PRÁCTICA 2a 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 o un 4 que se añade al tablero en una casilla vacía seleccionada aleatoriamente.

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

Hay que 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 sé pueden hacer parejas y no hemos llegado a 2048.

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

Os proporcionamos 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 tenéis que implementar en ensamblador las nuevas funcionalidades y modificar las subrutinas que habéis hecho en la primera parte para trabajar el paso de parámetros entre las subrutinas de ensamblador y también con las funciones de C.

Esta **segunda parte** se tiene que entregar antes de las **23:59 del 18 de diciembre de 2015**. Si en la primera entrega se superó la 1a parte se puede llegar a una puntuación de A en les prácticas.

En cambio, si no se hizo la primera entrega esta primera entrega o no fue satisfactoria se puede entregar la primera parte juntamente con esta 2a parte, 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 se tiene que explicar que 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_p2.asm

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 hacer 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:

```
//=====================================================
int p=score;
showNumberP2(RowScreenIni+DimMatrix*2,ColScreenIni+8,p); //Mostrar puntos
//showNumberP2_C(RowScreenIni+DimMatrix*2,ColScreenIni+8,p); //Mostrar puntos
//=====================================================
getch_C();
```

El código llama a la subrutina de ensamblador showNumberP2(), podemos cambiar el comentario y llamar a la función de C.

```
//=====================================================
int p=score;
//showNumberP2(RowScreenIni+DimMatrix*2,ColScreenIni+8,p); //Mostrar puntos
showNumberP2_C(RowScreenIni+DimMatrix*2,ColScreenIni+8,p); //Mostrar puntos
//=====================================================
getch_C();
```

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

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

```
;;;;;
; Convierte el valor recibido como parámetro (edx) de tipo int (DWORD)
; a los 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.
; En 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 la fila (edi) y la columna (esi) recibidos como
; parámetro, 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 subrutina gotoxyP2 y para
; mostrar los caracteres a la subrutina printchP2 implementando
; correctament el paso de parámetros.
;
; Variables utilizadas:
; Ninguna.
;
; Parámetros de entrada:
; rdi (edi): Fila donde lo queremos mostrar en pantalla.
; rsi (esi): Columna donde lo queremos mostrar en pantalla.
; rdx (edx): Valor que queremos mostrar en pantalla.
;
; Parámetros de salida :
; Ninguno.
;;;;;
showNumberP2:

;;;;;
; 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 subrutina gotoxyP2, y para
; mostrar los números de la matriz y los puntos a la subrutina showNumberP2
; implementado correctamente el paso de parámetros.
;
; Variables utilizadas:
; m     : matriz 4x4 donde hay los números del tablero de Juego.
; score : puntos acumulados hasta el el momento.
;
; Parámetros de entrada:
; Ninguno.
;
; Parámetros de salida :
; Ninguno.
;;;;;
updateBoardP2:


;;;;;
; Calcular el valor del índice para acceder a una matriz (4x4) que
; guardaremos en el registro (eax) a partir de la fila (edi) y
; la columna (esi) recibidos como parámetro.
; eax=((edi*DimMatrix)+(esi))*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:
; Ninguna.
;
; Parámetros de entrada:
```

```
; rdi(edi) : fila para acceder a la matriz (4x4).
; rsi(esi) : columna para acceder a la matriz (4x4).
;
; Parámetros de salida :
; rax(eax) : índice para acceder a la matriz (4x4) de tipo WORD.
;;;;;
calcIndexP2:

;;;;;
; Rotar a la derecha la matriz recibida como parámetro (edi), 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 fer la matriz traspuesta.
; La matriz recibida como parámetro 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] és DWORD[m+12].
; No se tiene que mostrar la matriz.
;
; Variables utilizadas:
; Ninguna.
;
; Parámetros de entrada:
; rdi(edi): Dirección de la matriz que queremos rotar.
;
; Parámetros de salida :
; Ninguno.
;;;;;
rotateMatrixRP2:

;;;;;
; Copiar una matriz del juego en otra matriz y retornar los puntos que
; tenga asociados. Esto permitirá copiar dos matrices después de una
; rotación y gestionar la opción '(u)Undo' del juego.
; Copia la matriz origen, segundo parámetro (esi), sobre la matriz
; destino, primer parámetro (edi) y devuelve los puntos asociados,
; tercer parámetro (rdx).
;
; Variables utilizadas:
; Ninguna.
;
; Parámetros de entrada:
; rdi(edi): Dirección de la matriz destino.
; rsi(esi): Dirección de la matriz origen.
; rdx(edx): Puntos asociados a la matriz origen
;
; Parámetros de salida :
; rax(eax) : Puntos asociados a la matriz origen.
;;;;;
copyStatusP2:

;;;;;
; Desplazar a la izquierda los números de cada fila de la matriz
; recibida como parámetro (edi), 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) se tienen que contar los
; desplazamientos.
; Si se ha movido algún número (número de desplazamientos>0) pondremos
; la variable (moved) a 1.
; Retornaremos el número de desplazamientos hechos.
; Si una fila de la matriz es: [0,2,0,4] y el número de desplazamientos
; es 0, quedará [2,4,0,0] y el número de desplazamientos será 2.
; 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] és 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.
;
; Parámetros de entrada:
; rdi(edi): Dirección de la matriz que queremos desplazar los números.
;
; Parámetros de salida :
; rax(eax) : Número de desplazamientos hechos.
;;;;;
shiftNumbersLP2:

;;;;;
; Emparejar números iguales des la izquierda de la matriz recibida como
; parámetro (edi) y acumular los puntos de las parejas que se hagan.
; Recorrer la matriz por filas de izquierda a derecha y de arriba a bajo.
; 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],
; puntos = (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.
; Retornamos los puntos obtenidos de hacer la parejas (no sumarlos a
; la variable (score)).
; 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] és DWORD[m+12].
; No se tiene que mostrar la matriz.
;
; Variables utilizadas:
; moved : Para indicar si se han hecho cambios en la matriz.
;
; Parámetros de entrada:
; rdi(edi): Dirección de la matriz que queremos hacer las parejas.
;
; Parámetros de salida :
; rax(eax) : Puntos obtenidos de hacer las parejas.
;;;;;
addPairsLP2:


;;;;;
; Verificar si se ha llegado a 2048 o si no se puede hacer algún movimiento.
; Si hay el número 2048 en la matriz (m), cambiar el estado a 3
; (state=3) para indicar que se ha ganado (WIN!).
; Si no hemos ganado, mirar si se puede hacer algun movimento,
; Si no se puede hacer ningún movimiento cambiar el estado a 4
; (state=4) para indicar que se ha perdido (GAME OVER!!!).
; Recorrer la matriz (m) por filas de derecha a izquierda y de a bajo a
; arriba contando las casillas vacías y mirando si hay el número 2048.
; Si hay el número 2048 poner (state=3) y acabar.
; Si no hay el número 2048 y no hay casillas vacías, mirar si se puede
; hacer alguna pareja en horizontal o en vertical. Par hacerlo
; hay que copiar la matriz (m) sobre la matriz (mAux) llamando
; (copyStatusP2), hacer parejas en la matriz (mAux) para mirar si se
; pueden hacer parejas en horizontal llamando (addPairsLP2) y guardar
; los puntos obtenidos, rotar la matriz (mAux) llamando (rotateMatrixRP2)
; y copiar la matriz rotada (mRotated) en la matriz (mAux) llamando
; (copyStatusP2), volver a hacer parejas en la matriz (mAux) para
; mirar si se pueden hacer parejas en vertical llamando (addPairsLP2)
; y acumular los puntos obtenidos con los puntos obtenidos antes, si
; los puntos acumulados son cero, quiere decir que no se pueden hacer
; parejas y se tiene que poner (state=4).
```

```
; Finalmente poner (moved=0), los movimientos que se hayan hecho no son
; cambios o movimientos hechos en la matriz (m) y no se tiene que considerar.
; 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] és DWORD[m+12].
; No se puede modificar ni la matriz (m), ni la matriz (mUndo).
;
; Variables utilizadas:
; m        : Dirección de la matriz que queremos verificar.
; mAux     : Dirección de la matriz donde copiaremos los datos para hacer
;            verificaciones.
; mRotated : Dirección de la matriz donde se hace la rotación.
; state    : Indica el estado del juego. 0:salir, 1:jugar, 2:gana, 3:pierde
; moved    : Para indicar si se han hecho cambios en la matriz.
;
; Parámetros de entrada:
; Ninguno
;
; Parámetros de salida :
; Ninguno
;;;;;
checkEndP2:
```

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

```
gotoxyP2
printchP2
getchP2
readKeyP2
playP2
```