Trabajo Práctico Integrador

Introducción

En los primeros días del correo electrónico, este solo podía manejar caracteres imprimibles, por lo que se desarrollaron distintos métodos para poder enviar información binaria a través del correo electrónico.

Uno de estos métodos era la codificación BASE64, en esta codificación se convierten 3 bytes binarios en 4 caracteres imprimibles de la siguiente manera:

1) Se agrupa una secuencia de 3 bytes binarios formando una secuencia de 24 bits, por ejemplo, si se tiene la secuencia 0xFA, 0x17, 0x6B se tiene

0xFA	0x17	0x6B
11111010	00010111	01101011
111110100001011101101011		

2) Esta secuencia de 24 bits se divide en 4 grupos de 6 bits, obteniéndose 4 números binarios (y su equivalente decimal)

111110-100001-011101-101011			
111110	100001	011101	101011
62	**33**	**29**	**43**

3) Estos números binarios se utilizan como índice de una tabla (similar a la tabla ascii) que tiene los caracteres alfabéticos en mayúscula y minúscula (A-Z y a-z), los dígitos decimales (0-9) y los símbolos “+” y “/” con la siguiente equivalencia

Índice	Caracter
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I
9	J
10	K
11	L
12	M
13	N
14	O
15	P

Índice	Caracter
16	Q
17	R
18	S
19	T
20	U
21	V
22	W
23	X
24	Y
25	Z
26	a
27	b
28	c
29	d
30	e
31	f

Índice	Caracter
32	g
33	h
34	i
35	j
36	k
37	l
38	m
39	n
40	o
41	p
42	q
43	r
44	s
45	t
46	u
47	v

Índice	Caracter
48	w
49	x
50	y
51	z
52	0
53	1
54	2
55	3
56	4
57	5
58	6
59	7
60	8
61	9
62	+
63	/

Quedando la secuencia de caracteres imprimibles

62	33	29	43
+	h	d	r

4) De esta manera, la secuencia binaria **0xFA, 0x17, 0x6B** se codifica como la secuencia de 4 caracteres imprimibles **+hdr**

5) El algoritmo trabaja agrupando de a 3 bytes de la secuencia binaria y convirtiéndolos a una secuencia de 4 caracteres imprimibles. Si la cantidad de bytes binarios es divisible por 3, la conversión termina con el último grupo de 3 bytes. Puede darse el caso que al final de la secuencia no quede un grupo de 3 bytes, sino un grupo de 2 bytes o un byte solo.

   a. Si quedan 2 bytes, se agrega un byte 0x00 y se indica esto agregando un símbolo "=" al final de la conversión (notar que el símbolo "=" no está en la tabla de conversión). Ejemplo: el ultimo grupo es la secuencia 0xAA, 0xD4

   b. Relleno al final con 0x00

0xAA	**0xD4**	***0x00***
10101010	**11010100**	**00000000**
101010101101010000000000		

   c. Divido en 4 grupos de 6 bits

101010-101101-010000-000000			
101010	**101101**	**010000**	**000000**
42	**45**	**16**	**00**

   d. Utilizo los números como índice de la tabla

42	45	16	00
q	t	Q	A

   e. De esta manera el grupo binario final **0xAA, 0xD4** se codifica como la secuencia de caracteres imprimibles **qtQA=** (el igual "=" se agrega para indicar en la decodificación que el tercer byte binario 0x00 fue colocado de relleno y no forma parte de la secuencia binaria original)

   f. Si queda un solo byte al final, se agregan dos bytes 0x00 y se indica esto agregando dos símbolos "=" al final de la conversión. Ejemplo: el último byte de la secuencia es 0x18

g. Relleno al final con dos bytes 0x00

0x18	*0x00*	*0x00*
00011000	00000000	00000000
000110000000000000000000		

h. Divido en 4 grupos de 6 bits

000110-000000-000000-000000			
000110	000000	000000	000000
6	00	00	00

i. Utilizo los números como índice de la tabla

6	00	00	00
G	A	A	A

j. De esta manera el byte final **0x18** se codifica como la secuencia de caracteres imprimibles **GAAA==** (los dos símbolos igual "==" se agregan para indicar en la decodificación que el segundo y tercer byte binario 0x00 fueron colocados de relleno y no forman parte de la secuencia binaria original)

La decodificación aplica el proceso inverso, se toman grupos de 4 caracteres imprimibles y se genera la secuencia de 24 bits binarios que la originaron

1) Se agrupa una secuencia de 4 caracteres imprimibles y se reemplaza cada carácter por su índice en la tabla, por ejemplo, si se tiene la secuencia +PrT se tiene

+PrT			
62	15	43	19
111110	001111	101011	010011

2) Se agrupan los números binarios de los 4 índices formando una secuencia de 24 bits

111110	001111	101011	010011
111110001111101011010011			

3) Se divide la secuencia de 24 bits en 3 grupos de 8 bits

11111000-11111010-11010011		
11111000	11111010	11010011
0xF8	0xFA	0xD3

4) De esta manera, la secuencia de 4 caracteres imprimibles **+PrT** se decodifica a la secuencia binaria **0xF8, 0xFA, 0xD3**.

5) Puede darse el caso que a continuación de la última secuencia de 4 caracteres imprimibles haya un símbolo "=", esto indica que el ultimo byte 0x00 en la decodificación esta puesto de relleno, pero no forma parte de la secuencia binaria original. Por ejemplo, se tiene la última secuencia de 4 caracteres imprimibles **fZ8A=** (siempre que hay un símbolo "=", el carácter inmediato anterior es una letra "A"

6) Se agrupa la secuencia de 4 caracteres imprimibles y se reemplaza cada carácter por su índice en la tabla

fZ8A			
31	25	60	00
011111	011001	111100	000000

7) Se agrupan los números binarios de los 4 índices formando una secuencia de 24 bits

011111	011001	111100	000000
011111-011001-111100-000000			

8) Se divide la secuencia de 24 bits en 3 grupos de 8 bits

01111101-10011111-00000000		
01111101	10011111	00000000
0x7D	0x9F	0x00

9) De esta manera, la secuencia de 4 caracteres imprimibles fZ8A se decodifica a la secuencia binaria 0x7D, 0x9F, 0x00, pero como el carácter siguiente es un símbolo "=" (la secuencia de caracteres era fZ8A=), significa que hay que descartar el último byte 0x00 por no pertenecer a la secuencia original. En consecuencia, la cadena **fZ8A=** se decodifica a la secuencia binaria **0x7D, 0x9F**.

10) También puede darse el caso que a continuación de la última secuencia de 4 caracteres imprimibles haya dos símbolos "=", esto indica que los dos últimos bytes 0x00 en la decodificación están puestos de relleno, pero no forma parte de la secuencia binaria original. Por ejemplo, se tiene la secuencia final de caracteres imprimibles **NgAA==** (siempre que hay dos símbolos "=", los dos carácteres inmediatos anteriores son dos letras "A"

11) Se agrupa la secuencia de 4 caracteres imprimibles y se reemplaza cada carácter por su índice en la tabla

NgAA			
13	32	00	00
001101	100000	000000	000000

12) Se agrupan los números binarios de los 4 índices formando una secuencia de 24 bits

001101	100000	000000	000000
001101-100000-000000-000000			

13) Se divide la secuencia de 24 bits en 3 grupos de 8 bits

00110110-00000000-00000000		
00110110	00000000	00000000
0x36	0x00	0x00

14) De esta manera, la secuencia de 4 caracteres imprimibles NgAA se decodifica a la secuencia binaria 0x36, 0x00, 0x00, pero como los carácteres siguientes son dos símbolos "=" (la secuencia final de caracteres era NgAA==), significa que hay que descartar los dos últimos bytes 0x00 por no pertenecer a la secuencia original. En consecuencia, la cadena **NgAA==** se decodifica al byte binario **0x36**.

Enunciado del Trabajo Práctico

Para grupos de 1 o 2 Alumnos:
Implementar en C y assembler Intel x86_64 un programa que codifique una secuencia binaria de tamaño arbitrario que se encuentra en un archivo "inputBinario.bin" y genere como salida el archivo codificado "outputTexto.txt" con la codificación pedida.
Se debe implementar en C el manejo de archivos (apertura, lectura, escritura, cierre) y se debe implementar en assembler la codificación.

Para grupos de 3 o 4 Alumnos:
Implementar en C y assembler Intel x86_64 un programa que cumpla el enunciado anterior y que además decodifique una secuencia de caracteres imprimibles de tamaño arbitrario que se encuentra en un archivo "inputTexto.txt" y genere como salida el archivo codificado "outputBinario.txt" con la decodificación pedida.
Se debe implementar en C el manejo de archivos (apertura, lectura, escritura, cierre) y se debe implementar en assembler la codificación.

Recomendaciones:
Implementar la codificación/decodificación:

- En funciones implementadas en assembler invocadas desde C en archivos .ASM separados del archivo principal .C
- Las funciones en assembler pueden recibir como parámetros un puntero al comienzo de los datos de entrada y un unsigned int con la cantidad de datos de entrada
- Es posible usar funciones de C para saber la cantidad de datos en el archivo de entrada, reservar memoria dinámica del tamaño apropiado para la entrada y la salida, realizar la lectura e invocar a la función de assembler correspondiente.

Entregables:
Se debe subir al campus un archivo comprimido conteniendo todo el código fuente (c y assembler), las instrucciones de compilación para generar el archivo ejecutable y un archivo de codificación (y uno de decodificación si corresponde) con el que se pueda probar el software.

Calificación
La funcionalidad mínima para aprobar en grupos de 1 o 2 alumnos es que el programa codifique un archivo binario de contenido arbitrario de 189 Bytes (tamaño fijo y divisible por 3). Esta Funcionalidad genera una calificación de 4 (cuatro)

La funcionalidad mínima para aprobar en grupos de 3 o 4 alumnos es que el programa implemente la funcionalidad anterior y además decodifique un archivo de texto de contenido arbitrario de 252 bytes (tamaño fijo y divisible por 4). Esta funcionalidad genera una calificación de 4 (cuatro)

La corrección se realizará probando los archivos de codificación (y decodificación si corresponde) subidos al campus y archivos de prueba de las mismas características generados por los docentes. El software debe poder procesar correctamente ambos tipos de archivos.