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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de septiembre: 06-09-2008**

Se ha diseñado un programa ensamblador para trabajar sobre conjuntos. Los elementos de estos conjuntos son letras mayúsculas, debiendo cumplirse que todas las letras pertenecientes a un mismo conjunto sean diferentes.

En este programa los conjuntos se almacenan en listas de datos de tipo *byte*. El primer elemento de la lista es un número indicando el número de elementos que tiene el conjunto. A continuación de este número se almacenan los elementos del conjunto, es decir, una serie de letras mayúsculas. En la sección de datos del programa puedes observar la definición de tres conjuntos: *conjuntoA*, *conjuntoB* y *conjuntoI*.

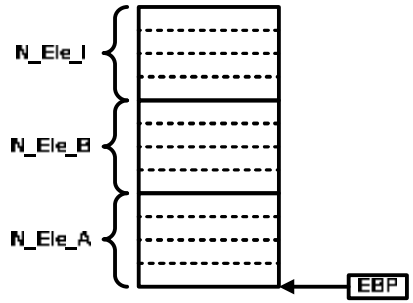
En este programa se ha diseñado un procedimiento, denominado *Interseccion*, cuyo objetivo es realizar la operación de intersección entre dos conjuntos. Recuerda que la intersección de dos conjuntos es otro conjunto que contiene solamente aquellos elementos que son comunes a los dos conjuntos sobre los que se hace la intersección. Así en el caso de los conjuntos definidos en la sección de datos de nuestro programa, *conjuntoA* y *conjuntoB*, su intersección será el conjunto formado por los elementos 'J', 'A' y 'B', que son los tres elementos comunes a *conjuntoA* y *conjuntoB*. A continuación se indican las pautas de diseño seguidas en la construcción del procedimiento *Interseccion*.

1) El procedimiento recibe tres parámetros: las direcciones de las dos listas conteniendo los datos de los conjuntos sobre los que realizar la intersección y la dirección de otra lista en la que almacenar el resultado de la intersección. En este programa en concreto, el programa principal pasará al procedimiento las direcciones de *conjuntoA*, *conjuntoB* y *conjuntoI*. *conjuntoA* y *conjuntoB* contienen los datos de los conjuntos sobre los que se va a hacer la intersección y *conjuntoI* es la lista en la que se dejará el resultado de la intersección. Los parámetros son pasados a través de la pila y destruidos por el llamador.

2) Para llevar a cabo el procesamiento requerido, se diseña un bucle que va procesando uno a uno los elementos de uno de los conjuntos (en concreto aquel cuya dirección se apila en primer lugar), en nuestro caso, los elementos de *conjuntoA*. En el programa se ha llamado a este bucle *bucleExt* (bucle exterior). Para cada elemento de *conjuntoA*, se busca si se encuentra en el otro conjunto (o sea, aquel cuya dirección se pila en segundo lugar), en nuestro caso, *conjuntoB*. Esto requiere comparar el elemento del *conjuntoA* que se esté procesando con cada elemento de *conjuntoB* hasta que se encuentre una coincidencia o hasta que se hayan recorrido todos los elementos de *conjuntoB*. Si se ha encontrando una coincidencia, el elemento se llevará a *conjuntoI* y si no, se descartará. Para llevar a cabo estas comparaciones se diseñan un bucle llamado *bucleInt* (bucle interior), debido a que éste se encuentra embebido dentro del bucle exterior.

3) En el procedimiento se utilizan tres registros como punteros: EBX, ESI y EDI. EBX se usa para apuntar a los elementos del conjunto cuya dirección se apila en primer lugar, en nuestro caso, *conjuntoA*. ESI se usa para apuntar a los elementos del conjunto cuya dirección se apila en segundo lugar, en nuestro caso, *conjuntoB*. Finalmente, EDI se usa para apuntar a la lista donde se almacena el conjunto intersección, en nuestro caso *conjuntoI*.

4) El procedimiento utiliza tres variables locales de tipo doble palabra, cuyas posiciones en la pila se indican en la figura siguiente:



El objetivo de estas variables es almacenar temporalmente el número de elementos que hay en cada uno de los conjuntos, así por ejemplo, en la variable *N_Ele_A* se almacena el número de elementos que hay en *conjuntoA*.

5) El procedimiento retorna en el registro EAX el valor 0 si el resultado de la intersección es el conjunto vacío y el valor 1 en el caso contrario.

El programa principal llama al procedimiento pasándole los parámetros adecuados y chequea el resultado devuelto por el procedimiento, entonces si el conjunto intersección obtenido como resultado no es el conjunto vacío, se almacena el primer elemento de *conjuntoI* en la variable *primerEleI* de la sección de datos. En el caso contrario, se almacena el carácter '*' en dicha variable.

A continuación se proporciona el listado del programa.

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

.DATA
conjuntoA  DB  5
           DB  'J', 'A', 'C', 'B', 'F'
conjuntoB  DB  4
           DB  'J', 'B', 'R', 'A'
conjuntoI  DB  0
           DB  '*', '*', '*', '*', '*'
primerEleI DB  0
```

A

```
.CODE
inicio:


  (--1--)


  ; Retorno al Sistema Operativo
  push 0
  call ExitProcess

Interseccion PROC
  push ebp
  mov  ebp, esp
  ; reserva de espacio para variables locales
  (--2--)

  ; Salvaguarda de registros
  push ebx
  push esi
  push edi
  push ecx
  push edx

  ; Acceso a los parámetros
  mov  ebx, [ebp+16]
  mov  esi, [ebp+12]
  mov  edi, [ebp+8]

  ; Inicializar variables locales
  mov  al, [ebx]
  mov  [ebp-4], eax  ◄◄◄
  mov  al, [esi]
  mov  [ebp-8], eax
  mov  al, [edi]
  mov  [ebp-12], eax

  ; Incrementar cada registro para que apunte
  ; al primer elemento de su respectivo conjunto
  add  ebx, 1
  add  esi, 1
  add  edi, 1

  ; Se guarda en EDX la dirección del primer elemento del
  ; conjunto cuya dirección de pasa como segudo parámetro
  ; (conjunto B en este ejemplo)
  mov  edx, esi

  ; Cargar iteraciones bucle exterior
  (--3--)
  jcxz vacio
bucleExt:
  mov  al, [ebx]
  ; Salvar en la pila el ECX del bucle exterior
  push ecx
  ; Cargar iteraciones bucle interior
  (--4--)
  jcxz vacio
bucleInt:
  cmp  al, [esi]
  jne  sigue
  ; Elemento en ambos conjuntos
  ; Incluirlo en el conjunto interseccion
  mov  [edi], al
  inc  edi
  ; Incrementar el número de lementos del conjunto interseccion
  ; (variable local N_Ele_I)
      No se muestra esta instrucción
  ; Se abandona el procesamiento del bucle interior
  jmp  sigue2

sigue:
  inc  esi
loop bucleInt

sigue2:
  ; Reiniciar ESI para que apunte al primer elemento del
  ; conjunto cuya dirección se pasa como segudo parámetro
  ; (conjunto B en este ejemplo)
  mov  esi, edx

  inc  ebx
  ; Restaurar el ECX del bucle exterior
  pop ecx
loop bucleExt

  ; En la lista que almacena el conjunto intersección
  ; actualizar el elemento que guarda su tamaño
  (--5--)
```

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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de septiembre: 06-09-2008**

```
vacio:
  ; Alguno de los conjuntos está vacío
  ; No se hace nada sobre el conjunto intersección

  ; Se retorna el resultado
  No se muestra esta instrucción ; Instrucción que compara
                                 ; N_Ele_I con 0
  jne   sigue3
  mov   eax, 0
  jmp   sigue4
sigue3:
  mov   eax, 1
sigue4:

  ; Restauración de registros
  pop  edx
  pop  ecx
  pop  edi
  pop  esi
  pop  ebx

  ; Eliminación de variables locales
  mov  esp, ebp

  pop  ebp
  ret
Interseccion ENDP

END inicio
```

Como datos adicionales se conoce que la dirección de comienzo de la sección de datos del programa es 00404000 y que el valor del registro SP, justo cuando el programa comienza a ejecutarse, es 0012FFC4.

Teniendo en cuenta toda esta información contesta a las pregustas que se proporcionan a continuación.

— El hueco (—**1**—) del listado corresponde al cuerpo del programa principal. Escribe las instrucciones necesarias para que el programa principal lleve a cabo las acciones indicadas en el enunciado del programa. Si necesitas etiquetas, utiliza la palabra *continua* seguida de los números 1, 2, 3, etc. según el número de etiquetas que necesites.

1

```
push OFFSET conjuntoA
push OFFSET conjuntoB
push OFFSET conjuntoI
call Interseccion
add  esp, 12
```

```
  cmp  eax, 0
  je   continua1
  mov  bl, [conjuntoI+1]
  mov  [primerEleI], bl
  jmp  continua2
continua1:
  mov  [primerEleI], '*'
continua2:
```

— ¿Qué instrucción es necesaria en el hueco (—**2**—) del listado?

0,5

```
sub  esp, 12
```

— ¿Qué instrucciones son necesarias en los huecos (—**3**—) y (—**4**—) del listado?

0,5

```
Hueco (--3--): mov  ecx, [ebp-4]

Hueco (--4--): mov  ecx, [ebp-8]
```

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

0,5

```
mov  edi, [ebp+8]
mov  eax, [ebp-12]
mov  [edi], al
```

— Codifica la instrucción "`mov [ebp-4], eax`" marcada con el símbolo ◄◄◄ en el listado.

0,5

```
89 45 FC
```

— Indica el rango de direcciones ocupado por la variable local *N_Ele_I* durante la ejecución del procedimiento *Interseccion*. (Ejemplo de respuesta: 00000000 – 00000006)

0,5

```
0012FFA4 – 0012FFA7
```

A

A continuación se muestra el listado de un programa C simple. La función *menorDe()* retorna el menor de los dos enteros que recibe como parámetros. El cuerpo de esta función no se muestra ya que no es necesario.

```
int R=0, A=5;


main()
{

    R = menorDe( A, 9 ); ♠♠♠

}

int menorDe( int x, int y )
{
    ...
}
```

— Tenido en cuenta toda esta información, determina el código máquina que generaría el compilador de C al compilar la sentencia de asignación marcada con el símbolo ♠♠♠ en el listado anterior.

0,75

```
push  9
push  [A]
call  menorDe
add   esp, 8
mov   [R], eax
```

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

0,75

¿Para qué se utilizan?
Se utilizan para señalizar errores catastróficos que no permiten continuar la ejecución del sistema de una forma estable.

¿Qué causas las provocan?
La corrupción de las estructuras de datos del sistema o bien errores hardware.

¿Cómo se tratan una vez producidas?

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

Indica un ejemplo de excepción de este tipo:

La doble falta.

— Indica y describe brevemente los componentes básicos de un disco duro

0,5

Platos: Están formados por una aleación rígida de aluminio y recubiertos por una capa de material magnético sobre la que se graba la información.

Motor de giro: Su objetivo es hacer girar los platos a velocidad constante.

Cabezas de lectura/escritura: Su objetivo es escribir información sobre la superficie de los platos y leer de ellos. Hay una cabeza por cada superficie.

Brazo: Su objetivo es servir de soporte para la cabeza.

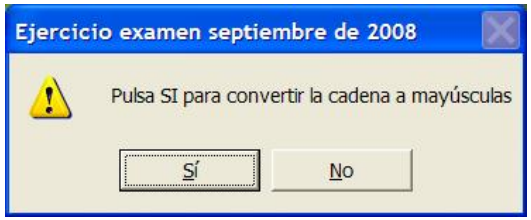
Actuador: Es un servomotor encargado de mover los brazos para posicionar las cabezas en las posiciones de los platos requeridas.

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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de septiembre: 06-09-2008**

El objetivo de un programa es procesar una cadena de caracteres introducida a través de la consola. El procesamiento a realizar es convertir todas las letras minúsculas que se encuentren en la cadena a mayúsculas, pero sin realizar modificación alguna sobre aquellos caracteres que no sean minúsculas. Así si la cadena a convertir fuera "JJab12;r", la conversión realizada por el programa debería generar la cadena "JJAB12;R".

El programa permite al usuario decidir si la cadena introducida a través de la consola se convierte o no. Para ello el programa genera la siguiente ventana de mensaje:



Entonces si el usuario pulsa el botón SÍ la cadena se convierte, pero si pula NO, se deja como está. Finalmente el programa imprime la cadena en consola, haya sido ésta convertida o no.

Para llevar a cabo la conversión se apunta a la cadena mediante el puntero *p*. Entonces mediante un bucle *while* se recorre la cadena carácter a carácter hasta que se encuentre su terminador, que es el número 0. Para determinar si el carácter es minúscula, debe comparase con los números 61(hex) y 7A(hex), que corresponden a las letras 'a' y 'z' respectivamente. Para convertir una minúscula en mayúscula se puede restar el valor 20(hex) al carácter procesado.

A continuación se proporciona el listado del programa perfilado mediante comentarios. Debes rellenar los huecos del programa teniendo en cuenta los comentarios.

```
// Incluir los ficheros de cabecera necesarios
   #include <windows.h>                                          0,25
   #include <stdio.h>

main()
{
   char mensaje[]="Pulsa SI para convertir la cadena a mayúsculas";
   char titulo[]="Ejercicio examen septiembre de 2008";
   char cadena[80];
   char *p;        // Puntero auxiliar para procesar la cadena
   int valor_ret;  // Para recoger el valor devuelto por MessageBox()

   // Introducir la cadena por consola
   printf("Introduzca la cadena a procesar: ");
   scanf_s("%s", cadena, 80);                                    0,5
```

```
   // Solicitar conversión de la cadena mediante MessageBox
   valor_ret = MessageBox( NULL, mensaje, titulo,                0,5
                           MB_YESNO | MB_ICONEXCLAMATION );

   // Si se ha pulsado SI convertir la cadena a mayúsculas
   if ( valor_ret == IDYES )                                     0,25
   {
       p=cadena;
       while( *p!=0 )                                            0,5
       {
           if ( (*p >= 0x61) && (*p <= 0x7A) )
               *p = *p - 0x20;
           p++;
       }
   }

   // Imprimir cadena
      printf("\n\n%s", cadena );
}
```

— Un programa desarrollado para la plataforma Windows ejecuta una sentencia *VirtualAlloc()* y como resultado de la misma resulta reservada y comprometida la siguiente región de memoria: [0102 0000 – 0103 2FFF]. Además la región se configura para que pueda ser leída y escrita. Escribe la sentencia *virtualAlloc()* apropiada para realizar la reserva y compromiso de dicha región. El valor devuelto por *VirtualAlloc()* será asignado a un puntero *p* definido en el programa.

```
p=VirtualAlloc( (void *)0x01020000,                              0,5
                19*4096,
                MEM_RESERVE | MEM_COMMIT,
                PAGE_READWRITE );
```

A

Se dispone de un computador cuyo sistema de memoria virtual trabaja con direcciones virtuales de 16 bits, direcciones físicas de 14 bits y páginas de 1 KB. 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. 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 400C). Rellena las columnas de la tabla que se encuentran en blanco, indicando para cada dirección virtual los datos siguientes: 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 una dirección virtual se mueve alguna página del disco a la memoria física, la página física usada para albergar la página proveniente del disco será la menos significativa que se encuentre libre. Se supone que las únicas páginas físicas utilizadas son las indicadas en la tabla de páginas del proceso.

Tabla de páginas

Página Virtual	Página Física
00	-
01	-
0E	A
0F	B
10	0
11	Disco
12	1
13	Disco
3E	-
3F	-

A rellenar — 1

Dir. Virtual	Fallo Pág. (Si/No)	¿Continúa ejecución? (Si/No)	Dir. Física
400C	NO	SI	000C
4487	SI	SI	0887
4544	NO	SI	0944
FE23	SI/NO	NO	-
0178	-	-	-
E992	-	-	-
13AF	-	-	-

— Contesta a las preguntas que se indican en el siguiente cuadro:

0,5

Indica los nombres de los dos tipos de colas utilizadas en en los sistemas operativos para gestionar la planificacion de procesos:

Cola de procesos listos y cola de dispositivo

Un mecanismo usado en los sistemas operativos tiene por objetivo sacar procesos de ejecución, salvándolos en el disco, para luego volver a ponerlos en ejecución cuando sea requerido. Indica el nombre anglosajón dado a este mecanismo:

swapping

Define el concepto de quantum de ejecución:

Es el tiempo máximo que un proceso puede ocupar la CPU de forma continuada

— Contesta las preguntas relativas a la arquitectura IA-32 indicadas en el siguiente cuadro:

0,5

Define el IOPL:

Es un campo de 2 bits del registro de estado que determina el nivel de privilegio necesario para que se puedan ejecutar instrucciones de E/S

¿Cúantos niveles de privilegio de ejecución tiene la arquitectura IA-32?

4

¿Cuándo se producen las conmutaciones de privilegio de ejecución?

cuando se produce una transferencia de control al SO mediante una llamada al sistema, una interrupción o una excepción.