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

Examen de Arquitectura de Computadores (Telemática) **Convocatoria de Junio: 2 – 06 – 2010**

El código mostrado a continuación pretende determinar el valor numérico de una cadena de caracteres en la que se encuentra un número escrito en hexadecimal. En la sección de datos se ha declarado una variable de nombre *numero* cuyo contenido es "DB3", cuyo valor numérico expresado en decimal es 3507. La cadena puede tener cualquier tamaño, pero es indispensable que termine con un ASCII 0, como en el ejemplo. Además, las letras deben estar escritas en mayúsculas. La variable global *valor* será donde se guarde el valor numérico de la cadena una vez terminado el cálculo. El programa principal, únicamente tiene que llamar a la función *calcula* y guardar en *valor* el resultado devuelto por ésta. Los procedimientos escritos para realizar este cálculo se describen a continuación:

- Calcula: Recibe como único parámetro la **dirección** de comienzo de la cadena que contiene el número hexadecimal. Como la cadena puede tener cualquier longitud, hace un recorrido inicial sobre ella para determinar el número de dígitos que contiene. Para ello itera sobre la cadena hasta encontrar un ASCII 0. Una vez determinado el número de dígitos que tiene la cadena, se vuelve a iterar sobre ella desde el principio. Mediante un bucle controlado por contador, utilizando la instrucción *loop*, se calcula el valor numérico de cada dígito de la cadena. Este valor es calculado por el procedimiento *valorHex*, que lo devuelve en el registro EAX. Sumando el valor correspondiente a cada dígito de la cadena se obtiene el valor de la cadena completa. El registro encargado de guardar esta suma es EDX. El procedimiento *Calcula* retorna el valor numérico de la cadena en el registro EAX salvo que ésta contenga algún carácter no válido, en cuyo caso retornaría -1.

- valorHex: Calcula el valor numérico de **un** dígito hexadecimal (es decir, un caracter que ha de ser o bien un número 0-9 o bien una letra mayúscula A-F). Para poder realizar este cálculo, también se tiene que disponer de la **posición** del dígito. El valor de la cadena de ejemplo, **DB3**, se hallaría del siguiente modo: valor = $13x16^{2} + 11x16^{1} + 3x16^{0}$ (D=13, B=11). El procedimimento *valorHex* recibe dos parámetros **por valor y en 32 bits**. Éstos son (en orden de apilación) el código ASCII del caracter a procesar y la posición que ocupa en la cadena (empezando a contar por 1 de derecha a izquierda). Este procedimiento utiliza una variable local para realizar los cálculos. Si el código ASCII pasado como parámetro no se corresponde con el de un dígito o una letra mayúscula comprendida en el rango A-F, devuelve -1. En caso contrario, devuelve el valor asociado a ese dígito, que será: ASCII x $16^{(posición - 1)}$.

- potencia: Eleva una base a un exponente. Recibe dos parámetros **por valor y en 32 bits**. El primer parámetro en orden de apilación es la base de la potencia, y el segundo el exponente. Retorna en el registro EAX el resultado de $base^{exponente}$.

- multiplica: Multiplica dos números enteros. Recibe dos parámetros **por valor y en 32 bits** correspondientes a los dos factores de la multiplicación. Retorna en el registro EAX el producto de los dos factores pasados por parámetro.

Se sabe que la sección de datos comienza en la dirección **0x00404000**. Al comienzo de la ejecución del programa, el registro ESP contiene la dirección **0x0012FFC4**. El registro ESP debe contener esta misma dirección al finalizar la ejecución.

```
.386
.MODEL FLAT, stdcall

ExitProcess PROTO, :DWORD

.DATA
      numero DB "DB3", 0
      valor  DD 0

.CODE

multiplica PROC
      No se muestran estas instrucciones

      ret 8
multiplica ENDP

potencia PROC
      No se muestran estas instrucciones

      ret 8
potencia ENDP

valorHex PROC
      push  ebp
      mov   ebp, esp
      sub   esp, 4 ;Variable local auxiliar

      push  ebx ;Guardará el caracter ASCII
♠♠♠   push  edx ;Guardará la posición del dígito

      mov   ebx, [ebp+12]
      mov   edx, [ebp+8]
      dec   edx

      cmp   ebx, '0'
      jb    nonumero
      cmp   ebx, '9'
      ja    nonumero
      ; Si no saltó, es un número
```

```
       mov   [ebp-4], ebx
       sub   [ebp-4], DWORD PTR '0' ;Vble Local: Valor del dígito


       (--1--)


       jmp   final
nonumero:
       cmp   ebx, 'A'
       jb    novalido
       cmp   ebx, 'F'
       ja    novalido
       ; Si llegó hasta aquí, es una letra

       mov   [ebp-4], ebx
       sub   DWORD PTR [ebp-4], 'A'
       add   DWORD PTR [ebp-4], 10 ;Vble Local = Valor de la letra
```

Mismo bloque de instrucciones que (- - 1 - -)

```
       jmp   final

novalido: ;Si salta a esta etiqueta, es que el código ASCII que se
          ;pasó por parámetro no es ni un dígito ni una letra A-F
       mov   eax, -1
final:


       (--3--)


valorHex ENDP

Calcula PROC
       push  ebp
       mov   ebp, esp

       push  edx ;Se utilizará para ir sumando el valor de cada
                 ;caracter de la cadena
       push  ecx ;Guardará el número de caracteres de la cadena
       push  esi ;Guardará el parámetro

       mov   esi, [ebp+8] ;Lectura del parámetro


       xor   edx, edx ;Inicializar a cero ecx y edx
       xor   ecx, ecx

contar:


       (--4--)


fincontar:

       mov   esi, [ebp+8]   ;Nos situamos de nuevo al comienzo
                            ;de la cadena

bucle:
       movzx ebx, BYTE PTR [esi] ;Guardar el ebx el ASCII del
                                 ;caracter procesado en esta iteración
       push  ebx
       push  ecx
       call  valorHex ;Calcular su valor numérico
       cmp   eax, -1
       je    final ;Dígito no válido -> mantengo eax = -1

       add   edx, eax ;Sumar a edx el valor del caracter actual

       inc   esi  ;Dejar esi preparado para que apunte al siguiente
       loop  bucle
       mov   eax, edx;Fin del procesamiento.Dejar el resultado en eax

final:
       pop   esi
       pop   ecx
       pop   edx
       pop   ebp
       ret
Calcula ENDP

inicio:


       (--2--)

       push  0
       call  ExitProcess
END inicio
```

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

Examen de Arquitectura de Computadores (Telemática) **Convocatoria de Junio: 2 – 06 – 2010**

— Escribe las instrucciones necesarias del hueco (--**1**--), donde se calcula el valor correspondiente a un carácter de la cadena mediante la expresión: $16^{posicion} * valor$.

0,75

```
push  16
push  edx
call  potencia
push  eax
push  [ebp-4]
call  multiplica
```

— Escribe el programa principal que debería aparecer en el hueco (--**2**--).

0,5

```
push  OFFSET numero
call  Calcula
add   esp, 4
mov   [valor], eax
```

— Escribe el conjunto de instrucciones que serían necesarias en la parte final del procedimiento *valorHex*, hueco (--**3**--).

0,5

```
pop  edx
pop  ebx
add  esp, 4
pop  ebp
ret  8
```

— Construye el bucle situado en el hueco (--**4**--), encargado de contar el número de caracteres que forman la cadena de entrada (el terminador ASCII 0 **NO** se cuenta). Utiliza las etiquetas *contar* y *fincontar* definidas en el código. El número de caracteres tiene que guardarse en el registro ECX.

0,75

```
cmp  BYTE PTR [esi], 0
je   fincontar
inc  ecx
inc  esi
jmp  contar
```

— Determina el valor del registro ESP **después** de que se ejecute por primera vez la instrucción marcada en el código con el símbolo ♠♠♠, en la función *valorHex*. Responder en hexadecimal.

0,5

```
00 12 FF 90
```

— Codifica la instrucción `mov [esi+256], al`. Responder en hexadecimal.

0,5

```
88 86 00 01 00 00
```

— En la definición técnica de un proceso se indica cuáles son los cuatro elementos fundamentales que lo forman. Indica a continuación dichos elementos y describe brevemente cada uno de ellos.

0,5

Imagen binaria de un programa. Está formada por las instrucciones y datos globales 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 las que se encuentran almacenadas.

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

— Responde a las siguientes preguntas.

0,5

¿Cuáles son los cuatro componentes típicos del núcleo de un sistema operativo moderno?

Gestor de procesos, gestor de memoria, gestor de ficheros y gestor de entrada-salida.

En el ámbito de los sistemas operativos, ¿qué se entiende por multitarea?

Que se permite la ejecución "concurrente" de múltiples procesos.

¿y por multiusuario?

Que se permite la interacción de varios usuarios simultáneamente con el sistema.

A

— Contesta a las siguientes preguntas breves:

¿Cuáles son las causas que pueden provocar una transferencia de control desde un programa de usuario al sistema operativo? **1**

Interrupciones, excepciones y llamadas al sistema.

Escribe las instrucciones necesarias para leer un byte del puerto 6060h del espacio de direcciones de E/S y almacenarlo en una variable global (definida en la sección de datos) de tipo byte llamada *dato*.

```
mov dx, 6060h
in al, dx
mov [dato], al
```

Escribe las instrucciones necesarias para escribir el contenido de una variable global (definida en la seccioón de datos) de tipo byte llamada *dato*, en el puerto 20h del espacio de direcciones de E/S.

```
mov al, [dato]
out 20h, al
```

¿Qué dos tipos de interrupciones existen en la arquitectura IA-32?

Enmascarables y no enmascarables.

En un determinado instante, un computador ejecuta un proceso cuya tabla de páginas se muestra a continuación (sólo se muestran las entradas de la tabla de páginas correspondientes a páginas utilizadas en ese momento). En este computador, las direcciones virtuales son de 32 bits, y las posiciones de memoria albergan un byte. El programa en ejecución utiliza dos páginas para el código, dos para los datos y una para la pila. El código comienza en la página virtual 0x03000, los datos en la página 0x21000 y la pila en la página 0x7FFFF. También se sabe que la pila siempre empieza en la última página virtual.

— ¿Cuántos bits utiliza este computador para codificar la página virtual y desplazamiento en sus direcciones virtuales?

Pag. Virtual: 19 Desplazamiento: 13 **0,75**

Tabla de Páginas

Nº Pag. Virtual (Hex)	Pre-sencia	Nº Pag. Fis. (Hex) / Offset Dis.
03000	Si	0BB
03001	Si	0FF
21000	No	Offset Y
21001	Si	015
7FFFF	No	Offset X

— Teniendo en cuenta que se utilizan 11 bits para direccionar las páginas físicas, y que la segunda página de la sección de código se ubicó en el último marco de la memoria principal, ¿qué porcentaje del espacio de direcciones físico puede ser utilizado?

12,5 % **0,75**

— ¿Cuál sería la dirección virtual más significativa que se podría utilizar este caso? Responder en hexadecimal.

FF FF FF FF **0,25**

— ¿Cuál sería la dirección física más significativa que se podría utilizar este caso? Responder en hexadecimal.

1F FF FF **0,25**

Se desea escribir un programa en C que muestre por consola cuál fue el último día que se escribió en un fichero. Para ello se utilizarán unas funciones de WIN32 cuya especificación se encuentra en el Anexo, en las últimas páginas del examen.

En la siguiente página hay un código incompleto que se debe rellenar de tal modo que la aplicación compile sin errores, teniendo en cuenta la especificación de las funciones *GetFileInformationByHandle* y *FileTimeToSystemTime*. Aunque la primera de esas dos funciones ya proporcione el instante de tiempo en que fue modificado un fichero por última vez, haremos uso de la función *FileTimeToSystemTime* para convertir ese dato en una estructura SYSTEMTIME, que es más sencilla de representar (su especificación también aparece en el Anexo).

Nota: No se pueden declarar más variables. Sólo se pueden utilizar las que ya están declaradas en el programa.

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

Examen de Arquitectura de Computadores (Telemática) **Convocatoria de Junio: 2 – 06 – 2010**

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

void mostrarError(){
	printf("Se ha producido un error: %d\n", GetLastError());
	getch();
	exit(-1);
}

int main ()
{
	BOOL resultado;
	BY_HANDLE_FILE_INFORMATION info;
	HANDLE fh;
	SYSTEMTIME time;

	 //Obtenemos un manejador del fichero "fichero.txt"
	fh = CreateFile("c:\\fichero.txt",
		GENERIC_READ, FILE_SHARE_WRITE, 0, OPEN_EXISTING, 0, 0);

	//Obtener la información del fichero utilizando
	//la función GetFileInformationByHandle

	resultado = GetFileInformationByHandle(fh, &info);          0,5



	//Si se produjo un error, llamo a la función mostrarError

	if (resultado == 0)                                         0,5
	{
		mostrarError();
	}

	//Convertir a SYSTEMTIME el instante del último acceso
	//guardado en el campo correspondiente de la estructura info

	resultado = FileTimeToSystemTime(&(info.ftLastWriteTime),   0,5
				&time);



	//Si se produjo un error, llamo a la función mostrarError
	if ( NO SE MUESTRA ESTA CONDICION )
	{
		mostrarError();
	}
	//Mostrar por pantalla el dia y el mes del último acceso
	//utilizando la estructura time

	printf("Ultimo acceso el dia %d del mes %d\n",              0,5
		time.wDay,time.wMonth);


	getch();
	return 0;
}
```

A continuación se muestra el listado de un programa C muy simple. En este programa se define la función *esImpar()*, que recibe un número entero sin signo como parámetro y devuelve como resultado un 1, si el número que recibe como parámetro es impar y un 0, si no lo es. La función *main()* llama a la función *esImpar()* para determinar si la variable global A es impar y almacena el resultado en la variable global B.

```
unsigned int esImpar( unsigned int );  // Prototipo

unsigned int A=5, B=0;

int main()
{
	B = esImpar( A );  ♦♦♦
	return 0;
}

unsigned int esImpar( unsigned int r )
{
	unsigned int aux;
	aux = r & 1;
	return aux;
}
```

— Escribe el conjunto de instrucciones ensamblador que generaría el compilador de C al compilar la sentencia marcada con el símbolo ♦♦♦ en el listado.

```
push [A]
call esImpar
add  esp, 4
mov  [B], eax
```

0,5

GetFileInformationByHandle

La función **GetFileInformationByHandle** proporciona información del fichero especificado.

```
BOOL GetFileInformationByHandle(

  HANDLE hFile,

  LPBY_HANDLE_FILE_INFORMATION lpFileInformation

);
```

Parámetros

hFile
[in] Manejador del fichero del que se va a obtener la información.
Éste no debe ser un manejador de tubería. La función **GetFileInformationByHandle** no funciona con manejadores de tubería.

lpFileInformation
[out] Puntero a una estructura de tipo **BY_HANDLE_FILE_INFORMATION** que recibe la información del fichero.

Valores de retorno
Si la función tiene éxito, el valor de retorno es distinto de cero. Para conseguir más información sobre el error, llama a la función **GetLastError**.

FileTimeToSystemTime

La función **FileTimeToSystemTime** convierte una variable FILETIME en una variable SYSTEMTIME.

```
BOOL FileToSystemTime(

  const FILETIME* lpFileTime,

  LPSYSTEMTIME lpSystemTime

);
```

Parámetros

lpFileTime
[in] Puntero la una estructura **FILETIME** que contiene el instante de tiempo que se desea convertir al formato **SYSTEMTIME**.

Este valor debe ser menor que 0x8000000000000000. En caso contrario, falla.

lpSystemTime
[out] Puntero a la estructura **SYSTEMTIME** que recibirá el instante de tiempo.

Valores de retorno

Si la función tiene éxito, el valor de retorno es distinto de cero. Para conseguir más información sobre el error, llama a la función **GetLastError**.

BY_HANDLE_FILE_INFORMATION

La estructura **BY_HANDLE_FILE_INFORMATION** contiene la información solicitada por la función **GetFileInformationByHandle**.

```
typedef struct _BY_HANDLE_FILE_INFORMATION {
  DWORD dwFileAttributes;
  FILETIME ftCreationTime;
  FILETIME ftLastAccessTime;
  FILETIME ftLastWriteTime;
  DWORD dwVolumeSerialNumber;
  DWORD nFileSizeHigh;
  DWORD nFileSizeLow;
  DWORD nNumberOfLinks;
  DWORD nFileIndexHigh;
  DWORD nFileIndexLow;

} BY_HANDLE_FILE_INFORMATION,
*PBY_HANDLE_FILE_INFORMATION;
```

Miembros

dwFileAttributes
Atributos del fichero.

ftCreationTime
Estructura de tipo **FILETIME** que especifica cuando se ha creado el fichero. Si el sistema de ficheros utilizado no da soporte al instante de creación de fichero, este miembro es 0 (cero).

ftLastAccessTime
Estructura de tipo **FILETIME**. Para un fichero, la estructura especifica el último instante en el que ha sido leído o escrito. Para un directorio, la estructura especifica cuando se ha creado. Tanto para ficheros como para directorios, la fecha especificada es correcta, pero la hora siempre se fija en media noche. Si el sistema de ficheros utilizado no da soporte al instante del último acceso, este miembro es 0 (cero).

ftLastWriteTime
Estructura de tipo **FILETIME**. Para un fichero, la estructura especifica el último instante en el que ha sido escrito. Para un directorio, la estructura especifica cuando se ha creado. Si el sistema de ficheros utilizado no da soporte al instante del último acceso, este miembro es 0 (cero).

dwVolumeSerialNumber
Número de serie del volúmen que contiene el fichero.

nFileSizeHigh
Parte alta del tamaño del fichero.

nFileSizeLow
Parte baja del tamaño del fichero.

nNumberOfLinks
Número de enlaces a este fichero. Para sistemas de ficheros FAT, este miembro es siempre 1. Para sistemas de ficheros NTFS, puede ser superior a 1.

nFileIndexHigh
Parte alta del identificador único asociado con el fichero. Para más información, ver **nFileIndexLow**.

nFileIndexLow
Parte baja del identificador único asociado con el fichero.

Este valor sólo es útil mientras el fichero está abierto por al menos un proceso. Si ningún proceso lo ha abierto, el índice puede cambiar la próxima vez que se abra el fichero.

SYSTEMTIME

La estructura **SYSTEMTIME** representa una fecha y una hora utilizando miembros individuales para el mes, día, año, día de la semana, hora, minuto, segundo y milisegundo.

```
typedef struct _SYSTEMTIME {
  WORD wYear;
  WORD wMonth;
  WORD wDayOfWeek;
  WORD wDay;
  WORD wHour;
  WORD wMinute;
  WORD wSecond;
  WORD wMilliseconds;

} SYSTEMTIME,
*PSYSTEMTIME;
```

Miembros

wYear
Año (1601 - 30827).

wMonth
Mes.

Enero = 1
Febrero = 2
Marzo = 3
Abril = 4
Mayo = 5
Junio = 6
Julio = 7
Agosto = 8
Septiembre = 9
Octubre = 10
Noviembre = 11
Diciembre = 12

wDayOfWeek
Día de la semana.

Domingo = 0
Lunes = 1
Martes = 2
Miércoles = 3
Jueves = 4
Viernes = 5
Sábado = 6

wDay
Día del mes (1-31).

wHour
Hora (0-23).

wMinute
Minuto (0-59).

wSecond
Segundo (0-59).

wMilliseconds
Milisegundo (0-999).