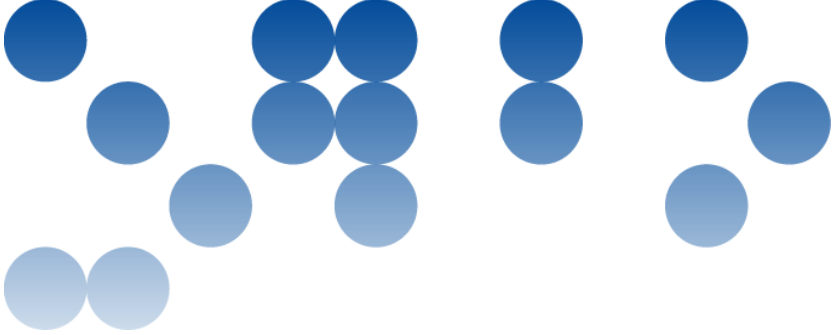


PRÁCTICA 1.ª parte: 2048

Estructura de Computadores

Grado en Ingeniería Informática

feb20-jun20

Estudios de Informática, Multimedia y Telecomunicación

EIMT.UOC.EDU

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 hacer 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 en 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 entran en contacto se suman, 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 o un 4 que se añade al tablero en una casilla vacía seleccionada aleatoriamente.

Podéis ver una descripción más detallada del juego en:

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

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

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

También hay una versiones del juego por iOS y Android.

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

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

La práctica se ha dividido en dos partes:

PRIMERA PARTE OBLIGATORIA:

Para esta primera parte os proporcionamos dos ficheros: 2Kp1c-es.c y 2Kp1-es.asm. El código C (2Kp1c-es.c) no lo tenéis que modificar solo tenéis que implementar en ensamblador en el fichero p1-es.asm las funcionalidades siguientes:

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

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

- ShowNumber, convertir el valor de la variable *number* a caracteres ASCII y mostrarlos por pantalla.
- UpdateBoard, mostrar por pantalla sobre el tablero los valores de la matriz *m* y el valor del marcador *score*.
- CopyMatrix, copiar los valores de la matriz *mRotated* a la matriz *m*.
- RotateMatrix, rotar a la derecha los valores de la matriz *m*, copiando los valores sobre la matriz *mRotated*.
- ShiftNumbers, desplazar a la izquierda todos los valores de la matriz *m*, dejando a la derecha los ceros que haya en cada fila.

- 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.
- PlayGame, permite jugar llamando a las subrutinas de ensamblador implementadas por vosotros, para comprobar que toda la práctica funciona correctamente.
- PlayGame C, permite jugar llamando a todas las funciones implementadas en lenguaje C. Solo para ver el funcionamiento del juego.
- Exit, finalizar el programa.

Os recomendamos que vayáis desarrollando el código siguiendo el orden de estas opciones del menú hasta llegar a la opción que permite jugar utilizando todas las funcionalidades anteriores. Cada opción permite comprobar el funcionamiento de cada subrutina de forma independiente.

En esta primera parte el juego del 2048 no estará completamente implementado, faltará para 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 implementarán las funcionalidades necesarias para tener un juego totalmente funcional.

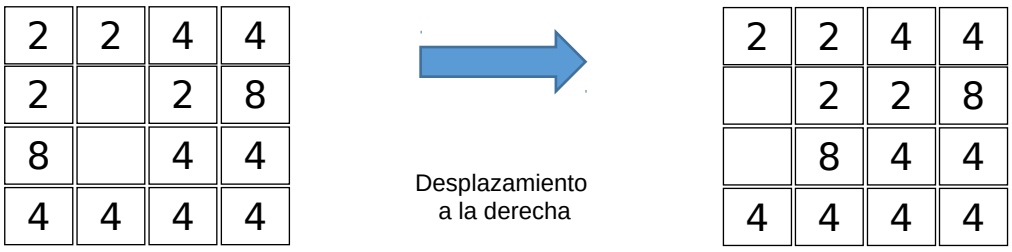
Implementación de los movimientos:
Para implementar cada movimiento disponemos de 3 funcionalidades: rotar, desplazar y hacer parejas.

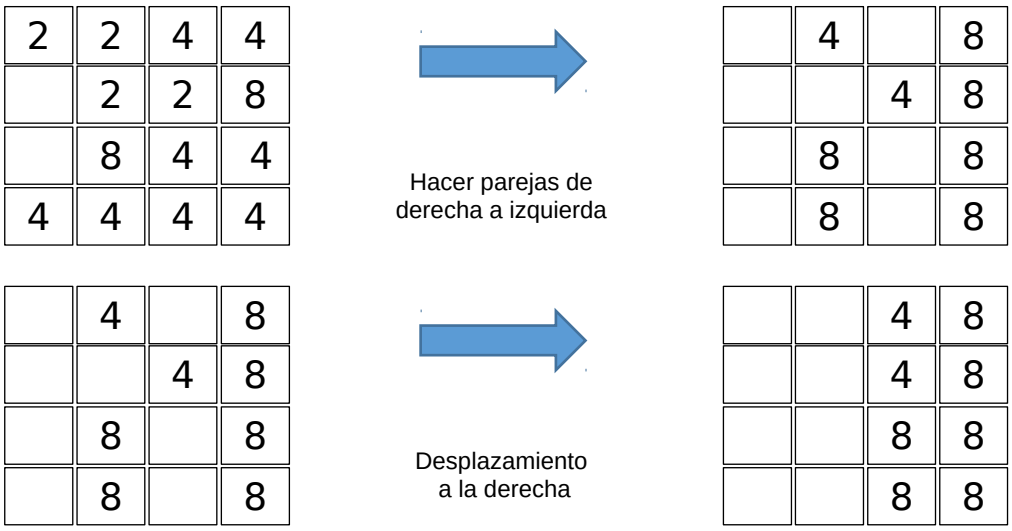
En la práctica implementaremos los 4 tipos de movimientos, izquierda, derecha, arriba y abajo, utilizando solo el desplazamiento a la derecha (subrutina *shiftNumbersRP1*), buscar parejas de derecha a izquierda (subrutina *addPairsRP1*) y haciendo una o más rotaciones a la derecha de 90° de la matriz (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 a la matriz *m*.

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

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

Ejemplo de un movimiento a la derecha:





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

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

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

Movimiento arriba:
- Rotación de 90° a la derecha (por cada rotación copiar nuevamente la matriz *mRotated* a m).
- Desplazamiento a la derecha.
- Buscar parejas de derecha a izquierda.
- Volver a hacer desplazamiento a la derecha para juntar las parejas que se hayan hecho.

- Rotación de 270° a la derecha = 3 rotaciones de 90° a la derecha (por cada rotación copiar nuevamente la matriz *mRotated* a m).

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

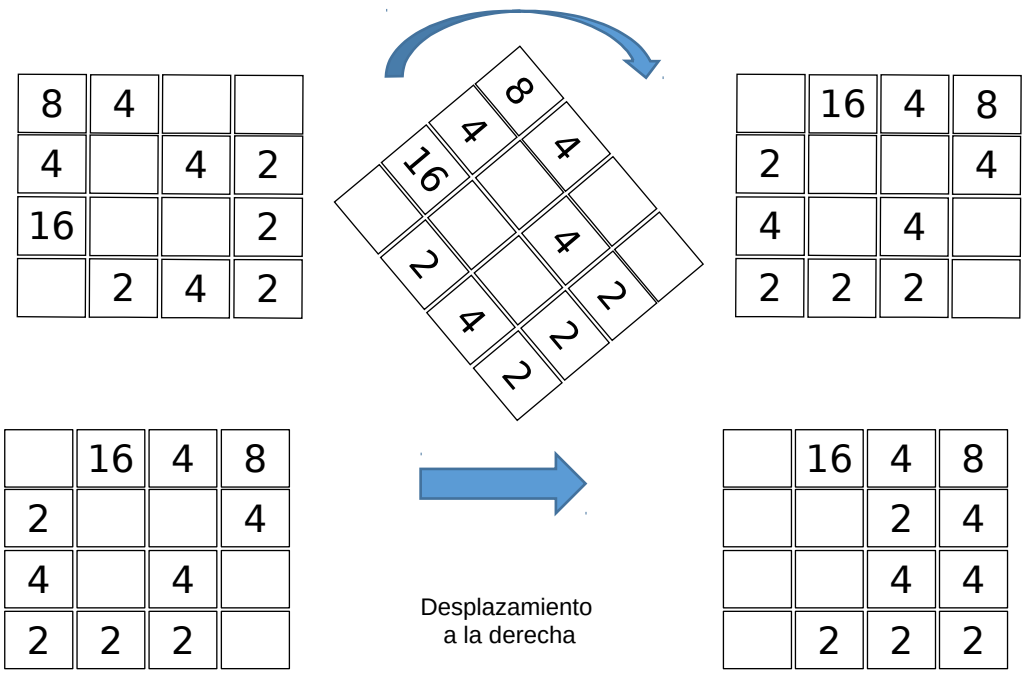
Haremos la implementación de esta forma para evitar la implementación de cuatro subrutinas de desplazamiento y cuatro de hacer parejas

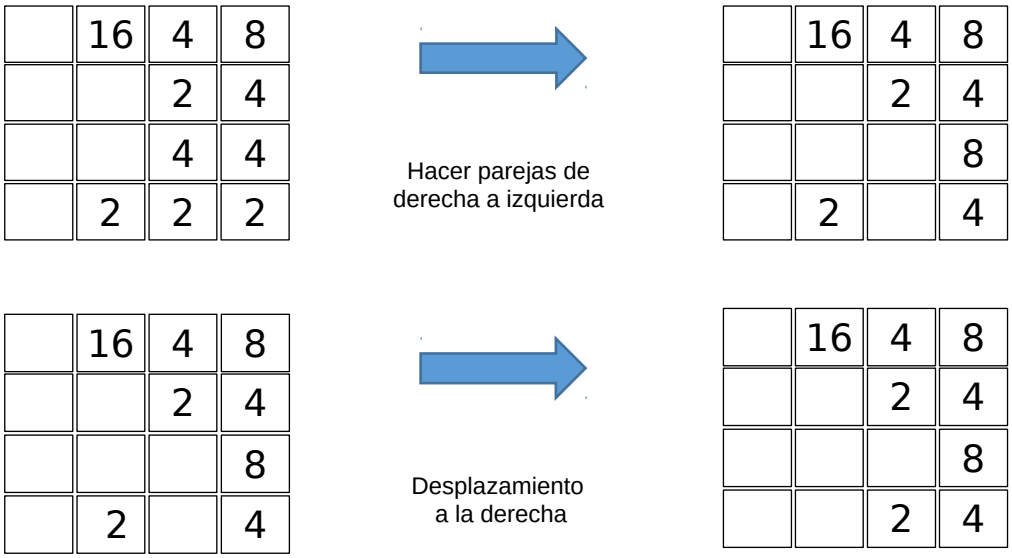
Veamos a continuación a modo de ejemplo qué operaciones se harían para hacer un desplazamiento abajo de los valores de la matriz.

Movimiento arriba
Tiene que hacer esto:

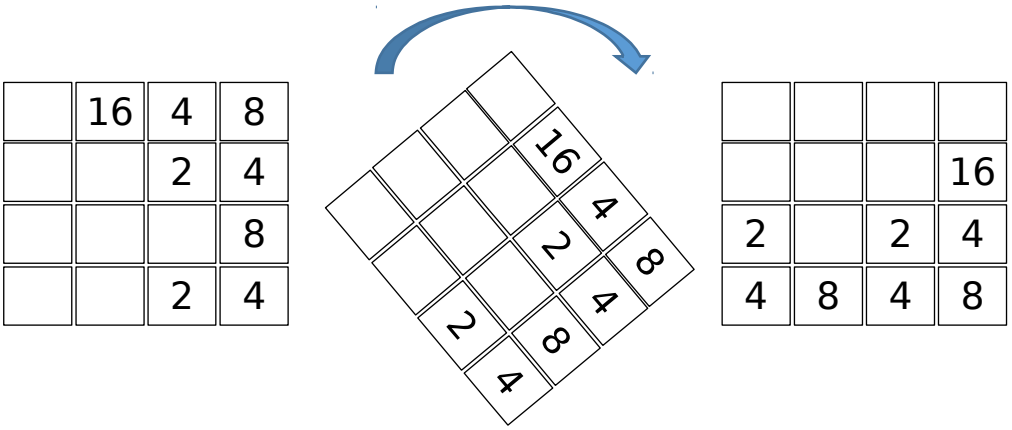
Pasos que sigue el programa:

Rotación de 90° a la derecha

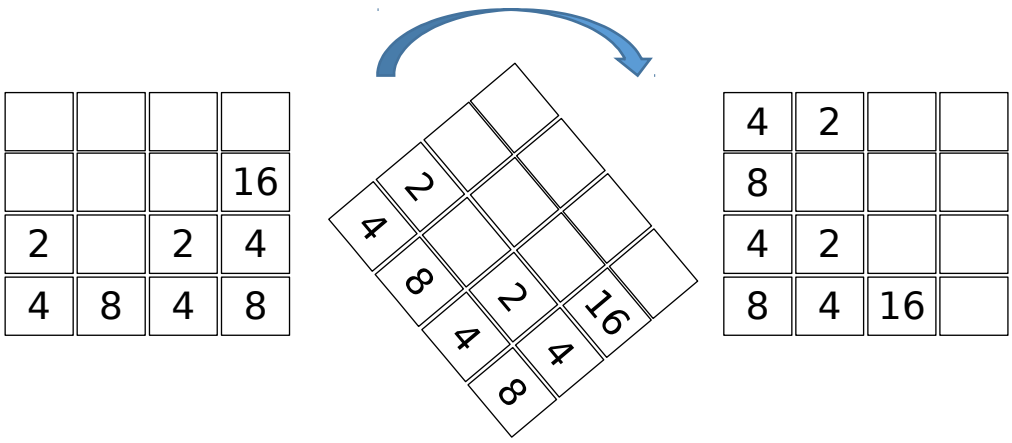




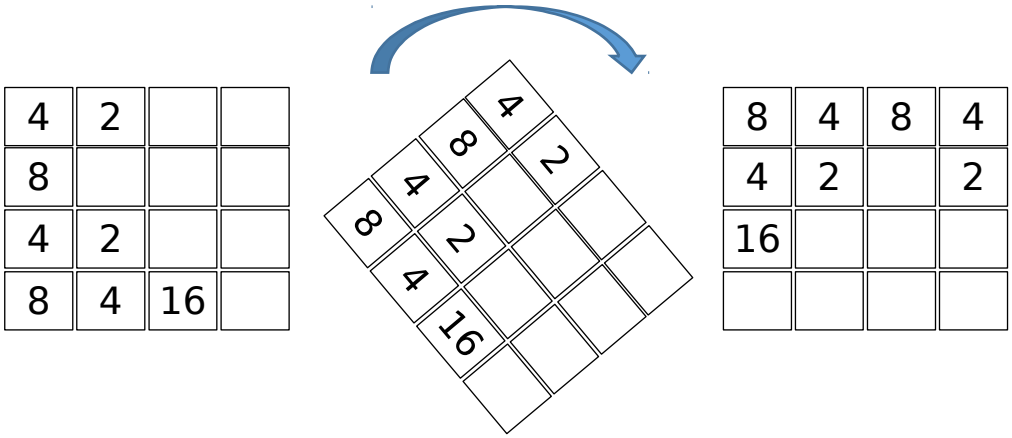
1.ª Rotación de 90° a la derecha



2a Rotación de 90° a la derecha



3a Rotación de 90° a la derecha



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

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

Os proporcionaremos también dos ficheros para esta segunda parte: 2Kp2c-es.c y 2Kp2-es.asm. De forma parecida a la primera parte, el código C no lo tenéis que modificar, solo tenéis 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 1 de mayo de 2020 os daremos los programas .c y .asm correspondientes a la 2.ª parte opcional.

Formato y fechas de entrega

La primera **parte** se puede entregar antes de las **23:59 del día 10 de abril de 2020** 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 15 de mayo de 2020** 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 estado satisfactoria se puede hacer una **segunda entrega** antes de las **23:59 del 15 de mayo de 2020** . 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 10-04-2020	**Primera parte Superada**	**Primera parte NO Superada NO Presentado**	**Primera parte Superada**	**Primera parte NO Superada NO Presentado**	**Primera parte NO Superada NO Presentado**
Segunda Entrega 15-05-2020	**Segunda parte NO Superada NO Presentado**	**Primera parte Superada**	**Segunda parte Superada**	**Primera parte Superada Segunda parte Superada**	**Primera parte NO Superada NO Presentado**
Nota Final Práctica	**B**	**C+**	**A**	**B**	**D/N**

Los alumnos que no superen la PRÁCTICA tendrán un suspenso (calificación 0-2) 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.

La entrega se tiene que hacer a través de l 'aplicación **Entrega y registro d'AC** de l 'aula. Se tiene que entregar solo un fichero con el código ensamblador muy comentado.

Es muy importante que el nombre del fichero tenga vuestros apellidos y nombre con el formato:

cognom1_cognom2_nombre_2Kp1.asm

Fecha tope Primera Entrega:

Viernes, 10 de abril de 2020 a las 23:59:59

Fecha tope Segunda Entrega:

Viernes, 15 de mayo de 2020 a las 23:59:59

Criterios de valoración

La práctica tiene que funcionar completamente para considerarse superada, y hay que implementar todas las funcionalidad pedidas en el enunciado, **la opción del menú correspondiente a ljuego completo en ensamblador tiene que funcionar correctamente.**

Las otras opciones del menú son solo para comprobar individualmente cada una de las subrutinas que se tienen que implementar.

No es suficiente para aprobar la práctica que las opciones correspondientes a las subrutinas individuales funcionen.

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

- Eficacia: que haga el que s'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 compacte pero claro). Usar modos d'adrizamiento adecuados: los que hagan falta. Evitar l'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+rsi], [rbx+rdi]. No se pueden utilizar índices con valores fijos para acceder a los vectores.

Ejemplo del que **NO** se puede hacer:

```
mov WORD [m+0], 0
mov WORD [m+4], 0
mov WORD [m+8], 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 s'ha d'explicar qué hace la instrucción (se da por supuesto que quién la lee sabe ensamblador) si no que s'ha d'explicar porque se usa una instrucción o grupo d'instrucciones (para hacer qué tarea de más alto nivel, relacionada con el problema que queremos resolver).

Implementación

Como 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 se tienen 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 hacer sacando el comentario de la llamada en 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:

```
//=====================================================
  rowScreen = 18;
  colScreen = 26;
  number = score;
  showNumberP1();   //Mostrar puntos
  //showNumberP1_C();
//=====================================================
```

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

```
//=====================================================
  rowScreen = 18;
  colScreen = 26;
  number = score;
  //showNumberP1();   //Mostrar puntos
  showNumberP1_C();
//=====================================================
```

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

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

```
gotoxyP1
printchP1
getchP1
readKeyP1
playP1
```

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

```
showNumberP1
updateBoardP1
rotateMatrixRP1
copyMatrixP1
shiftNumbersRP1
addPairsRP1
```