A **Apellidos** _____ **Nombre** _____ **DNI**_____

Examen de Arquitectura de Computadores (Telemática) **Convocatoria de Julio: 8 – 07 – 2011**

Se dispone de un programa escrito en ensamblador encargado de encriptar cadenas de caracteres utilizando sustituciones monoalfabéticas. Éste es uno de los sistemas de encriptación más sencillos que existen y su uso se remonta a la época de Julio César. Su funcionamiento consiste en sustituir una letra por otra realizando una suma. El caso más sencillo sería sumando 1, con lo que cada letra del abecedario sería sustituida por la *siguiente* letra, es decir, la A se sustituiría por la B, la B por la C, y así sucesivamente hasta llegar a la Z, que se sustituiría por la A. Se puede utilizar cualquier número para realizar las sumas, siendo éste considerado como la *clave* de la encriptación. Dicho número sería necesario para desencriptar el texto, que se utilizaría para restar a cada una de las letras y así reconstruir el texto original.

El programa mostrado en la página siguiente utiliza este método para codificar cadenas de caracteres (no se proporciona el método para desencriptar). Dicho programa sustituye sólo las letras del código ASCII, es decir, letras mayúsculas y minúsculas de la A a la Z. Cualquier otro carácter permanecerá inalterado, como por ejemplo el espacio, los signos de puntuación, la letra ñ (mayúscula y minúscula) y las vocales con tilde. Además, las letras minúsculas son sustituidas por letras minúsculas y las mayúsculas por mayúsculas. Por ejemplo, a partir de la frase:

```
¡En esta casa obedecemos las leyes de la termodinámica!
```

Con una clave 3, la frase resultante sería:

```
¡Hq hvwd fdvd rehghfhprv odv ohbhv gh od whuprglqáplfd!
```

Cada letra del código ASCII ha sido sustituida por la que ocupa 3 sitios más adelante dentro del abecedario (respetando mayúsculas y minúsculas) salvo la letra *y* de la palabra *leyes*. En los casos en los que el abecedario "se acabe", se vuelve a empezar por la letra a, de tal modo que tres posiciones por delante de la letra *y* está la letra *b* (y → z → a → b).

El programa utiliza dos procedimientos, ***Codificar*** y ***CodigoCaracter*** descritos a continuación.

- Procedimiento ***Codificar***: Recibe dos parámetros, la cadena a codificar y la clave. El primer parámetro (en orden de apilación) es la dirección de la cadena a codificar y el segundo la clave. Ambos parámetros son de 32 bits. Este procedimiento no devuelve ningún valor y transforma la cadena de caracteres que se le ha pasado del modo descrito anteriormente. La transformación se realiza mediante un bucle que trabaja carácter a carácter. Dicho bucle comprueba si el carácter actual es el ASCII 0 en cada iteración. En caso de serlo, se habría llegado al final de la cadena y el bucle termina. En caso contrario transforma el carácter. En primer lugar determina si el carácter a procesar en la iteración actual es una letra mayúscula, una letra minúscula u otro carácter distinto (esto lo realiza mediante una llamada a un procedimiento auxiliar llamado ***CodigoCaracter***). Si el procedimiento ***CodigoCaracter*** determina que el carácter no es una letra, el bucle continúa sin hacer ninguna transformación. En caso contrario, suma al código ASCII del carácter la clave que se ha pasado como parámetro. Una vez hecho esto y antes de pasar a la siguiente iteración del bucle, se ha de comprobar que el código ASCII obtenido no es superior al código ASCII de la 'z' o 'Z'. Para ello, el procedimiento comprueba si el carácter es mayúscula y el ASCII resultante de la suma es superior a 122 o si el carácter es minúscula y el ASCII resultante de la suma es superior a 90. Si se cumple una de estas dos condiciones, al ASCII resultante se le ha de restar 26 (el numero de letras del abecedario).

- Procedimiento ***CodigoCaracter***. Recibe un único parámetro. Dicho parámetro debe ser un código ASCII. Aunque los caracteres se codifiquen con 1 BYTE, éstos deben pasarse al procedimiento ***CodigoCaracter*** en 32 bits y por valor. Este procedimiento comprueba si el valor pasado como parámetro está comprendido en el intervalo [65-90] o en el intervalo [97-122]. En el primer caso, el código ASCII se correspondería con una letra mayúscula, mientras que en el segundo caso se correspondería con una letra minúscula. Si el código ASCII pasado por parámetro no se encuentra dentro de ninguno de estos dos intervalos, es porque éste no se corresponde con el código de una letra. El procedimiento devuelve uno de los siguientes valores en el registro EAX:
  - 1: Si el código ASCII pasado por parámetro es de una letra mayúscula.
  - 2: Si el código ASCII pasado por parámetro es de una letra minúscula.
  - 3: Si el código ASCII pasado por parámetro no es de una letra.

Información adicional: Al inicio de la ejecución del programa el contenido del registro ESP es **00 12 FF C4h**.

```
.386
.MODEL FLAT, stdcall

ExitProcess PROTO, :DWORD

.DATA

 frase  DB "¡En esta casa obedecemos las leyes de la
           termodinámica!", 0

.CODE

Codificar PROC
       push ebp
       mov  ebp, esp

       push esi
       push ebx
```

```
      (--1--) ;Lectura de parámetros. ESI=Dirección Cadena
              ;                       EBX=Clave


bucle:
      cmp  BYTE PTR [esi], 0    ;Si el carácter es el ASCII 0
      je   salir                ;hemos llegado al final de la cadena


      (--2--)     ;Llamar al procedimiento CodigoCaracter
                  ;pasándole por parámetro el carácter actual
                  ;apuntado por esi.

      cmp  eax, 3
      je  finBucle
      ;Si llego aquí, el caracter es una letra

      add  [esi], bl      ;Sumamos la clave al ASCII de la letra
      cmp  eax, 1
      jne  es_min

      ;Si llego aquí, el caracter es mayúscula
      ;Comprobar que el ASCII resultante sigue siendo una mayúscula
      ;(es decir, que el código ASCII no sea superior a 90)
      ;Si lo es, restar 26
      (--3--)
      jmp finBucle
es_min:

      ;Si llego aquí, el caracter es minúscula
      ;Comprobar que el ASCII resultante sigue siendo una minúscula
      ;(es decir, que el código ASCII no sea superior a 122)
      ;Si lo es, restar 26

      No se muestran estas instrucciones

finBucle:
      inc esi
      jmp bucle
salir:
      pop  ebx
      pop  esi
      pop  ebp
      ret  8
Codificar ENDP
CodigoCaracter PROC
      push ebp
      mov  ebp, esp

♠♠♠   push ebx
♦♦♦   mov  ebx, [ebp+8]

      ;Comprobar si el código ASCII está en el intervalo [65-90]
      (--4--)

      ;Código a ejecutar si está en dicho intervalo
      (--4--)

no_mayuscula:

      ;Comprobar si el código ASCII está en el intervalo [97-122]
      No se muestran estas instrucciones

      ;Código a ejecutar si está en dicho intervalo
      No se muestran estas instrucciones

no_minuscula:
      ;Si llega aquí, no es mayúscula ni minúscula
      mov  eax, 3
final:
      pop  ebx
      pop  ebp
      ret  4
CodigoCaracter ENDP


inicio:

      push OFFSET frase
      push 3
      call Codificar

      push 0
      call ExitProcess

END inicio
```

A **Apellidos** _____ **Nombre** _____ **DNI** _____

Examen de Arquitectura de Computadores (Telemática) **Convocatoria de Julio: 8 – 07 – 2011**

— Escribe las instrucciones que hacen falta en el hueco (--**1**--) para realizar la lectura de los parámetros siguiendo las indicaciones de los comentarios presentes en el código. **0,5**

```
mov  esi, [ebp+12]
mov  ebx, [ebp+8]
```

— Escribe las instrucciones necesarias en el hueco (--**2**--) para llamar al procedimiento *CodigoCaracter* pasándole por parámetro el carácter correspondiente de la frase. **0,5**

```
mov   al, [esi]
movzx eax, al
push  eax
call  CodigoCaracter
```

— Escribe las instrucciones que harían falta en el hueco (--**3**--) para realizar la comprobación de que el código ASCII resultante tras sumar la clave no sea superior al código ASCII de la Z mayúscula (ASCII 90). En caso de que sea así, debe restarse 26 a dicho código ASCII para volver a empezar por la 'A'. **0,5**

```
cmp  BYTE PTR [esi], 90
jbe  finBucle
sub  BYTE PTR [esi], 26
```

— Escribe las instrucciones necesarias que harían falta en el hueco (--**4**--) para comprobar si el ASCII pasado como parámetro al procedimiento *CodigoCaracter* se corresponde con una letra mayúscula (está en el intervalo [65-90]). En caso de que así sea, escribir las líneas de código necesarias para que el procedimiento funcione como se especifica en el enunciado. **0,75**

```
;Comprobar si el ASCII está en el intervalo [65-90]
 cmp  ebx, 65
 jb   no_mayuscula
 cmp  ebx, 90
 ja   no_mayuscula

 ;Código a ejecutar si está en dicho intervalo
 mov  eax, 1
 jmp final
```

— Determina el valor del registro ESP **después** de ejecutar la instrucción `push ebx` del procedimiento *CodigoCaracter*, marcada con el símbolo ♠♠♠. Contestar en hexadecimal. **0,5**

```
00 12 FF 9C
```

— Codifica la instrucción `mov ebx, [ebp+8]` del listado, marcada con el símbolo ♦♦♦. Contestar en hexadecimal. **0,5**

```
8B 5D 08
```

— Define la TLB indicando claramente su objetivo **0,5**

```
Es una memoria de alta velocidad totalmente integrada con
la MMU, que almacena las entradas de la tabla de páginas
más frecuentemente utilizadas. Así las direcciones
virtuales pueden ser traducidas sin necesidad de acceder
a la memoria física. Un mecanismo hardware permite
actualizar dinámicamente el TLB con las entradas de la
tabla de páginas más frecuentemente utilizadas.
```

— Contesta a las siguientes preguntas breves: **0,5**

```
¿Por qué resulta muy problemático introducir cambios
en la arquitectura de un determinado modelo de
computador?

Porque los programas realizados para la arquitectura
original dejarían de funcionar en la arquitectura
modificada.



Escribe una secuencia de instrucciones que envíen el dato
50h al puerto 2000h de E/S.

mov dx, 2000h
mov al, 50h
out dx, al
```

A

— Indica cuáles son los cuatro elementos fundamentales que forman un proceso (según su definición técnica) y describe brevemente cada uno de ellos.

0,5

Imagen binaria de un programa. Está formada por las instrucciones y datos del programa.

La pila. Área de memoria en la que se almacenan datos temporales.

La tabla de páginas. Traduce las direcciones virtuales generadas por el proceso en las direcciones físicas en la que se encuentra almacenado.

El PCB. Estructura utilizada por el sistema operativo para controlar la ejecución del proceso.

— Explica los pasos correspondientes al protocolo de comunicación que se establece entre el PIC y la CPU IA-32, para que ésta pueda identificar al periférico que solicita una interrupción. No dibujar la figura correspondiente, solamente indicar y explicar los pasos.

0,5

1) El periférico solicita interrupción al PIC a través de una línea INTX.

2) El PIC pasa la solicitud de interrupción a la CPU a través de la línea INTR.

3) Si la CPU acepta la interrupción, se lo indica al PIC a través de la línea INTA.

4) El PIC envía a la CPU el número de interrupción correspondiente al periférico que solicitó la interrupción. Cada línea INTX tiene asociado un número distinto, identificándose de esta forma a los diferentes periféricos. La asociación de números de interrupción a las líneas INTX es programable.

A continuación se muestra el código de un programa escrito en C que intercambia el valor de dos variables globales utilizando una variable local de forma auxiliar.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>

int A, B;

int main()
{
	int local;

	A = 10;		//Instrucción 1
	B = 20;

	local = A;	//Instrucción 2
	A = B;
	B = local;

	return 0;
}
```

— ¿Qué instrucciones generaría un compilador de C para traducir a ensamblador la instrucción 1?

0,25

```
mov [A], 10
```

— ¿Qué instrucciones generaría un compilador de C para traducir a ensamblador la instrucción 2?

0,5

```
mov eax, [A]
mov [ebp-4], eax
```

A **Apellidos** _____ **Nombre** _____ **DNI** _____

Examen de Arquitectura de Computadores (Telemática) **Convocatoria de Julio: 8 – 07 – 2011**

A continuación se muestra un programa escrito en C encargado de comprobar el nivel de batería de un ordenador portátil. Éste pretende dar un aviso al usuario si la batería del portátil está por debajo del 5% de carga. Esta información la proporciona la función de WIN32 `GetSystemPowerStatus` mediante una estructura de tipo `SYSTEM_POWER_STATUS`. Completa los huecos según lo indicado en los comentarios adyacentes a ellos.

La especificación tanto de la función como de la estructura se proporciona en el anexo.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>

int main()
{
	SYSTEM_POWER_STATUS sps;
	BOOL resultado;
	BYTE flags;

	//Llama a la función GetSystemPowerStatus para obtener
	//información sobre la carga y el estado de la batería
	resultado = GetSystemPowerStatus(&sps);          0,5


	//Si la función falló, dar un mensaje de error

	if ( resultado==0                                0,25 )
	{
		printf("La función ha fallado\n");
	}

	//Dar un mensaje de aviso SÓLO cuando el nivel de carga de
	//la batería sea inferior al 5%
	                                                 0,75
	flags = sps.BatteryFlag & 4;

	if (flags!=0                              )
	{
		printf("Atención: La batería está a punto de
			agotarse!\n");
	}
	return 0;
}
```

Un computador utiliza un sistema de memoria virtual con las siguientes características:

Direcciones virtuales: 20 bits

Direcciones físicas: 16 bits

Tamaño de página: 512 bytes (cada posición de memoria alberga un byte).

Se está ejecutando en este computador un proceso que consta de 1 página para la sección de datos (D1), 2 páginas para la sección de pila (P1 y P2) y 2 páginas para el código (C1, C2). Se sabe que la memoria física instalada en este computador sólo ocupa el 25% del espacio de direcciones físico. Las secciones de datos y código crecen desde direcciones menos significativas a direcciones más significativas. Sin embargo la pila crece desde direcciones más significativas a direcciones menos significativas.

A continuación se muestra la tabla de páginas (incompleta) en un momento dado durante la ejecución del proceso descrito anteriormente. Completa la tabla sabiendo que la pila se ubica siempre comenzando por la dirección virtual más significativa del espacio de direcciones virtual y que la primera página de la sección de pila se ubicó en la página más significativa de la memoria física.

Tabla de Páginas

	Nº Pag. Virtual (Hex)	Pre-sencia	Nº Pag. Fis. (Hex) / Offset Dis.
C1	200	Si	05
C2	201	No	Offset X
D1	500	Si	10
P2	7FE	Si	09
P1	7FF	Si	1F

1

— ¿Qué rango de direcciones virtuales es utilizado por la sección de código de este programa? Contestar en hexadecimal.

4 00 00 – 4 03 FF 0,5

— En la sección de datos del programa está declarado en primer lugar un vector de 64 enteros (cada entero ocupa 4 bytes) y a continuación una variable de tipo entero (el resto de la página queda sin utilizar). Determina el rango de direcciones virtuales y físicas ocupado por dicha variable de tipo entero. Contestar en hexadecimal.

Direcciones virtuales: **A 01 00 – A 01 03** 1

Direcciones físicas: **21 00 – 21 03**

GetSystemPowerStatus

La función **GetSystemPowerStatus** proporciona información del estado de la energía del sistema. El estado indica si el sistema está funcionando con corriente continua o alterna, si la batería está actualmente cargándose y cuanta batería queda.

```
BOOL GetSystemPowerStatus (

  LPSYSTEM_POWER_STATUS lpSystemPowerStatus

);
```

Parámetros

lpSystemPowerStatus
[out] Puntero a una estructura de tipo SYSTEM_POWER_STATUS que recibirá la información sobre la energía del sistema.

Valores de retorno
Si la función tiene éxito, el valor de retorno es distinto de cero.
Si la función falla, el valor de retorno es cero.

Estructura SYSTEM_POWER_STATUS

La estructura **SYSTEM_POWER_STATUS** contiene la información sobre el estado de la energía del sistema.

```
typedef struct _SYSTEM_POWER_STATUS {
  BYTE ACLineStatus;
  BYTE BatteryFlag;
  BYTE BatteryLifePercent;
  BYTE Reserved1;
  DWORD BatteryLifeTime;
  DWORD BatteryFullLifeTime;
} SYSTEM_POWER_STATUS,
 *LPSYSTEM_POWER_STATUS;
```

Miembros

BatteryFlag
El estado de la carga de la batería. Este miembro puede tener activo uno o varios de los siguientes flags:

- 1: Alto: La carga de la batería es superior al 66%.
- 2: Bajo: La carga de la batería es inferior al 33%.
- 4: Crítico: La carga de la batería es inferior al 5%.
- 8: Cargando la batería.
- 128: El sistema no tiene batería.
- 255: Estado desconocido.

Nota: No se proporciona la especificación del resto de miembros de esta estructura.