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

Examen de Arquitectura de Computadores (Telemática) **Parcial de diciembre: 9-12-2009**

```
.386
.MODEL flat, stdcall

ExitProcess PROTO, :DWORD

.DATA

      vector1      DD  3, 36,  8
      vector2      DD 38,  5, 14
      tamanio      DB  3

      resultado    DD 0

.CODE

ProdEsc PROC
      push ebp
      mov  ebp, esp
      sub  esp, 4         ;Reserva Vble local

      push ecx            ;Para guardar el tamaño de los vectores
      push esi            ;Para guardar la dirección de vector1
      push edi            ;Para guardar la dirección de vector2


      ;Lecura de parámetros e incialización de variable local a 0
      (--2--)


bucle:
      push [esi]
      push [edi]
      call multiplica
      add  [ebp-4], eax


      (--3--)
      loop bucle

      mov  eax, [ebp-4] ♠♠♠

      pop  edi
      pop  esi
      pop  ecx

      add  esp, 4
      pop  ebp
      ret
ProdEsc ENDP

multiplica PROC
      push  ebp
      mov   ebp, esp

      push  ecx           ;Para guardar un factor (parámetro)
      push  ebx           ;Para guardar el otro factor (parámetro)

      xor  eax, eax

      ;Lectura de parámetros del procedimiento multiplica
      No se muestran estas instrucciones
bucleM:
      add  eax, ebx
      loop bucleM

      pop  ebx
      pop  ecx
      pop  ebp
      ret  8
multiplica ENDP

inicio:
      (--1--)

      push  0
      call  ExitProcess

END inicio
```

El código mostrado en el cuadro anterior realiza el producto escalar de dos vectores de números enteros. Esta operación se define a continuación:

Supongamos que tenemos dos vectores de 3 componentes:

$$u = (x_u, y_u, z_u) \text{ y } v = (x_v, y_v, z_v)$$

El producto escalar $u \cdot v$ será:

$$u \cdot v = x_u \cdot x_v + y_u \cdot y_v + z_u \cdot z_v$$

(El procedimiento sería análogo para vectores con cualquier número de componentes.)

A

Para hacer este cálculo se dispone de dos procedimientos: *ProdEsc* y *multiplica*, descritos a continuación.

Procedimiento ProdEsc: Es el encargado de realizar el producto escalar. Recibe tres parámetros a través de la pila en el siguiente orden de apilación:

- Dirección de comienzo del vector1
- Dirección de comienzo del vector2
- Tamaño de los vectores (por valor)

Para realizar el producto escalar, el procedimiento ejecuta un bucle en el que en cada iteración *i* se multiplican las componentes *i*-ésimas de los dos vectores. Dicho producto se acumula en una variable local al procedimiento, que al final del bucle contendrá el producto escalar de los dos vectores. El resultado se devuelve en el registro EAX.

Procedimiento **multiplica**: Este procedimiento es llamado desde *ProdEsc*. Realiza la multiplicación de dos números enteros, recibidos como parámetros a través de la pila por valor. El resultado se devuelve en el registro EAX.

Ambos procedimientos deben dejar todos los registros en el mismo estado en que se encontraban antes de comenzar su ejecución. Asimismo, tras hacer una llamada a un procedimiento, la pila también debe estar en el mismo estado en que se encontraba antes de realizar la llamada.

Al final de la ejecución del programa, debe cargarse el resultado obtenido en la variable global *resultado*.

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

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

Contesta a las preguntas relativas a este programa que se presentan a continuación:

— Escribe las instrucciones que deberían estar en el hueco (--**1**--), correspondientes al programa principal. Se desean multiplicar los vectores *vector1* y *vector2*, ambos definidos en la sección de datos del programa. El tamaño de los vectores está contenido en otra variable de la sección de datos llamada *tamanio* de tipo BYTE. El resultado del producto escalar debe guardarse en la variable global *resultado*.

1

```
push  OFFSET vector1
push  OFFSET vector2
movzx eax, [tamanio]
push  eax
call  ProdEsc
add   esp, 12
mov   [resultado], eax
```

— ¿Qué byte contiene la posición de memoria **0040400C**h al inicio del programa? Responde en hexadecimal.

0,5

```
26
```

— ¿Cuál es el menor valor que llega a alcanzar el registro ESP durante la ejecución del programa? Responde en hexadecimal.

0,5

```
00 12 FF 88
```

— Escribe las instrucciones que deberían estar en el hueco (--**2**--). En este hueco se debe realizar la lectura de los parámetros y poner a 0 la variable local al procedimiento. Asignar los parámetros a los registros indicados en el comentario situado en el código.

0,5

```
mov ecx, [ebp + 8]
mov esi, [ebp + 16]
mov edi, [ebp + 12]
mov DWORD PTR [ebp – 4], 0
```

— Para que el bucle del procedimiento *ProdEsc* funcione correctamente, ¿qué instrucciones habría que añadir en el hueco (--**3**--)?

0,5

```
add esi, 4
add edi, 4
```

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

Examen de Arquitectura de Computadores (Telemática) **Parcial de diciembre: 9-12-2009**

— Codifica la instrucción "`mov bh, [ebx + ecx]`" (NOTA: esta instrucción no pertenece al programa)

`8A 3C 19  (también valdría 8A 3C 0B)` **0,5**

— ¿Qué valor contienen los registros esi y edi en el instante en que se ejecuta la instrucción marcada con el símbolo ♠♠♠? Contestar en hexadecimal.

`esi: 00 40 40 0C  edi: 00 40 40 18` **0,5**

Se sabe que para configurar el puerto paralelo LPT1 como puerto de salida hay que escribir un 0 en el puerto de E/S 37Ah.

— Escribe las instrucciones necesarias para configurar el puerto LPT1 como puerto de salida:

```
mov al, 00h
mov dx, 37Ah
out dx, al
```
0,5

Responde a las siguientes preguntas breves:

1

Define el concepto de plataforma de ejecución:

Es el conjunto formado por el hardware de un computador y el sistema operativo que controla dicho hardware.

¿Qué datos se almacenan en el marco de pila de un procedimiento?

Sus parámetros, la dirección de retorno, el valor del registro EBP del procedimiento llamador y sus variables locales.

¿Cuál es el objetivo de la IDT?

Proporcionar las direcciones de los manejadores que atienden a interrupciones, excepciones y llamadas al sistema.

¿Cómo se llama el componente hardware encargado de realizar la transformación de direcciones virtuales a direcciones físicas?

MMU

— Define el concepto de jerarquía de memoria.

0,75

Se trata de combinar diferentes tecnologías de memoria, rápidas y grandes, de modo que:

- La memoria rápida contenga en cada momento la parte del programa o programas a los que se está accediendo con mayor frecuencia.
- La memoria lenta contenga el resto del programa o programas en ejecución.

— ¿Qué instrucciones escribirías para poner a '1' todos los bits del registro `al` salvo los bits 0, 1 y 2 (que permanecerían inalterados)?

```
or al, 0F8h
```
0,25

— ¿Qué instrucciones escribirías para poner a '0' los 4 bits más bajos del registro `al`, dejando el resto inalterados?

```
and al, 0F0h
```
0,25

En un determinado instante, el contenido del registro ebx es **0040500C**h y el contenido del registro edi es **00000002**h.

— Determina el rango de direcciones de memoria modificadas por la instrucción "`mov [ebx+edi*4+8], WORD PTR 26`". Contestar en hexadecimal.

`00 40 50 1C – 00 40 50 1D` **0,5**

A

En el siguiente código escrito en C, se calcula la suma desde 1 hasta n para un número entero dado.

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

int variable;

int sumatorio(int n)
{
	int auxiliar, contador;	//Instrucción 1

	auxiliar=0;
	contador=n;		//Instrucción 2

	while (contador>0){	//Instrucción 3
		auxiliar=auxiliar+contador;
		contador=contador-1;
	}
	return auxiliar;
}

int main ()
{

	variable=sumatorio(5);	//Instrucción 4

	_getch();
	return 0;
}
```

Escribe las instrucciones ensamblador que generaría un compilador de C para traducir las siguientes que se piden a continuación.

Nota: Puedes utilizar las etiquetas y los registros que consideres oportuno.

— Instrucción 1.

```
sub esp, 8
```

0,5

— Instrucción 4

```
push 5
call sumatorio
add  esp, 4
mov [variable], eax
```

0,75

— El bucle while completo etiquetado como `Instrucción 3`.

```
principio:
  cmp DWORD PTR [ebp-8], 0
  jle fin

  mov eax, [ebp-4]
  add eax, [ebp-8]
  mov [ebp-4], eax
  dec [ebp-8]

  jmp principio
fin:
```

1

— Instrucción 2.

```
mov eax, [ebp+8]
mov [ebp-8], eax
```

0,5