración se realiza un shift de manera tal que los 6 bits necesarios para la codificación queden en la parte más significativa y luego se desplaza a derecha 26 bits en todos los casos para obtener un número representado de 6 bits, cuyo valor es el código a buscar en la tabla 3 para saber qué carácter representa en Base64 y se posteriormente se lo almacena en el arreglo.

3.3. Implementación función de decodificación

A diferencia de la codificación, en este caso en vez de leer 3 bytes del archivo pasado por parámetro se leen 4 (ya que representan los tres caracteres a decodificar) y se los almacena en un buffer. Luego, al igual que *codification*, se pasa el buffer a la función *combineBytes* y el resultado que devuelve es utilizado por la función *decodification* que se muestra a continuación:

```

1 | char* decodification(int binaryCode, size_t readBytes){
2 |     int shiftsLeft[] = {8, 16, 24};
3 |     int shiftRight = 24;
4 |     char* code = malloc(readBytes);
5 |     code[readBytes - 1] = '\\0';
6 |     size_t posCode = 0;
7 |     for (size_t i = 0; i < readBytes - 1; i++){
8 |         int binaryCodeAux = binaryCode;
9 |         binaryCodeAux = binaryCodeAux << shiftsLeft[i];
10 |        binaryCodeAux = (int)((unsigned) binaryCodeAux >> shiftRight);
11 |        code[posCode++] = (char) binaryCodeAux;
12 |    }
13 |    return code;
14 | }

```

En este caso las iteraciones a realizar son la cantidad de bytes leídos menos 1, ya que 4 bytes en base 64 equivalen a 3 bytes al realizar la decodificación.

Al igual que con la función *codification* realizamos los shifts correspondientes a izquierda de manera que nos quede en los bits más significativos el valor del carácter que hay que decodificar. Luego realizamos un shift a derecha de 24 bits, de esta forma nos queda un número representado en 8 bits del cual obtenemos su correspondiente valor y el carácter que representa en la tabla ASCII.

4. Compilación y ejecución

4.1. Compilación

Para compilar el programa implementado se debe ejecutar el siguiente comando que generará el archivo ejecutable *tp0*:

```
gcc -Wall -Werror -O0 tp0.c -o tp0
```

Para generar el código assembly en el sistema MIPS32:

```
gcc -Wall -Werror -O0 -S tp0.c
```

Los parámetros que recibe `gcc` [3] se utilizan:

- **-O0** desactiva las optimizaciones que realiza el compilador habitualmente.
- **-S** compila el código hasta el ensamblado.
- **-Wall** devolverá todos los *warnings* posibles por el compilador.
- **-Werror** tratará los *warnings* como errores.

4.2. Ejecución

Para ejecutar el archivo compilado:

```
./tp0 [Opciones]
```

4.2.1. Descripción de parámetros

- V --version: Muestra la versión y sale del programa.
- h --help: Muestra información de ayuda de cómo ejecutar el programa y sale del programa.
- o --output: Path para el archivo de salida (en caso de no indicar uno se utilizará el *stdout*).
- i --input: Path para el archivo de entrada (en caso de no indicar uno se utilizará el *stdin*).
- d --decode: Indica que la operación será de decodificación.

4.2.2. Ejemplos de ejecución

Codificar desde el archivo *ejemplo.txt* y mostrarlo en pantalla, se puede realizar de cualquiera de las siguientes maneras:

```
./tp0 -i ejemplo.txt
```

```
./tp0 < ejemplo.txt
```

```
cat ejemplo.txt | ./tp0
```

Codificar desde el archivo *ejemplo.txt* y escribir el resultado en *salida.txt*:

```
./tp0 -i ejemplo.txt -o salida.txt
```

Además de las variantes con pipe (‘|’) y redirección (‘<’, ‘>’).

En caso de querer decodificar el archivo previamente codificado:

```
./tp0 -i salida.txt -d
```

5. Pruebas

Se realizaron ocho sets de pruebas donde en cada una se prueban distintos casos. Para cada caso se procede a:

1. Mostrar el contenido del archivo origen.
2. Ejecutar el programa con el input archivo origen.
3. Mostrar el resultado esperado.
4. Comparar los resultados anteriores con el comando `diff`.

5.1. Prueba 1

Prueba sobre una cadena corta de tres caracteres (ancho de un bloque de codificación).

```
cat test/test1.txt
Man

./tp0 < test/test1.txt
TWFu

cat test/res_test1.txt
TWFu

./tp0 < test/test1.txt | diff test/res_test1.txt -
```

Como se puede observar, el comando `diff` no tiene una salida, indicando de que ambos coinciden, lo mismo ocurrirá en las siguientes pruebas.

5.2. Prueba 2

Prueba sobre una cadena corta de un carácter (caso borde).

```
cat test/test2.txt
M

./tp0 < test/test2.txt
TQ==

cat test/res_test2.txt
TQ==

./tp0 < test/test2.txt | diff test/res_test2.txt -
```


5.3. Prueba 3

Prueba sobre una cadena corta con dos ‘\n’ al final del archivo.

```
cat test/test3.txt
Man

./tp0 < test/test3.txt
TWFCgo=

cat test/res_test3.txt
TWFCgo=

./tp0 < test/test3.txt | diff test/res_test3.txt -
```

5.4. Prueba 4

Prueba sobre una cadena larga.

```
cat test/test4.txt
Man is distinguished , not only by his reason , but by this singular
    ↪ passion from other animals , which is a lust of the mind , that by
    ↪ a perseverance of delight in the continued and indefatigable
    ↪ generation of knowledge , exceeds the short vehemence of any
    ↪ carnal pleasure .

./tp0 < test/test4.txt
TWfUIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpYByZWfZb24sIGJ1dCBieS
B0aGlzIH Npbmd1bGFyIH Bhc3Npb24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBh
IGx1c3Qgb2YgdGhlIGlpbmQsIHRoYXQgYnkgYSBwZXJzZXZlcF uY2Ugb2YgZGVsaWdodC
Bpb iB0aGUgY29udGlu dWV kIGFuZCBpbmRlZmF0aWdhYm xIIGdlbmVyYXRpb24gb2Yga25v
d2xlZGdlLCBleGNlZWRzIHRoZSBzaG9ydCB2ZW hlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbG
Vhc3VyZS4=

cat test/res_test4.txt
TWfUIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpYByZWfZb24sIGJ1dCBieS
B0aGlzIH Npbmd1bGFyIH Bhc3Npb24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBh
IGx1c3Qgb2YgdGhlIGlpbmQsIHRoYXQgYnkgYSBwZXJzZXZlcF uY2Ugb2YgZGVsaWdodC
Bpb iB0aGUgY29udGlu dWV kIGFuZCBpbmRlZmF0aWdhYm xIIGdlbmVyYXRpb24gb2Yga25v
d2xlZGdlLCBleGNlZWRzIHRoZSBzaG9ydCB2ZW hlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbG
Vhc3VyZS4=

./tp0 < test/test4.txt | diff test/res_test4.txt -
```

5.5. Prueba 5

Prueba sobre una cadena mediana, con ‘\n’ al final (prueba de la cátedra).

```
cat test/test5.txt  
En un lugar de La Mancha de cuyo nombre no quiero acordarme  
  
./tp0 < test/test5.txt  
RW4gdW4gbHVnYXIgZGUgTGEgTWFuY2hhIGRIIGNleW8gbm9tYnJlIG5vIHF1aWVyb3BlY2  
9yZGFyZWUK  
  
cat test/res_test5.txt  
RW4gdW4gbHVnYXIgZGUgTGEgTWFuY2hhIGRIIGNleW8gbm9tYnJlIG5vIHF1aWVyb3BlY2  
9yZGFyZWUK  
  
./tp0 < test/test5.txt | diff test/res_test5.txt -
```

5.6. Prueba 6

Prueba sobre una cadena larga, con ‘\n’ entre el texto.

```
cat test/test6.txt
Man is distinguished , not only by his reason ,
but by this singular passion from other animals ,
which is a lust of the mind, that by a perseverance
of delight in the continued and indefatigable
generation of knowledge, exceeds the short
vehemence of any carnal pleasure.

./tp0 < test/test6.txt
TWfUIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWZzb24sIApidXQgYn
kgdGhpcyBzaW5ndWxhciBwYXNzaW9uIGZyb20gb3RoZXIgaW5pbWFscywgCndoZWNoIGlz
IGEgbHVzdCBvZiB0aGUgbWluZCwgGdGhhdCBieSBhIHBlcnNldmVyYW5jZSAKb2YgZGVsaW
dodCBpb20aGUgY29udGluZGVkIGFuZCBpbmRlZmF0aWdhYmxlIApnZW5lcmF0aW9uIG9m
IGtub3dsZWRnZSsgZXhjZWNkcyB0aGUgc2hvcnQgCnZlaGVtZW5jZSBvZiBhbnkgY2Fybml
FsIHBSZWZzdXJlLg==

cat test/res_test6.txt
TWfUIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWZzb24sIApidXQgYn
kgdGhpcyBzaW5ndWxhciBwYXNzaW9uIGZyb20gb3RoZXIgaW5pbWFscywgCndoZWNoIGlz
IGEgbHVzdCBvZiB0aGUgbWluZCwgGdGhhdCBieSBhIHBlcnNldmVyYW5jZSAKb2YgZGVsaW
dodCBpb20aGUgY29udGluZGVkIGFuZCBpbmRlZmF0aWdhYmxlIApnZW5lcmF0aW9uIG9m
IGtub3dsZWRnZSsgZXhjZWNkcyB0aGUgc2hvcnQgCnZlaGVtZW5jZSBvZiBhbnkgY2Fybml
FsIHBSZWZzdXJlLg==

./tp0 < test/test6.txt | diff test/res_test6.txt -
```

5.7. Prueba 7

Prueba sobre una cadena larga, con '\n' entre el texto y al final.

```
cat test/test7.txt
```

Man is distinguished, not only by his reason, but by this singular passion from other animals, which is a lust of the mind, that by a perseverance of delight in the continued and indefatigable generation of knowledge, exceeds the short vehemence of any carnal pleasure.

```
./tp0 < test/test7.txt  
TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIApidXQgYn  
kgdGhpcyBzaW5ndWxhciBwYXNzaW9uIGZyb20gb3RoZXIgaWY5pbWFSyYzZSAKb2YgZGVsaW  
IEEgHVzdCBvZiB0aGUgbWluZCwgZGhhdBieSBhIHBlcnNldmVyaWY5jZSAKb2YgZGVsaW  
dodCBpbIB0aGUgY29udGludWVkaGFuZCBpbmRlZmF0aWdhYmxiIApnZW5lcmlF0aW9uIG9m  
IGtub3dsZWRnZSwgZXBkY29udGludWVkaGFuZCBpbmRlZmF0aWdhYmxiIApnZW5lcmlF0aW9uIG9m  
FsIHBSZWZkdXJlgo=
```

```
cat test/res_test7.txt  
TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIApidXQgYn  
kgdGhpcyBzaW5ndWxhciBwYXNzaW9uIGZyb20gb3RoZXIgaWY5pbWFscywgCndoawNoIGlz  
IGEgbHVzdCBvZiB0aGUgbWluZCwgdGhhdBieSBhIHBlcnNldmVyYW5jZSAKb2YgZGVsaW  
dodCBpbIB0aGUgY29udGludWVkaGFuZCBpbmRlZmF0aWdhYmxiIApnZW5lcmlF0aW9uIG9m  
IGtub3dsZWRnZSwgZXBkYzVkaW9uZG9uc2hvbnQgc2lnZSByZiBhbnkgY2Fybmc  
FsIHBSZWZkdXJlgo=
```

```
./tp0 < test/test7.txt | diff test/res test7.txt -
```

5.8. Prueba 8

Prueba sobre una cadena corta, con ‘\n’ entre el texto.

```
cat test/test8.txt
M
a
n

./tp0 < test/test8.txt
TQphCm4=

cat test/res_test8.txt
TQphCm4=

./tp0 < test/test8.txt | diff test/res_test8.txt -
```

5.9. Pruebas automatizadas

Para correr las pruebas de forma automática se usó un *script* de hecho en *Python* (que se encuentra en el apéndice C.1).

Se asume la siguiente estructura de archivos:

tp /	
---	test /

```

| |--- test1.txt
| |--- res_test1.txt
| |--- test2.txt
| |--- res_test2.txt
. .
. .
. .
| |--- test8.txt
| |--- res_test8.txt
|--- test.py
|--- tp0.c
|--- tp0*

```

Para correr las pruebas previamente descriptas, se debe ejecutar:

```
python test.py ./tp0 test/test test/res_test
```

Donde el primer parámetro es el programa que se está ejecutando (en este caso el programa en el modo de codificación), el segundo es el prefijo de los archivos de entrada (en este caso los archivos estarán en la carpeta `test`, y todos comenzarán con `test`, por ejemplo, `test/test1.txt`) y el último parámetro es el prefijo de los archivos que contienen el valor esperado para cada prueba.

```
$ python test.py ./tp0 test/test test/res_test
.....
```

Cada punto indica una prueba exitosa, en caso de falla, se muestra la diferencia entre el valor esperado y el obtenido.

Ahora, a partir de esta estructura, se puede probar la funcionalidad de decodificar con el siguiente comando:

```
python test.py "./tp0 -d" test/res_test test/test
```

Donde ahora se está ejecutando el programa en modo **decode** (**-d**), los archivos de entrada serán los textos codificados y los de comparación serán los no codificados.

Ejecutándolo se obtiene:

```
$ python test.py "./tp0 -d" test/res_test test/test
.....
```

Indicando que pasaron todas las pruebas.

5.10. Pruebas de volumen

Para cada uno de los casos que se detallan a continuación se realizaron varias ejecuciones del programa y se tomó la moda.

5.10.1. Caso 1: 1Kb

Para este caso se utilizó:

```
$ time head -c $((1024)) /dev/zero | ./tp0 > /dev/null
```

El tiempo de ejecución en la máquina host fue de:

```
real 0m0,005s
user 0m0,003s
sys 0m0,005s
```

Mientras que en la máquina guest fue:

```
real 0m0,162s
user 0m0,120s
sys 0m0,036s
```

5.10.2. Caso 2: 1Mb

El comando para este caso fue:

```
$ time head -c $((1024 * 1024)) /dev/zero | ./tp0 > /dev/null
```

El tiempo de ejecución en la máquina host fue de:

```
real 0m0,108s
user 0m0,106s
sys 0m0,010s
```

Y en la máquina guest fue:

```
real 0m3,961s
user 0m3,808s
sys 0m0,144s
```

5.10.3. Caso 3: 1Gb

Al igual que en el caso anterior, debemos modificar una vez más el comando multiplicandolo por 1024, el cual nos queda:

```
$ time head -c $((1024 * 1024 * 1024)) /dev/zero | ./tp0 > /dev/null
```

El tiempo de ejecución en la máquina host fue de:

```
real 0m30,398s
user 0m30,301s
sys 0m1,801s
```

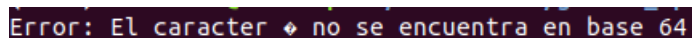
En este caso el tiempo de ejecución en la máquina guest no se pudo determinar, se cortó la ejecución del programa luego de 10 minutos aproximadamente. Se plantea la hipótesis de que al ser un entorno de simulación (QEMU), no se contaría con los mismos recursos que en la máquina host, reduciendo así la capacidad de procesamiento, haciendo que el programa no termine en un tiempo razonable.

5.11. Pruebas con archivos binarios

Para estas pruebas se utilizaron los siguientes comandos:

```
$ head -25 /dev/urandom | ./tp0
$ head -50 /dev/urandom | ./tp0
$ head -100 /dev/urandom | ./tp0
```

En todos los casos se obtuvo el mismo resultado:



```
Error: El caracter 4 no se encuentra en base 64
```

Figura 1: Resultado de las pruebas con archivos binarios

Esto ocurre porque la codificación en base64 trabaja con un conjunto de valores correspondientes a la tabla ASCII y cuando se ingresan caracteres que no pertenecen a esta tabla no puede codificarlos y falla.

6. Código

6.1. Código en C

Ver apéndice A.1

6.2. Código assembly en MIPS32

Ver apéndice A.2

7. Conclusiones

Se realizó un programa totalmente funcional que permite la codificación de texto a Base64 y recibe instrucciones por línea de comandos, se incluyeron pruebas que corroboran la validez del código y se logró obtener el código assembly específico de la arquitectura MIPS. Esto permitió el entendimiento de herramientas necesarias por la cátedra así como chequear su funcionamiento.

Se realizaron pruebas con archivos de texto, binarios y archivos de gran tamaño y las conclusiones son las siguientes:

- Todas las pruebas con archivos de textos resultaron exitosas, se validaron los casos bordes en donde el programa era posible que fallara en la codificación y regresando los mismos archivos a su forma de texto se comprobó que la decodificación también era correcta.
- No todas las pruebas con archivos binarios resultan exitosas, en ciertos casos se obtiene un carácter que no se encuentra en la tabla de índice de base64 lo cual vuelve imposible la codificación del archivo, razón por la cual el programa lanza el error que se muestra en la figura 1.
- Al realizar pruebas con grandes entradas de datos el programa funciona correctamente en la máquina host, tardando relativamente poco (menos de un minuto aproximadamente). Sin embargo, al ejecutar el mismo programa con el mismo archivo de entrada en la máquina guest (MIPS32) el mismo es muchísimo más lento, al punto que se tuvo que detener la ejecución del programa 10 minutos después de iniciar y no se pudo determinar el tiempo total de ejecución.
- Al realizar la misma prueba (de volumen) con un mega de bytes en lugar de un giga en la máquina host se obtiene un tiempo de ejecución mucho menor a la máquina guest, en esta ocasión no fue necesario detener la ejecución pues el programa dio el resultado esperado, sin embargo se puede notar como al cambiar entorno de ejecución el programa se ralentiza al punto de ser aproximadamente 27 veces más lento (2.744s -tiempo en máquina guest y 0,100s -tiempo en máquina host).

A. Código

A.1. Código en C

```
1 #define _GNU_SOURCE
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <stddef.h>
6 #include <ctype.h>
7 #include <stdbool.h>
8 #define _POSIX_C_SOURCE 200809L //para incluir getline
9 char* BASE64 =
    ↪ "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
10
11 void printHelp();
12 int combineBytes(char* block, size_t readBytes, bool code);
13 char* codification(int binaryCode, size_t readBytes);
14 char* decodification(int binaryCode, size_t readBytes);
15 int findPosition(char character);
16 void removeCharacter(char* buffer, char character, size_t* readBytes);
17 void resetBuffer(char* buffer);
18 void modifyBuffer(char* buffer);
19
20
21
22 int main(int argc, char const *argv[]) {
23     bool code = true; //flag para saber que operacion realizar
24     FILE* inputFile = stdin;
25     FILE* outputFile = stdout;
26     char* (*func)(int, size_t) = codification;
27
28     for (size_t i = 1; i < argc; i++){
29         if (!strcmp(argv[i], "-h") || !strcmp(argv[i], "--help")) {
30             printHelp();
31             return 0;
32         }
33
34         else if (!strcmp(argv[i], "-V") || !strcmp(argv[i], "--version")){
35             fprintf(stdout, "Version 1.0.0\n");
36             return 0;
37         }
38
39         else if (!strcmp(argv[i], "-d") || !strcmp(argv[i], "--decode")){
40             code = false;
41             func = decodification;
42         }
43
44         else if (!strcmp(argv[i], "-i") || !strcmp(argv[i], "--input")){
45             inputFile = fopen(argv[i + 1], "r");
46
47             if(!inputFile){
48                 fprintf(stderr, "Error: Cannot open/find the specified
                    ↪ input file");
```

```

49         return 1;
50     }
51 }
52
53 else if (!strcmp(argv[i], "-o") || !strcmp(argv[i], "--output")){
54     outputFile = fopen(argv[i + 1], "w");
55
56     if(!outputFile){
57         fprintf(stderr, "Error: Cannot open/find the specified
58             ↪ input file");
59         return 1;
60     }
61 }
62
63 int bufferSize = (code) ? 4 : 5; // ambos estan +1 para incluir el
64     ↪ \0
65 char buffer[bufferSize];
66 buffer[bufferSize - 1] = '\0';
67 size_t readBytes = fread(buffer, 1, sizeof(buffer) - 1, inputFile);
68
69 while(!feof(inputFile) || readBytes != 0){
70     if (!code && (strstr(buffer, "=") || strstr(buffer, "\n"))){
71         char remove = (strstr(buffer, "=") ? '=' : '\n');
72         removeCharacter(buffer, remove, &readBytes);
73         if (!readBytes) break;
74     }
75
76     int combinedBytes = combineBytes(buffer, readBytes, code);
77
78     if (combinedBytes < 0){
79         return 1;
80     }
81
82     char* result = func(combinedBytes, readBytes);
83     fwrite(result, sizeof(char), strlen(result), outputFile);
84     free(result);
85     resetBuffer(buffer); //Lo limpiamos para que no quede con basura
86     ↪ en cada iteracion
87     readBytes = fread(buffer, 1, sizeof(buffer) - 1, inputFile);
88 }
89 return 0;
90 }
91
92 void printHelp(){
93     fprintf(stdout, "Usage:\n\tttp0 -h\n\tttp0 -V\n\tttp0 [options]\n");
94     fprintf(stdout, "Options:\n\t-V, --version \tPrint version and
95         ↪ quit.\n\t-h, --help\
96     Print this information.\n\t-o, --output \tPath to output
97         ↪ file.\n\t-i, --input\
98     Path to input file.\n\t-d, --decode \tDecode a base64-encoded
99         ↪ file.\n");

```



```

97     fprintf(stdout, "Examples: \n\ttp0 -i input.txt -o output.txt\n");
98
99 }
100
101 int combineBytes(char* block, size_t readBytes, bool code){
102     int resultado = (0x00000000);
103     int SHIFTS = 24;
104     int mult = (code) ? 8 : 6;
105     for (int i = 0; i < readBytes; i++){
106         int shiftLeft = SHIFTS - (i + 1) * mult;
107         int toShift = (code) ? (int)block[i] : findPosition(block[i]);
108         if (toShift < 0){
109             fprintf(stderr, "Error: El caracter %c no se encuentra en base
110                 ↪ 64\n", block[i]);
111             return -1;
112         }
113         resultado |= toShift << shiftLeft;
114     }
115     return resultado;
116 }
117
118 char* codification(int binaryCode, size_t readBytes){
119     int shiftsLeft[] = {8, 14, 20, 26};
120     int shiftRight = 26;
121     char* code = malloc(5);
122     code[0] = '=';
123     code[1] = '=';
124     code[2] = '=';
125     code[3] = '=';
126     code[4] = '\\0';
127     size_t posCode = 0;
128     for (size_t i = 0; i < readBytes + 1; i++){
129         int binaryCodeAux = binaryCode; //Para no modificar el codigo
130             ↪ binario original
131         binaryCodeAux = binaryCodeAux << shiftsLeft[i];
132         binaryCodeAux = (int)((unsigned) binaryCodeAux >> shiftRight);
133         code[posCode++] = BASE64[binaryCodeAux];
134     }
135     return code;
136 }
137
138 char* decodification(int binaryCode, size_t readBytes){
139     int shiftsLeft[] = {8, 16, 24};
140     int shiftRight = 24;
141     char* code = malloc(readBytes);
142     code[readBytes - 1] = '\\0';
143
144     size_t posCode = 0;
145     for (size_t i = 0; i < readBytes - 1; i++){
146         int binaryCodeAux = binaryCode;
147         binaryCodeAux = binaryCodeAux << shiftsLeft[i];
148         binaryCodeAux = (int)((unsigned) binaryCodeAux >> shiftRight);
149         code[posCode++] = (char) binaryCodeAux;

```

```
149     }
150     return code;
151 }
152
153 int findPosition(char character){
154     for (int i = 0; i < strlen(BASE64); i++){
155         if (character == BASE64[i]) return i;
156     }
157     return -1;
158 }
159
160 void removeCharacter(char* buffer, char character, size_t* readBytes){
161     for (size_t i = 0; i < strlen(buffer); i++){
162         if (buffer[i] == character) *readBytes -= 1;
163     }
164 }
165
166 void resetBuffer(char* buffer){
167     size_t pos = 0;
168     while(buffer[pos] != '\0') buffer[pos++] = '\0';
169 }
```

A.2. Código MIPS32

```
1  .file 1 "tp0.c"
2  .section .mdebug.abi32
3  .previous
4  .nan legacy
5  .module fp=xx
6  .module nooddspreg
7  .abicalls
8  .globl BASE64
9  .rdata
10 .align 2
11 $LC0:
12 .ascii "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123"
13 .ascii "456789+/\000"
14 .section .data.rel.local,"aw",@progbits
15 .align 2
16 .type BASE64, @object
17 .size BASE64, 4
18 BASE64:
19 .word $LC0
20 .rdata
21 .align 2
22 $LC1:
23 .ascii "\000"
24 .align 2
25 $LC2:
26 .ascii "\000"
27 .align 2
```

```

28 $LC3:
29     .ascii  "--V\000"
30     .align  2
31 $LC4:
32     .ascii  "--version\000"
33     .align  2
34 $LC5:
35     .ascii  "Version 1.0.0\012\000"
36     .align  2
37 $LC6:
38     .ascii  "--d\000"
39     .align  2
40 $LC7:
41     .ascii  "--decode\000"
42     .align  2
43 $LC8:
44     .ascii  "--i\000"
45     .align  2
46 $LC9:
47     .ascii  "--input\000"
48     .align  2
49 $LC10:
50     .ascii  "r\000"
51     .align  2
52 $LC11:
53     .ascii  "Error: Cannot open/find the specified input file\000"
54     .align  2
55 $LC12:
56     .ascii  "--o\000"
57     .align  2
58 $LC13:
59     .ascii  "--output\000"
60     .align  2
61 $LC14:
62     .ascii  "w\000"
63     .text
64     .align  2
65     .globl  main
66     .set   nomips16
67     .set   nomicromips
68     .ent   main
69     .type  main, @function
70 main:
71     .frame $fp,120,$31    # vars= 56, regs= 10/0, args= 16, gp= 8
72     .mask  0xc0ff0000,-4
73     .fmask 0x00000000,0
74     .set   noreorder
75     .cplod $25
76     .set   nomacro
77     addiu  $sp,$sp,-120
78     sw     $31,116($sp)
79     sw     $fp,112($sp)
80     sw     $23,108($sp)
81     sw     $22,104($sp)

```

```

82  sw  $21,100($sp)
83  sw  $20,96($sp)
84  sw  $19,92($sp)
85  sw  $18,88($sp)
86  sw  $17,84($sp)
87  sw  $16,80($sp)
88  move $fp,$sp
89  .cprestore 16
90  sw  $4,120($fp)
91  sw  $5,124($fp)
92  li  $2,1      # 0x1
93  sb  $2,24($fp)
94  lw  $2,%got(stdin)($28)
95  lw  $2,0($2)
96  sw  $2,28($fp)
97  lw  $2,%got(stdout)($28)
98  lw  $2,0($2)
99  sw  $2,32($fp)
100 lw  $2,%got(codification)($28)
101 sw  $2,36($fp)
102 li  $2,1      # 0x1
103 sw  $2,40($fp)
104 b   $L2
105 nop
106
107 $L15:
108 lw  $2,40($fp)
109 sll $2,$2,2
110 lw  $3,124($fp)
111 addu $2,$3,$2
112 lw  $3,0($2)
113 lw  $2,%got($LC1)($28)
114 addiu $5,$2,%lo($LC1)
115 move $4,$3
116 lw  $2,%call16(strcmp)($28)
117 move $25,$2
118 .reloc 1f,R_MIPS_JALR,strcmp
119 1: jalr $25
120 nop
121
122 lw  $28,16($fp)
123 beq $2,$0,$L3
124 nop
125
126 lw  $2,40($fp)
127 sll $2,$2,2
128 lw  $3,124($fp)
129 addu $2,$3,$2
130 lw  $3,0($2)
131 lw  $2,%got($LC2)($28)
132 addiu $5,$2,%lo($LC2)
133 move $4,$3
134 lw  $2,%call16(strcmp)($28)
135 move $25,$2

```

```

136 | .reloc 1f,R_MIPS_JALR,strcmp
137 | 1: jalr $25
138 | nop
139 |
140 | lw $28,16($fp)
141 | bne $2,$0,$L4
142 | nop
143 |
144 | $L3:
145 | lw $2,%got(printHelp)($28)
146 | move $25,$2
147 | .reloc 1f,R_MIPS_JALR,printHelp
148 | 1: jalr $25
149 | nop
150 |
151 | lw $28,16($fp)
152 | move $2,$0
153 | b $L27
154 | nop
155 |
156 | $L4:
157 | lw $2,40($fp)
158 | sll $2,$2,2
159 | lw $3,124($fp)
160 | addu $2,$3,$2
161 | lw $3,0($2)
162 | lw $2,%got($LC3)($28)
163 | addiu $5,$2,%lo($LC3)
164 | move $4,$3
165 | lw $2,%call16(strcmp)($28)
166 | move $25,$2
167 | .reloc 1f,R_MIPS_JALR,strcmp
168 | 1: jalr $25
169 | nop
170 |
171 | lw $28,16($fp)
172 | beq $2,$0,$L6
173 | nop
174 |
175 | lw $2,40($fp)
176 | sll $2,$2,2
177 | lw $3,124($fp)
178 | addu $2,$3,$2
179 | lw $3,0($2)
180 | lw $2,%got($LC4)($28)
181 | addiu $5,$2,%lo($LC4)
182 | move $4,$3
183 | lw $2,%call16(strcmp)($28)
184 | move $25,$2
185 | .reloc 1f,R_MIPS_JALR,strcmp
186 | 1: jalr $25
187 | nop
188 |
189 | lw $28,16($fp)

```

```

190     bne $2,$0,$L7
191     nop
192
193 $L6:
194     lw  $2,%got(stdout)($28)
195     lw  $2,0($2)
196     move $7,$2
197     li  $6,14      # 0xe
198     li  $5,1       # 0x1
199     lw  $2,%got($LC5)($28)
200     addiu $4,$2,%lo($LC5)
201     lw  $2,%call16(fwrite)($28)
202     move $25,$2
203     .reloc 1f,R_MIPS_JALR,fwrite
204 1:    jalr $25
205     nop
206
207     lw  $28,16($fp)
208     move $2,$0
209     b $L27
210     nop
211
212 $L7:
213     lw  $2,40($fp)
214     sll $2,$2,2
215     lw  $3,124($fp)
216     addu $2,$3,$2
217     lw  $3,0($2)
218     lw  $2,%got($LC6)($28)
219     addiu $5,$2,%lo($LC6)
220     move $4,$3
221     lw  $2,%call16(strcmp)($28)
222     move $25,$2
223     .reloc 1f,R_MIPS_JALR,strcmp
224 1:    jalr $25
225     nop
226
227     lw  $28,16($fp)
228     beq $2,$0,$L8
229     nop
230
231     lw  $2,40($fp)
232     sll $2,$2,2
233     lw  $3,124($fp)
234     addu $2,$3,$2
235     lw  $3,0($2)
236     lw  $2,%got($LC7)($28)
237     addiu $5,$2,%lo($LC7)
238     move $4,$3
239     lw  $2,%call16(strcmp)($28)
240     move $25,$2
241     .reloc 1f,R_MIPS_JALR,strcmp
242 1:    jalr $25
243     nop

```

```

244
245     lw    $28,16($fp)
246     bne   $2,$0,$L9
247     nop
248
249 $L8:
250     sb     $0,24($fp)
251     lw     $2,%got(decodification)($28)
252     sw     $2,36($fp)
253     b      $L10
254     nop
255
256 $L9:
257     lw     $2,40($fp)
258     sll    $2,$2,2
259     lw     $3,124($fp)
260     addu   $2,$3,$2
261     lw     $3,0($2)
262     lw     $2,%got($LC8)($28)
263     addiu  $5,$2,%lo($LC8)
264     move   $4,$3
265     lw     $2,%call16(strcmp)($28)
266     move   $25,$2
267     .reloc 1f,R_MIPS_JALR,strcmp
268 1:     jalr    $25
269     nop
270
271     lw     $28,16($fp)
272     beq    $2,$0,$L11
273     nop
274
275     lw     $2,40($fp)
276     sll    $2,$2,2
277     lw     $3,124($fp)
278     addu   $2,$3,$2
279     lw     $3,0($2)
280     lw     $2,%got($LC9)($28)
281     addiu  $5,$2,%lo($LC9)
282     move   $4,$3
283     lw     $2,%call16(strcmp)($28)
284     move   $25,$2
285     .reloc 1f,R_MIPS_JALR,strcmp
286 1:     jalr    $25
287     nop
288
289     lw     $28,16($fp)
290     bne    $2,$0,$L12
291     nop
292
293 $L11:
294     lw     $2,40($fp)
295     addiu  $2,$2,1
296     sll    $2,$2,2
297     lw     $3,124($fp)

```

```

298     addu    $2,$3,$2
299     lw      $3,0($2)
300     lw      $2,%got($LC10)($28)
301     addiu   $5,$2,%lo($LC10)
302     move    $4,$3
303     lw      $2,%call16(fopen)($28)
304     move    $25,$2
305     .reloc   1f,R_MIPS_JALR,fopen
306 1:     jalr   $25
307     nop
308
309     lw      $28,16($fp)
310     sw      $2,28($fp)
311     lw      $2,28($fp)
312     bne     $2,$0,$L10
313     nop
314
315     lw      $2,%got(stderr)($28)
316     lw      $2,0($2)
317     move    $7,$2
318     li      $6,48      # 0x30
319     li      $5,1       # 0x1
320     lw      $2,%got($LC11)($28)
321     addiu   $4,$2,%lo($LC11)
322     lw      $2,%call16(fwrite)($28)
323     move    $25,$2
324     .reloc   1f,R_MIPS_JALR,fwrite
325 1:     jalr   $25
326     nop
327
328     lw      $28,16($fp)
329     li      $2,1       # 0x1
330     b       $L27
331     nop
332
333 $L12:
334     lw      $2,40($fp)
335     sll     $2,$2,2
336     lw      $3,124($fp)
337     addu    $2,$3,$2
338     lw      $3,0($2)
339     lw      $2,%got($LC12)($28)
340     addiu   $5,$2,%lo($LC12)
341     move    $4,$3
342     lw      $2,%call16(strcmp)($28)
343     move    $25,$2
344     .reloc   1f,R_MIPS_JALR,strcmp
345 1:     jalr   $25
346     nop
347
348     lw      $28,16($fp)
349     beq     $2,$0,$L14
350     nop
351

```



```

352 lw $2,40($fp)
353 sll $2,$2,2
354 lw $3,124($fp)
355 addu $2,$3,$2
356 lw $3,0($2)
357 lw $2,%got($LC13)($28)
358 addiu $5,$2,%lo($LC13)
359 move $4,$3
360 lw $2,%call16(stremp)($28)
361 move $25,$2
362 .reloc 1f,R_MIPS_JALR,stremp
363 1: jalr $25
364 nop
365
366 lw $28,16($fp)
367 bne $2,$0,$L10
368 nop
369
370 $L14:
371 lw $2,40($fp)
372 addiu $2,$2,1
373 sll $2,$2,2
374 lw $3,124($fp)
375 addu $2,$3,$2
376 lw $3,0($2)
377 lw $2,%got($LC14)($28)
378 addiu $5,$2,%lo($LC14)
379 move $4,$3
380 lw $2,%call16(fopen)($28)
381 move $25,$2
382 .reloc 1f,R_MIPS_JALR,fopen
383 1: jalr $25
384 nop
385
386 lw $28,16($fp)
387 sw $2,32($fp)
388 lw $2,32($fp)
389 bne $2,$0,$L10
390 nop
391
392 lw $2,%got(stderr)($28)
393 lw $2,0($2)
394 move $7,$2
395 li $6,48 # 0x30
396 li $5,1 # 0x1
397 lw $2,%got($LC11)($28)
398 addiu $4,$2,%lo($LC11)
399 lw $2,%call16(fwrite)($28)
400 move $25,$2
401 .reloc 1f,R_MIPS_JALR,fwrite
402 1: jalr $25
403 nop
404
405 lw $28,16($fp)

```

```
406    li    $2,1          # 0x1
407    b     $L27
408    nop
409
410 $L10:
411    lw     $2,40($fp)
412    addiu  $2,$2,1
413    sw     $2,40($fp)
414 $L2:
415    lw     $2,120($fp)
416    lw     $3,40($fp)
417    sltu   $2,$3,$2
418    bne    $2,$0,$L15
419    nop
420
421    lbu    $2,24($fp)
422    beq    $2,$0,$L16
423    nop
424
425    li     $2,4          # 0x4
426    b     $L17
427    nop
428
429 $L16:
430    li     $2,5          # 0x5
431 $L17:
432    sw     $2,44($fp)
433    lw     $2,44($fp)
434    move    $4,$2
435    sw     $4,72($fp)
436    move    $2,$sp
437    sw     $2,76($fp)
438    move    $2,$4
439    addiu  $2,$2,-1
440    sw     $2,48($fp)
441    move    $2,$4
442    move    $23,$2
443    move    $22,$0
444    srl    $2,$23,29
445    sll    $18,$22,3
446    or     $18,$2,$18
447    sll    $19,$23,3
448    move    $2,$4
449    move    $21,$2
450    move    $20,$0
451    srl    $2,$21,29
452    sll    $16,$20,3
453    or     $16,$2,$16
454    sll    $17,$21,3
455    move    $2,$4
456    addiu  $2,$2,7
457    srl    $2,$2,3
458    sll    $2,$2,3
459    subu    $sp,$sp,$2
```

```

460    addiu $2,$sp,16
461    addiu $2,$2,0
462    sw    $2,52($fp)
463    lw    $2,44($fp)
464    addiu $2,$2,-1
465    lw    $3,52($fp)
466    addu  $2,$3,$2
467    sb    $0,0($2)
468    lw    $2,52($fp)
469    move  $3,$4
470    addiu $3,$3,-1
471    lw    $7,28($fp)
472    move  $6,$3
473    li    $5,1          # 0x1
474    move  $4,$2
475    lw    $2,%call16(fread)($28)
476    move  $25,$2
477    .reloc 1f,R_MIPS_JALR,fread
478 1:    jalr $25
479    nop
480
481    lw    $28,16($fp)
482    sw    $2,68($fp)
483    b     $L18
484    nop
485
486 $L26:
487    lbu   $2,24($fp)
488    xori  $2,$2,0x1
489    andi  $2,$2,0x00ff
490    beq   $2,$0,$L19
491    nop
492
493    lw    $2,52($fp)
494    li    $5,61          # 0x3d
495    move  $4,$2
496    lw    $2,%call16(strchr)($28)
497    move  $25,$2
498    .reloc 1f,R_MIPS_JALR,strchr
499 1:    jalr $25
500    nop
501
502    lw    $28,16($fp)
503    bne   $2,$0,$L20
504    nop
505
506    lw    $2,52($fp)
507    li    $5,10          # 0xa
508    move  $4,$2
509    lw    $2,%call16(strchr)($28)
510    move  $25,$2
511    .reloc 1f,R_MIPS_JALR,strchr
512 1:    jalr $25
513    nop

```

```

514
515     lw    $28,16($fp)
516     beq   $2,$0,$L19
517     nop
518
519 $L20:
520     lw    $2,52($fp)
521     li    $5,61      # 0x3d
522     move  $4,$2
523     lw    $2,%call16(strchr)($28)
524     move  $25,$2
525     .reloc 1f,R_MIPS_JALR,strchr
526 1:     jalr $25
527     nop
528
529     lw    $28,16($fp)
530     beq   $2,$0,$L21
531     nop
532
533     li    $2,61      # 0x3d
534     b     $L22
535     nop
536
537 $L21:
538     li    $2,10      # 0xa
539 $L22:
540     sb    $2,56($fp)
541     lw    $2,52($fp)
542     addiu $4,$fp,68
543     lb    $3,56($fp)
544     move  $6,$4
545     move  $5,$3
546     move  $4,$2
547     lw    $2,%got(removeCharacter)($28)
548     move  $25,$2
549     .reloc 1f,R_MIPS_JALR,removeCharacter
550 1:     jalr $25
551     nop
552
553     lw    $28,16($fp)
554     lw    $2,68($fp)
555     beq   $2,$0,$L28
556     nop
557
558 $L19:
559     lw    $2,52($fp)
560     lw    $3,68($fp)
561     lbu   $4,24($fp)
562     move  $6,$4
563     move  $5,$3
564     move  $4,$2
565     lw    $2,%got(combineBytes)($28)
566     move  $25,$2
567     .reloc 1f,R_MIPS_JALR,combineBytes

```

```

568 1: jalr $25
569     nop
570
571     lw $28,16($fp)
572     sw $2,60($fp)
573     lw $2,60($fp)
574     bgez $2,$L24
575     nop
576
577     li $2,1          # 0x1
578     b $L25
579     nop
580
581 $L24:
582     lw $3,68($fp)
583     lw $2,36($fp)
584     move $5,$3
585     lw $4,60($fp)
586     move $25,$2
587     jalr $25
588     nop
589
590     lw $28,16($fp)
591     sw $2,64($fp)
592     lw $4,64($fp)
593     lw $2,%call16(strlen)($28)
594     move $25,$2
595     .reloc 1f,R_MIPS_JALR,strlen
596 1: jalr $25
597     nop
598
599     lw $28,16($fp)
600     lw $7,32($fp)
601     move $6,$2
602     li $5,1          # 0x1
603     lw $4,64($fp)
604     lw $2,%call16(fwrite)($28)
605     move $25,$2
606     .reloc 1f,R_MIPS_JALR,fwrite
607 1: jalr $25
608     nop
609
610     lw $28,16($fp)
611     lw $4,64($fp)
612     lw $2,%call16(free)($28)
613     move $25,$2
614     .reloc 1f,R_MIPS_JALR,free
615 1: jalr $25
616     nop
617
618     lw $28,16($fp)
619     lw $2,52($fp)
620     move $4,$2
621     lw $2,%got(resetBuffer)($28)

```

```

622  move    $25,$2
623  .reloc   1f,R_MIPS_JALR,resetBuffer
624  1:  jalr   $25
625  nop
626
627  lw      $28,16($fp)
628  lw      $2,52($fp)
629  lw      $3,72($fp)
630  addiu   $3,$3,-1
631  lw      $7,28($fp)
632  move    $6,$3
633  li      $5,1      # 0x1
634  move    $4,$2
635  lw      $2,%call16(fread)($28)
636  move    $25,$2
637  .reloc   1f,R_MIPS_JALR,fread
638  1:  jalr   $25
639  nop
640
641  lw      $28,16($fp)
642  sw      $2,68($fp)
643  $L18:
644  lw      $4,28($fp)
645  lw      $2,%call16(feof)($28)
646  move    $25,$2
647  .reloc   1f,R_MIPS_JALR,feof
648  1:  jalr   $25
649  nop
650
651  lw      $28,16($fp)
652  beq     $2,$0,$L26
653  nop
654
655  lw      $2,68($fp)
656  bne     $2,$0,$L26
657  nop
658
659  b       $L23
660  nop
661
662  $L28:
663  nop
664  $L23:
665  move    $2,$0
666  $L25:
667  lw      $sp,76($fp)
668  $L27:
669  move    $sp,$fp
670  lw      $31,116($sp)
671  lw      $fp,112($sp)
672  lw      $23,108($sp)
673  lw      $22,104($sp)
674  lw      $21,100($sp)
675  lw      $20,96($sp)

```

```

676 lw $19,92($sp)
677 lw $18,88($sp)
678 lw $17,84($sp)
679 lw $16,80($sp)
680 addiu $sp,$sp,120
681 jr $31
682 nop
683
684 .set macro
685 .set reorder
686 .end main
687 .size main, .-main
688 .rdata
689 .align 2
690 $LC15:
691 .ascii "Usage:\012\011tp0 -h\012\011tp0 -V\012\011tp0
        ↪ [options]\012"
692 .ascii "\000"
693 .align 2
694 $LC16:
695 .ascii "Options:\012\011-V, --version \011Print version and quit"
696 .ascii ".\012\011-h, --help\011Print this information.\012\011-o"
697 .ascii ", --output \011Path to output file.\012\011-i, --input\011"
698 .ascii "Path to input file.\012\011-d, --decode \011Decode a bas"
699 .ascii "e64-encoded file.\012\000"
700 .align 2
701 $LC17:
702 .ascii "Examples: \012\011tp0 -i input.txt -o output.txt\012\000"
703 .text
704 .align 2
705 .globl printHelp
706 .set nomips16
707 .set nomicromips
708 .ent printHelp
709 .type printHelp, @function
710 printHelp:
711 .frame $fp,32,$31 # vars= 0, regs= 2/0, args= 16, gp= 8
712 .mask 0xc0000000,-4
713 .fmask 0x00000000,0
714 .set noreorder
715 .cpload $25
716 .set nomacro
717 addiu $sp,$sp,-32
718 sw $31,28($sp)
719 sw $fp,24($sp)
720 move $fp,$sp
721 .cpstore 16
722 lw $2,%got(stdout)($28)
723 lw $2,0($2)
724 move $7,$2
725 li $6,38 # 0x26
726 li $5,1 # 0x1
727 lw $2,%got($LC15)($28)
728 addiu $4,$2,%lo($LC15)

```

```

729 lw $2,%call16(fwrite)($28)
730 move $25,$2
731 .reloc 1f,R_MIPS_JALR,fwrite
732 1: jalr $25
733 nop
734
735 lw $28,16($fp)
736 lw $2,%got(stdout)($28)
737 lw $2,0($2)
738 move $7,$2
739 li $6,199 # 0xc7
740 li $5,1 # 0x1
741 lw $2,%got($LC16)($28)
742 addiu $4,$2,%lo($LC16)
743 lw $2,%call16(fwrite)($28)
744 move $25,$2
745 .reloc 1f,R_MIPS_JALR,fwrite
746 1: jalr $25
747 nop
748
749 lw $28,16($fp)
750 lw $2,%got(stdout)($28)
751 lw $2,0($2)
752 move $7,$2
753 li $6,43 # 0x2b
754 li $5,1 # 0x1
755 lw $2,%got($LC17)($28)
756 addiu $4,$2,%lo($LC17)
757 lw $2,%call16(fwrite)($28)
758 move $25,$2
759 .reloc 1f,R_MIPS_JALR,fwrite
760 1: jalr $25
761 nop
762
763 lw $28,16($fp)
764 nop
765 move $sp,$fp
766 lw $31,28($sp)
767 lw $fp,24($sp)
768 addiu $sp,$sp,32
769 jr $31
770 nop
771
772 .set macro
773 .set reorder
774 .end printHelp
775 .size printHelp,.-printHelp
776 .rdata
777 .align 2
778 $LC18:
779 .ascii "Error: El caracter %c no se encuentra en base 64\012\000"
780 .text
781 .align 2
782 .globl combineBytes

```



```

783 .set nomips16
784 .set nomicromips
785 .ent combineBytes
786 .type combineBytes, @function
787 combineBytes:
788 .frame $fp,56,$31      # vars= 24, regs= 2/0, args= 16, gp= 8
789 .mask 0xc0000000,-4
790 .fmask 0x00000000,0
791 .set noreorder
792 .cpload $25
793 .set nomacro
794 addiu $sp,$sp,-56
795 sw $31,52($sp)
796 sw $fp,48($sp)
797 move $fp,$sp
798 .cpstore 16
799 sw $4,56($fp)
800 sw $5,60($fp)
801 move $2,$6
802 sb $2,64($fp)
803 sw $0,24($fp)
804 li $2,24      # 0x18
805 sw $2,32($fp)
806 lbu $2,64($fp)
807 beq $2,$0,$L31
808 nop
809
810 li $2,8      # 0x8
811 b $L32
812 nop
813
814 $L31:
815 li $2,6      # 0x6
816 $L32:
817 sw $2,36($fp)
818 sw $0,28($fp)
819 b $L33
820 nop
821
822 $L38:
823 lw $2,28($fp)
824 addiu $3,$2,1
825 lw $2,36($fp)
826 mul $2,$3,$2
827 lw $3,32($fp)
828 subu $2,$3,$2
829 sw $2,40($fp)
830 lbu $2,64($fp)
831 beq $2,$0,$L34
832 nop
833
834 lw $2,28($fp)
835 lw $3,56($fp)
836 addu $2,$3,$2

```

```

837     lb    $2,0($2)
838     b     $L35
839     nop
840
841 $L34:
842     lw    $2,28($fp)
843     lw    $3,56($fp)
844     addu   $2,$3,$2
845     lb    $2,0($2)
846     move   $4,$2
847     lw    $2,%got(findPosition)($28)
848     move   $25,$2
849     .reloc 1f,R_MIPS_JALR,findPosition
850 1:     jalr  $25
851     nop
852
853     lw    $28,16($fp)
854 $L35:
855     sw    $2,44($fp)
856     lw    $2,44($fp)
857     bgez   $2,$L36
858     nop
859
860     lw    $2,%got(stderr)($28)
861     lw    $4,0($2)
862     lw    $2,28($fp)
863     lw    $3,56($fp)
864     addu   $2,$3,$2
865     lb    $2,0($2)
866     move   $6,$2
867     lw    $2,%got($LC18)($28)
868     addiu  $5,$2,%lo($LC18)
869     lw    $2,%call16(fprintf)($28)
870     move   $25,$2
871     .reloc 1f,R_MIPS_JALR,fprintf
872 1:     jalr  $25
873     nop
874
875     lw    $28,16($fp)
876     li    $2,-1      # 0xffffffffffffffff
877     b     $L37
878     nop
879
880 $L36:
881     lw    $3,44($fp)
882     lw    $2,40($fp)
883     sll   $2,$3,$2
884     lw    $3,24($fp)
885     or    $2,$3,$2
886     sw    $2,24($fp)
887     lw    $2,28($fp)
888     addiu $2,$2,1
889     sw    $2,28($fp)
890 $L33:

```

```

891 lw $3,28($fp)
892 lw $2,60($fp)
893 sltu $2,$3,$2
894 bne $2,$0,$L38
895 nop
896
897 lw $2,24($fp)
898 $L37:
899 move $sp,$fp
900 lw $31,52($sp)
901 lw $fp,48($sp)
902 addiu $sp,$sp,56
903 jr $31
904 nop
905
906 .set macro
907 .set reorder
908 .end combineBytes
909 .size combineBytes,.-combineBytes
910 .align 2
911 .globl codification
912 .set nomips16
913 .set nomicromips
914 .ent codification
915 .type codification,@function
916 codification:
917 .frame $fp,72,$31 # vars= 40, regs= 2/0, args= 16, gp= 8
918 .mask 0xc0000000,-4
919 .fmask 0x00000000,0
920 .set noreorder
921 .cpload $25
922 .set nomacro
923 addiu $sp,$sp,-72
924 sw $31,68($sp)
925 sw $fp,64($sp)
926 move $fp,$sp
927 .cpstore 16
928 sw $4,72($fp)
929 sw $5,76($fp)
930 li $2,8 # 0x8
931 sw $2,44($fp)
932 li $2,14 # 0xe
933 sw $2,48($fp)
934 li $2,20 # 0x14
935 sw $2,52($fp)
936 li $2,26 # 0x1a
937 sw $2,56($fp)
938 li $2,26 # 0x1a
939 sw $2,32($fp)
940 li $4,5 # 0x5
941 lw $2,%call16(malloc)($28)
942 move $25,$2
943 .reloc 1f,R_MIPS_JALR,malloc
944 1: jalr $25

```

```

945  nop
946
947  lw   $28,16($fp)
948  sw   $2,36($fp)
949  lw   $2,36($fp)
950  li   $3,61      # 0x3d
951  sb   $3,0($2)
952  lw   $2,36($fp)
953  addiu $2,$2,1
954  li   $3,61      # 0x3d
955  sb   $3,0($2)
956  lw   $2,36($fp)
957  addiu $2,$2,2
958  li   $3,61      # 0x3d
959  sb   $3,0($2)
960  lw   $2,36($fp)
961  addiu $2,$2,3
962  li   $3,61      # 0x3d
963  sb   $3,0($2)
964  lw   $2,36($fp)
965  addiu $2,$2,4
966  sb   $0,0($2)
967  sw   $0,24($fp)
968  sw   $0,28($fp)
969  b    $L40
970  nop
971
972  $L41:
973  lw   $2,72($fp)
974  sw   $2,40($fp)
975  lw   $2,28($fp)
976  sll  $2,$2,2
977  addiu $3,$fp,24
978  addu  $2,$3,$2
979  lw   $2,20($2)
980  lw   $3,40($fp)
981  sll  $2,$3,$2
982  sw   $2,40($fp)
983  lw   $3,40($fp)
984  lw   $2,32($fp)
985  srl  $2,$3,$2
986  sw   $2,40($fp)
987  lw   $2,24($fp)
988  addiu $3,$2,1
989  sw   $3,24($fp)
990  lw   $3,36($fp)
991  addu  $2,$3,$2
992  lw   $3,%got(BASE64)($28)
993  lw   $4,0($3)
994  lw   $3,40($fp)
995  addu  $3,$4,$3
996  lb   $3,0($3)
997  sb   $3,0($2)
998  lw   $2,28($fp)

```

```

999     addiu $2,$2,1
1000    sw    $2,28($fp)
1001    $L40:
1002    lw     $2,76($fp)
1003    addiu   $3,$2,1
1004    lw     $2,28($fp)
1005    sltu    $2,$2,$3
1006    bne     $2,$0,$L41
1007    nop
1008
1009    lw     $2,36($fp)
1010    move    $sp,$fp
1011    lw     $31,68($sp)
1012    lw     $fp,64($sp)
1013    addiu   $sp,$sp,72
1014    jr      $31
1015    nop
1016
1017    .set     macro
1018    .set     reorder
1019    .end     codification
1020    .size    codification,.-codification
1021    .align   2
1022    .globl   decodification
1023    .set     nomips16
1024    .set     nomicromips
1025    .ent     decodification
1026    .type    decodification, @function
1027    decodification:
1028    .frame   $fp,64,$31      # vars= 32, regs= 2/0, args= 16, gp= 8
1029    .mask    0xc0000000,-4
1030    .fmask   0x00000000,0
1031    .set     noreorder
1032    .cpload  $25
1033    .set     nomacro
1034    addiu    $sp,$sp,-64
1035    sw       $31,60($sp)
1036    sw       $fp,56($sp)
1037    move     $fp,$sp
1038    .cprestore 16
1039    sw       $4,64($fp)
1040    sw       $5,68($fp)
1041    li       $2,8           # 0x8
1042    sw       $2,44($fp)
1043    li       $2,16          # 0x10
1044    sw       $2,48($fp)
1045    li       $2,24          # 0x18
1046    sw       $2,52($fp)
1047    li       $2,24          # 0x18
1048    sw       $2,32($fp)
1049    lw       $4,68($fp)
1050    lw       $2,%call16(malloc)($28)
1051    move     $25,$2
1052    .reloc   1f,R_MIPS_JALR,malloc

```

```

1053 1: jalr $25
1054     nop
1055
1056     lw $28,16($fp)
1057     sw $2,36($fp)
1058     lw $2,68($fp)
1059     addiu $2,$2,-1
1060     lw $3,36($fp)
1061     addu $2,$3,$2
1062     sb $0,0($2)
1063     sw $0,24($fp)
1064     sw $0,28($fp)
1065     b $L44
1066     nop
1067
1068 $L45:
1069     lw $2,64($fp)
1070     sw $2,40($fp)
1071     lw $2,28($fp)
1072     sll $2,$2,2
1073     addiu $3,$fp,24
1074     addu $2,$3,$2
1075     lw $2,20($2)
1076     lw $3,40($fp)
1077     sll $2,$3,$2
1078     sw $2,40($fp)
1079     lw $3,40($fp)
1080     lw $2,32($fp)
1081     srl $2,$3,$2
1082     sw $2,40($fp)
1083     lw $2,24($fp)
1084     addiu $3,$2,1
1085     sw $3,24($fp)
1086     lw $3,36($fp)
1087     addu $2,$3,$2
1088     lw $3,40($fp)
1089     seb $3,$3
1090     sb $3,0($2)
1091     lw $2,28($fp)
1092     addiu $2,$2,1
1093     sw $2,28($fp)
1094 $L44:
1095     lw $2,68($fp)
1096     addiu $3,$2,-1
1097     lw $2,28($fp)
1098     sltu $2,$2,$3
1099     bne $2,$0,$L45
1100     nop
1101
1102     lw $2,36($fp)
1103     move $sp,$fp
1104     lw $31,60($sp)
1105     lw $fp,56($sp)
1106     addiu $sp,$sp,64

```

```

1107 jr    $31
1108 nop
1109
1110 .set    macro
1111 .set    reorder
1112 .end    decodification
1113 .size   decodification , .-decodification
1114 .align  2
1115 .globl  findPosition
1116 .set    nomips16
1117 .set    nomicromips
1118 .ent    findPosition
1119 .type   findPosition , @function
1120 findPosition:
1121 .frame   $fp,40,$31      # vars= 8, regs= 2/0, args= 16, gp= 8
1122 .mask    0xc0000000,-4
1123 .fmask    0x00000000,0
1124 .set     noreorder
1125 .cpload  $25
1126 .set     nomacro
1127 addiu   $sp,$sp,-40
1128 sw      $31,36($sp)
1129 sw      $fp,32($sp)
1130 move    $fp,$sp
1131 .cprestore 16
1132 move    $2,$4
1133 sb      $2,40($fp)
1134 sw      $0,24($fp)
1135 b       $L48
1136 nop
1137
1138 $L51:
1139 lw      $2,%got(BASE64)($28)
1140 lw      $3,0($2)
1141 lw      $2,24($fp)
1142 addu    $2,$3,$2
1143 lb      $2,0($2)
1144 lb      $3,40($fp)
1145 bne     $3,$2,$L49
1146 nop
1147
1148 lw      $2,24($fp)
1149 b       $L50
1150 nop
1151
1152 $L49:
1153 lw      $2,24($fp)
1154 addiu   $2,$2,1
1155 sw      $2,24($fp)
1156 $L48:
1157 lw      $2,%got(BASE64)($28)
1158 lw      $2,0($2)
1159 move    $4,$2
1160 lw      $2,%call16(strlen)($28)

```

```

1161  move    $25,$2
1162  .reloc   1f,R_MIPS_JALR,strlen
1163  1:  jalr   $25
1164  nop
1165
1166  lw      $28,16($fp)
1167  move    $3,$2
1168  lw      $2,24($fp)
1169  sltu    $2,$2,$3
1170  bne     $2,$0,$L51
1171  nop
1172
1173  li      $2,-1      # 0xffffffffffffffff
1174  $L50:
1175  move    $sp,$fp
1176  lw      $31,36($sp)
1177  lw      $fp,32($sp)
1178  addiu   $sp,$sp,40
1179  jr      $31
1180  nop
1181
1182  .set     macro
1183  .set     reorder
1184  .end     findPosition
1185  .size    findPosition,.-findPosition
1186  .align   2
1187  .globl   removeCharacter
1188  .set     nomips16
1189  .set     nomicromips
1190  .ent     removeCharacter
1191  .type    removeCharacter, @function
1192  removeCharacter:
1193  .frame   $fp,40,$31      # vars= 8, regs= 2/0, args= 16, gp= 8
1194  .mask    0xc0000000,-4
1195  .fmask   0x00000000,0
1196  .set     noreorder
1197  .cpload  $25
1198  .set     nomacro
1199  addiu    $sp,$sp,-40
1200  sw       $31,36($sp)
1201  sw       $fp,32($sp)
1202  move     $fp,$sp
1203  .cprestore 16
1204  sw       $4,40($fp)
1205  move     $2,$5
1206  sw       $6,48($fp)
1207  sb       $2,44($fp)
1208  sw       $0,24($fp)
1209  b        $L53
1210  nop
1211
1212  $L55:
1213  lw       $3,40($fp)
1214  lw       $2,24($fp)

```



```

1215     addu    $2,$3,$2
1216     lb     $2,0($2)
1217     lb     $3,44($fp)
1218     bne    $3,$2,$L54
1219     nop
1220
1221     lw     $2,48($fp)
1222     lw     $2,0($2)
1223     addiu   $3,$2,-1
1224     lw     $2,48($fp)
1225     sw     $3,0($2)
1226 $L54:
1227     lw     $2,24($fp)
1228     addiu   $2,$2,1
1229     sw     $2,24($fp)
1230 $L53:
1231     lw     $4,40($fp)
1232     lw     $2,%call16(strlen)($28)
1233     move    $25,$2
1234     .reloc  1f,R_MIPS_JALR,strlen
1235 1:     jalr   $25
1236     nop
1237
1238     lw     $28,16($fp)
1239     move    $3,$2
1240     lw     $2,24($fp)
1241     sltu    $2,$2,$3
1242     bne     $2,$0,$L55
1243     nop
1244
1245     nop
1246     move    $sp,$fp
1247     lw     $31,36($sp)
1248     lw     $fp,32($sp)
1249     addiu   $sp,$sp,40
1250     jr      $31
1251     nop
1252
1253     .set    macro
1254     .set    reorder
1255     .end    removeCharacter
1256     .size   removeCharacter,.-removeCharacter
1257     .align  2
1258     .globl  resetBuffer
1259     .set    nomips16
1260     .set    nomicromips
1261     .ent    resetBuffer
1262     .type   resetBuffer,@function
1263 resetBuffer:
1264     .frame  $fp,24,$31    # vars= 8, regs= 1/0, args= 0, gp= 8
1265     .mask   0x40000000,-4
1266     .fmask  0x00000000,0
1267     .set    noreorder
1268     .set    nomacro

```

```
1269     addiu $sp,$sp,-24
1270     sw    $fp,20($sp)
1271     move   $fp,$sp
1272     sw    $4,24($fp)
1273     sw    $0,8($fp)
1274     b     $L57
1275     nop
1276
1277 $L58:
1278     lw    $2,8($fp)
1279     addiu  $3,$2,1
1280     sw    $3,8($fp)
1281     lw    $3,24($fp)
1282     addu   $2,$3,$2
1283     sb    $0,0($2)
1284 $L57:
1285     lw    $3,24($fp)
1286     lw    $2,8($fp)
1287     addu   $2,$3,$2
1288     lb    $2,0($2)
1289     bne    $2,$0,$L58
1290     nop
1291
1292     nop
1293     move   $sp,$fp
1294     lw    $fp,20($sp)
1295     addiu  $sp,$sp,24
1296     jr     $31
1297     nop
1298
1299     .set   macro
1300     .set   reorder
1301     .end   resetBuffer
1302     .size  resetBuffer,.-resetBuffer
1303     .ident "GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516"
```

B. Enunciado del trabajo práctico

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

1. Objetivos

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

2. Alcance

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

3. Requisitos

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

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

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, y se valorarán aquellos escritos usando la herramienta $\text{T}_{\text{E}}\text{X}$ / $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$.

4. Recursos

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

5. Base 64

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

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

6. Programa

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

6.1. Ejemplos

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

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

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

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

```
$ cat quijote.txt
En un lugar de La Mancha de cuyo nombre no quiero acordarme
$ ./tp0 -i quijote.txt -o qb64
$ cat qb64
RW4gdW4gbHVnYXJlZGUgTGEGTWFuY2hhIGRlIGN1eW8gbm9tYnJlIG5vIHF1aWVybY29yZGFy
bWUK
```

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

```
cat quijote.txt | ./tp0
RW4gdW4gbHVnYXJlZGUgTGEGTWFuY2hhIGRlIGN1eW8gbm9tYnJlIG5vIHF1aWVybY29yZGFy
bWUK
```

También se puede usar para decodificar:

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

7. Informe

El informe deberá incluir:

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

8. Fecha de entrega

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

Referencias

- [1] QEMU, <https://www.qemu.org/>.
- [2] Debian, the Universal Operating System, <https://www.debian.org/>.
- [3] Codificación base64, <https://es.wikipedia.org/wiki/Base64>

C. Código auxiliar

C.1. Pruebas automatizadas

```

import subprocess
import sys
import glob
import tempfile
def run_command(command):
    proc = subprocess.Popen(command,
        stdout=subprocess.PIPE, stderr= subprocess.PIPE,
        shell=True, universal_newlines=True)
    std_out, std_err = proc.communicate()
    return proc.returncode, std_out, std_err

def main(command, input_prefix, result_prefix):
    input_files = glob.glob(f'{input_prefix}*')
    test_files = { file_name[len(input_prefix):]:
        [file_name,] for file_name in input_files }
    for id, files in test_files.items():
        files.append(f'{result_prefix}{id}')

    for file, output_file in test_files.values():
        result_obtained = run_command(f'{command} < {file}')[1]
        with tempfile.NamedTemporaryFile() as tmp:
            tmp.write(bytes(result_obtained, encoding='utf-8'))
            tmp.read()
            code_diff, stdout_diff, _ = run_command(f"""
                diff {output_file} {tmp.name}
            """)
            if code_diff == 0:
                print('.', end='')
                continue
            print()
            print(f'Ejecución con {file} comparación con {output_file}')
            print(stdout_diff)

if __name__ == "__main__":
    if len(sys.argv) == 4:
        command, input_prefix, result_prefix = sys.argv[1:]
        main(command, input_prefix, result_prefix)
    else:
        print("""
        Use: python test.py command input_prefix result_prefix
        command: comando a ejecutar.
        input_prefix: prefijo de los archivos que se usaran como entrada.
        result_prefix: prefijo de los archivos a comparar.
        """)

```

Referencias

- [1] *Artículo de Wikipedia sobre Base64*. <https://es.wikipedia.org/wiki/Base64>
- [2] *Tabla ascii*. <https://elcodigoascii.com.ar/>
- [3] *Parámetros de gcc*. <https://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html>