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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de junio: 15-06-2004**

A continuación se muestra el listado de un programa cuyo objetivo es procesar una cadena de caracteres. La cadena a procesar está formada por palabras separadas por guiones, siendo variable el número de guiones de separación. El objetivo del procesamiento es copiar cada palabra existente en la cadena en una ubicación separada para cada palabra, descartándose los guiones de separación que hay entre las palabras.

La cadena a procesar se encuentra en la sección de datos del programa y es referenciada mediante la etiqueta *frase*. La ubicación para almacenar las palabras procesadas de la cadena se establece también en la sección de datos, mediante un array formado 8 filas con capacidad para 16 caracteres cada una. La posición de comienzo de este array se encuentra marcado con la etiqueta *palabras*. Las filas del array están rellenadas inicialmente con guiones. Cuando se lleva a cabo el procesamiento, en cada fila se almacena una palabra. Así en nuestro caso, tras el procesamiento las tres primeras filas quedarán de la siguiente forma:

```
Frase-----------
de--------------
prueba----------
```

Y en las filas siguientes se almacenarán el resto de palabras de la frase.

Para procesar la frase se diseña un bucle. En cada una de sus iteraciones se procesa un carácter de la frase. En cada iteración pueden ocurrir cuatro cosas diferentes:

1) Se estaban procesando guiones y hay que procesar un nuevo guión.
2) Se estaban procesando guiones y hay que comenzar a procesar una palabra.
3) Se estaba procesando una palabra y hay que procesar un nuevo carácter de la palabra.
4) Se estaba procesando una palabra y hay que terminar de procesar la palabra.

Con objeto de poder saber en una iteración lo que estaba ocurriendo en la iteración anterior se utiliza la variable de la sección de datos *car_previo*, que se carga al final de cada iteración con el carácter que se procesó en dicha iteración.

Uno de los problemas fundamentales a resolver en este programa es el direccionamiento del array *palabras*. Para ello se utiliza, en la instrucción del listado marcada con ◄◄◄, el modo de direccionamiento '`[ebx+edi]`'. EBX se usa como un registro base que apunta al comienzo del array palabras. EDI proporciona un desplazamiento que permite ubicarse en cada momento en la posición adecuada del array. Cada vez que hay que hay comenzar a procesar una nueva palabra, hay que cargar EDI con el desplazamiento correspondiente a una nueva fila del array de palabras. Para ello se multiplica el contador de palabras procesadas por 16, que es el número de caracteres que hay en una fila. Con objeto de llevar a cabo esta multiplicación el programa utiliza el procedimiento *Multiplica*. Este procedimiento recibe a través de la pila los dos números a multiplicar y devuelve el resultado en el registro EDX .

Como dato adicional se sabe que el código ascii de la 'a' es 61h y que los códigos de las letras son continuos, o sea, 62h el de la 'b' y así sucesivamente. También se conoce que la primera posición de la sección de datos del programa se encuentra en la dirección 00402000.

```
.386
.model flat
Extern ExitProcess:PROC

.DATA
palabras   db "----------------"
           db "----------------"
           db "----------------"
           db "----------------"
           db "----------------"
           db "----------------"
           db "----------------"
           db "----------------"
frase      db "Frase---de--prueba-para----este-programa", 0
car_previo db '-'

.CODE
Multiplica PROC
  push ebp
  mov  ebp, esp
  push eax
  push ecx
  xor  edx, edx
  mov  eax, [ebp+8]
  mov  ecx, [ebp+12]
  jcxz sigue

bucle_multiplicacion:

  (--1--)

sigue:
  pop  ecx
  pop  eax
  pop  ebp
  ret  8
Multiplica ENDP


inicio:
  ; Inicializar registros
  mov  ebx, OFFSET palabras
  mov  esi, OFFSET frase
  mov  eax, 0  ; inicializar contador de palabras
```

A

```
bucle:
  cmp [esi], BYTE PTR 0
  je  fuera
  cmp [esi], BYTE PTR '-'
  jne es_caracter

es_guion:
  cmp [car_previo], '-'
  je  procesar_guion
  jmp terminar_procesamiento_palabra

es_caracter:
  cmp [car_previo], '-'
  je  comenzar_procesamiento_palabra
  jmp procesar_caracter

comenzar_procesamiento_palabra:
  ; Cargamos edi con el valor apropiado
  (--2--)
  jmp  procesar_caracter

terminar_procesamiento_palabra:
  inc eax ; incrementar contador de palabras
  ; Continuar en procesar_blanco

procesar_guion:
  ; no hacer nada
  jmp preparacion_iteracion_siguiente

procesar_caracter:
  mov cl, [esi]
  mov [ebx+edi], cl ◄◄◄
  inc edi
  ; Continuar en preparacion_iteracion_siguiente

preparacion_iteracion_siguiente:
  mov cl, [esi]
  mov [car_previo], cl
  inc esi
  jmp bucle
fuera:

  ; Retorno al sistema operativo
  push 0
  call ExitProcess
END inicio
```

Contesta a las siguientes preguntas relativas al programa anterior:

— ¿Qué instrucciones son necesarias en el hueco (—**2**—) del listado? **0,5**

```
push eax
push 16
call Multiplica
mov  edi, edx
```

— Codifica la instrucción ‘`mov [ebx+edi], cl`’ (marcada en el listado con el símbolo ◄◄◄). Indica su codificación en hexadecimal. **0,5**

```
88 0C 3B
```

— ¿Qué instrucciones son necesarias en el hueco (—**1**—) del listado? **0,5**

```
add  edx, eax
loop bucle_multiplicacion
```

— Indica cuál es el valor más grande que alcanza el registro EDI durante el procesamiento. Contesta en hexadecimal. **0,5**

```
58h
```

— ¿Qué otra instrucción, también de tipo `mov`, podría utilizarse para sustituir a la instrucción ‘`mov [ebx+edi], cl`’ (marcada en el listado con ◄◄◄), para que sin realizar ningún otro cambio en el programa, éste siga funcionando correctamente? Indica también si al hacer este cambio el tamaño en bytes de la sección de código del programa pasaría a ser mayor, igual o menor que antes. **0,5**

```
Instrucción: mov [palabras+edi], cl

Nuevo tamaño (contesta menor, igual o mayor): mayor
```

— ¿Qué valor hay almacenado en la dirección de memoria 00402043 justo antes de que se ejecute la instrucción `call ExitProcess`? Contesta en hexadecimal. **0,5**

```
65h
```

— Determina el valor del registro ESI en el momento que el registro EAX se carga con el valor 3. Contesta en haxadecimal. **0,5**

```
00402092
```

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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de junio: 15-06-2004**

A continuación se muestra el listado de un programa que se ejecuta sobre el sistema operativo Windows. El objetivo de dicho programa es procesar un conjunto de cadenas de caracteres que son introducidas por pantalla por el usuario. El programa funciona de la siguiente forma: el usuario introduce por pantalla el número de cadenas que quiere procesar, entonces el programa compromete una región de memoria virtual para almacenar las cadenas. Después, mediante un bucle, el programa le pide al usuario que introduzca las cadenas y las almacena en la región de memoria comprometida. Finalmente, mediante otro bucle, el programa visualiza las cadenas almacenadas.

Las cadenas manejadas por el programa son de tamaño variable, pero con un límite máximo de 1024 caracteres. Debido a esto, para cada cadena a procesar el programa deberá habilitar un área de memoria de 1024 caracteres. Así se asegura que haya sitio para la cadena sea cual sea su tamaño.

La salida realizada por pantalla por el programa cuando se han introducido las cadenas “aaa”, “bbbb” y “ccccc” debe ser la siguiente (sólo se muestra la salida correspondiente a la impresión de las cadenas):

```
Cadena[0]: aaa
Cadena[1]: bbbb
Cadena[2]: ccccc
```

Debe tenerse en cuenta esta salida al rellenar el hueco marcado con el símbolo ◄◄◄.

Rellena los huecos existentes en el listado prestando atención a los comentarios.

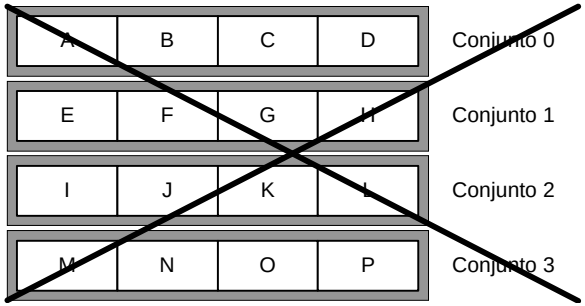
```
// Inlcuye los ficheros de cabecera que consideres oportunos
#include <windows.h>                                          0,25
#include <stdio.h>


main()
{
    char (*p)[1024];    // Para apuntar a la region comprometida
                        // y manejar en ella arrays de 1024 caracteres
    int  num_cad;       // Numero de cadenas a procesar
    int  i;             // Contador


    // Pedir por pantalla el numero de cadenas a procesar
    printf("\n\nNumero de cadenas a procesar: ");
    scanf("%d", &num_cad);

    // Reservar y comprometer la memoria necesaria (las páginas
    // justas, ni más ni menos) para almacenar las cadenas.
    // Hacer que el SO elija la region.
    p=VirtualAlloc(NULL,                                      0,5
                   num_cad*1024,
                   MEM_RESERVE | MEM_COMMIT,
                   PAGE_READWRITE);

    // Introducir cadenas
    printf("\n========== Introducir cadenas ==========\n");
    for(i=0; i<num_cad; i++)
    {
        printf("Cadena[%d]: ", i);
        scanf("%s", p+i);
    }

    // Visualizar cadenas
    printf("\n========== Visualizar cadenas ==========\n");
    for(i=0; i<num_cad; i++)
       printf("Cadena[%d]: %s\n", i, p+i);                    0,5   ◄◄◄

    // liberar completamente la memoria indicando si la operación
    // tiene exito o no
    if ( VirtualFree(p, 0, MEM_RELEASE)                       0,25 )
       printf("\nLiberacion OK\n");
    else
       printf("\nFallo en la liberacion de memoria\n");
}
```

— Se sabe que durante la ejecución del programa anterior el puntero ***p*** ha tomado el valor 00509000 y ***num_cad***, 29 (decimal). Teniendo en cuenta esta información, ¿cuál es la dirección final de la región comprometida por el programa?

00510FFF 0,5

A

Se dispone de un computador que tiene una memoria principal de 4 Kbytes, manejada con direcciones de 12 bits. Con objeto de mejorar el rendimiento de este computador, se diseña para él una memoria *cache* formada por 16 bloques de 8 bytes cada uno, organizados en 4 vías. A continuación se muestra un esquema de esta cache, en el que se da un nombre a cada bloque y se indica el conjunto al que cada bloque pertenece.



A partir de estos datos contestar a las siguientes preguntas:

- ¿En cuál o cuáles de los bloques de la *cache* puede ubicarse la dirección 7ACh de la memoria principal? Contesta utilizando las letras utilizadas en la figura superior para nombrar los bloques.

E, F, G y H — **0,5**

- Determina cuál es la dirección más grande de la memoria principal que puede ubicarse en el bloque K de la *cache*.

FF7 — **0,5**

- Supongamos que los bloques de la *cache* de la figura superior se reorganizan de las siguientes formas:

  A) Siguiendo una estrategia de correspondencia totalmente asociativa.

  B) Siguiendo una estrategia de correspondencia directa.

Se desea saber, para cada uno de los casos anteriores, cuántos bloques diferentes de la memoria principal se podrían ubicar en un determinado bloque de la *cache*. Contesta en decimal.

0,5

Nº bloques caso A: 512

Nº bloques caso B: 32

Se dispone de un computador cuyo sistema de memoria virtual trabaja con páginas de 1 KB. Además se sabe que su espacio de direcciones virtual y su espacio de direcciones físico están formados por 64 y 16 páginas, respectivamente. Se supone que el tamaño de la memoria instalada en el sistema coincide con el tamaño del espacio de direcciones físico.

En la tabla inferior izquierda se muestra el estado de la tabla de páginas de un proceso, que se encuentra en ejecución en el sistema. En esta tabla, tanto las páginas virtuales como las físicas están expresadas en decimal. Las entradas de la tabla de páginas que contienen un '-', así como aquellas que no se muestran, corresponden a páginas no utilizadas por el proceso; si contienen la palabra 'disco', significa que se encuentran en el disco; y si en ellas se encuentra un número, éste representa la página física en la que se encuentra ubicada la página virtual correspondiente.

En la tabla inferior derecha (a rellenar) se indica una secuencia de direcciones generadas por el proceso (empezando por la dirección 4C22). Rellena las columnas de la tabla que se encuentran en blanco, indicando para cada dirección virtual: 1) si se produce fallo de página o no, 2) si el proceso pude continuar su ejecución, y 3) la dirección física generada. Se debe rellenar la tabla hasta que se llegue a una dirección virtual que no se pueda traducir.

Nota: Si debido a las direcciones virtuales generadas se mueve alguna página del disco a la memoria física, las páginas físicas usadas para albergar las páginas provenientes del disco comenzarán en la página 0 y, después, serían consecutivas a ésta.

Tabla de páginas

Página Virtual (Dec.)	Página Física (Dec.)
0	-
1	-
2	Disco
3	Disco
4	9
5	11
18	14
19	15
20	12
62	Disco
63	-

A rellenar

1

Dir. Virtual (Hex.)	Fallo Pág. (Si/No)	¿Continúa ejecución? (Si/No)	Dir. Física (Hex.)
4C22	NO	SI	3C22
0D30	SI	SI	0130
F908	SI	SI	0508
0F0A	NO	SI	030A
7DC3	NO	NO	-
7121	-	-	-
0398	-	-	-
FFFF	-	-	-

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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de junio: 15-06-2004**

— Indica cinco tipos de información que sean típicamente mantenidos en el PCB de un proceso. Comenta brevemente cada uno de ellos:

0,5

A elegir entre los siguientes:

Estado del proceso
Identificador del proceso
Contador de programa
Registros
Información de planificación
Información de manejo de memoria
Información de E/S
Información de uso de recursos

— Contesta a las siguientes cuestiones sobre las interrupciones enmascarables utilizadas en los procesadores IA-32.

0,5

Objetivo:
son utilizadas por los periféricos para solicitar la atención del procesador.

Nombre y finalidad de las líneas de control usadas para gestionar este tipo de interrupciones:
INTR (Interrupt Request): Se utiliza para solicitar interrupción al procesador.

INTA (Interrupt Acknowledge): Es utilizada por el procesador para inidicar que ha aceptado una solicitud de interrupción.

Mecanismo de enmascaramiento:
Bit IF del registro de estado. Si IF=0 no se aceptan las interrupciones. Si IF=1 sí se aceptan.

— Define la TLB indicando claramente su objetivo

0,5

Es una memoria de alta velociada totalmente integrada c la MMU, que almacena las entradas de la tabla de páginas más frecuentemente utilizadas. Así las direcciones virtuales pueden ser traducidas sin necesidad de acceder a la memoria física. Es un mecanidmos hardware permite actualizar dinámicamente el TLB con las entradas de la tabla de páginas más frecuentemente utilizadas.

— Determina cuál o cuáles de las siguientes afirmaciones relativas al procesador IA-32 son CIERTAS. Contesta "ninguna" si crees que ninguna lo es.

A) La instrucción '`inc eax`' no podrá ser ejecutada en el modo supervisor del procesador.

B) Las direcciones lógicas de la arquitectura IA-32 son de 48 bits.

~~C) La dirección lógica de la siguiente instrucción a ejecutar viene determinada por ES:EIP.~~

D) El sistema operativo Windows utiliza dos niveles de privilegio de la arquitectura IA-32: 0 para supervisor y 1 para usuario.

0,5

B