

Laboratorio de Sistemas Basados en Microprocesadores

Práctica 2b: Juego de instrucciones: Sudoku

En esta práctica vamos a hacer una sencilla implementación de un sudoku 4x4. Se trata de una continuación de la práctica 2a, en la que ya trabajamos con matrices, direccionamiento e instrucciones. El alumno deberá realizar el correspondiente análisis de los requisitos expuestos y realizar su implementación.

Algunas de las reglas básicas del Sudoku son:

1. En una fila solo puede haber números del 1 al 4, y sin repetición.
2. En una columna solo puede haber números del 1 al 4, y sin repetición.
3. En cada una de las 4 regiones solo puede haber números del 1 al 4, y sin repetición.

En el desarrollo de esta práctica, será necesario emplear una buena parte del juego de instrucciones del 8086.

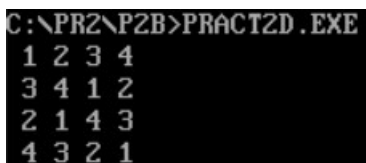
Programa 1: pract2d.asm

- A. Realice una subrutina (`imprime_matriz`) basada en el desarrollo realizado en la práctica 2A para imprimir un conjunto de 4 vectores 1x4, inicializados por defecto en el segmento de datos.

Si tenemos la siguiente definición de vectores:

```
vector1 db 1,2,3,4
vector2 db 3,4,1,2
vector3 db 2,1,4,3
vector4 db 4,3,2,1
```

La salida de nuestro programa será la siguiente:



```
C:\PR2\PR2B>PRACT2D.EXE
1 2 3 4
3 4 1 2
2 1 4 3
4 3 2 1
```

- B. Desarrolle una subrutina (`comprueba_filas`) en lenguaje ensamblador que compruebe que una matriz, inicializada mediante vectores en el segmento de datos, cumple con las siguientes reglas:
- Regla 1: en cada fila de la matriz no existen elementos repetidos.
 - Regla 2: cada fila de la matriz estará compuesta por 4 elementos pertenecientes al conjunto [1, 2, 3, 4]
 - Regla 3: Siempre se imprimirá la matriz como salida del programa. Además, en caso de cumplirse estas reglas, se imprimirá por pantalla el mensaje **"FILAS VALIDAS"**. En caso contrario se imprimirá el mensaje **"FILAS NO VALIDAS"**

Veamos un conjunto de posibles entradas y salidas:

vector1 db 1,2,3,4 vector2 db 3,4,1,2 vector3 db 2,1,4,3 vector4 db 4,3,2,1	vector1 db 1,1,3,4 vector2 db 3,4,1,2 vector3 db 2,1,4,3 vector4 db 4,3,2,1	vector1 db 1,5,3,4 vector2 db 3,4,1,2 vector3 db 2,1,4,3 vector4 db 4,3,2,1
C:\PR2\P2B>PRACT2E.EXE 1 2 3 4 3 4 1 2 2 1 4 3 4 3 2 1 FILAS VALIDAS	C:\PR2\P2B>PRACT2E.EXE 1 1 3 4 3 4 1 2 2 1 4 3 4 3 2 1 FILAS NO VALIDAS	C:\PR2\P2B>PRACT2E.EXE 1 5 3 4 3 4 1 2 2 1 4 3 4 3 2 1 FILAS NO VALIDAS

C. Desarrolle una subrutina en lenguaje ensamblador(*comprueba_columnas*), que compruebe que una matriz, inicializada mediante vectores en el segmento de datos, cumple con las siguientes reglas:

- Regla 4: en cada columna de la matriz no existen elementos repetidos.
- Regla 5: Siempre se imprimirá la matriz como salida del programa. A continuación, se escribirá el mensaje del resultado de las comprobaciones a nivel de fila. Por último, en caso de cumplirse estas nuevas reglas, se imprimirá por pantalla el mensaje **"COLUMNAS VALIDAS"**. En caso contrario se imprimirá el mensaje **"COLUMNAS NO VALIDAS"**

Veamos un conjunto de posibles entradas y salidas:

vector1 db 1,2,3,4 vector2 db 3,4,1,2 vector3 db 2,1,4,3 vector4 db 4,3,2,1	vector1 db 1,2,3,4 vector2 db 3,4,1,2 vector3 db 2,1,4,3 vector4 db 1,2,3,4	vector1 db 1,5,3,4 vector2 db 3,4,1,2 vector3 db 2,1,4,3 vector4 db 4,3,2,1
C:\PR2\P2B>PRACT2F.EXE 1 2 3 4 3 4 1 2 2 1 4 3 4 3 2 1 FILAS VALIDAS COLUMNAS VALIDAS	C:\PR2\P2B>PRACT2F.EXE 1 2 3 4 3 4 1 2 2 1 4 3 1 2 3 4 FILAS VALIDAS COLUMNAS NO VALIDAS	C:\PR2\P2B>PRACT2F.EXE 1 5 3 4 3 4 1 2 2 1 4 3 4 3 2 1 FILAS NO VALIDAS COLUMNAS VALIDAS

Notas:

- Para acceder a los números de la matriz se ha de utilizar un direccionamiento base-indexado con desplazamiento fijo.
- Para la resolución de este ejercicio, recomendamos utilizar la plantilla *pract2.asm* que encontraréis en el ANEXO II.

Programa 2: pract2e.asm

- A. Desarrolle una subrutina en lenguaje ensamblador, ampliando el ejercicio anterior, que compruebe que una matriz, inicializada mediante vectores en el segmento de datos, cumple con las siguientes reglas:
- Regla 6: en cada región de la matriz no existen elementos repetidos.
 - Regla 7: Siempre se imprimirá la matriz como salida del programa. A continuación, se escribirá el mensaje del resultado de las comprobaciones a nivel de fila y columna. Por último, en caso de cumplirse estas nuevas reglas, se imprimirá por pantalla el mensaje **“REGIONES VALIDAS”**. En caso contrario se imprimirá el mensaje **“REGIONES NO VALIDAS”**

Veamos un conjunto de posibles entradas y salidas:

<pre>vector1 db 1,2,3,4 vector2 db 3,4,1,2 vector3 db 2,1,4,3 vector4 db 4,3,2,1</pre>	<pre>vector1 db 1,2,3,4 vector2 db 2,1,4,3 vector3 db 3,4,1,2 vector4 db 4,3,2,1</pre>
<pre>C:\PR2\P2B>PRACT2G.EXE 1 2 3 4 3 4 1 2 2 1 4 3 4 3 2 1 FILAS VALIDAS COLUMNAS VALIDAS REGIONES VALIDAS</pre>	<pre>C:\PR2\P2B>PRACT2G.EXE 1 2 3 4 2 1 4 3 3 4 1 2 4 3 2 1 FILAS VALIDAS COLUMNAS VALIDAS REGIONES NO VALIDAS</pre>

Notas:

- Para acceder a los números de la matriz se ha de utilizar un direccionamiento base-indexado con desplazamiento fijo.
- Para la resolución de este ejercicio, recomendamos utilizar la plantilla *pract2.asm* que encontraréis en el ANEXO II.
- Si se han comprobado las filas y las columnas previamente, únicamente será necesario comparar las diagonales de cada región.
- Recordemos que nuestro Sudoku está compuesto por 4 regiones de 4 elementos:

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

- B. Desarrolle una subrutina en lenguaje ensamblador, para solicitar al usuario que vaya introduciendo la matriz por teclado, vector a vector. Este programa deberá cumplir con las siguientes reglas:
- Regla 8: Se imprimirá un mensaje pidiendo al usuario un vector.
 - Regla 9: Se irán guardando los vectores para trabajar con ellos a modo de matriz. Una vez recibidos los 4 vectores, se ejecutarán las comprobaciones de los ejercicios anteriores sobre la nueva matriz.
 - Regla 10: En caso de que el usuario introduzca un vector de tamaño incorrecto, se imprimirá el mensaje “**VECTOR INCORRECTO**” y se parará la ejecución del programa.
- C. Desarrolle un programa que incorpore todas las subrutinas desarrolladas en los apartados anteriores. El contenido final es libre pero como mínimo debe hacer uso de todas las funciones. Ejemplo de posibles entradas y salidas:

<pre> C:\PR2\P2B>PRACT2H.EXE INTRODUZCA UN VECTOR (EJ.: 1 2 3 4): 1 2 3 4 INTRODUZCA UN VECTOR (EJ.: 1 2 3 4): 3 4 1 2 INTRODUZCA UN VECTOR (EJ.: 1 2 3 4): 2 1 4 3 INTRODUZCA UN VECTOR (EJ.: 1 2 3 4): 4 3 2 1 1 2 3 4 3 4 1 2 2 1 4 3 4 3 2 1 FILAS VALIDAS COLUMNAS VALIDAS REGIONES VALIDAS </pre>
<pre> C:\PR2\P2B>PRACT2H.EXE INTRODUZCA UN VECTOR (EJ.: 1 2 3 4): 12 VECTOR INCORRECTO </pre>

Notas:

- Recomendamos la utilización de subrutinas tanto para pedir los vectores, como para almacenar la información.
- Recomendamos limitar la lectura de caracteres para facilitar el cumplimiento de la Regla 10.
- Para la resolución de este ejercicio, recomendamos utilizar la plantilla *pract2.asm* que encontraréis en el ANEXO II.

ENTREGA DE LA PRÁCTICA:

Ejercicio 1:

Se deberán entregar un zip con el fichero fuente *pract2d.asm* y el makefile.

Ejercicios 2:

Se deberá entregar un zip con el fichero fuente *pract2e.asm* y el makefile.

Para todos los ejercicios, el código generado deberá estar correctamente tabulado y comentado. La falta de comentarios, o la baja calidad de éstos, será calificada negativamente.

Las fechas límite para la subida de los archivos son:

Ejercicio 1:

Grupo del Jueves: 8 de Abril a las 20.15

Grupos del Viernes: 9 de Abril a las 19:15h

Ejercicio 2:

Grupos del Jueves: 14 de Abril a las 23:55

Grupos del Viernes: 15 de Abril a las 23:55h

Anexo I: Funciones del sistema operativo para imprimir texto y finalizar la ejecución.

El sistema operativo MS-DOS facilita la mayor parte de sus servicios a través de la interrupción 21h. Antes de que nuestro programa invoque la llamada éste mediante la instrucción INT 21H, se debe cargar en el registro AH el número de la función solicitada. Además, cada función puede requerir de otros parámetros de entrada cuyos valores deberán almacenarse en otra serie de registros. A continuación, se detallan tres funciones de uso habitual en los programas que se desarrollan en el laboratorio.

Función 2H: Envío de un código ASCII al periférico pantalla

Descripción: Imprime un único carácter por pantalla.
Parámetros de entrada: AH = 2h.
DL = código ASCII del carácter que se imprimirá en pantalla.
Parámetros de salida: Ninguno.
Registros afectados: Ninguno.

Ejemplo:

```
mov ah, 2      ; Número de función = 2
mov dl, 'A'    ; Se desea imprimir la letra A
int 21h        ; Interrupción software al sistema operativo
```

Función 9H: Envío de una cadena de caracteres ASCII al periférico pantalla

Descripción: Impresión de una cadena de caracteres que termina con el carácter ASCII=\$ por pantalla.
Parámetros de entrada: AH = 9h.
DX = Desplazamiento en memoria desde el origen del segmento de datos donde se ubica en memoria el primer carácter de la cadena.
Parámetros de salida: Ninguno.
Registros afectados: Ninguno.

Ejemplo:

```
.DATA
Texto DB "Hello world",13,10,'$' ; string terminado con los caracteres CR,LF y'$'
.CODE
.....
; Si DS es el segmento donde está el texto a imprimir:
mov dx, offset Texto      ; DX : offset al inicio del texto a imprimir
mov ah, 9                 ; Número de función = 9 (imprimir string)
int 21h                   ; Ejecuta el servicio del sistema operativo
```

Función 0AH: Lectura de caracteres introducidos por el periférico teclado

Descripción: Captura los caracteres introducidos por el teclado, los imprime por pantalla a modo de eco local (local echo) y se los transfiere posteriormente a la aplicación. La función finaliza cuando el usuario introduce el ASCII 13 (Retorno de carro). El sistema operativo permitirá la entrada de caracteres siempre que no se supere el máximo permitido definido en uno de los parámetros de entrada. En el caso contrario, estos no se mostrarán por pantalla ni se almacenarán.

Parámetros de entrada: DX = Desplazamiento en memoria desde el origen del segmento de datos donde se ubica el espacio reservado para capturar los caracteres. En el primer byte de dicho espacio reservado se ha de almacenar el número de caracteres que se desea capturar.

Parámetros de salida: En el segundo byte de la zona de memoria apuntada por DX el sistema operativo almacena el número de caracteres almacenados. A partir del tercer byte, el sistema operativo almacena los caracteres capturados.

Registros afectados: Ninguno.

Ejemplo:

MOV AH,0AH	;Función captura de teclado
MOV DX,OFFSET NOMBRE	;Area de memoria reservada = etiqueta NOMBRE
MOV NOMBRE[0],60	;Lectura de caracteres máxima=60
INT 21H	

En NOMBRE[1] el sistema operativo almacenará el número de caracteres registrados.

Función 4C00H: Fin de programa

Descripción: Esta función indica al sistema operativo que el proceso(programa) ha finalizado correctamente.

Parámetros de entrada: Ninguno.

Parámetros de salida: Ninguno.

Registros afectados: Ninguno.

Ejemplo:

mov ax, 4C00h	; Fin de programa
int 21h	; Ejecuta el servicio del sistema operativo

Anexo II: Plantillas asm

pract2.asm

Esta plantilla contiene la información necesaria para la realización de todos los ejercicios. En el último ejercicio no harán falta gran parte de las inicializaciones contenidas en el segmento de datos.

```
; Autor:

; Grupo

; Práctica 2, ejercicio X


NUM_VECTORES EQU 4 ; AQUI INDICAMOS EL NUMERO DE VECTORES QUE VAMOS A ANALIZAR
NUM_COMPONENTES EQU 4 ; INDICAMOS EL NUMERO DE ELEMENTOS EN CADA VECTOR
NUM_TOTAL_ELEMENTOS EQU NUM_VECTORES * NUM_COMPONENTES ; NUMERO TOTAL DE ELEMENTOS
SUMANDO TODOS LOS VECTORES

NUM_MAXIMO_DE_COMPONENTE EQU 4 ; LIMITE SUPERIOR QUE PUEDEN TOMAR LOS ELEMENTOS
NUM_MINIMO_DE_COMPONENTE EQU 1 ; LIMITE INFERIOR QUE PUEDEN TOMAR LOS ELEMENTOS


; DEFINICION DEL SEGMENTO DE DATOS
DATOS SEGMENT

    vector1 db 1,2,3,4
    vector2 db 3,4,1,2
    vector3 db 2,1,4,3
    vector4 db 4,3,2,1

    conversionASCII DB "      ", 13, 10, '$' ; CADENA DONDE SE VAN A GUARDAR LAS
CONVERSIONES DE LOS NUMEROS

    imprimirFilasSI DB "FILAS VALIDAS", 13, 10, '$'
    imprimirFilasNO DB "FILAS NO VALIDAS", 13, 10, '$'
    imprimirColumnasSI DB "COLUMNAS VALIDAS", 13, 10, '$'
    imprimirColumnasNO DB "COLUMNAS NO VALIDAS", 13, 10, '$'
    imprimirRegionesSI DB "REGIONES VALIDAS $"
    imprimirRegionesNO DB "REGIONES NO VALIDAS $"

DATOS ENDS
```



```
; *****  
; DEFINICION DEL SEGMENTO DE PILA  
PILA SEGMENT STACK "STACK"  
        DB 40H DUP (0) ;ejemplo de inicialización, 64 bytes inicializados a 0  
PILA ENDS  
; *****  
; DEFINICION DEL SEGMENTO EXTRA  
EXTRA SEGMENT  
EXTRA ENDS  
; *****  
; DEFINICION DEL SEGMENTO DE CODIGO  
CODE SEGMENT  
ASSUME CS: CODE, DS: DATOS, ES: EXTRA, SS: PILA  
; COMIENZO DEL PROCEDIMIENTO PRINCIPAL  
INICIO PROC  
; INICIALIZA LOS REGISTROS DE SEGMENTO CON SU VALOR  
        MOV AX, DATOS  
        MOV DS, AX  
        MOV AX, PILA  
        MOV SS, AX  
        MOV AX, EXTRA  
        MOV ES, AX  
        MOV SP, 64 ; CARGA EL PUNTERO DE PILA CON EL VALOR MAS ALTO  
        ; FIN DE LAS INICIALIZACIONES  
        ; COMIENZO DEL PROGRAMA  
  
; FIN DEL SEGMENTO DE CODIGO  
CODE ENDS  
; FIN DEL PROGRAMA INDICANDO DONDE COMIENZA LA EJECUCION  
END INICIO
```