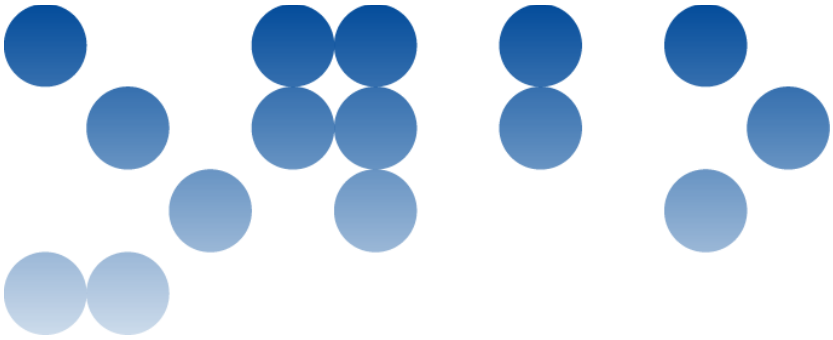


PRÁCTICA 1a parte: 2048

Estructura de Computadores

Grado en Ingeniería Informática

set15-feb16

Estudios de Informática, Multimedia y Telecomunicaciones

EIMT.UOC.EDU

UOC Universitat Oberta de Catalunya

Presentación

La práctica que se describe a continuación consiste en la programación en lenguaje ensamblador x86_64 de un conjunto de subrutinas, que se tienen que poder llamar desde un programa en C. El objetivo es implementar un juego del 2048.

La PRÁCTICA es una **actividad evaluable individual**, por lo tanto no se pueden hacer comentarios muy amplios en el foro de la asignatura. Se puede hacer una consulta sobre un error que tengáis a la hora de ensamblar el programa o de algún detalle concreto, pero no se puede poner el código de una subrutina o bucles enteros.

Competencias

Las competencias específicas que persigue la PRÁCTICA son:

- [13] Capacidad para identificar los elementos de la estructura y los principios de funcionamiento de un ordenador.
- [14] Capacidad para analizar la arquitectura y organización de los sistemas y aplicaciones informáticos en red.
- [15] Conocer las tecnologías de comunicaciones actuales y emergentes y saberlas aplicar convenientemente para diseñar y desarrollar soluciones basadas en sistemas y tecnologías de la información.

Objetivos

Introducir al estudiante en la programación de bajo nivel de un computador, utilizando el lenguaje ensamblador de la arquitectura Intel x86-64 y el lenguaje C.

Recursos

Podéis consultar los recursos del aula pero no podéis hacer uso intensivo del foro.

El material básico que podéis consultar es:

- Módulo 6: Programación en ensamblador (x86_64)
- Documento “Entorno de trabajo”

La Práctica: 2048

La práctica consiste en implementar el juego del “2048” que consiste en ir moviendo los valores que se encuentran en un tablero de 4 x 4 de forma que cuando dos valores iguales entren en contacte se sumen, hasta llegar a que en una casilla el valor sea igual a 2048. Se pueden desplazar los valores del tablero a la izquierda, derecha, arriba o abajo, al hacer un desplazamiento se arrastran todos los valores del tablero en la dirección seleccionada, eliminando los espacios en blanco que haya entre las casillas, si al hacer el desplazamiento quedan dos casillas consecutivas con el mismo valor, se suman los valores, dejando una casilla con el valor sumado y la otra casilla en blanco.

Al finalizar un movimiento el programa genera un número 2 que se añade al tablero en una casilla vacía seleccionada aleatoriamente.

Podéis ver una descripción más detallada del juego en:

https://es.wikipedia.org/wiki/2048_(videojuego)

Y podéis ver una versión online del juego en la página web del autor:

http://gabrielecirulli.github.io/2048/

También hay versiones del juego para iOS y Android.

La práctica consta de un programa en C, que os damos hecho i que NO TENÉIS QUE MODIFICAR, y un programa en ensamblador que contiene algunas subrutinas ya hechas y otras que tenéis que implementar vosotros. Más adelante encontraréis la información detallada sobre la implementación de la práctica.

El archivo C contiene una versión completa de la práctica para que os sirva de guía a la hora de implementar las subrutinas en ensamblador. También os permite ejecutar el juego para ver cómo tiene que funcionar.

La práctica se ha dividido en dos partes:

PRIMERA PARTE OBLIGATORIA:

Para esta primera parte os proporcionamos dos archivos: p1c.c y p1.asm. El código C (p1c.c) no lo tenéis que modificar sólo tenéis que implementar en ensamblador en el archivo p1.asm las funcionalidades siguientes:

- Actualizar el contenido del tablero con los valores del juego y los puntos del marcador.
- Desplazar los valores de las casillas a la izquierda, derecha, arriba y abajo.
- Sumar los valores de aquellas casillas consecutivas que tengan el mismo valor, dejando sólo el valor de la suma y acumular en un contador los valores de las sumas que es vayan realizando, actualizando el valor de la variable que hace de marcador, *score*.

- Rotar la matriz *m* a la derecha.

El programa en C genera un menú principal con 9 opciones:

1. ShowNumber, convertir el valor de la variable *number* a caracteres ASCII y mostrarlos por pantalla.
2. UpdateBoard, mostrar por pantalla sobre el tablero los valores de la matriz *m* y el valor del marcador *score*.
3. CopyMatrix, copiar los valores de la matriz *mRotated* en la matriz *m*.
4. RotateMatrix, rotar a la derecha los valores de la matriz *m*, copiando los valores sobre la matriz *mRotated*.
5. ShiftLeft, desplazar a la izquierda todos los valores de la matriz *m*, dejando a la derecha los ceros que haya en cada fila.
6. AddPairs, buscar parejas de valores iguales consecutivos por filas de izquierda a derecha, sumando los valores, dejando la suma en la casilla de la izquierda y poniendo un cero en la casilla de la derecha.
7. PlayGame, permite jugar llamando a las subrutinas de ensamblador implementadas por vosotros, para comprobar que toda la práctica funciona correctamente.
8. Play Game C, permite jugar llamando a todas las funciones implementadas en lenguaje C. Sólo para ver el funcionamiento del juego.
1. Exit, finalizar el programa.

En esta primera parte el juego del 2048 no estará completamente implementado, faltará por implementar la funcionalidad que comprueba que el juego se ha acabado si se llega a 2048, y la comprobación de que se ha perdido el juego si el tablero está lleno, no se pueden hacer parejas y no hemos llegado a 2048. En la segunda parte opcional se implementaran las funcionalidades necesarias para tener un juego totalmente funcional.

Implementación de los movimientos:
Para implementar cada movimiento disponemos de 3 funcionalidades: rotar, desplazar y hacer parejas.

En la práctica implementaremos los 4 tipos de movimientos, izquierda, derecha, arriba y abajo, utilizando sólo el desplazamiento a la izquierda (subrutina *shiftNumbersLP1*), buscar parejas de izquierda a derecha (subrutina *addPairsLP1*) y haciendo una o más rotaciones a la derecha de 90º de la matriz (subrutina *rotateMatrixRP1* y *copyMatrixP1*), después de cada rotación se tiene que ejecutar la subrutina copyMatrixP1 porque la subrutina *rotateMatrixRP1* deja los valores que se han rotado en la matriz *mRotated* y se tienen que volver a copiar en la matriz *m*.

Para cada tipo de movimiento haremos los pasos que se indican a continuación:

Movimiento a la izquierda:

- Desplazamiento a la izquierda.
- Buscar parejas de izquierda a derecha.
- Volver a hacer un desplazamiento a la izquierda para juntar las parejas que se hayan hecho.

Ejemplo de un movimiento a la izquierda:

2	2	4	4
2		2	8
8		4	4
4	4	4	4

Desplazamiento a la izquierda

2	2	4	4
2	2	8	
8	4	4	
4	4	4	4

2	2	4	4
2	2	8	
8	4	4	
4	4	4	4

Hacer parejas de izquierda a derecha

4		8	
4		8	
8	8		
8		8	

Desplazamiento a la izquierda

4	8		
4	8		
8	8		
8	8		

Para el resto de movimientos habrá que rotar la matriz para poder hacer los desplazamientos y las parejas hacia la izquierda y después dejarla como estaba.

Movimiento a la derecha:

- Rotación de 180º a la derecha = 2 rotaciones de 90º a la derecha (para cada rotación copiar nuevamente la matriz mRotated sobre m).
- Desplazamiento a la izquierda.
- Buscar parejas de izquierda a derecha.
- Volver a hacer un desplazamiento a la izquierda para juntar las parejas que se hayan hecho.

– Rotación de 180º a la derecha = 2 rotaciones de 90º a la derecha (para cada rotación copiar nuevamente la matriz mRotated sobre m).

Movimiento arriba:
– Rotación de 270º a la derecha = 3 rotaciones de 90º a la derecha (para cada rotación copiar nuevamente la matriz mRotated sobre m).
– Desplazamiento a la izquierda.
– Buscar parejas de izquierda a derecha.
– Volver a hacer un desplazamiento a la izquierda para juntar las parejas que se hayan hecho.
– Rotación de 90º a la derecha (para cada rotación copiar nuevamente la matriz mRotated sobre m).

Movimiento abajo:
– Rotación de 90º a la derecha (para cada rotación copiar nuevamente la matriz mRotated sobre m).
– Desplazamiento a la izquierda.
– Buscar parejas de izquierda a derecha.
– Volver a hacer un desplazamiento a la izquierda para juntar las parejas que se hayan hecho.
– Rotación de 270º a la derecha = 3 rotaciones de 90º a la derecha (para cada rotación copiar nuevamente la matriz mRotated sobre m).

En todos los casos acabamos haciendo 4 rotaciones de 90º, y por tanto hacemos una vuelta entera de 360º de la matriz volviéndola a dejar con la misma orientación que tenía.

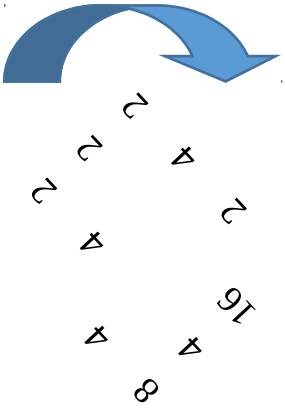
Hacemos la implementación de esta forma para evitar la implementación de cuatro subrutinas de desplazamiento y cuatro de hacer parejas.

Veamos a continuación un ejemplo de las operaciones que se harían para un desplazamiento debajo de los valores de la matriz.

Movimiento abajo
Tiene que hacer lo siguiente:

Pasos que sigue el programa:

Rotación de 90º a la derecha



8	4				16	4	8
4		4	2		2		4
16			2		4		4
	2	4	2		2	2	2

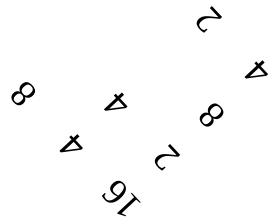
	16	4	8		16	4	8
2			4		2	4	
4		4		Desplazamiento a la izquierda	4	4	
2	2	2			2	2	2

16	4	8			16	4	8
2	4				2	4	
4	4			Hacer parejas de izquierda a derecha	8		
2	2	2			4		2

16	4	8			16	4	8
2	4				2	4	
8				Desplazamiento a la izquierda	8		
4		2			4	2	

1a Rotación de 90º a la derecha

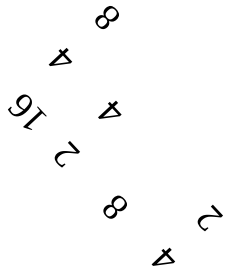
16	4	8	
2	4		
8			
4	2		



4	8	2	16
2		4	4
			8

2a Rotación de 90º a la derecha

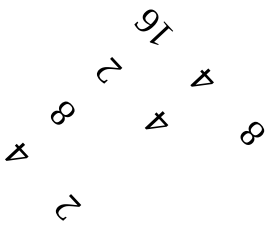
4	8	2	16
2		4	4
			8



		2	4
			8
		4	2
	8	4	16

3a Rotación de 90º a la derecha

		2	4
			8
		4	2
	8	4	16



8			
4	4		2
16	2	8	4

SEGUNDA PARTE OPCIONAL: En la segunda parte se tendrán que implementar las funcionalidades adicionales necesarias para completar todas las funcionalidades del juego del 2048.

Además habrá que trabajar el paso de parámetros entre las diferentes subrutinas, modificando la implementación hecha en la primera parte.

Os proporcionaremos también dos archivos para esta segunda parte: p2c.c y p2.asm. De forma parecida a la primera parte, el código C no lo tendréis que modificar, sólo tendréis que implementar en ensamblador nuevas funcionalidades y habrá que modificar las subrutinas que habréis hecho en la primera parte para trabajar el paso de parámetros entre las subrutinas de ensamblador y también con las funciones de C.

El 6 de noviembre de 2015 os daremos los archivos .c y .asm correspondientes a la 2a parte opcional.

La **primera parte** se puede entregar antes de las **23:59 del día 6 de noviembre de 2015** para obtener una puntuación de prácticas que puede llegar a una B. Si en esta fecha se ha hecho la entrega de la primera parte de manera satisfactoria se puede entregar la **segunda parte** antes de las **23:59 del 18 de diciembre de 2015** para poder llegar a una puntuación de A en las prácticas.

En cambio, si no se ha podido hacer la primera entrega o esta primera entrega no ha sido satisfactoria se puede hacer una **segunda entrega** antes de las **23:59 del 18 de diciembre**. En esta segunda fecha de entrega se puede entregar la primera parte, para obtener una calificación máxima de C, o ambas partes (la práctica completa), para obtener una calificación máxima de B.
Este esquema de entregas y calificación se puede resumir en la siguiente tabla.

Primera Entrega 06-11-2015	**Primera parte Superada**	**Primera parte NO Superada** NO Presentada	**Primera parte Superada**	**Primera parte NO Superada** NO Presentada	**Primera parte NO Superada** NO Presentada
Segunda Entrega 18-12-2015	**Segunda parte NO Superada** NO Presentada	**Primera parte Superada**	**Segunda parte Superada**	**Primera parte Superada Segunda parte Superada**	**Primera parte NO Superada** NO Presentada
Nota Final Práctica	**B**	**C+**	**A**	**B**	**D/N**

Los alumnos que no superen la PRÁCTICA tendrán un suspenso (calificación 2-3) o N si no se ha presentado nada, en la nota final de prácticas y con esta nota no se puede aprobar la asignatura, por este motivo la PRÁCTICA es obligatoria.

Criterios de valoración

Dado que se trata de hacer un programa, sería bueno recordar que las dos virtudes a exigir, por este orden son:

- Eficacia: que haga lo que se ha pedido y tal como se ha pedido, es decir, que todo funcione correctamente según las especificaciones dadas. Si las subrutinas no tienen la funcionalidad pedida, aunque la práctica funcione correctamente, se podrá considerar suspendida.
- Eficiencia: que lo haga de la mejor forma. Evitar código redundante (que no haga nada) y demasiado código repetido (que el código sea compacto pero claro). Usar modos de direccionamiento adecuados: los que hagan falta. Evitar el uso excesivo de variables y usar registros para almacenar valores temporales.

IMPORTANTE: Para acceder a los vectores en ensamblador se ha de utilizar direccionamiento relativo o direccionamiento indexado: [m+esi], [ebx+edi]. No se pueden utilizar índices con valores fijos para acceder a los vectores.

Ejemplo de lo que **NO** se puede hacer:

```
mov WORD [m+0], 0
mov WORD [m+2], 0
mov WORD [m+4], 0
...
```

Y repetir este código muchas veces.

Otro aspecto importante es la documentación del código: que clarifique, dé orden y estructura, que ayude a entenderlo mejor. No es ha de explicar qué hace la instrucción (se da por supuesto que quién la lee sabe ensamblador) sino que se ha de explicar por qué se usa una instrucción o grupo de instrucciones (para hacer qué tarea de más alto nivel, relacionada con el problema que queremos resolver).

Formato y fecha de entrega

La entrega se tiene que hacer a través de la aplicación **Entrega y registro de EC** del aula. Se tiene que entregar sólo un fichero con el código ensamblador bien comentado.

Es importante que el nombre de los ficheros tenga vuestros apellidos y nombre con el formato:

apellido1_apellido2_nombre_p1.asm

Fecha límite de la primera entrega:

Viernes, 06 de noviembre de 2015 a las 23:59:59

Fecha límite de la segunda entrega:

Viernes, 18 de diciembre de 2015 a les 23:59:59

Implementación

Cómo ya hemos dicho, la práctica consta de una parte de código en C que os damos hecho y **NO PODÉIS MODIFICAR** y un programa en ensamblador que contiene algunas subrutinas ya implementadas y las subrutinas que tenéis que implementar. Cada subrutina de ensamblador tiene una cabecera que explica que hace esta subrutina y como se tiene que implementar, indicando las variables, funciones de C y subrutinas de ensamblador que hay que llamar.
No tenéis que añadir otras variables o subrutinas.

Para ayudaros en el desarrollo de la práctica, en el fichero de código C encontraréis implementado en este lenguaje las subrutinas que tenéis que hacer en ensamblador para que os sirvan de guía durante la codificación.

En el código C se hacen llamadas a las subrutinas de ensamblador que tenéis que implementar, pero también encontraréis comentadas las llamadas a las funciones de C equivalentes. Si queréis probar las funcionalidades hechas en C lo podéis quitando el comentario de la llamada de C y poniéndolo en la llamada a la subrutina de ensamblador.

Por ejemplo en la opción 1 del menú hecho en C hay el código siguiente:

```
//=====================================================
  rowCur = RowScreenIni+(DimMatrix*2);
  colCur = ColScreenIni+8;
  number = score;
  showNumberP1();   //Mostrar puntos
  //showNumberP1_C();
//=====================================================
```

El código llama a la subrutina de ensamblador showNumberP1(), podemos cambiar el comentario y llamar a la función de C.

```
//=====================================================
  rowCur = RowScreenIni+(DimMatrix*2);
  colCur = ColScreenIni+8;
  number = score;
  //showNumberP1();   //Mostrar puntos
  showNumberP1_C();
//=====================================================
```

Recordad volver a dejar el código como estaba para probar vuestras subrutinas.

Subrutinas que hay que implementar en ensamblador para la Primera Parte:

```
;;;;;
; Convierte el valor de la variable (number) de tipo int (DWORD)
; a caracteres ASCII que representen su valor.
; Hay que dividir el valor entre 10,  de forma iterativa,
; hasta que el cociente de la división sea 0.
; A cada iteración, el residuo de la división que es un valor
; entre (0-9) indica el valor del dígito que tenemos que convertir
; a ASCII ('0' - '9') sumando '0' (48 decimal) para poderlo mostrar.
; Se tienen que mostrar los dígitos (carácter ASCII) desde la posición
; indicada por las variables (rowCur) y (colCur), posición de les
; unidades, hacia la izquierda.
; Como el primer dígito que obtenemos son las unidades, después las decenas,
; ..., para mostrar el valor se tiene que desplazar el cursor una posición
; a la izquierda en cada iteración.
; Para posicionar el cursor se llamada a la función gotoxyP1 y para
; mostrar los caracteres a la función printchP1.
;
; Variables utilizadas:
; rowCur : Fila para posicionar el cursor a la pantalla.
; colCur : Columna para posicionar el cursor a la pantalla.
; charac : Carácter que queremos mostrar
; number : Valor que queremos mostrar.
;
; Parámetros de entrada:
; Ninguno
;
; Parámetros de salida :
; Ninguno.
;;;;;
showNumberP1:


;;;;;
; Actualizar el contenido del Tablero de Juego con los datos de
; la matriz (m) y los puntos del marcador (score) que se han hecho.
; Se tiene que recorrer toda la matriz (m), y para cada elemento de
; la matriz posicionar el cursor en pantalla y mostrar el número de
; esa posición de la matriz.
; Después, mostrar el marcador (score) en la parte inferior del tablero.
; Finalmente posicionar el cursor a la derecha de la última fila del tablero.
; Para posicionar el cursor se llama a la función gotoxyP1, y para
; mostrar los números de la matriz y el marcador de puntos a la función
; showNumberP1.
;
; Variables utilizadas:
; rowCur : Fila para posicionar el cursor en pantalla.
; colCur : Columna per a posicionar el cursor en pantalla.
; number : Número que queremos mostrar.
; m      : matriz 4x4 donde hay los números del tablero de juego.
; score  : puntos acumulados en el marcador hasta el momento.
;
; Parámetros de entrada:
; Ninguno
;
; Parámetros de salida :
; Ninguno
;;;;;
updateBoardP1:


;;;;;
; Calcular el valor del índice para acceder a una matriz (4x4) que
; guardaremos en la variable (indexMat) a partir de la fila indicada
; por la variable (row) y la columna indicada por la variable (col).
; indexMat=((row*DimMatrix)+(col))*2
; multiplicamos por 2 porque es una matriz de tipo short (WORD) 2 bytes.
;
; Esta subrutina no tiene una función en C equivalente.
```

```
;
; Variables utilizadas:
; row     : fila para acceder a la matriz m.
; col     : columna para acceder a la matriz m.
; indexMat: índice para acceder a la matriz m.
;
; Parámetros de entrada:
; Ninguno
;
; Parámetros de salida :
; Ninguno
;;;;;

;;;;;
; Rotar a la derecha la matriz (m), sobre la matriz (mRotated).
; La primera fila pasa a ser la cuarta columna, la segunda fila pasa
; a ser la tercera columna, la tercera fila pasa a ser la segunda
; columna y la cuarta fila pasa a ser la primer columna.
; En el enunciado se explica con más detalle como hacer la rotación.
; NOTA: NO es lo mismo que hacer la matriz traspuesta.
; La matriz (m) no se tiene que modificar,
; los cambios se tiene que hacer en la matriz (mRotated).
; Para acceder a matrices desde ensamblador hay que calcular el índice
; a partir de la fila y la columna llamando a la subrutina calcIndexP1.
; m[row][col], en C, es equivalente a WORD[m+eax], en ensamblador, si
; eax = ((row*DimMatrix)+(col))*2. m[1][2] es DWORD[m+12].
; No se tiene que mostrar la matriz.
;
; Variables utilizadas:
; m       : matriz 4x4 donde hay los números del tablero de juego.
; mRotated: matriz 4x4 para hacer la rotación.
; row     : fila para acceder a la matriz m.
; col     : columna para acceder a la matriz m.
; indexMat: índice para acceder a la matriz m.
;
; Parámetros de entrada:
; Ninguno.
;
; Parámetros de salida :
; Ninguno.
;;;;;
rotateMatrixRP1:


;;;;;
; Copiar los valores de la matriz (mRotated) a la matriz (m).
;
; Variables utilizadas:
; m       : matriz 4x4 donde hay los números del tablero de juego.
; mRotated: matriz 4x4 para hacer la rotación.
;
; Parámetros de entrada:
; Ninguno.
;
; Parámetros de salida :
; Ninguno.
;;;;;
copyMatrixP1:



;;;;;
; Desplazar a la izquierda los números de cada fila de la matriz (m),
; manteniendo el orden de los números y poniendo los ceros a la derecha.
; Recorrer la matriz por filas de izquierda a derecha y de arriba a bajo.
; Si se desplaza un número (NO LOS CEROS) pondremos la variable
; (moved) a 1.
; Si una fila de la matriz es: [0,2,0,4] y moved = 0, quedará [2,4,0,0]
; y moved = 1.
```

```
; Para acceder a matrices desde ensamblador hay que calcular el índice
; a partir de la fila y la columna llamando a la subrutina calcIndexP1.
; m[row][col], en C, es equivalente a WORD[m+eax], en ensamblador, si
; eax = ((row*DimMatrix)+(col))*2. m[1][2] es DWORD[m+12].
; Los cambios se tienen que hacer sobre la misma  matriz.
; No se tiene que mostrar la matriz.
;
; Variables utilizadas:
; moved   : Para indicar si se han hecho cambios en la matriz.
; m       : matriz 4x4 donde hay los números del tablero de juego.
; row     : fila para acceder a la matriz m.
; col     : columna para acceder a la matriz m.
; indexMat: índice para acceder a la matriz m.
;
; Parámetros de entrada:
; Ninguno.
;
; Parámetros de salida :
; Ninguno.
;;;;;
shiftNumbersLP1:


;;;;;
; Emparejar números iguales des la izquierda de la matriz (m) y acumular
; los puntos en el marcador sumando los puntos de las parejas que se hagan.
; Recorrer la matriz por filas de izquierda a derecha y de arriba abajo.
; Cuando se encuentre una pareja, dos casillas consecutivas con el mismo
; número, juntamos la pareja poniendo la suma de los números de la
; pareja en la casilla de la izquierda y un 0 en la casilla de la derecha y
; acumularemos esta suma puntos que se ganan).
; Si una fila de la matriz es: [8,4,4,2] y moved = 0, quedará [8,8,0,2],
; p = (4+4)= 8 y moved = 1.
; Si al final se ha juntado alguna pareja (puntos>0), pondremos la variable
; (moved) a 1 para indicar que se ha movido algún número y actualizaremos
; la variable (score) con los puntos obtenidos de hacer las parejas.
; Para acceder a matrices desde ensamblador hay que calcular el índice
; a partir de la fila y la columna llamando a la subrutina calcIndexP1.
; m[row][col], en C, es equivalente a WORD[m+eax], en ensamblador, si
; eax = ((row*DimMatrix)+(col))*2. m[1][2] es DWORD[m+12].
; No se tiene que mostrar la matriz.
;
; Variables utilizadas:
; moved   : Para indicar si se han hecho los cambios en la matriz.
; score   : Puntos acumulados hasta el momento.
; m       : matriz 4x4 donde hay los números del tablero de juego.
; row     : fila para acceder a la matriz m.
; col     : columna para acceder a la matriz m.
; indexMat: índice para acceder a la matriz m.
;
; Parámetros de entrada:
; Ninguno.
;
; Parámetros de salida :
; Ninguno.
;;;;;
addPairsLP1:
```

Subrutinas que ya están implementadas y que no tenéis que modificar para la Primera Parte:

```
gotoxyP1
printchP1
getchP1
readKeyP1
playP1
```