

# SISTEMAS BASADOS EN MICROPROCESADORES

## Grado en Ingeniería Informática Escuela Politécnica Superior – UAM

### COLECCIÓN DE PROBLEMAS DE LOS TEMAS 2.7 A 5.4

**P1.** Si **SP=0006h** y **FLAGS=0210h** al inicio de la ejecución del código que se adjunta, indicar los valores contenidos en las **primeras seis posiciones de la pila** al ejecutar la primera instrucción del procedimiento `Leer_Datos`, tanto cuando todos los procedimientos del programa son cercanos (**NEAR**), como cuando son lejanos (**FAR**). La pila está inicializada a ceros.

```
2100:2250 E8A8FD    call Leer_Datos
2100:2253 89161000    mov Datos[0], dx
```

0	1	2	3	4	5
0	0	0	0	53h	22h

*Caso NEAR*

0	1	2	3	4	5
0	0	53h	22h	0	21h

*Caso FAR*

**P2.** Al inicio de la ejecución de una función invocada desde lenguaje C, se tiene que **SP=2** y que las 16 primeras posiciones de la pila contienen los siguientes valores:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
11h	A0h	25h	00h	32h	00h	A2h	E9h	00h	C1h	24h	F1h	00h	63h	41h	12h

La signatura de dicha función es: `int fun ( char* p, long n );`

Indicar el valor de los tres parámetros con que esa función fue invocada desde C, tanto cuando todas las direcciones son cercanas (**NEAR**), como cuando son lejanas (**FAR**).

*Caso NEAR:*     `p = 0032h`                      `n = C100E9A2h`

*Caso FAR:*        `p = C100:E9A2h`     `n = 6300F124h`

**P3.** Escribir en ensamblador el código necesario para **poner a 1 los bits 5, 10 y 14 del registro AX**, dejando todos los demás bits de ese registro intactos, y **poner a 0 los bits 5, 10 y 14 del registro BX**, dejando intactos los demás bits. Se valorará la eficiencia del código.

--- -- 0100010000010000001                      44201

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

Cartagena99

**P4.** Usando los procedimientos lejanos `enviar0` y `enviar1`, escribir en ensamblador un procedimiento eficiente que envíe secuencialmente los bits del **registro AL**, desde el más significativo al menos significativo. Se valorará la eficiencia del código.

```

enviarAL PROC

    push cx

    mov cx, 8          ; Itera los ocho bits de AL

bucle:    rcl al, 1    ; Pasa el bit más alto de AL al acarreo
          jc envial    ; Si hay acarreo envía 1, si no envía 0

          call enviar0
          jmp finbucle

envial:    call enviar1

finbucle:  dec cx
          jnz bucle

          rcl al, 1    ; Deja AL igual que al principio

    pop cx
    ret

enviarAL ENDP

```

**P5.** Al inicio de la ejecución de una función invocada desde lenguaje C, se tiene que **SP=0** y que las 16 primeras posiciones de la pila contienen los siguientes valores:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
11h	A0h	25h	00h	32h	00h	A2h	E9h	00h	C1h	24h	F1h	00h	63h	41h	12h

La signatura de dicha función es: `int fun ( char c, int n, char* p );`

Indicar el valor de los tres parámetros con que esa función fue invocada desde C, tanto cuando todas las direcciones son cercanas (**NEAR**), como cuando son lejanas (**FAR**).

*Caso NEAR:*    `c = 25h`            `n = 0032h`            `p = E9A2h`

---

*Caso FAR:*        `c = 32h`            `n = E9A2h`            `p = F124:C100h`

---



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP:689 45 44 70

```
4253:0007 E8F6FF call _fun
4253:000A B8004C mov ax, 4C00h
```

```
fun ( int n, char c, char* p );
```

Indicar el valor de las 16 posiciones iniciales de la pila en el momento de ejecutarse la primera instrucción de código máquina de la función `fun`, tanto cuando todas las direcciones son cercanas (**NEAR**), como cuando son lejanas (**FAR**).

**Caso NEAR:**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						0Ah	00h	34h	12h	ABh	00h	78h	56h		

**Caso FAR:**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		0Ah	00h	53h	42h	34h	12h	ABh	00h	78h	56h	53h	42h		

**P7.** Escribir en ensamblador la función `_Multiply8U`, que **multiplica dos enteros sin signo de 8 bits mediante sumas sucesivas**. El primer operando ha de estar almacenado en **BH** y el segundo en **BL**. El **resultado ha de ser un entero sin signo de 16 bits** que se retornará en **AX**. La multiplicación se realizará sumando el primer operando tantas veces como indique el segundo. Se valorará la eficiencia del código.

```
_Multiply8U proc near
    mov ax, 0

    cmp bh, 0
    je fin      ; Primer operando es cero
    cmp bl, 0
    je fin      ; Segundo operando es cero

    push bx dx

    ; Pasa primer operando a dx
    mov dl, bh
    mov dh, 0

    ; Suma primer operando (dx) tantas veces como
    ; indica el segundo (bl)
seguir: add ax, dx

    dec bl
```



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

**P8.** Llamando a la función de multiplicar desarrollada en el problema anterior, escribir en ensamblador la función de C que se reproduce en el siguiente recuadro, que calcula el producto escalar de dos vectores de **n** dimensiones cuyos elementos son enteros sin signo de 8 bits. Las variables locales están almacenadas en registros. Se supone que el programa de C está compilado en **modelo compacto**. Se valorará la eficiencia del código.

```
_DotProd8U proc near
```

```
    push bp
    mov bp, sp
    push bx cx dx si di ds es

    mov dx, 0    ; dx = res
```

```
    ; Salta @retorno (2 bytes por ser código NEAR) y bp (2 bytes)
    mov cx, [bp+4]    ; cx := n
    cmp cx, 0
    je fin           ; n es cero
```

```
    ; Punteros v1 y v2 ocupan 4 bytes por ser datos FAR
```

```
    lds si, [bp+6]    ; ds:si := v1
    les di, [bp+10]   ; es:di := v2
```

```
seguir: mov bh, [si]    ; bh := v1[i]
        mov bl, es:[di] ; bl := v2[i]
        call _Multiply8U ; ax := v1[i] * v2[i]
        add dx, ax      ; dx := dx + v1[i] * v2[i]
```

```
    ; i := i+1
    inc si
    inc di
```

```
    dec cx
    jnz seguir
```

```
fin:    mov ax, dx
        pop es ds di si dx cx bx
        pop bp
```

```
    ret                ; Devuelve resultado en ax
```

```
_DotProd8U endp
```

```
int DotProd8U (int n, char *v1, char *v2)
{
    register int i;
    register int res=0;

    for (i=0; i<n; i++)
        res=res + Multiply8U( v1[i], v2[i] );

    return res;
}
```

**P9.** La función de lenguaje C cuya signatura se indica en el recuadro de la derecha es invocada desde el programa de código máquina que se muestra en el recuadro de la izquierda. En el

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

---

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70**



Indicar el valor de las 16 posiciones iniciales de la pila en el momento de ejecutarse la primera instrucción de código máquina de la función `fun`, tanto cuando todas las direcciones son cercanas (**NEAR**), como cuando son lejanas (**FAR**).

**Caso NEAR:**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
								0Dh	FFh	65h	87h	21h	43h	12h	00h

**Caso FAR:**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				0Dh	FFh	42h	53h	65h	87h	34h	12h	21h	43h	12h	00h

**P10.** Indicar el vector de la interrupción de **impresión por pantalla** dado el siguiente volcado de memoria.

```
0000:0000 54 02 CF 15 CE 01 CF 15 04 00 70 00 D7 01 CF 15
0000:0010 04 00 70 00 30 00 00 C8 30 00 00 C8 30 00 00 C8
```

Segmento = **C800h**      Offset = **0030h**

**P11.** Escribir en ensamblador el código necesario para **poner a 1 los bits 0, 7 y 14 del registro AX**, dejando todos los demás bits de ese registro intactos, y **poner a 0 los bits 2, 10 y 15 del registro BX**, dejando intactos los demás bits. Se valorará la eficiencia del código.

```
or ax, 0100000010000001b                   ; 4081h
and bx, 0111101111111011b                 ; 7BFBh
```

**P12.** Escribir en ensamblador utilizando instrucciones básicas (sin instrucciones de manipulación de cadenas ni de bucles) la función `strlen` de C, cuya signature se reproduce a continuación. Esta función retorna la longitud de la cadena de caracteres que recibe como argumento. Dicha cadena acaba con un byte a cero. Se supone que el programa de C está compilado en **modelo largo**. Se valorará la eficiencia del código.

```
int strlen (char *s);
```

`_strlen PROC far`

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70



```

itera:  cmp BYTE PTR [bx], 0 ; Test de final de cadena
        je fin
        inc ax                ; Incrementa contador de caracteres
        inc bx                ; Apunta al siguiente carácter
        jmp itera

fin:    pop bx ds
        pop bp

        ret

_strlen ENDP

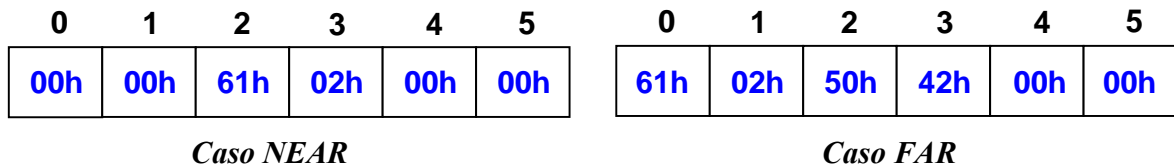
```

**P13.** Si **SP=0004h** y **FLAGS=0200h** al inicio de la ejecución del código que se adjunta, indicar los valores contenidos en las **primeras seis posiciones de la pila** al ejecutar la primera instrucción del procedimiento **Leer\_Datos**, tanto cuando todos los procedimientos del programa son cercanos (**NEAR**), como cuando son lejanos (**FAR**). Se considera que la pila está inicializada a ceros.

```

4250:025E E8A8FD    call Leer_Datos
4250:0261 89161000    mov Datos[0], dx

```

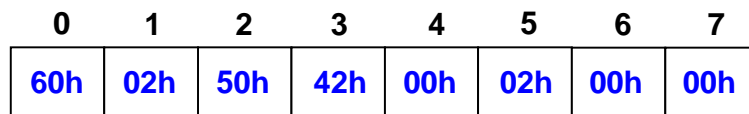


**P14.** Si **SP=0006h** y **FLAGS=0200h** al inicio de la ejecución del código que se adjunta, indicar los valores contenidos en las **primeras ocho posiciones de la pila** en el momento de ejecutar la primera instrucción de la rutina de servicio de la interrupción 61h. Se considera la pila inicializada a ceros.

```

4250:025E CD61      int 61h
4250:0260 89161000    mov Datos[0], dx

```



**P15.** Escribir en ensamblador un programa residente asociado a la interrupción 65h, que ejecute un retardo igual a **65536\*N iteraciones**, con **N** siendo el valor recibido en el **registro AX**.



**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE**  
**LLAMA O ENVÍA WHATSAPP: 689 45 44 70**  
 ---  
**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS**  
**CALL OR WHATSAPP:689 45 44 70**

```

                                push ax cx
iteral:  xor cx, cx
itera2:  dec cx
                                jnz itera2

                                dec ax
                                jnz iteral

                                pop cx ax

                                iret

retardo ENDP

instalar: xor ax, ax
                                mov es, ax
                                mov ax, offset retardo
                                mov bx, cs

                                cli
                                mov es:[65h*4], ax
                                mov es:[65h*4+2], bx
                                sti

                                ; Deja residente el procedimiento de retardo
                                mov dx, offset instalar
                                int 27h

codigo ENDS
END inicio

```

**P16.** Escribir en ensamblador un procedimiento lejano, **suma32**, que sume las variables globales de 32 bits **op1** y **op2**, dejando el resultado en la variable global de 32 bits **res**.

```

op1 dd ?          suma32 PROC FAR
op2 dd ?
res dd ?

                                push ax
                                push si

                                mov si, 0

                                ; Suma las dos palabras de menor peso

                                mov ax, WORD PTR op1[si]
                                add ax, WORD PTR op2[si]
                                mov WORD PTR res[si], ax

                                ; Pasa a apuntar a las palabras de mayor peso

                                inc si
                                inc si

```



Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

```

ret
suma32 ENDP

```

**P17.** Si **SP=0006h** y **FLAGS=0210h** al inicio de la ejecución del código que se adjunta, indicar los valores contenidos en las **primeras seis posiciones de la pila** al ejecutar la primera instrucción del procedimiento **Leer\_Datos**, tanto cuando todos los procedimientos del programa son cercanos (**NEAR**), como cuando son lejanos (**FAR**). La pila está inicializada a ceros.

```

2100:2250 E8A8FD call Leer_Datos
2100:2253 89161000 mov Datos[0], dx

```

0	1	2	3	4	5
0	0	0	0	53h	22h

*Caso NEAR*

0	1	2	3	4	5
0	0	53h	22h	0	21h

*Caso FAR*

**P18.** Indicar el vector de la interrupción de **punto de ruptura** (*breakpoint*) dado el siguiente volcado de memoria.

```

0000:0000 54 02 CF 15 CE 01 CF 15 04 00 70 00 D7 01 CF 15
0000:0010 04 00 70 00 30 00 00 C8 30 00 00 C8 30 00 00 C8

```

Segmento = 15CFh      Offset = 01D7h

**P19.** Se tiene una matriz bidimensional de tamaño (**FILAS** x **COLUMNAS**) almacenada por filas en la variable **Matriz2D**. Escribir en ensamblador un procedimiento lejano, **escribelcol**, que reciba la **dirección de la matriz en el registro BX** y ponga a **uno** todos los elementos de la **columna indicada en el registro AX**. Se valorará la eficiencia del código.

```

FILAS = 10
COLUMNAS = 20
Matriz2D db FILAS*COLUMNAS dup (?)

mov bx, offset Matriz2D
mov ax, 4
call escribelcol      ; Pone a 1 los elementos de la columna 4
                       ; de Matriz2D

```

**escribelcol PROC FAR**

**push ax si**

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP:689 45 44 70





```

    dec cx
    jnz buclecol

    pop si, cx
    ret

escribelcol ENDP

```

**P20.** Suponiendo que **SS=424Dh**, **SP=14**, **AX=3412h** y **BX=5678h**, indicar el **valor hexadecimal de los 16 primeros bytes del segmento SS** una vez ejecutado el siguiente programa.

```

    push AX
    pop BX
    push SS
    push BX

```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
										12h	34h	4Dh	42h		

**P21.** Declarar mediante directivas de ensamblador de 8086 las mismas variables que aparecen en el siguiente extracto en lenguaje C, teniendo en cuenta que las cadenas de caracteres en C acaban con el byte 0.

```

char nombre[20];           // Cadena de caracteres de 20 bytes
short edad = 17;          // Entero de 2 bytes inicializado
char tabla2D[10][2];      // Tabla de bytes de 10 filas por 2 columnas
short valores[5] = { 1, 2, 3, 4, 5 };
char despedida[20] = "Hasta luego";

_nombre db 20 dup (?)
_edad dw 17
_tabla2D db 10*2 dup (?)
_valores dw 1, 2, 3, 4, 5
_despedida db "Hasta luego", 0, 20-($-despedida) dup (?)

```

**P22.** El siguiente programa en lenguaje ensamblador de 8086, que debe **invertir el orden de los caracteres de una cadena dada de 512 bytes como máximo**, tiene varios errores. Proponer una versión correcta del mismo programa haciendo el **menor número de cambios**. Sólo es necesario reescribir las líneas erróneas.



CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP:689 45 44 70

```

datos segment
    cadena    dw  "Hola"
    longitud  db  cadena-$
datos ends

resultados segment
    resultado db  200 dup (?)
resultados ends

codigo segment
    assume cs:codigo, ds:datos
    invertir proc far
        mov ax, datos
        mov ds, ax
        mov ax, resultado
        mov es, ax
        mov si, longitud
        mov di, 0
seguir:    mov al, cadena[si-1]
            mov resultado[di], al
            dec si
            inc di
            jz seguir
            mov ax, 4C00h
            int 21h
    invertir endp
codigo ends
end codigo

```

```

datos segment
    cadena    db  "Hola"
    longitud  dw  $-cadena
datos ends

resultados segment
    resultado db  200h dup (?)
resultados ends

codigo segment
    assume cs:codigo, ds:datos, es:resultados
    invertir proc far
        mov ax, datos
        mov ds, ax
        mov ax, resultados
        mov es, ax
        mov si, longitud
        mov di, 0
seguir:    mov al, cadena[si-1]
            mov resultado[di], al
            inc di
            dec si
            jnz seguir
            mov ax, 4C00h
            int 21h
    invertir endp
codigo ends
end invertir

```

**P23.** Escribir en ensamblador un procedimiento lejano (descontar2\_32) que **decremente en dos unidades** el valor de la variable de 32 bits cuya dirección recibe mediante los registros AX y BX tal como se indica en el código adjunto. Tras su ejecución, este procedimiento no deberá alterar los valores previos de ningún registro del banco general ni de segmento. Se valorará la eficiencia del código.

```

datos segment
    contador    dd  0FFFFFFFFh
datos ends

...

mov ax, OFFSET contador
mov bx, SEG contador

call descontar2_32

```

descontar2\_32 PROC far

```

push bx es

mov es, bx
mov bx, ax

```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

Cartagena99

descontar2\_32 ENDP

**P24.** Al inicio de la ejecución de una función invocada desde lenguaje C, se tiene que **SP=2** y que las 16 primeras posiciones de la pila contienen los siguientes valores:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
11h	A0h	25h	00h	32h	00h	A2h	E9h	00h	C1h	24h	F1h	00h	63h	41h	12h

La signatura de dicha función es: `int fun ( char* p, long n );`

Indicar el valor de los tres parámetros con que esa función fue invocada desde C, tanto cuando todas las direcciones son cercanas (**NEAR**), como cuando son lejanas (**FAR**).

**Caso NEAR:**         p = 0032h                      n = C100E9A2h    

**Caso FAR:**         p = C100:E9A2h                      n = 6300F124h    

**P25.** Si **SP=0004h** y **FLAGS=0210h** al inicio de la ejecución del código que se adjunta, indicar los valores contenidos en las **primeras seis posiciones de la pila** al ejecutar la primera instrucción del procedimiento `Leer_Datos`, tanto cuando todos los procedimientos del programa son cercanos (**NEAR**), como cuando son lejanos (**FAR**). La pila está inicializada a ceros.

```
20FF:4000 E8A8FD     call Leer_Datos
20FF:4003 89161000    mov Datos[0], dx
```

0	1	2	3	4	5
0	0	03h	40h	0	0

**Caso NEAR**

0	1	2	3	4	5
03h	40h	FFh	20h	0	0

**Caso FAR**

**P26.** Declarar mediante directivas de ensamblador de 8086 las mismas variables que aparecen en el siguiente extracto en lenguaje C, teniendo en cuenta que las cadenas de caracteres en C acaban con el byte 0 y que el tipo `short` ocupa 2 bytes.

```
char caracter = 'A';
short edad;
short tabla[100];
char valores[5] = { 1, 2, 3, 4, 5 };
char despedida[12] = "Hasta luego";
```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

**P27.** El siguiente programa en lenguaje ensamblador de 8086, que debe **contar el número de caracteres de una cadena dada de 512 bytes como máximo**, tiene varios errores. Proponer una versión correcta del mismo programa haciendo el **menor número de cambios**. Sólo es necesario reescribir las líneas erróneas.

```

datos segment
    cadena db "Hola", 0
datos ends

resultados segment
    resultado db 2 dup(?)
resultados ends

codigo segment
    assume cs:codigo, ds:datos,
es:resultados

    contar proc far
        mov ax, cadena
        mov ds, ax
        mov ax, resultados
        mov es, ax
        mov si, 4
seguir: mov cadena[si], 0
        jz fin
        dec si
        jmp fin
fin:    mov resultado, si
        mov ax, 4C00h
        int 21h
    contar endp
codigo ends
end contar

```

```

datos segment
    cadena db "Hola", 0
datos ends

resultados segment
    resultado db 2 dup(?)
resultados ends

codigo segment
    assume cs:codigo, ds:datos, es:resultados

    contar proc far
        mov ax, datos
        mov ds, ax
        mov ax, resultados
        mov es, ax
        mov si, 0
seguir: cmp cadena[si], 0
        jz fin
        inc si
        jmp seguir
fin:    mov WORD PTR resultado, si
        mov ax, 4C00h
        int 21h
    contar endp
codigo ends
end contar

```

**P28.** Escribir en ensamblador un procedimiento lejano (contar4\_48) que **incremente en cuatro unidades** el valor de la variable de 48 bits cuya dirección recibe mediante los registros AX y BX tal como se indica en el código adjunto. Tras su ejecución, este procedimiento no deberá alterar los valores previos de ningún registro del banco general ni de segmento. Se valorará la eficiencia del código.

```

datos segment
    contador db 6 dup(0)
datos ends

...

mov ax, OFFSET contador
mov bx, SEG contador

call contar4_48

```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

Cartagena99

```

mov bx, ax

add WORD PTR es:[bx], 4
adc WORD PTR es:[bx+2], 0
adc WORD PTR es:[bx+4], 0

pop es bx

ret

contar4_48 ENDP

```

**P29.** Si **SP=000Ah** y **FLAGS=1234h** al inicio de la ejecución del código que se adjunta, indicar los valores contenidos en las **primeras dieciséis posiciones de la pila** en el momento de ejecutar la primera instrucción de la rutina de servicio de la interrupción 1Ch. Los valores desconocidos deben dejarse en blanco.

```

549A:025E CD1C      int 1Ch
549A:0260 89161000  mov Datos[0], dx

```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				60h	02h	9Ah	54h	34h	12h						

**P30.** Al inicio de la ejecución de una función invocada desde lenguaje C, se tiene que **SP=4** y que las 16 primeras posiciones de la pila contienen los siguientes valores:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
11h	A0h	25h	00h	32h	00h	A2h	E9h	00h	C1h	24h	F1h	00h	63h	41h	12h

La signatura de dicha función es: `int fun ( int *p, char c, int n );`

Indicar el valor de los tres parámetros con que esa función fue invocada desde C, tanto cuando todas las direcciones son cercanas (**NEAR**), como cuando son lejanas (**FAR**).

**Caso NEAR:**         p = E9A2h                      c = 00h                      n = F124h

**Caso FAR:**         p = F124h:C100h                      c = 00h                      n = 1241h

**P31.** Escribir en ensamblador de 8086 un procedimiento cercano denominado `_Instalar_61h` que modifique el **vector de la interrupción 61h** con la dirección de otro procedimiento denominado `_RSI_61h`. El valor anterior de ese vector de interrupción debe almacenarse previamente en una única variable que deberá declararse dentro del propio procedimiento

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP:689 45 44 70

```

inicio: push ax es
        mov ax, 0
        mov es, ax
        mov ax, es:[61h*4]           ; Guarda Offset de rsi anterior
        mov cs:rsi61, ax
        mov ax, es:[61h*4 + 2]       ; Guarda Segmento de rsi anterior
        mov cs:rsi61[2], ax

        ; Cambia rsi de 61h
        cli
        mov word ptr es:[61h*4], offset _RSI_61h
        mov word ptr es:[61h*4 + 2], seg _RSI_61h
        sti

        pop es ax
        ret

```

`_Instalar_61h ENDP`

**P32.** Escribir en ensamblador de 8086 utilizando instrucciones básicas (sin instrucciones de manipulación de cadenas ni de bucles) la función `strcmp` de C, cuya signatura se reproduce a continuación. Esta función retorna un entero que indica la relación entre las dos cadenas que recibe como argumentos: Un valor de cero indica que ambas cadenas son iguales. Un valor mayor que cero indica que el primer carácter que no coincide tiene un valor mayor en `str1` que en `str2`. Un valor menor que cero indica lo contrario. Se considera que el programa en C está compilado en **modelo pequeño** (*small*). Las cadenas de caracteres en C acaban con un cero. Se valorará la eficiencia del código.

```
int strcmp (const char *str1, const char* str2);
```

```

_strcmp PROC NEAR
        push bp
        mov bp, sp
        push bx si di

        mov si, bp[4]           ; si <= str1
        mov di, bp[6]           ; di <= str2

        mov ax, 0               ; Por defecto son iguales

continuar: mov bl, [si]           ; bl <= str1[i]
          cmp bl, [di]           ; str1[i] - str2[i]
          je iguales
          ja str1_mayor
          mov ax, -1              ; str1 es menor que str2
          jmp final
str1_mayor: mov ax, 1              ; str1 es mayor que str2
          imb final

```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

Cartagena99

```

final:      pop di si bx bp
           ret
_strncmp   ENDP

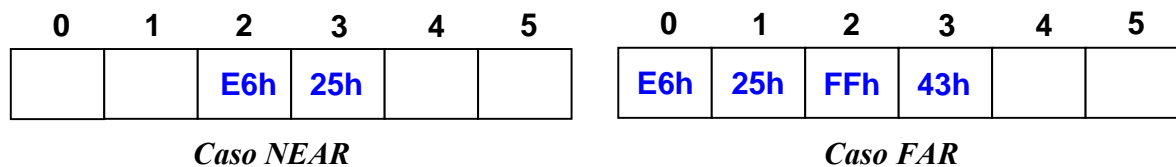
```

**P33.** Si **SP=0004h** y **FLAGS=1234h** al inicio de la ejecución del código que se adjunta, indicar los valores contenidos en las **primeras seis posiciones de la pila** al ejecutar la primera instrucción del procedimiento **Leer\_Datos**, tanto cuando todos los procedimientos del programa son cercanos (**NEAR**), como cuando son lejanos (**FAR**). Los valores desconocidos de la pila deben dejarse en blanco.

```

43FF:25E3 E8A8FD   call Leer_Datos
43FF:25E6 89161000  mov Datos[0], dx

```

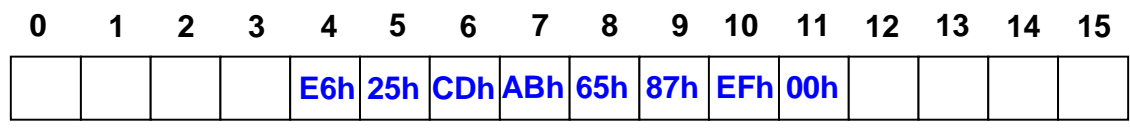


**P34.** La función de lenguaje C cuya signatura se indica en el recuadro de la derecha es invocada desde el programa de código máquina que se muestra en el recuadro de la izquierda. En el momento anterior de la llamada, se suponen los siguientes valores del puntero de pila y de los parámetros de la función: **FLAGS=1234h**, **SP = 12**, **n = ABCDh**, **p = 1234h:8765h**, **c = EFh**,

<pre> 43FF:25E3 E8F6FF   call _fun 43FF:25E6 B8004C   mov ax, 4C00h </pre>	<pre> fun ( int n, int* p, char c ); </pre>
--	---

Indicar el valor de las 16 posiciones iniciales de la pila en el momento de ejecutarse la primera instrucción de código máquina de la función **fun**, tanto cuando todas las direcciones son cercanas (**NEAR**), como cuando son lejanas (**FAR**). Los valores desconocidos de la pila deben dejarse en blanco.

*Caso NEAR:*



*Caso FAR:*



**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE**  
**LLAMA O ENVÍA WHATSAPP: 689 45 44 70**  
 ---  
**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS**  
**CALL OR WHATSAPP:689 45 44 70**

**P35.** Escribir en ensamblador de 8086 un procedimiento lejano `Abrir_Lectura` que invoque a la función `fopen` de la librería de C, cuya signatura se reproduce a continuación. Se considera que la librería de C está compilada en **modelo largo** (*large*). El **nombre del fichero y el modo de apertura están almacenados como cadenas ASCIIZ en las variables globales `fichero` y `modo`** respectivamente. El procedimiento ha de **almacenar en la variable global `descriptor` el descriptor de fichero retornado por `fopen`**. Las tres variables globales son accesibles a través del registro **DS**. Se valorará la eficiencia del código.

```

FILE * fopen ( const char * filename, const char * mode );

fichero db "datos.csv", 0, 256 dup (?)
modo db "r", 0
descriptor dw ?, ?

extrn _fopen:FAR

Abrir_Lectura PROC FAR

    push ax dx

    ; Apila dirección larga de modo
    push ds                    ; Apila segmento de modo
    mov ax, offset modo
    push ax

    ; Apila dirección larga de fichero
    push ds                    ; Apila segmento de modo
    mov ax, offset fichero
    push ax

    call _fopen

    add sp, 8                  ; Elimina parámetros de la pila

    ; fopen retorna en dx:ax la dirección larga del descriptor
    mov ds:descriptor, ax
    mov ds:descriptor[2], dx

    pop dx ax
    ret
Abrir_Lectura ENDP

```

**P36.** Escribir en ensamblador de 8086 un procedimiento lejano `Nombre_Fichero_C` que **almacene en la variable `fichero` del problema anterior el nombre de un fichero pasado como primer argumento** en una invocación del programa tal como la mostrada a continuación. El nombre del fichero ha de almacenarse en la variable como cadena ASCIIZ. La variable global es

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

Cartagena99



```

Nombre_Fichero_C PROC FAR
    push bx si di

    ; 82h = Offset de ler carácter de nombre fichero en PSP
    mov si, 82h
    mov ax, 0    ; Por defecto no tiene extensión .c
    mov di, 0    ; Índice a la cadena fichero

busca_punto: mov bl, es:[si]
             cmp bl, 13
             je final          ; Encuentra final de línea (13)
             mov ds:fichero[di], bl ; Copia carácter a fichero
             inc si
             inc di
             cmp bl, '.'
             jne busca_punto

             ; registro SI apunta después del punto
hay_punto:  cmp byte ptr es:[si], 'c'
             jne final          ; No es ".c"
             mov bl, es:[si+1] ; Lee carácter después de .c
             cmp bl, ' '
             je hay_punto_c     ; Espacio en blanco después de .c
             cmp bl, 13
             jne final          ; Otro carácter tras .c (no es .c)

hay_punto_c: mov ds:fichero[di], 'c'
             mov ds:fichero[di+1], 0 ; Guarda final de cadena
             mov ax, 1

final:      pop di si bx
             ret

Nombre_Fichero_C ENDP

```

**P37.** Al inicio de la ejecución de una función invocada desde lenguaje C, se tiene que **SP=8** y que las 16 primeras posiciones de la pila contienen los siguientes valores:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
11h	A0h	25h	00h	32h	00h	A2h	E9h	00h	C1h	24h	00h	63h	00h	41h	12h

La signatura de dicha función es: `int fun ( char p, int n );`

Indicar el valor de los dos parámetros con que esa función fue invocada desde C, tanto cuando todas las direcciones son cercanas (**NEAR**), como cuando son lejanas (**FAR**).



**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

---

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70**

**P38.** Escribir en ensamblador de 80x86 utilizando instrucciones básicas (sin instrucciones de manipulación de cadenas ni de bucles) la función `sumatorio` de C cuyo código se reproduce a continuación. Esta función calcula de forma recursiva el sumatorio de los `n` primeros números naturales, con `n` siendo el entero de 32 bits que recibe como argumento. Se supone que la función no detecta desbordamiento del resultado y que el programa en C está compilado en **modelo largo**. Se valorará la eficiencia del código.

```

long sumatorio( long n )
{
    if (n == 1) return 1;
    else return n + sumatorio( n-1 );
}

_sumatorio PROC FAR

    push bp
    mov bp, sp

    ; Accede a parámetro de entrada de 32 bits (n)
    mov ax, [bp+6]    ; AX <= Parte baja de n
    mov dx, [bp+8]    ; DX <= Parte alta de n

    cmp dx, 0
    jne noes1        ; n != 1
    cmp ax, 1
    je final        ; n == 1 => Retorna 1 en DX:AX

    ; n != 1

noes1:    ; Decrementa n

    dec ax          ; Decrementa parte baja
    sbb dx, 0       ; Resta acarreo (borrow) a parte alta

    ; Apila n-1 (parte alta primero) y llama recursivamente

    push dx ax
    call _sumatorio ; Llamada recursiva
    add sp, 4       ; Reequilibra la pila

    ; sumatorio( n-1 ) retornado en DX:AX

    ; DX:AX := DX:AX + n

    add ax, [bp+6]  ; Suma parte baja
    adc dx, [bp+8]  ; Suma parte alta y acarreo

final:    pop bp
          ret

```



Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

**P39.** Al inicio de la ejecución de una función invocada desde lenguaje C, se tiene que **SP=0** y que las 16 primeras posiciones de la pila contienen los siguientes valores:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
11h	A0h	25h	00h	32h	00h	A2h	E9h	00h	C1h	24h	00h	63h	00h	41h	12h

La signatura de dicha función es: `int fun ( long p, int n );`

Indicar el valor de los tres parámetros con que esa función fue invocada desde C, tanto cuando todas las direcciones son cercanas (**NEAR**), como cuando son lejanas (**FAR**).

**Caso NEAR:**     p = **00320025h**   n = **E9A2h**

**Caso FAR:**     p = **E9A20032h**   n = **C100h**

**P40.** Escribir en ensamblador de 80x86 utilizando instrucciones básicas (sin instrucciones de manipulación de cadenas ni de bucles) la función `maximo` de C cuyo código se reproduce a continuación. Esta función determina el valor máximo de una tabla de enteros con signo que recibe como argumento. Se supone que el programa en C está compilado en **modelo pequeño (small)**. Se valorará la eficiencia del código.

```

                                int maximo( int n, int *tabla )
                                {
                                    int i, max;
                                    max = tabla[0];
                                    for (i=1; i<n; i++)
                                        if (tabla[i] > max)
                                            max = tabla[i];
                                    return max;
                                }

_maximo PROC NEAR

    push bp
    mov bp, sp
    push bx cx

    mov bx, [bp+6]    ; bx == &tabla
    mov ax, [bx]     ; ax == max
    mov cx, 1        ; cx == i

bucle:    cmp cx, [bp+4];    ; ¿i == n?
           je final

           add bx, 2
           cmp [bx], ax    ; ¿tabla[i] > max?
           jle no_maximo  ; tabla[i] <= max

           mov ax, [bx]    ; max = tabla[i]

no_maximo: inc cx          ; i++
           jmp bucle

final:    pop cx bx

```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

Cartagena99

**P41.** Suponiendo que **SP=8** y que las primeras 16 posiciones del segmento de pila contienen los siguientes valores:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FFh	A0h	25h	00h	32h	00h	A2h	E9h	00h	C1h	24h	F1h	00h	63h	41h	12h

Indicar el valor de los cuatro registros después de la ejecución del siguiente programa.

```

pop AX      AX = C100h  BX = F124h  CX = C100h  DX = 6300h
pop BX
push AX
pop CX
pop DX

```

**P42.** Al inicio de la ejecución de una función invocada desde lenguaje C, se tiene que **SP=0** y que las 16 primeras posiciones de la pila contienen los siguientes valores:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
11h	A0h	25h	00h	32h	00h	A2h	E9h	00h	C1h	24h	F1h	00h	63h	41h	12h

La signatura de dicha función es: `int fun( char c, int n, char *s );`

Indicar el valor de los tres parámetros con que esa función fue invocada desde C, tanto cuando todas las direcciones son cercanas (**NEAR**), como cuando son lejanas (**FAR**).

*Caso NEAR:*     `c = 25h`     `n = 0032h`     `s = E9A2h`

*Caso FAR:*     `c = 32h`     `n = E9A2h`     `s = F124h:C100h`

**P43.** Escribir en ensamblador de 80x86 el código necesario para **invocar a la función Maximo** de C, cuya signatura se reproduce a continuación, así como para **reequilibrar la pila** después de dicha invocación. Los parámetros de la invocación son una tabla de enteros y su tamaño. Se supone que el programa de C está compilado en **modelo largo** (*large*). Se valorará la eficiencia del código.

```

int Maximo( int n, int *tabla );

int Tabla[10] = {2, 4, 5, 7, 1, 20, -2, 8, -100, 9};

Maximo( 10, Tabla );     // Implementar en ensamblador.

mov ax, SEG _Tabla
push ax
mov ax, OFFSET _Tabla

```

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70

Cartagena99

**P44.** Escribir en ensamblador de 80x86 la función de C que se reproduce en el siguiente recuadro, que calcula el **valor máximo de una tabla de n enteros con signo de 16 bits**. Las variables locales están almacenadas en registros. Se supone que el programa de C está compilado en **modelo largo (large)**. Se valorará la eficiencia del código.

Maximo PROC FAR

```

push bp
mov bp, sp
push bx cx dx si es

mov dx, [bp+6] ; dx == n
les bx, [bp+8] ; es:bx == tabla
mov ax, -32768 ; ax == max
mov cx, 0 ; cx == i

```

for:

```

cmp cx, dx ; i < n?
jge fin_for ; i >= n

; i < n
mov si, cx
shl si, 1 ; si == i * sizeof(int)
cmp es:[bx][si], ax ; tabla[i] > max?
jle fin_if ; tabla[i] <= max

```

```

; tabla[i] > max
mov ax, es:[bx][si] ; max = tabla[i]

```

fin\_if:

```

inc cx ; i++
jmp for

```

fin\_for:

```

pop es si dx cx bx
pop bp
ret

```

Maximo ENDP

```

int Maximo( int n, int *tabla )
{
    register int i;
    register int max = -32768;

    for (i=0; i<n; i++)
        if (tabla[i] > max) max = tabla[i];

    return max;
}

```

**P45.** Suponiendo que **SP=0** y que las primeras 16 posiciones del segmento de pila contienen los siguientes valores:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FFh	A0h	25h	00h	32h	00h	A2h	E9h	00h	C1h	24h	F1h	00h	63h	41h	12h

Indicar el valor de los cuatro registros después de la ejecución del siguiente programa.

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

---

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70**

Cartagena99

**P46.** Al inicio de la ejecución de una función invocada desde lenguaje C, se tiene que **SP=0** y que las 16 primeras posiciones de la pila contienen los siguientes valores:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
11h	A0h	25h	00h	32h	00h	A2h	00h	00h	C1h	24h	F1h	00h	63h	41h	12h

La signatura de dicha función es: `int fun( int n, char c, char *s );`

Indicar el valor de los tres parámetros con que esa función fue invocada desde C, tanto cuando todas las direcciones son cercanas (**NEAR**), como cuando son lejanas (**FAR**).

**Caso NEAR:**    `n = 0025h`    `c = 32h`    `s = 00A2h`

---

**Caso FAR:**    `n = 0032h`    `c = A2h`    `s = F124h:C100h`

---

**P47.** Escribir en ensamblador de 80x86 el código necesario para **invocar a la función `Minimo`** de C, cuya signatura se reproduce a continuación, así como para **reequilibrar la pila** después de dicha invocación. Los parámetros de la invocación son una tabla de enteros y su tamaño. Se supone que el programa de C está compilado en **modelo compacto**. Se valorará la eficiencia del código.

```
int Minimo( int n, int *tabla );

int Tabla[10] = {2, 4, 5, 7, 1, 20, -2, 8, -100, 9};

Minimo( 10, Tabla );    // Implementar en ensamblador.

mov ax, SEG _Tabla
push ax
mov ax, OFFSET _Tabla
push ax
mov ax, 10
push ax
call _Minimo
add sp, 6
```

**P48.** Escribir en ensamblador de 80x86 la función de C que se reproduce en el siguiente recuadro, que calcula el **valor mínimo de una tabla de n enteros con signo de 16 bits**. Las variables locales están almacenadas en registros. Se supone que el programa de C está compilado en **modelo compacto**. Se valorará la eficiencia del código.

`_Minimo PROC NEAR`

```
push bp
mov bp, sp
```

```
int Minimo( int n, int *tabla )
{
    register int i;
    register int min = 32767;
```

**CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
LLAMA O ENVÍA WHATSAPP: 689 45 44 70**

---

**ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
CALL OR WHATSAPP:689 45 44 70**



```

for:
  cmp cx, dx      ; i < n?
  jge fin_for    ; i >= n

      ; i < n
  mov si, cx
  shl si, 1      ; si == i * sizeof(int)
  cmp es:[bx][si], ax ; tabla[i] < min?
  jge fin_if     ; tabla[i] >= min

      ; tabla[i] < min
  mov ax, es:[bx][si] ; min = tabla[i]
fin_if:
  inc cx      ; i++
  jmp for

fin_for:
  pop es si dx cx bx
  pop bp
  ret

_Minimo ENDP

```



Cartagena99

CLASES PARTICULARES, TUTORÍAS TÉCNICAS ONLINE  
 LLAMA O ENVÍA WHATSAPP: 689 45 44 70

---

ONLINE PRIVATE LESSONS FOR SCIENCE STUDENTS  
 CALL OR WHATSAPP:689 45 44 70