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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de febrero: 10-02-2004**

A continuación se muestra el listado de un programa cuyo objetivo es determinar el número que representa una cadena de dígitos decimales. La cadena a procesar debe contener cuatro y sólo cuatro caracteres. Los caracteres de la cadena sólo pueden ser dígitos entre el 0 y el 9, ambos inclusive. Consecuentemente, la cadena representa un numero decimal.

Para procesar la cadena se define un procedimiento llamado *Convierte*. Este procedimiento recibe dos parámetros a través de la pila: la dirección de la cadena a procesar y la dirección de un *array* conteniendo las cuatro primeras potencias de 10, que serán usadas en el procedimiento de conversión. Para entender cómo se lleva a cabo la conversión nos fijaremos en la cadena almacenada en la sección de datos del programa. Se trata de "3419". Para procesar esta cadena, el procedimiento *Convierte* toma en primer lugar el carácter '3', lo convierte en el número 3 y luego lo multiplica por el primer valor del *array* de potencias, que es 1000. El resultado obtenido lo almacena en un acumulador. Después toma el siguiente carácter, '4', lo convierte en número y lo multiplica por 100, agregando este resultado al acumulador. Este procedimiento se repite hasta que se han procesado los cuatro caracteres de la cadena, obteniéndose finalmente el resultado. El procedimiento *Convierte* retorna el resultado en el registro EAX.

Para llevar a cabo las multiplicaciones necesarias, el procedimiento *Convierte* utiliza otro procedimiento, llamado *Multiplica*. Este procedimiento recibe a través de la pila los dos números a multiplicar y devuelve el resultado en el registro EDX.

En el programa principal se llama al procedimiento *Convierte* para procesar la cadena *CadenaNumero* de la sección de datos y se almacena el resultado en la variable *Numero*.

Como datos adicionales se sabe que la sección de datos se ubica a partir de la dirección 00402000 y que justo antes de que el programa comience a ejecutarse ESP=0012FFC4. A la izquierda de cada instrucción mostrada en el listado se indica la dirección de memoria en la que se encuentra ubicada.

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

        .DATA
        CadenaNumero     db "3419"
        Numero           dd 0
        Potencias        dd 1000, 100, 10, 1

        .CODE
        Multiplica PROC
00401000   push ebp
00401001   mov  ebp, esp
00401003   push eax
00401004   push ecx
00401005   xor  edx, edx
00401007   mov  eax, [ebp+8]
0040100A   mov  ecx, [ebp+12]
0040100D   jcxz sigue     ; No hacer el bulce si ecx=0
        bucle1:
00401010   add  edx, eax
00401012   loop bucle1

        sigue:
00401014   pop  ecx
00401015   pop  eax
00401016   pop  ebp

          (--2--)
        Multiplica ENDP

        Convierte PROC
0040101A   push ebp
0040101B   mov  ebp, esp
0040101D   push esi
0040101E   push edi
0040101F   push ebx
00401020   push ecx

00401021   mov  esi, [ebp+12]
00401024   mov  edi, [ebp+8]
00401027   xor  ebx, ebx
00401029   xor  eax, eax ; Acumulador
0040102B   mov  ecx, 4

        bucle2:
00401030   mov  bl, [esi]
00401032   sub  bl, '0'  ; Convierte el ascii del numero en la numero
  (--1--)
00401038   call Multiplica   ; Multiplica digito por la potencia que
                      ; le correponde
0040103D   add  eax, edx
0040103F   inc  esi
00401040   add  edi, 4
00401043   loop bucle2

00401045   pop  ecx
00401046   pop  ebx
00401047   pop  edi
00401048   pop  esi
00401049   pop  ebp
```

A

```
          (--2--)
        Convierte ENDP

        Inicio:
          ; Convertir cadena
0040104D  push OFFSET CadenaNumero
00401052  push OFFSET Potencias
00401057  call Convierte

          ; Almacenar resultado
0040105C  mov [Numero], eax

          ; Retorno al sistema operativo
00401061  push    0
00401063  call    ExitProcess
00401068  nop
        END Inicio
```

Contesta a las siguientes preguntas relativas al programa anterior.

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

```
push ebx
push DWORD PTR [edi]
```

— ¿Qué instrucción falta en los huecos (—**2**—) del listado? **0,5**

```
Ret 8
```

— Codifica la instrucción `mov [Numero], eax`. Indica su codificación en hexadecimal. **0,5**

```
89 05 04 20 40 00
```

— En el momento de la ejecución del programa en el que el registro ESP=0012FFB8, determina el valor de 32 bits que hay almacenado en la cabecera de la pila. Contesta con ocho dígitos hexadecimales **0,5**

```
0040105C
```

— ¿Cuál es el valor más grande que alcanza el registro EDI durante la ejecución del programa? **0,5**

```
00402018
```

—

Cuando la instrucción `call convierte` se ejecuta se le suma un valor de 32 bits al registro EIP. Determina cuál es dicho valor.

```
FF FF FF BE
```
0,5

— Imagina que en la sección de datos del programa se encuentra definida la siguiente variable:

```
MenorQueMil    db    ' '
```

Escribe un fragmento de código ensamblador que cargue la variable *MenorQueMil* con una 'S' o una 'N' en función de que el resultado almacenado en la variable *Numero* sea menor que mil o no. Este fragmento de código se ubicaría justo a continuación de la instrucción `mov [Numero], eax`. Utiliza las instrucciones y etiquetas que consideres oportunas.

```
  cmp [Numero], 1000
  jae mayor_o_igual
  mov [MenorQueMil], 'S'
  jmp sigue
mayor_o_igual:
  mov [MenorQueMil], 'N'
sigue:
```
0,5

— Determina cuál o cuáles de las siguientes afirmaciones son CIERTAS. Contesta NINGUNA, si crees que ninguna lo es.

A) Un procesador IA-32 tiene una sola línea de petición de interrupciones.

B) En la arquitectura IA-32 la excepción de código de operación inválido tiene asociada la entra 6 de la IDT.

C) La IDT contiene direcciones lineales que apuntan a manejadores de interrupción o de excepción.

D) Cuando en una CPU IA-32 se produce una transferencia de control al sistema operativo, su privilegio de ejecución conmuta del 0 al 3.

```
B
```
0,5

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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de febrero: 10-02-2004**

A continuación se muestra el listado de un programa que se ejecuta sobre el sistema operativo Windows.

```
#include <stdio.h>

int A=0;

main()
{
      int B=0;
      void *p;

      printf("Dir. A = %p\n", &A);
      printf("Dir. B = %p\n", &B);
      printf("Dir. main = %p\n, main);
}
```

Al ejecutar el programa se obtiene la siguiente salida por pantalla:

```
Dir. A = 004068F0
Dir. B = 0012FF7C
Dir. main = 00401000
```

Las direcciones **físicas** en las que se ubican las variables A y B y la función *main()* son las siguientes:

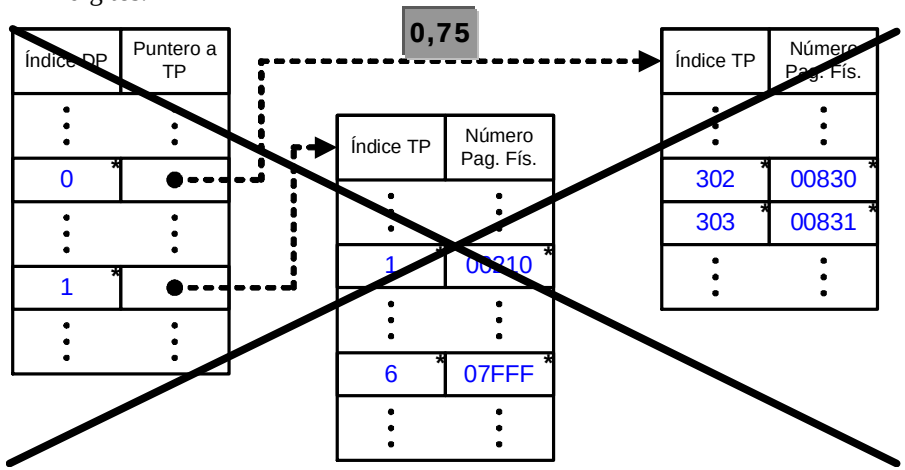
Dir. física (var A) = 07FFF8F0
Dir. física (var B) = 00831F7C
Dir. física (main) = 00210000

Se sabe que la imagen binaria del programa tiene una sola página de código y otra de datos. También se sabe que mientras que el programa se ejecuta hay dos páginas comprometidas para la pila, pero sólo una de ellas es utilizada por el programa. En esta página es en la que se encuentra la variable B. En la memoria física las páginas de pila se ubican en páginas contiguas.

Teniendo en cuenta toda esta información contesta a las preguntas A, B, C y D:

— A) Rellena la figura siguiente en la que se muestra el sistema de translación de direcciones del proceso. Para rellenar la figura ten en cuenta las siguientes indicaciones:

- Los campos a rellenar están marcados con un pequeño asterisco en la esquina superior derecha.
- Los campos índice en el DP e índice en la TP se rellanan en **decimal**. Cada tabla tiene 1024 entradas, que se numeran desde 0 a 1023. Se trata de determinar que números de entrada en estas tablas son utilizados por el proceso. Ten en cuenta que aunque haya puntos suspensivos entre las entradas, éstas pueden ser contiguas.
- El campo número de página física se rellena en **hexadecimal** utilizando cinco dígitos.



— B) Imagina que en este programa quieres reservar una región de memoria de 16 Mbytes a partir de la dirección 0100 0000 (hex). Las operaciones a realizar sobre esta región son lectura y escritura. El puntero **p** del programa debe quedar apuntando a la dirección base de esta región. Escribe la función *VirtualAlloc()* con los parámetros adecuados para llevar a cabo la operación de reserva requerida.

0,5

```
p=VirtualAlloc( (void *)0x01000000,
                16*1024*1024
                MEM_RESERVE
                PAGE_READWRITE );
```

— C) ¿Cuántas tablas de páginas deberá crear el sistema operativo para controlar las direcciones de la región anterior?

0,25

```
4
```

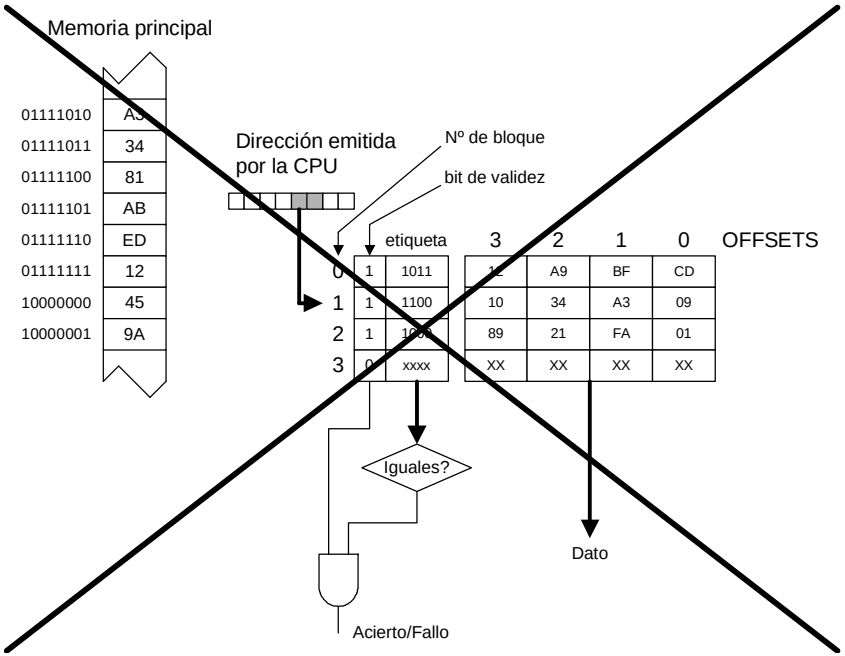
— D) Si la memoria física del computador en la que se ejecuta este programa se ubica a partir de la dirección 00000000 del espacio de direcciones físico, y la variable A del programa se encuentra en la última página física, cuál es el tamaño de la memoria física expresado en megabytes.

0,5

```
128
```

A

En la Figura siguiente se muestra la estructura de la memoria *cache* de un computador, cuya memoria principal es de 256 *bytes*. En la figura se muestra el estado de la *cache* en un momento dado, así como el contenido de algunas de posiciones de la memoria principal.



Teniendo en cuenta toda esta información, contesta a las preguntas A y B:

— A) Partiendo del estado representado en la figura, si la CPU emite la dirección 7Eh, ¿qué información se almacenaría en la etiqueta y en los cuatro bytes del bloque 3 de la cache? (En el campo etiqueta, contesta con cuatro bits. En los campos OFFSET, contesta con dos dígitos hexadecimales.)

0,5

Etiqueta: 0111
OFFSET 0: 81
OFFSET 1: AB
OFFSET 2: ED
OFFSET 3: 12

— B) Sí a partir del estado de la *cache* representado en la figura, la CPU emite las direcciones B3h, C8h, CBh y C2h (siguiendo el orden dado, o sea, primero B3h, luego C8h, etc.), determina cuáles de ellas provocarían FALLO de *cache*.

0,5

C8h y C2h

— Indica y define brevemente los diferentes estados por los que puede pasar un proceso durante su ejecución.

0,5

Nuevo: Estado del proceso durante su creación.

Listo: El proceso está esperando ser asignado al procesador para su ejecución.

En ejecución: El proceso tiene la CPU y ésta ejecuta sus intrucciones.

En espera: El proceso está esperando a que ocurra algún sucero, como por ejemplo la terminación de una operación de E/S.

Terminado: Estado del proceso durante su eleminación

— Indica brevemente los diferentes tipos de mecanismos utilizados por la paginación para llevar a cabo la protección de memoria.

0,5

Mecanismo 1:
Bit R/W para determinar el tipo de acceso permitido para cada página (sólo lectura o bien lectura/escritura)

Mecanismo 2:
Bit U/S para establecer el privilegio necesario para acceder a cada página

Mecanismo 3:
El uso de una tabla de página en cada proceso evita que un proceso pueda acceder a la memoria utilzia por otro proceso.

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

Examen de Arquitectura de Computadores. (Telemática.) **Convocatoria de febrero: 10-02-2004**

A continuación se muestra el listado de un programa cuyo objetivo es poner en ejecución a otros programas utilizando la función *CreateProcess()*. Sin embargo, esta función requiere muchos parámetros, la mayoría de los cuales suelen tomar siempre el mismo valor. Debido a ello se ha decidido encapsular la función *CreateProcess()* mediante otra función, llamada *AbreProceso()*, que recibe sólo tres parámetros: la dirección de una estructura del tipo STARTUPINFO, la dirección de otra estructura del tipo PROCESS_INFORMATION y la dirección de una cadena indicando el nombre del programa que se desea ejecutar. La función *AbreProceso()* abre el nuevo programa asignándole una nueva ventana de tipo consola. En la función *main()* del programa se llama a *AbreProceso()* para ejecutar un programa cuyo nombre se pide por pantalla.

Datos adicionales: estructura PROCESS_INFORMATION

```
typedef struct _PROCESS_INFORMATION
    HANDLE hProcess;
    HANDLE hThread;
    DWORD dwProcessId;
    DWORD dwThreadId;
} PROCESS_INFORMATION;
```

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

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

main()
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    char NombrePrograma[80];

    // Pedir por pantalla en nombre del programa a ejecutar
    printf("Nombre programa: ");
    scanf("%s", NombrePrograma);


    // Inicializacion de la STARTUPINFO (mostrada parcialmente)
    si.cb = sizeof(si);
    (...)

    // Apertura del proceso y mensaje indicando si ha tenido exito
    // o no la apertura

    if ( AbreProceso(&si, &pi, NombrePrograma) )   0,25
       printf("Apertura de proceso OK\n\n");

       // Sacar por pantalla el PID del proceso abierto
       printf("PID = %d\n", pi.dwProcessId);   0,25
    else
       printf("Fallo en la apertura de proceso\n\n");
}

int AbreProceso( STARTUPINFO *pStUpIn, PROCESS_INFORMATION *pPrInf,
                 char *Nombre )
{
    int ExitoApertura;

    ExitoApertura = CreateProcess(
                    NULL,                  // pszApplicationName
                    Nombre,                // pszCommandLine
                    NULL,                  // psaProcess
                    NULL,                  // psaThread
                    TRUE,                  // bInheritHandles
                    CREATE_NEW_CONSOLE ,   // fdwCreate
                    NULL,                  // pvEnvironment
                    NULL,                  // pszCurDir
                    pStUpIn,               // psiStartInfo
                    pPrInf );              // ppiProcInfo
                                                          0,5
    return(ExitoApertura);
}
```

A

— Un sistema de memoria virtual paginada utiliza direcciones virtuales de 32 bits, direcciones físicas de 24 bits y tamaño de página de 16Kbytes. Además, se sabe que la memoria instalada en el computador ocupa el 25% del espacio de direcciones físico, y que se ubica a partir de la dirección 000000 de éste. En un momento dado la unidad de ejecución emite la dirección A1347FABh. Si se sabe que esta dirección se encuentra mapeada en la última página de la memoria física, ¿cuál será la dirección física generada por la MMU de la CPU? Contesta con 6 dígitos hexadecimales.

0,5

3FFFAB

— Indica los diferentes tipos de excepciones disponibles en la arquitectura IA-32 y define brevemente su cometido:

0,5

Fallos: Se utilizan para señaliar errores no catastróficos que pueden ser tratados sin que se pierda la estabilidad del sistema.

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

Trampas: Se utilizan para transferir el control a rutinas de depuración.