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

Examen de Arquitectura de Computadores. (Telemática.) **Parcial de diciembre: 17-12-2008**

```
.386
.MODEL FLAT, stdcall

ExitProcess PROTO, :DWORD

.DATA

num   DD 0
frase DB "Homer no funciona cerveza bien sin", 0

.CODE

cuentapalabras PROC
      push ebp  ♠♠♠
      mov ebp, esp

      ; Salvaguarda de registros
      push esi
      push edx

      ; Inicializamos el registro edx, utilizado como contador
      ; de palabras
      xor edx, edx

      ; En esi se guardará la dirección de comienzo del array
      ; pasada por parámetro
      mov esi, [ebp+8]


bucle:
      ; En cada iteración, el registro bl guardará el código ASCII
      ; del carácter apuntado por esi
      mov bl, [esi]

      ; Condición de parada: si el código leído es 0, terminamos
      cmp bl, 0
      je final

      ; Para saber si el código contenido en bl corresponde al del
      ; espacio en blanco, se lo paso como parámetro a la función
      ; esEspacio.
      ; El parámetro debe de ser de 32 bits
      (--1--)

      call esEspacio

      (--2--)

      inc esi
      jmp bucle
final:

      inc edx
      mov eax, edx

      ; Restauración de registros y retorno
      (--3--)
cuentapalabras ENDP

esEspacio PROC
      push ebp
      mov ebp, esp

      ; Salvaguarda de registros
      push ebx

      xor eax, eax
      mov ebx, [ebp+8]
      cmp ebx, 32
      jne salir
      mov eax, 1
salir:

      ; Restauración de registros y retorno
```

No se muestran estas instrucciones

```
esEspacio ENDP

inicio:

      push OFFSET frase
      call cuentapalabras

      mov [num], eax

      push 0
      call ExitProcess

END inicio
```

A

El listado anterior corresponde a un programa encargado de contar el número de palabras escritas en una cadena de caracteres. Para realizar esta tarea, se encargará de contar el número de espacios (código ASCII 32 decimal) presentes en la cadena. De este modo, el número total de palabras escritas corresponderá al número de espacios contados más uno (por ejemplo, en la frase escrita en el código fuente hay 5 espacios, y por tanto, 6 palabras).

Se ha escrito una función cuentapalabras que recibe como parámetro la dirección de comienzo de una lista. El último carácter de dicha lista deberá ser forzosamente un 0, que se utilizará como terminador. La función cuentapalabras retorna el número de palabras escritas en la cadena que se le pase como parámetro. Para ello, iterará mediante un bucle sobre cada una de las letras de la cadena recibida comparando su código ASCII con el del espacio en blanco. En caso de que estos códigos sean iguales se incrementará un contador de espacios (edx) y en caso contrario no se hará nada. Dicho bucle finalizará en el momento que se lea un ASCII 0. Como el número total de palabras se corresponde con el número de espacios más uno, antes de finalizar habrá que incrementar en una unidad el registro encargado de contar los espacios.

Para saber si un código ASCII es el del espacio en blanco, la función cuentapalabras hace uso de una función auxiliar llamada esEspacio. Esta función recibe un parámetro (que, como siempre, será de **32 bits**) que se corresponderá con un código ASCII. Si este código es el 32, la funcion esEspacio retornará un 1, y en caso contrario retornará un 0.

El programa principal realiza una invocación a la función cuentapalabras, pasándole como parámetro la dirección de comienzo de un array llamando *frase* declarado en la sección de datos del programa. Los elementos de dicho array son de tipo byte.

Además de este array, en la sección de datos también hay declarada una variable de tipo doble palabra llamada *num* que se utilizará para almacenar el número de palabras existentes en *frase*, valor retornado por la función cuentapalabras.

Nota: en la pila no se pueden meter datos de 1 byte

Datos adicionales Dirección de comienzo de la sección de datos: 00404000
Valor de ESP al inicio del programa: 0012FFC4

Contesta a las preguntas relativas a este programa que se indican a continuación.

— Escribe a continuación las instrucciones que faltan en el hueco (--**1**--) para realizar correctamente la llamada a la función esEspacio.

0,75

```
movzx ebx, bl
push ebx
```

— Escribe las instrucciones que faltan en el hueco (--**2**--). Para ello, utiliza sólo una etiqueta.

0,75

```
  cmp eax, 0
  je noEsEspacio
  inc edx
noEsEspacio:
```

— ¿Qué instrucciones harían falta en el hueco (--**3**--) para realizar correctamente la restauración de registros y el retorno de la función cuentapalabras?

0,5

```
pop edx
pop esi
pop ebp
ret 4
```

— Determina el tamaño expresado en bytes que alcanza la pila del programa cuando ésta se encuentra en su máxima expansión. Contesta en decimal.

0,5

```
36 bytes
```

— ¿Qué valor de 32 bits hay guardado en la pila a partir de la dirección 00 12 FF C0 en el instante en que se ejecuta la instrucción "push ebp" marcada en el programa con el símbolo ♠♠♠? Contesta en hexadecimal.

0,5

```
00 40 40 04
```

— Codifica la instrucción "`mov DWORD PTR [esi+15], 12`" (NOTA: esta instrucción no pertenece al programa)

0,75

```
C7 46 0F 0C 00 00 00
```

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

Examen de Arquitectura de Computadores. (Telemática.) **Parcial de diciembre: 17-12-2008**

A continuación se muestra una función escrita en C encargada de calcular potencias.

```
int potencia(int base, int exponente){

	int aux,i; // Instrucción 1

	i=exponente;
	aux=1;  //Instrucción 2

	while(i>0){
		aux=multiplica(aux, base); //Instrucción 3
		i=i-1;
	}

	return aux;
}
```

El objetivo de la función potencia es devolver *base* elevado a *exponente*. Para ello hace uso de otra función, llamada *multiplica* encargada de retornar el producto de dos números pasados como parámetro.

— Determina el código máquina que generaría un compilador de C al compilar las instrucciones que tienen los comentarios "Instrucción 1" y "Instrucción 2".

Nota: No incluir etiquetas en las instrucciones generadas.

1

```
Instrucción 1:
     sub esp, 8
Instrucción 2:
     mov [ebp-4], DWORD PTR 1
```

— Escribe a continuación las instrucciones que se generarían para codificar la instrucción con el comentario "Instrucción 3".

Nota: No incluir etiquetas en las instrucciones generadas.

1

```
push [ebp+8]
push [ebp-4]
call multiplica
add esp, 8
mov [ebp-4], eax
```

— Un procedimiento (escrito en ensamblador) recibe a través de la pila dos parámetros. El primero (en orden de apilación) se pasa por valor y el segundo **por referencia**. Además, el procedimiento reserva en la pila espacio para dos variables locales, cada una de 32 bits. En base a esto, escribe las instrucciones necesarias para sumar los dos parámetros y almacenar el resultado en la segunda variable local (es decir, la que ocupa las direcciones menos significativas).

0,75

```
Mov esi, [ebp+8]
mov edx, [ebp+12]
add edx, [esi]
mov [ebp-8], edx
```

— Explica los conceptos de localidad espacial y localidad temporal en el acceso a las instrucciones y datos de un programa.

0,5

Localidad espacial:

Cuando un programa accede a una instrucción o a un dato, existe una elevada probabilidad de que instrucciones o datos cercanos sean accedidos pronto.

Localidad temporal:

Cuando un programa accede a una instrucción o a un dato, existe una elevada probabilidad de que esa misma instrucción o dato vuelva a ser accedido pronto.

— ¿Cuántos niveles de privilegio define la arquitectura IA-32, y cuántos de ellos utiliza el sistema operativo Windows?

0,5

La arquitectura define 4 niveles de privilegio, de los cuales, sólo 2 son utilizados por Windows.

A

— Contesta a las siguientes preguntas breves:

1

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

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.

— Indica en cuál o cuáles de las siguientes instrucciones de ensamblador es necesario utilizar el operador PTR (BYTE PTR, WORD PTR o DWORD PTR). Si crees que no hace falta en ninguna, responde NINGUNA.

A) mov eax, 35h

B) mov [ebp-4], 35h

C) mov [ebp-4], eax

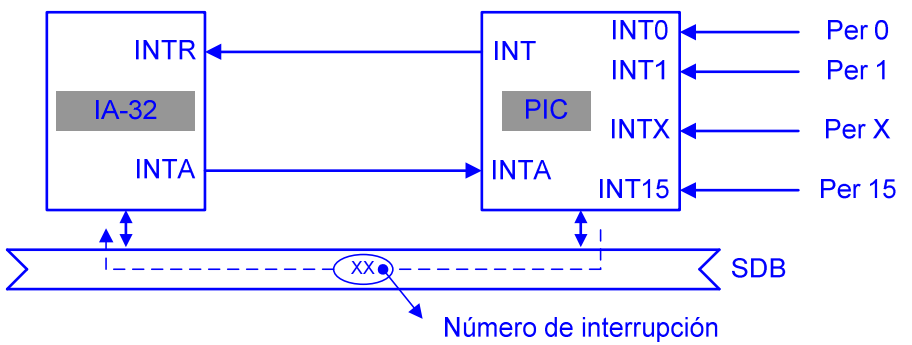
D) movsx eax, [ebx]

0,5

B, D

— Dibuja el esquema de gestión de interrupciones en la arquitectura IA-32 mediante el dispositivo PIC e indica los 4 primeros pasos llevados a cabo el PIC y el procesador para gestionar la petición de una interrupción por parte de un periférico.

1



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.