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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de febrero, primera parte: 09-02-2008**

A continuación se muestra el listado de un programa cuyo objetivo es calcular, dada una lista de números, cuáles de ellos son divisibles por otro determinado número. Para llevar a cabo este procesamiento se diseña un procedimiento llamado *CalculaDivisibles*. Este procedimiento recibe los parámetros que se describen a continuación, en el orden indicado:

1) Número del cual queremos obtener los divisibles que se encuentran en la lista de números. (Si queremos calcular los divisibles por 4, este número sería 4).
2) Dirección de la lista de números a procesar.
3) Dirección de un *buffer* (*array*) en el que se almacenarán los números divisibles encontrados.
4) Número de elementos que se pueden almacenar en el *buffer* anterior. El objetivo de este parámetro es poder finalizar la ejecución de *CalculaDivisibles* cuando se agota el espacio en el *buffer* para almacenar números divisibles.

La lista de números a procesar puede tener cualquier tamaño, pero debe estar formada por números positivos y terminada con el número 0. Tanto la lista a procesar como el buffer para almacenar los divisibles están formados por datos de tipo doble palabra.

El procedimiento *CalculaDivisibles* retorna en el registro EAX los siguientes valores:

- 0 **à** si el procedimiento falla, es decir, si se produce alguna incidencia en el cálculo de los divisibles. Las incidencias pueden ser: 1) se ha encontrado un número negativo en la lista de números a procesar; y 2) se ha agotado el *buffer* en el que se almacenan los números divisibles.
- 1 **à** si el procedimiento tiene éxito, es decir, si no se produce ninguna incidencia en el cálculo de divisibles.

El procedimiento *CalculaDivisibles* procesa la lista de números cuya dirección recibe como parámetro hasta que encuentre un 0 (que indica el final de la lista) retornando en este caso un 1 en EAX. El procedimiento finalizará antes de procesar la lista completa si encuentra un número negativo o si se llena el buffer, retornado entonces un 0 en EAX.

Para llevar a cabo las divisiones necesarias, el procedimiento *CalculaDivisibles* utiliza la instrucción "`idiv {R32|M32}`". Esta instrucción divide un número de 64 bits almacenado en EDX:EAX (operando implícito) entre el operando R32 (registro de 32 bits) o M32 (posición de memoria de 32 bits) indicado explícitamente en la instrucción. La instrucción genera el cociente de la división en EAX, y el resto en EDX. En este programa los números a dividir son pequeños por lo que caben en EAX, así que cuando se realicen las divisiones, el registro EDX se pondrá a cero.

El programa principal simplemente llama al procedimiento *CalculaDivisibles*, pasándole los parámetros adecuados, para determinar los números de la lista *ListaNumeros* (de la sección de datos) que son divisibles entre 3, y almacenarlos en el buffer *DivisiblesTres*.

Como datos adicionales se sabe que la sección de datos comienza en la dirección 00404000 y que justo al comienzo de la ejecución del programa, ESP = 0012FFC4.

```
.386
.MODEL FLAT, stdcall
ExitProcess PROTO, :DWORD

.DATA
ListaNumeros    DD  3, 7, 21, 18, 10, 15, 9, 0
DivisiblesTres  DD  0, 0, 0, 0

.CODE
CalculaDivisibles PROC
  push ebp
  mov  ebp, esp
  push edi
  push esi
  push ebx
  push edx

  ; cargar parámetros
  mov  edi, [ebp+12]  ; cargar dirección lista divisibles
  mov  esi, [ebp+16]  ; cargar dirección lista numeros

  ; NOTA: los parámetros 1 y 4 indicados en el enunciado
  ; no se cargan en registros. El procedimiento trabaja
  ; con ellos directamente en sus ubicaciones de pila

bucle:
  ; Si se acaba el buffer de divisibles indicar error

  cmp   No se muestra esto
  je    error
  ; Si el número a procesar es negativo indicar error
  cmp  DWORD PTR [esi], 0
  jl   error
  ; Si se acaba la lista de números abandonar el bucle
  je   fuera
  ; dividir

  (--1--)

  ; Si núemero divisible, almacenarlo en el buffer de divisibles
  ; Se usa EBX para mover el dato a dicho buffer
  cmp edx, 0
  jne sigue
  mov ebx, [esi]
```

A

```
  mov [edi], ebx

(--2--)

sigue:
  add  esi, 4
  jmp  bucle

fuera:
  ; no hay error
  mov  eax, 1
  jmp  RestaurarEstado

error:
  mov  eax, 0

RestaurarEstado:
  pop  edx  ◄◄◄
  pop  ebx
  pop  esi  ♠♠♠
  pop  edi
  pop  ebp
  ret
CalculaDivisibles ENDP

inicio:
  (--3--)
  ; Retorno al sistema operativo
  push 0
  call ExitProcess
END inicio
```

— ¿Qué instrucciones son necesarias en el hueco (—**1**—) del listado?

1

```
xor  edx, edx
mov  eax, [esi]
idiv DWORD PTR [ebp+20]
```

— ¿Qué instrucciones son necesarias en el hueco (—**2**—) del listado?

1

```
add  edi, 4
dec  DWORD PTR [ebp+8]
```

— ¿Qué instrucciones son necesarias en el hueco (—**3**—) correspondientes al programa principal?

1

```
push 3
push OFFSET ListaNumeros
push OFFSET DivisiblesTres
push 4
call CalculaDivisibles
add  esp, 16
```

— Determina los valores de los registros ESI y EDI justo antes de que se ejecute la instrucción “pop edx” marcada en el listado con el símbolo ◄◄◄.

1

```
ESI: 00404018          EDI: 00404030
```

— Determina el valor del registro ESP justo antes de que se ejecute la instrucción “pop esi” marcada en el listado con el símbolo ♠♠♠.

1

```
ESP: 0012FFA4
```

— Imagina que en algún lugar de la sección de código del programa anterior se encontrase la instrucción “`mov [listaNumeros+8], 7`”. Determina la codificación que tendría dicha instrucción.

1

```
C7 05 08 40 40 00 07 00 00 00
```

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

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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de febrero, primera parte: 09-02-2008**

A continuación se muestra el listado de una función C que recibe como parámetros dos números enteros y retorna como resultado el menor de ellos.

```
int menorDe( int x, int y )
{
    int menor;

    if ( x < y )
       menor = x;
    else
       menor = y;

    return menor;
}
```

En el listado siguiente se perfila mediante comentarios los bloques de código que genera el compilador de C para transformar a lenguaje máquina la función anterior. Escribe las instrucciones necesarias para cada uno de estos bloques. Donde sea pertinente sigue las instrucciones indicadas en los comentarios.

```
; Instrucciones estándar de entrada en la función
  push ebp
  mov  ebp, esp
  sub  esp, 4
```

0,5

```
; Instrucciones que calculan cúal es el menor de los parámetros.
; El menor de ellos se almacena en la variable local menor.
; Puedes utilizar como registros intermedios EBX y ECX.
; Utiliza las etiquetas que consideres oportunas.
  mov  ebx, [ebp+8]    ; Cargamos X en EBX
  mov  ecx, [ebp+12]   ; Cargamos Y en ECX
  cmp  ebx, ecx
  jl   xMenor
  ; Y es menor
  mov  [ebp-4], ecx
  jmp  sigue
xMenor:
  mov  [ebp-4], ebx
sigue:
```

1

```
; Retornar resultado
  mov  eax, [ebp-4]
```

0,25

```
; Instrucciones estándar de salida de la función
  mov  esp, ebp
  pop  ebp
  ret
```

0,25

— 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 las siguientes preguntas sobre las excepciones de tipo aborto en la arquitectura IA-32.

0,75

Objetivo:
Se utilizan para señalizar errores catastróficos que no permiten continuar la ejecución del sistema de una forma estable.

Causa:
Corrupción de las estructuras de datos del sistema, o bien errores hardware.

Tratamiento:
Se transfiere el control a una rutina que realiza el cierre controlado del sistema, deteniéndose la ejecución.