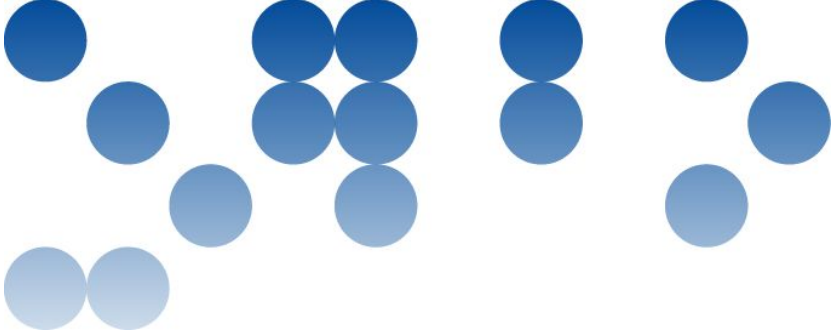

PRÁCTICA 1a parte: 8-puzle

Estructura de Computadores

Grado en Ingeniería Informática

sep20-feb21

Estudios de Informática, Multimedia y Telecomunicaciones

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 implementar un juego del 8-puzle.

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: 8-puzle

La práctica consiste en implementar el juego del “8-puzle” que consiste en ordenar 8 fichas en un tablero de 3 x 3 casillas hasta que el oden coincida con las posiciones del tablero solución, habrá una casilla vacía que nos permitirá hacer los movimientos para ordenar las fichas. Los movimientos que se

podrán realizar son mover una ficha: a la izquierda, derecha, arriba o abajo (nunca en diagonal). Cada movimiento consistirá en desplazar una ficha a la casilla vacía dejando su lugar vacío para el siguiente movimiento. Habrá unas teclas para desplazar el cursor por el tablero, la posición del cursor nos indicará la ficha que queremos mover y sólo se podrá hacer el movimiento si está al lado de la casilla vacía.

Las fichas del tablero serán caracteres alfanuméricos (números y letras) y consideraremos que el tablero está ordenado si las fichas están ordenadas siguiendo el mismo orden indicado en una matriz solución de valores, quedando la casilla vacía en la última posición (abajo – derecha).

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: 8P1c.c y 8P1.asm. El código C (8P1c.c) no lo tenéis que modificar sólo tenéis que implementar en ensamblador en el archivo 8P1.asm las funcionalidades siguientes:

- Mover el cursor: '**i**' (arriba), '**j**' (izquierda) , '**k**' (abajo) ,'**l**' (derecha). Controlando que al mover el cursor no salga fuera del tablero.
- Mover la ficha donde tenemos el cursor a la casilla vacía que tenga a su lado (arriba, abajo, izquierda o derecha): '**Espacio**'. Controlando que no se haga el movimiento si la casilla vacía no está al lado de la posición del cursor.
- Salir: si pulsamos '**ESC**' salir del programa (esta opción ya está implementada).

El programa en C genera un menú principal con 8 opciones:

1. updateBoard: llamar a la subrutina de ensamblador updateBoardP1 para actualizar el contenido del tablero.
2. spacePosScreen: llamar a la subrutina de ensamblador spacePosScreenP1 para buscar donde está el espacio en blanco dentro de la matriz tiles y posicionar el cursor en el lugar donde está el espacio.
3. copyMatrix: llamar a la subrutina de ensamblador copyMatrixP1 para copiar la matriz tilesIni en la matriz tiles.
4. moveTiles: llamar a la subrutina de ensamblador moveTilesP1 para mover la ficha en la dirección indicada por el carácter (charac): 'i':arriba, 'k':abajo, 'j':izquierda o 'l':derecha, a la posición donde está el

espacio, posición dentro de la matriz indicada por la variables (spacePos), controlando los casos en los cuales no se puede hacer el movimiento.

5. checkStatus: llamar a la subrutina de ensamblador checkStatusP1 para verificar el estado del juego. Si se han agotado los movimientos (moves == 0) poner state='3'; si no se ha podido hacer el movimiento (spacePos == newSpacePos), poner state='2'; sino poner state = '1' para continuar jugando.
6. Play Game: llamar a la subrutina de ensamblador playP1 que permite jugar llamando a las subrutinas de ensamblador implementadas por vosotros, para comprobar que toda la práctica funciona correctamente.
7. Play Game C: llamar a la función de C playP1_C que permite jugar llamando a todas las funciones implementadas en lenguaje C. **Sólo para ver el funcionamiento del juego.**
0. Exit, finalizar el programa.

En esta primera parte el juego del 8-puzle no estará completamente implementado, faltará por implementar la funcionalidad que comprueba que el juego se ha acabado si todas las fichas están ordenadas según la matriz solución y el espacio en blanco está en la última casilla. En la segunda parte opcional se implementaran las funcionalidades necesarias para tener un juego totalmente funcional.

SEGUNDA PARTE OPCIONAL: En la segunda parte se tendrán que implementar las funcionalidades adicionales necesarias para completar todas las funcionalidades del juego del 8-puzle.

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: 8P2c.c y 8P2.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 27 de noviembre de 2020 os daremos los archivos .c y .asm correspondientes a la 2a parte opcional.

La **primera parte** se puede entregar antes de las **24:00 del día 6 de noviembre 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 **24:00 del 11 de diciembre 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 sido satisfactoria se puede hacer una **segunda entrega** antes de las **24:00 del 11 de diciembre 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 06-11-2020	**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 11-12-2020	**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 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 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_8P1.asm

Fecha límite de la primera entrega:

Viernes, 6 de noviembre de 2020 a les 24:00:00

Fecha límite de la segunda entrega:

Viernes, 11 de diciembre de 2020 a les 24:00:00

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 al juego 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:

1. 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.
2. 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: [tiles+rsi], [rbx+rdi]. No se pueden utilizar índices con valores fijos para acceder a los vectores.

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

```
mov BYTE [tiles+0], 0
mov BYTE [tiles+1], 0
mov BYTE [tiles+2], 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 se tiene que 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).

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:

```
//=====================================================
updateBoardP1();
//updateBoardP1_C();
//=====================================================
```

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

```
//=====================================================
//updateBoardP1();
updateBoardP1_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
printMessageP1
playP1
```

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

```
updateBoardP1
spacePosScreenP1
copyMatrixP1
moveTileP1
checkStatusP1
```