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

Examen de Arquitectura de Computadores. (Telemática) **Convocatoria de febrero, primera parte: 08-02-2010**

Se pretende escribir un programa en el lenguaje ensamblador de la arquitectura IA-32 que realice la suma de dos matrices A y B de cualquier tamaño. Las matrices están definidas en la sección de datos, y sus elementos son de tipo doble palabra. Las dos matrices tienen que tener forzosamente las mismas dimensiones. El número de filas y columnas que tienen las matrices se guardan en sendas variables, también en la sección de datos y de tipo byte.

Nota: Téngase en cuenta que, aunque en el código fuente parezca que las matrices A y B están declaradas como si fuesen matrices, en realidad todos los elementos de cada matriz se guardan de forma consecutiva en memoria, como si fuese un vector. Es decir, en memoria tras el último elemento de la primera fila de la matriz A se guarda el primer elemento de la segunda fila de la matriz A.

El resultado de la operación de suma de A y B sobrescribirá el antiguo valor de A, es decir, la operación a realizar es: A = A + B

Para realizar este cálculo, el programa principal ha de llamar al procedimiento *sumaMatriz*. Éste realiza la suma fila a fila, utilizando el procedimiento *sumaFila*, encargado de realizar la suma de una fila completa de las matrices. El procedimiento *sumaMatriz* tiene que llamar a *sumaFila* tantas veces como filas tengan las matrices.

Las multiplicaciones que pudieran hacer falta en cualquier parte del programa deben hacerse utilizando el procedimiento *multiplica*.

La especificación de todos estos procedimientos se muestra a continuación:

Procedimimento *sumaMatriz*: Recibe cuatro parámetros, todos de 32 bits, a través de la pila. El orden de apilación de éstos es el siguiente:

1. Dirección de comienzo de la matriz A
2. Dirección de comienzo de la matriz B
3. Número de filas de las matrices
4. Número de columnas de las matrices

El procedimiento realiza un bucle con tantas iteraciones como filas tengan las matrices. En cada iteración, realizan la suma de una fila completa utilizando el procedimiento *sumaFila* de forma anidada. No se retorna ningún valor.

Procedimiento *sumaFila*: Recibe tres parámetros, todos de 32 bits, a través de la pila. El orden de apilación de éstos es el siguiente:

1. Dirección de comienzo de la **fila** de la matriz A
2. Dirección de comienzo de la **fila** de la matriz B
3. Número de columnas de las matrices A y B (o dicho de otro modo, el número de elementos que tiene cada fila).

El procedimiento realiza un bucle con tantas iteraciones como columnas tengan las matrices. En cada iteración se suma a un elemento de la fila de A su correspondiente elemento de la fila de B. No se retorna ningún valor.

Procedimiento *multiplica*: Recibe a través de la pila los dos números que se desea multiplicar. El orden en que sean pasados éstos es irrelevante. Los números tienen que ser enteros positivos codificados en 32 bits.

Se retorna en EAX el producto de los dos parámetros.

Ténganse en cuenta los comentarios incluidos dentro del código fuente.

```
.386
.MODEL FLAT, stdcall

ExitProcess PROTO, :DWORD

.DATA
      A  DD    3,  5, 10,  3,
               2, 14,  0,  2,
               9,  0,  5, -1

      B  DD    3, -3,  6, 10,
               0, 10, -2, -2,
               4,  1,  3,  0

      filas    DB  3
      columnas DB  4

.CODE

multiplica PROC
      push ebp
      mov ebp, esp
      push ecx

      xor eax, eax

      ; Se realiza la multiplicación mediante una serie de sumas
      mov ecx, [ebp+8]
bucle:
      add eax, [ebp+12]
      loop bucle

      pop ecx
      pop ebp
      ret
multiplica ENDP
```

A

```
sumaFila PROC
       push ebp
       mov  ebp, esp

       ;Salvaguarda de registros modificados por el procedimiento
       push ecx
       push edi
       push esi     ◄◄◄

       mov  esi, [ebp+12]  ;esi:Dirección de comienzo de la fila de B
       mov  edi, [ebp+16]  ;edi:Dirección de comienzo de la fila de A
       mov  ecx, [ebp+8]   ;ecx:Número de columnas de A y B

       ;En cada iteración de este bucle, sumar un elemento diferente
       ;Aij = Aij + Bij
       ;i es la fila que se desea sumar con esta llamada a sumaFilas
       ;j es el elemento de la fila i que hay que sumar en cada
        ;iteración del bucle
bucle2:

       ( -- 3 -- )

       loop bucle2

       ; Restauración de los registros modificados
       pop  esi
       pop  edi
       pop  ecx
       pop  ebp
       ret        ; y retorno al procedimiento llamador
sumaFila ENDP


sumaMatriz PROC
       push ebp
       mov  ebp, esp

       ; Salvaguarda de los registros modificados en el procedimiento
       push ecx
       push edi
       push esi

       mov esi, [ebp+16] ;esi: Dirección de comienzo de la matriz B
       mov edi, [ebp+20] ;edi: Dirección de comienzo de la matriz A
♠♠♠    mov ecx, [ebp+12] ;ecx: Número de filas de A y B
```

```
; En cada iteración de este bucle, se suma una fila completa
; utilizando el procedimiento sumaFila

bucle:
       ; Llamada al procedimiento sumaFila para sumar
       ; las filas i-ésimas
       push edi
       push esi
       push DWORD PTR [ebp+8]
       call sumaFila
       add  esp, 12

       ; Actualizar los registros esi y edi para que en la siguiente
       ; iteración apunten a las siguientes filas
       ( -- 2 -- )

       loop bucle

       pop  esi     ;Restauración de registros modificados y retorno
       pop  edi
       pop  ecx
       pop  ebp
       ret
sumaMatriz ENDP

inicio:
       ; Llamada al procedimiento sumaMatriz para sumar A y B

       ( -- 1 -- )

       push 0
       call ExitProcess
END inicio
```

— Escribe las instrucciones que serían necesarias en el hueco (-- 2 --) del listado.

0,75

```
push  DWORD PTR [ebp+8]
push  4
call  multiplica
add   esp, 8
add   esi, eax
add   edi, eax
```

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

Examen de Arquitectura de Computadores. (Telemática) **Convocatoria de febrero, primera parte: 08-02-2010**

— Escribe las instrucciones que serían necesarias en el hueco (-- **1** --) del listado para invocar correctamente al procedimiento *sumaMatriz*.

1

```
push   OFFSET A
push   OFFSET B
mov    al,  filas
movzx  eax, al
push   eax
mov    al, columnas
movzx  eax, al
push   eax
call   sumaMatriz
add    esp, 16
```

— Escribe las instrucciones necesarias en el hueco (-- **3** --) para completar el bucle del procedimiento *sumaFilas*.

0,75

```
mov  eax,  [esi]
add  [edi], eax

add  esi, 4
add  edi, 4
```

Se sabe que la sección de datos comienza a partir de la dirección virtual **0x00404000**, y que el registro ESP al comienzo de la ejecución del programa contiene el valor **0x0012FFC4**. Conociendo esta información, contesta a las preguntas A y B.

— A) ¿Qué BYTE está contenido en la dirección virtual **0x00404034**? Contestar en hexadecimal.

0,5

```
FD
```

— B) ¿Qué dirección contendrá el registro ESP **después** de ejecutar la instrucción marcada con el símbolo ◄◄◄ en el listado? Contestar en hexadecimal.

0,5

```
00 12 FF 80
```

— Codifica la instrucción `mov ecx, [ebp+12]`, marcada con el símbolo ♠♠♠ en el listado. Contestar en hexadecimal.

0,5

```
8B 4D 0C
```

Se desea programar un procedimiento en ensamblador IA-32 que compruebe si un código ASCII pasado como parámetro se corresponde con una letra minúscula o no. Dicho código ASCII es el único parámetro que recibe el procedimiento, y éste es de 32 bits.

Información adicional: Código ASCII de 'a' = 97. Código ASCII de 'z' = 122.

— Completa el cuerpo del procedimiento para que escriba en el registro EAX un '1' si el parámetro era una letra minúscula o un '0' en cualquier otro caso.

1

```
esMinuscula PROC
   push ebp
   mov  ebp, esp
   push ebx

   ;Lectra del parámetro
   mov  ebx, [ebp+8]

   ;Si es menor que 97 o mayor que 122: EAX=0, si no: EAX=1
   ;COMPLETAR

   cmp  ebx, 97
   jb   NoMinuscula
   cmp  ebx, 122
   ja   NoMinuscula
   mov  eax, 1
   jmp  final
NoMinuscula:
   mov  eax, 0
final:

   pop  ebx
   pop  ebp
   ret
esMinuscula ENDP
```

— Responde brevemente a las siguientes cuestiones:

0,5

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
```

¿Qué tipos de excepciones existen en la arquitectura IA32?

Fallos, abortos y trampas

A

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

0,75

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.

— ¿Qué tres problemas presenta, principalmente, el uso de un sistema de memoria sin ningún mecanismo de protección?

0,5

1. **Un programa podría escribir sobre el código o los datos del SO**
2. **Un programa podría escribir sobre el código o los datos de otro programa.**
3. **Un programa podría escribir sobre su propio código.**

— Define los siguientes conceptos:

0,75

Plataforma de ejecución:
Conjunto formado por el hardware de un computador y el SO que controla dicho hardware.
IDT:
Es una tabla que contiene las direcciones de los manejadores que atienden a interrupciones, excepciones y llamadas al sistema.

PIC:
Es un dispositivo utilizado para dar soporte a varias fuentes de interrupción.

Se tiene el siguiente código para convertir una cadena de caracteres a mayúsculas. El procesamiento se realiza a través de una función auxiliar, que recorre la cadena desde su comienzo hasta encontrar un carácter ASCII 00. Cada carácter es examinado, y si pertenece al rango de las letras minúsculas, lo transforma en mayúscula mediante una operación lógica. Los caracteres que no sean una letra minúscula permanecerán inalterados.

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


char cadena[]="¿Garantia adicional? No tengo nada que perder.";

void procesa (char * cadena){

	char * p;  //Instrucción 2
	int i;     //Instrucción 2

	p=cadena;  //Instrucción 3
	i=0;       //Instrucción 4

	while (*(p+i) != 0){

		if (*(p+i)>=97 && *(p+i)<=122){
			*(p+i)= *(p+i) & 0x0DF;   //Instrucción 5
		}

		i++;
	}
}


int main ()
{

	procesa(cadena);  //Instrucción 1


	_getch();
	return 0;
}
```

Nota: No usar etiquetas de variables locales o de parámetros.

Nota: Téngase en cuenta que en el procedimiento procesa, al estar declarada la variable local p antes que i, ocupa en la pila direcciones de memoria más significativas.

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

Examen de Arquitectura de Computadores. (Telemática) **Convocatoria de febrero, primera parte: 08-02-2010**

— Escribe las instrucciones que se generarían al compilar la **Instrucción 1**.

0,5

```
Push  OFFSET cadena
call  procesa
add   esp, 4
```

— Escribe el código que generarían las **Instrucciones 2, 3 y 4**.

1

```
Instrucción 2: (Declaración de las variables p e i)
sub  esp, 8

Instrucción 3:
mov  eax, [ebp+8]
mov [ebp-4], eax

Instrucción 4:
mov  DWORD PTR [ebp-8], 0
```

— Escribe las líneas de código que podrían ser generadas al compilar la **Instrucción 5**. Puedes utilizar todos los registros que consideres oportunos.

1

```
mov  esi,  [ebp-4]
add  esi,  [ebp-8]
mov  al,  [esi]
and  al,  0DFh
mov [esi], al
```