SOLUCIONES AL EXAMEN DE

LABORATORIO DE ESTRUCTURAS DE LOS COMPUTADORES

CURSO 96/97. FEBRERO 1.997

Parte de MS-DOS.

1º) En el disco fijo de un ordenador se dispone de cuatro programas de aplicación: un procesador de textos (TEXTOS.EXE) que se encuentra en el directorio C:\APLITEX\; un programa de gráficos (DIBUJOS.EXE) situado en el directorio C:\GRAFICOS\; una base de datos (BASEDATO.COM) en el directorio D:\DATOS\; y, por último, un programa de facturación (FACTURAS.EXE) localizado en el directorio C:\UTIL\FACT\. Se pide realizar un archivo de procesamiento por lotes llamado **MENU** que presente en pantalla las diferentes aplicaciones que se tienen, así como una opción de salida, y que mediante la pulsación de una tecla efectúe la llamada a la aplicación correspondiente.

Solución:

```
@echo off
echo ∂
echo ∂
echo 1.- Procesador de textos.
echo 2.- Programa de gráficos.
echo 3.- Base de datos.
echo 4.- Facturación.
echo 5.- Salir.
echo ∂
choice /c:12345 "Introduzca una opción, por favor."
if ERRORLEVEL 5 goto Fin
if ERRORLEVEL 4 goto Facturas
if ERRORLEVEL 3 goto BBDD
if ERRORLEVEL 2 goto Graficos
C:\APLITEX\TEXTOS.EXE
goto Fin
:Graficos
C:\GRAFICOS\DIBUJOS.EXE
goto Fin
:BBDD
D:\DATO\BASEDATO.COM
goto Fin
:Facturas
C:\UTIL\FACT\FACTURAS.EXE
:Fin
```

Nota: el símbolo ∂ representa al carácter ALT+255

2º) Escriba un programa BAT llamado **BUSCAR** que se encargue de localizar un fichero dentro de todo el disco duro y, completada la búsqueda, informe de la ruta a especificar para llamarlo. Se visualizará un mensaje de ayuda si se indica el parámetro /? o si no se especifica el nombre del fichero a buscar.

Sintaxis: BUSCAR [/?] <fichero>

Solución:

Existen dos posibles soluciones.

A)

```
@ECHO OFF
IF "%1" == "" goto Ayuda
IF "%1" == "/?" goto Ayuda
chkdsk /V | FIND /I "%1"
goto Fin
:Ayuda
echo Este programa busca un  archivo en todos los directorios
echo de la unidad de disco actual indicándonos todos aquellos
echo directorios en los que se encuentre.
echo La sintaxis es:
echo          "BUSCAR [[?] | [fichero]]"
:Fin
```

Al pedir que buscásemos el fichero LEAME.TXT entregaría como solución:

C:\DOS\LEAME.TXT

C:\SB16\LEAME.TXT

C:\NU\LEAME.TXT

B)

```
@ECHO OFF
IF "%1" == "" goto Ayuda
IF "%1" == "/?" goto Ayuda
DIR /S \%1 | FIND /I "\"
goto Fin
:Ayuda
echo Este programa busca un  archivo en todos los directorios
echo de la unidad de disco actual indicándonos todos aquellos
echo directorios en los que se encuentre.
echo La sintaxis es:
echo          "BUSCAR [[?] | [fichero]]"
:Fin
```

Al pedir que buscásemos el fichero LEAME.TXT entregaría como solución:

Directorio de C:\DOS
Directorio de C:\NU
Directorio de C:\SB16

Nota: el símbolo representa al carácter ALT+255

3º) Se pide un fichero BAT llamado **PEGAR** que, tomando como parámetros una secuencia de nombres de ficheros, efectúe la unión de los mismos de las siguientes maneras:

* Si se especifica el parámetro /A, el contenido de cada uno de los ficheros que sucedan a este parámetro deberá añadirse al final del primer fichero de la secuencia.
* Si se especifica el parámetro /B, el contenido del primer fichero deberá añadirse al final de cada uno de los ficheros indicados después de dicho parámetro.
* Si no se especifica ni el parámetro /A ni el /B, se creará un fichero llamado TODO.TXT donde se almacenará el contenido de todos los ficheros de la secuencia.

En caso de especificarse el parámetro /? o de no indicarse ninguno, se mostrará un mensaje de ayuda. Asimismo, si no hay indicados al menos dos ficheros se visualizará un mensaje de error. En caso de que alguno de los ficheros especificados no se encontrara en el disco, deberá mostrarse un mensaje de aviso pero no se interrumpirá la ejecución del programa.

Sintaxis: PEGAR [/?] <fichero> [/A|/B] <fichero> [fichero [fichero....]]

Solución:

```
@echo off
if "%1" == "" goto Ayuda
if "%1" == "/?" goto Ayuda
if not exist %1 goto NoExiste
if "%2" == "/A" goto AnadirAlA
if "%2" == "/B" goto AnadirElA
goto AnadirATodo
:Ayuda
echo El programa realiza las siguientes operaciones:
echo Con el parámetro /A concatena al primer fichero todos los demás.
echo Con el parámetro /B concatena el primer fichero a todos los demás.
echo Si no se le pasa ningún parámetro se copian todos los ficheros
echo a un fichero que se llamará TODO.TXT
echo La sintaxis es:
echo "PEGAR [[/?]|fichero] [/A|/B] fichero [fichero[fichero...]]]"
goto Fin

:AnadirAlA
set NombreFich=%1
:BucleAlA
IF not exist %3 echo No existe el fichero %3
IF exist %3 type %3 >> %NombreFich%
shift
IF "%3"=="" goto Fin
goto BucleAlA
```

```
:AnadirElA
:BucleElA
IF not exist %3 echo No existe el fichero %3
IF exist %3 type %NombreFich% >> %3
shift
IF "%3"=="" goto Fin
goto BucleElA

:AnadirATodo
IF not exist %1 goto NoExiste
type %1 >> TODO.TXT
:BucleATodo
IF not exist %2 echo No existe el fichero %2
IF exist %2 type %2 >> TODO.TXT
shift
IF "%2"=="" goto Fin
goto BucleATodo

:NoExiste
echo El fichero %1 no existe
:Fin
```

En los casos en que se pasen los parámetros /A y /B se debe almacenar el parámetro del primer fichero, ya que al hacer los SHIFT lo perdemos. La forma de hacerlo es dejar el parámetro en una variable, y luego acceder a ella cuando necesitemos ese valor.

La forma de definirnos la variable es SET NombreFich=%1 y cada vez que deseemos acceder a su contenido deberemos poner %NombreFich%

Parte de Ensamblador.

1º) Realice un programa que lea un número de cuatro cifras decimales por teclado y escriba por pantalla un mensaje que indique si el número es o no múltiplo de 4.

Solución:

```
DATOS SEGMENT
    Terminar        EQU     4C00h
    LeerCadena      EQU     0Ah
    EscribirCadena  EQU     09h
    LF              EQU     0Ah
    CR              EQU     0Dh
    Mascara         EQU     03h
    EsMultiplo      DB      'El número introducido es múltiplo de 4', LF, CR, '$'
    NoEsMultiplo    DB      'El número introducido no es múltiplo de 4', LF, CR, '$'
    Cadena          DB      5, 0, 0, 0, 0, 0, 0
DATOS ENDS
PILA SEGMENT STACK
    DB 1024 DUP (" ")
PILA ENDS
```

```
CODIGO SEGMENT
    ASSUME CS:CODIGO, SS:PILA, DS:DATOS
Principal PROC FAR
    MOV AX, DATOS
    MOV DS, AX
    CALL LeerNumero     ; Lee la cifra de 4 dígitos.
    CALL Convertir      ; Convierte de decimal a hexadecimal.
    CALL DeterminaMul   ; Mira si es múltiplo de 4 o no.
    MOV AX, Terminar
    INT 21h
    RET
Principal ENDP
LeerNumero PROC NEAR
    MOV AH, LeerCadena  ; Este procedimiento lee una cadena de 4 cifras
    LEA DX, Cadena      ; decimales, dejando  el resultado  en la cadena
    INT 21h             ; Cadena.
    RET
LeerNumero ENDP
Convertir PROC NEAR
    PUSH AX             ; Este  procedimiento se  encarga  de  pasar de las
    PUSH BX             ; 4  cifras decimales  leídas a un  único número en
    PUSH CX             ; hexadecimal. Se supone que el usuario introduce
    XOR AX, AX          ; los datos de forma correcta.
    XOR CX, CX          ; Salvamos los registros que vamos a necesitar y
    XOR DX, DX          ; los inicializamos a cero.
    LEA BX, Cadena
    INC BX
    MOV CL, [BX]        ; Haremos el bucle tantas veces como cifras leídas
    DEC CL              ; menos 1.
    BUCLE:
        INC BX
        MOV AL, [BX]
        SUB AL, 30h     ; Corregimos el  valor  de  código ASCII  a número.
        ADD DX, AX      ; La diferencia  para  unir  las  cifras  en  decimal  con
        MOV AL, 0Ah     ; respecto a  las hexadecimales  es  que  en las  segundas
        MUL DX          ; desplazábamos 4 veces  a  la izquierda  lo que equivale
        MOV DX, AX      ; a  multiplicar  por  10h. En  nuestro  caso  deberemos
        XOR AX, AX      ; hacer  lo  mismo  pero  por  10d (0Ah). En  todos  los
        LOOP BUCLE      ; casos  menos el  último, que  se  corresponde  con  las
    INC BX              ; unidades. Motivo por  el cuál realizamos  la suma  con
    MOV AL, [BX]        ; el resto del número fuera del bucle.
    SUB AL, 30h         ; El número en hexadecimal, así construido, se devuelve
    ADD DX, AX          ; en el registro DX.
    POP CX
    POP BX              ; Recuperamos el valor original de los registros.
    POP AX
    RET
Convertir ENDP
```

```
DeterminaMul PROC NEAR
    TEST DX, Mascara            ; Tomamos el  número decimal  que se pasa
    JZ EsMul                    ; en DX y lo comparamos  con la  máscara
    LEA DX, NoEsMultiplo        ; 3h (0011b) para detrminar  si el número es
    JMP Seguir                  ; o no es múltiplo de 4; ya que los múltiplos
    EsMul:                      ; de 4 tienen los dos últimos bits a 0.
        LEA DX, EsMultiplo
    SEGUIR:
        MOV AH, EscribirCadena
        INT 21h
    RET
DeterminaMul ENDP
CODIGO ENDS
END Principal
```

2º) Se pide escribir un programa que lea por teclado una cadena de caracteres (80 como máximo) y la modifique de forma que convierta a mayúscula el primer carácter de cada palabra, visualizándose por pantalla el resultado.

Solución:

```
DATOS SEGMENT
    Terminar        EQU     4C00h
    LeeCadena       EQU     0Ah
    EscribeCadena   EQU     09h
    EscribeC        EQU     02h
    LF              EQU     0Ah
    CR              EQU     0Dh
    Espacio         EQU     20h
    Conversion      EQU     20h
    Dolar           EQU     24h
    MinMinuscula    EQU     61h
    MsgPedirCadena   DB     'Introduzca una cadena de texto (máximo 80 caracteres)',
LF, CR, '$'
    MsgResultado     DB     'La cadena convertida queda:', LF, CR, '$'
    Cadena           DB     81, 0
                     DB     81 DUP (0)
DATOS ENDS

PILA SEGMENT STACK
    DB 1024 DUP (" ")
PILA ENDS
```

```
CODIGO SEGMENT
    ASSUME CS:CODIGO, SS:PILA, DS:DATOS
Principal PROC FAR
    MOV AX, DATOS
    MOV DS, AX
    CALL Enter              ; Hacemos un salto de línea.
    CALL LeerCadena         ; Leemos la cadena de 80 caracteres.
    CALL Enter              ; Hacemos un salto de línea.
    CALL Convertir          ; Realizamos la conversión de la cadena.
    CALL EscribirCadena     ; Escribimos la cadena.
    MOV AX, TERMINAR
    INT 21h
    RET
Principal ENDP
LeerCadena PROC NEAR
    MOV AH, EscribeCadena       ; Solicitamos que introduzcan una cadena de
    LEA DX, MsgPedirCadena      ; 80 caracteres como máximo.
    INT 21h
    MOV AH, LeeCadena           ; Leemos la cadena y dejamos el resultado en
    LEA DX, Cadena              ; la cadena Cadena.
    INT 21h
    RET
LeerCadena ENDP
Enter PROC NEAR
    MOV AH, EscribeC            ; Este procedimiento realiza un salto de línea.
    MOV DL, LF
    INT 21h
    MOV DL, CR
    INT 21h
    RET
Enter ENDP
Convertir PROC NEAR
    PUSH BX             ; Este procedimiento convierte las primera letra de
cada
    PUSH CX             ; palabra en mayúsculas. Toma como entrada la cadena
    PUSH DX             ; Cadena y deja el resultado de la conversión en la
misma.
    XOR CX, CX          ; No consideramos los signos de puntuación.
    XOR DX, DX          ; Salvamos los registros que vamos a utilizar y los
ponemos
    LEA BX, Cadena      ; a cero
    INC BX
    MOV CL, [BX]
    INC BX              ; Accedemos al primer elemento de la cadena. Miramos si
    DEC CL              ; se trata de un espacio, en cuyo caso puede que existan
    MOV DL, [BX]        ; más y debemos saltarlos.
    CMP DL, Espacio     ; En caso de que no sea un espacio, se tratará de una letra
    JE BucleEspacios    ; en minúsculas con lo que la convertiremos.
    SUB DL, Conversion
```

```
        MOV [BX], DL
        Bucle:
            INC BX
            MOV DL, [BX]
            CMP DL, Espacio
            JNE FinBucle
            BucleEspacios:
                INC BX              ; El bucle recorre los restantes elementos de la
                DEC CL              ; cadena, saltándose los espacios que encuentre
                MOV DL, [BX]        ; y convirtiendo a mayúscula el siguiente elemento
                CMP DL, Espacio     ; de la cadena que se encuentre detrás del último
                JE BucleEspacios    ; espacio.
                CMP DL, MinMinuscula
                JNGE FinBucle
                SUB DL, Conversion
                MOV [BX], DL
            FinBucle:
            LOOP Bucle
        POP DX
        POP CX                      ; Recuperamos el valor original de los registros.
        POP BX
        RET
Convertir ENDP
EscribirCadena PROC NEAR
        PUSH DX.            ; Salvamos los registros que vamos a utilizar y los
ponemos
        PUSH BX             ; a cero.
        PUSH CX
        LEA BX, Cadena      ; Este procedimiento pretende emplear la función 09h, que
        INC BX              ; imprime cadenas de caracteres terminados en $.
        MOV CL, [BX]        ; Como nuestra cadena no termina en $, deberemos hacer
        ADD BX, CX          ; que el $ aparezca en el último lugar de la cadena.
        INC BX
        MOV CH, Dolar
        MOV [BX], CH
        MOV AH, EscribeCadena
        LEA DX, MsgResultado
        INT 21h
        LEA DX, Cadena      ; Además, las dos primeras posiciones de la cadena
no
        INC DX              ; pertenecen a la frase introducida, son el máximo
número
        INC DX              ; de caracteres y el número que hemos leído realmente. Por
        INT 21h             ; ello, los saltamos incrementando dos veces la
dirección
        POP CX              ; que tenemos en DX.
        POP BX              ; Recuperamos el valor original de los registros.
        POP DX
        RET
```

```
EscribirCadena ENDP
CODIGO ENDS
END Principal
```

3º) Escriba un programa que lea un número de dos cifras hexadecimales por teclado y emita tantos pitidos como indique el número leído. Deberá existir un intervalo de tiempo constante entre sonidos. Para simplificar el código, se supondrá que se dispone de un procedimiento (de tipo "FAR") llamado PITAR, el cual hace sonar una vez el altavoz del ordenador.

Solución:

```
DATOS SEGMENT
    Terminar        EQU     4C00h
    EscribeC        EQU     02h
    Pitido          EQU     07h
    LeeCadena       EQU     0Ah
    EscribeCadena   EQU     09h
    LF              EQU     0Ah
    CR              EQU     0Dh
    DesdeLetra      EQU     07h
    DesdeNumero     EQU     30h
    EsNumero        EQU     3Ah
MsgPedir  DB 'Introduzca un número de dos cifras hexadecimales, por favor', LF, CR,
'$'
    Cadena          DB      3, 0, 0, 0, 0
DATOS ENDS

PILA SEGMENT STACK
    DB 1024 DUP (" ")
PILA ENDS

CODIGO SEGMENT
    ASSUME CS:CODIGO, SS:PILA, DS:DATOS
Principal PROC FAR
    MOV AX, DATOS
    MOV DS, AX
    CALL LeerCifra          ; Leemos la cifra de dos dígitos hexadecimales.
    CALL Convertir          ; Convertimos las cifras en un único número.
    CALL BuclePitar         ; Pitamos tantas veces como indique el número leído.
    MOV AX, Terminar
    INT 21h
    RET
Principal ENDP
LeerCifra PROC NEAR
    PUSH DX
    MOV AH, EscribeCadena       ; Solicitamos que se introduzca un número de
    LEA DX, MsgPedir            ; dos cifras hexadecimales.
    INT 21h
    MOV AH, LeeCadena           ; Leemos el número de dos cifras y lo guardamos
    LEA DX, Cadena              ; en la cadena Cadena.
```

```
        INT 21h
        POP DX
        RET
LeerCifra ENDP
Convertir PROC NEAR
        PUSH BX                 ; Este procedimiento convierte del código ASCII a
        PUSH CX                 ; el valor  numérico de  cada cifra  y luego las junta
        XOR CX, CX              ; en un único número hexadecimal.
        XOR DX, DX
        LEA BX, Cadena
        INC BX
        INC BX
        MOV DH, [BX]
        CALL Corregir           ; Este procedimiento cambia de ASCII a número.
        MOV DL, DH
        INC BX
        MOV DH, [BX]
        CALL Corregir
        MOV CL, 04h
        SHL DL, CL              ; Juntamos las dos cifras en un único número.
        ADD DL, DH
        POP CX
        POP BX
        RET
Convertir ENDP
Corregir PROC NEAR
        CMP DH, EsNumero        ; Este procedimiento corrige la cifra en
hexadecimal
        JL TrataNumero          ; a su valor numérico. Restará 30h si esta entre 0-9
        SUB DH, DesdeLetra      ; y 37h si se encuentra comprendida entre A-F
        TrataNumero:
        SUB DH, DesdeNumero
        RET
Corregir ENDP
BuclePitar PROC NEAR
        PUSH CX
        XOR CX, CX          ; Este procedimiento realiza el número de pitidos
solicitado
        MOV CL, DL          ; recibe en DL el número de veces que debe pitar y llamará
        Bucle:              ; al procedimiento pitar.
            CALL Pitar
            LOOP Bucle
        POP CX
        RET
BuclePitar ENDP
```

```
Pitar PROC NEAR
        PUSH CX
        PUSH DX
        XOR CX, CX
        MOV AH, EscribeC      ; Este procedimiento realiza un único pitido  y espera
        MOV DL, Pitido; dos segundos de tiempo. La espera se  ha  realizado
        INT 21h               ; mediante la función 86h de la INT 16h colocando el
        MOV AH, 86h           ; número de microsegundos deseado  en CX:DX pero
        MOV CX, 001Eh         ; en hexadecimal.
        MOV DX, 8480h
        INT 15h
        POP DX
        POP CX
        RET
Pitar ENDP
CODIGO ENDS
END Principal
```

3º) Localice los errores que se producirían al ensamblar el programa cuyo código se muestra. Asimismo, suponiendo corregidos éstos, indique si habría errores en la ejecución del programa. ¿Qué ocurriría? Justifique las respuestas.

```
DATOS SEGMENT
  HOLA      DB “Hola”
  QUE       DB 10, 13, 36
  ADIOS     DB “Hasta luego, Lucas”
  VUELTA    DB ‘Z’
  NUMERO    DB  0
DATOS ENDS

PILA SEGMENT
  PL     DB ? DUP (1024)
PILA ENDS

CODIGO SEGMENT
ASUME CS:CODIGO, DS:DATOS,
        SS:PILA
INICIO:   MOV BX, DATOS
          MOV DS, BX
          XOR CX, CX
; Escribimos el saludo inicial
SALUDO:   LEA DX, HOLA
          MOV AH, 9
          INT 21
; Se emiten quinientos pitidos
BUCLE:    MOV CL, 500
          MOV AH, 07H
          MOV AL, EH
          INT 10
          LOOP BUCLE
; Leemos un carácter por teclado
          MOV AH, 01
          INT 21
          MOV NUMERO, AL
; Si es “Z” volvemos a empezar
          CMP NUMERO, VUELTA
          JE SALUDO
; Mensaje de despedida
          LEA DX, ADIOS
          MOV AH, 9
          INT 21
; Fin del programa
          MOV AH, 4C
          INT 21
CODIGO ENDS
          END
```

Solución:

Los errores encontrados son:

- Los mensajes de texto deben de estar en comillas simples y terminar en $
- La pila esta mal definida le falta añadir el STACK
- PL dará error
- El valor para el DUP no es válido
- ASUME lleva dos eses ASSUME por lo que no encontrara el segmento de código.
- El símbolo EH no está definido, ya que deberíamos poner 0Eh para que lo reconozca.
- El número 500 no entra en CL, se corresponde con el número hexadecimal 1F4 que debe entrar en 16 bits, no en 8.
- La inicialización del bucle debe quedar fuera del cuerpo del bucle, o sería un bucle infinito.
- No se puede comparar NUMERO con VUELTA ya que al menos uno de los operandos debe ser un registro
- A todas las interrupciones les falta el H de hexadecimal.
- El número 4C debe indicar que es hexadecimal
- Falta poner END Inicio
- En el bucle de pitidos los valores de AL y de AH están intercambiados.
- No es un fallo grave, pero ya que empleamos BX para inicializar el segmento de datos, deberíamos ponerlo a cero al igual que el CX.

El código correcto es el que aparece a continuación. Con este código el programa escribirá HOLA emitirá 500h pitidos, solicitará una tecla y si es diferente de Z saldrá del programa escribiendo Hasta luego, Lucas. En caso de ser igual a Z, el programa volverá a escribir HOLA, hará 500h pitidos y solicitará otra tecla, y así sucesivamente.

```
DATOS SEGMENT
    HOLA   DB    'Hola$'
    QUE    DB    10, 13, 36
    ADIOS  DB    'Hasta luego, Lucas$'
    VUELTA DB    'Z'
    NUMERO DB    0
DATOS ENDS

PILA SEGMENT STACK
    DB 1024 DUP(?)
PILA ENDS

CODIGO SEGMENT
    ASSUME CS:CODIGO, DS:DATOS,
SS:PILA
INICIO: MOV BX, DATOS
    MOV DS, BX
    XOR CX, CX
;Escribimos el saludo inicial
SALUDO: LEA DX, HOLA
    MOV AH, 9
    INT 21H
;Se emiten quinientos pitidos
MOV CX, 500
BUCLE:
    MOV AH, 0EH
    MOV AL, 07
    INT 10H
    LOOP BUCLE
;Leemos un carácter por teclado
    MOV AH, 01
    INT 21H
    MOV NUMERO, AL
;Si es "Z" volvemos a empezar
    CMP AL, VUELTA
    JE SALUDO
;Mensaje de despedida
    LEA DX, ADIOS
    MOV AH, 9
    INT 21H
;Fin del programa
    MOV AH, 4CH
    INT 21H
CODIGO ENDS
    END   INICIO
```

Nota: en el listado se han resaltado en negrita las líneas que contenía error.