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

Examen de Arquitectura de Computadores (Telemática) **Parcial de noviembre: 24-11-2010**

A continuación se muestra el código fuente de un programa escrito en lenguaje ensamblador, encargado de calcular la media aritmética de una lista de números positivos. Los números cuya media se desea calcular están ubicados en una lista de la sección de datos llamada *lista,* cuyos elementos son todos positivos y de tipo DOBLE PALABRA. Adicionalmente, se cuenta con una variable auxiliar, también ubicada en la sección de datos y de tipo BYTE llamada *tamanio,* que contiene el número de elementos que componen dicha lista. Una vez que se haya calculado la media aritmética de todos los elementos de la lista, ese valor ha de guardarse en la variable *media* ubicada en la sección de datos y declarada como de tipo DOBLE PALABRA. El cálculo de la media se realiza mediante un procedimiento llamado *promedio* que a su vez utiliza otro procedimiento llamado *divide,* cuyo funcionamiento se describe a continuación:

- Procedimiento *promedio*: Recibe dos parámetros a través de la pila, ambos de 32 bits. El primer parámetro que se debe apilar es la DIRECCIÓN de la lista que contiene los números, y el segundo es el número de elementos que forma la lista a procesar. Este procedimiento calcula el valor medio de los elementos de la lista y lo deja en el registro EAX.
- Procedimiento *divide*: Este procedimiento realiza la **división entera** de dos números naturales mediante restas sucesivas. El modo en que se calcula el cociente entero es restando al dividendo el divisor sucesivamente, hasta que el dividendo sea menor que el divisor, momento en que se detiene el algoritmo. El número de restas necesarias coincide con el cociente. Por ejemplo, si se quisiera dividir 19 entre 3 (cociente real 6.33, cociente entero 6, resto 1) se restaría 3 a 19 tantas veces como fuese necesario hasta que el resultado de una resta fuese inferior a 3:

  19 → 16 → 13 → 10 → 7 → 4 → 1

  Al haberse realizado 6 restas, el cociente entero es 6 (y el número que queda al finalizar el algoritmo, un 1, es el resto).

  Este procedimiento recibe dos parámetros a través de la pila, ambos de 32 bits. El primer parámetro que se debe apilar es el dividendo, y el segundo el divisor. El cociente se deja en el registro EAX.

Datos: La sección de datos del programa comienza a partir de la dirección **00404000h.**

El valor del registro ESP al comenzar la ejecución **0012FFC4h.**

```
.386
.MODEL FLAT, stdcall

ExitProcess PROTO, :DWORD

.DATA
      lista   DD 3, 5, 15, 8
      tamanio DB 4
      media   DD 0

.CODE

divide PROC
      push ebp
      mov  ebp, esp

      push ebx
      push edx
      push ecx     ♠♠♠

      mov  ebx, [ebp+8] ;ebx: Divisor
      mov  edx, [ebp+12] ;edx: Dividendo
      xor  ecx, ecx

      (--2--)


Bucle:
      cmp  edx, ebx
      jbe  finBucle

      sub  edx, ebx
      inc  ecx
      jmp  Bucle
finBucle:


finFuncion:
      mov  eax, ecx

      pop  ecx
      pop  edx
      pop  ebx
      pop  ebp
      ret  8
divide ENDP
```

A

```
promedio PROC
      push ebp
      mov  ebp, esp

      push esi
      push ecx
      mov  esi, [ebp+12] ;ESI: Dirección de comienzo de lista
      mov  ecx, [ebp+8]  ;ECX: Número de elementos de la lista
      xor  eax, eax      ;EAX: Auxiliar. Guardar la suma de
                         ;     todos los elementos de la lista


      (--3--)


      pop  ecx
      pop  esi
      pop  ebp
      ret
promedio ENDP


inicio:
      (--1--)

      push 0
      call ExitProcess

END inicio
```

— Escribe las instrucciones que sean necesarias en el hueco (--**1**--) del listado correspondiente al programa principal, de tal modo que se llame a la función *promedio* para hallar el valor medio de la lista *lista* y guardar el resultado en la variable global *media*.

1

```
push  OFFSET lista
movzx eax, [tamanio]
push  eax
call  promedio
add   esp, 8
mov   [media], eax
```

— Escribe las instrucciones que sean necesarias en el hueco (--**2**--) del listado. Las instrucciones escritas en este hueco deben comprobar que el divisor sea distinto de cero. En el caso de que el divisor sea cero, se debe saltar a la etiqueta *finFuncion* para terminar la ejecución del procedimiento.

0,5

```
cmp  ebx, 0
je   finFuncion
```

— Escribe las instrucciones que sean necesarias en el hueco (--**3**--) del listado. Estas instrucciones deben calcular la media aritmética de todos los elementos de la lista que se pasa por parámetro. Utiliza el registro EAX para calcular la suma de todos ellos y después divide ese valor por el número de elementos de la lista. Se pueden definir cuantas etiquetas sean necesarias, siempre y cuando no se utilice el nombre de ninguna otra etiqueta ya existente en el código fuente.

1

```
Bucle2:
add  eax, [esi]
add  esi, 4
loop Bucle2
push eax
push DWORD PTR [ebp+8]
call divide
```

— Se sabe que el byte contenido en una posición de la sección de datos es **0F**. ¿Cuál es su dirección? Contesta en hexadecimal.

0,5

```
00 40 40 08
```

— ¿Qué valor contendrá el registro ESP **después** de ejecutar la instrucción `push ecx` del procedimiento divide, señalada con el símbolo ♠♠♠? Contestar en hexadecimal.

0,5

```
00 12 FF 90
```

— ¿Cuál sería la codificación de la instrucción `mov ebx, [ebp-8]`? Contestar en hexadecimal.

0,5

```
8B 5D F8
```

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

Examen de Arquitectura de Computadores (Telemática) **Parcial de noviembre: 24-11-2010**

A continuación se muestra un programa escrito en C en el que se ha definido una función encargada de encontrar la potencia enésima de 2. Dicha función recibe por parámetro un número entero *n* y devuelve 2^n.

```
int numero;

int multiplica(int a, int b){
      return a*b;
}

int Potencia2n(int n)
{
      int auxiliar; //Instrucción 1
      auxiliar=1;   //Instrucción 2

      while(n>0){   //Bucle while
            auxiliar=multiplica(auxiliar,2);
            n=n-1;
      }

      return auxiliar;
 }


int main()
{
   numero=Potencia2n(10);

   return 0;
}
```

Escribe a continuación las instrucciones ensamblador que generaría un compilador de C para transformar las instrucciones indicadas en el código fuente:

— Instrucciones estándar de entrada a la función y creación del marco de pila, junto con la declaración de la variable *auxiliar* realizada en la Instrucción 1.

0,75

```
push ebp
mov  ebp, esp
sub  esp, 4
```

— Instrucción 2:

0,5

```
mov DWORD PTR [ebp-4], 1
```

— Escribe las instrucciones que se generarían al traducir a ensamblador el bucle *while* completo de la función *Potencia2n*.

1

```
Bucle:
  cmp  DWORD PTR [ebp+8], 0
  jle  finBucle
  push 2
  push DWORD PTR [ebp-4]
  call multiplica
  add  esp, 8
  mov  [ebp-4], eax
  dec  DWORD PTR [ebp+8]
  jmp  Bucle
finBucle:
```

Contesta las siguientes preguntas relativas a los registros de sistema y de aplicación en la arquitectura IA-32:

0,5

Objetivo de los registros de aplicación:
Proporcionan el soporte básico para el diseño de algoritmos. Almacenan operandos y direcciones de memoria.

Un ejemplo de registro de aplicación: EAX

Objetivo de los registros de sistema:
Contienen información de configuración del sistema y las direcciones de las estructuras de datos fundamentales del SO.

Un ejemplo de registro de sistema: IDTR

— ¿Cuántos niveles de privilegio define la arquitectura IA-32, y cuántos de ellos se utilizan en un equipo que tiene instalado un sistema operativo Windows?

0,5

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

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.

— Contesta a las siguientes preguntas breves:

1

Escribe las instrucciones necesarias para enviar un dato de tipo byte, declarado en la sección de datos de un programa ensamblador y llamado *dato,* al puerto 6060h del espacio de direcciones de E/S.

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

Describe brevemente qué se conoce como LINKER.
Es una aplicación que genera, a partir del lenguaje máquina contenido en uno o varios ficheros objeto y librerías, un fichero ejecutable.

¿Cuál es el objetivo de las excepciones?
Provocar una transferencia de control al sistema operativo cuando se alcanza una determinada condición de estado (normalmente un error) durante la ejecución de una instrucción.

¿Qué es la IDT y para qué se utiliza?
La IDT es una estructura de información organizada en forma de tabla que sirve para obtener las direcciones de los manejadores que atienden a las interrupciones, excepciones y llamadas al sistema.

En un momento dado, los registros generales de la CPU tienen almacenados los valores que se indican en la siguiente tabla:

Registro	Contenido (en hexadecimal)
EAX	00 00 00 00
EBX	00 40 20 00
ECX	00 00 00 09
EDX	0A 00 0A 00
ESI	00 00 00 00
EDI	00 00 00 02
EIP	00 40 10 3A
ESP	00 12 44 FF
EBP	00 00 00 0C

En ese instante se ejecuta la instrucción `mov DWORD PTR [ebx+edi*4+8], 522`.

Contesta a las siguientes preguntas:

— Determina el rango de direcciones de memoria modificadas por la instrucción.

00 40 20 10 – 00 40 20 13 **0,5**

— Determina el valor del byte almacenado en la dirección menos significativa de las modificadas por la instrucción una vez que ésta se haya ejecutado. Contestar en hexadecimal.

0A **0,5**