

PRÁCTICA 4: SUBROUTINAS. C + ENSAMBLADOR

OBJETIVOS Y COMPETENCIAS A ADQUIRIR:

- Comprensión del uso de subrutinas (SBR).
- Construcción de un programa con una parte en lenguaje de alto nivel (C) y otra parte en ensamblador.

PROBLEMA A RESOLVER

Dada una tabla de N enteros de 32 bits (N constante arbitraria definida con EQU) almacenados de forma consecutiva en memoria, deben ordenarse de menor a mayor empleando el método de ordenación quicksort.

MÉTODO QUICKSORT:

Para realizar la ordenación emplear el método de ordenación Quicksort en su versión recursiva. Su codificación a alto nivel (para una tabla de N elementos) es la siguiente:

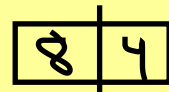
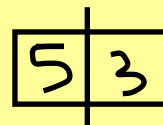
```
/* Ordenación tabla T de N elementos: T[0],T[1],...,T[N-1]
 * aplicando el método quicksort */
```

```
void ordena(int *T,int n) {
    qksort(0,n-1);
}

void qksort(int iz,int de) {
    int i,j,x,w;

    i=iz;j=de;x=T[(iz+de)/2];
    do {
        while (T[i]<x) i=i+1;
        while (x<T[j]) j=j-1;
        if (i <= j) {
            w=T[i];T[i]=T[j];T[j]=w;
            i=i+1;j=j-1;
        }
    } while (i<=j);
    if (iz<j) qksort(iz,j);
    if (i<de) qksort(i,de);
}
```

$i = 4$ $j = 3$
 $x = 1$



PARTE 1: SUBROUTINAS

La primera parte consiste en hacer unos ejercicios relacionados con SBR, trabajando exclusivamente a nivel ensamblador. Para ello crea un nuevo proyecto como en prácticas anteriores y sigue los siguientes pasos:

1. **Implementar una SBR básica con parámetros y resultados:** Como primer ejercicio, realizar un programa con una tabla T de N enteros de 32 bits (N constante arbitraria definidas con EQU) cuyos elementos queremos sumar. La suma se hará por medio de una SBR **suma** con dos parámetros y un resultado (todos en la pila). Los parámetros serán la dirección de comienzo de la tabla (parámetro por referencia) y el número de elementos de la misma (parámetro por valor). La SBR devuelve como resultado la suma de los elementos de la tabla. El programa principal debe llamar a la SBR **suma** y almacenar la suma de los elementos en la variable `stotal` en el área de datos.
2. **Implementar la SBR ordena del método quicksort:** Añadir al programa principal una llamada a la SBR **ordena**. Esta SBR tiene los mismos dos parámetros que la SBR **suma** del apartado anterior (T por referencia y N por valor), pero ahora la SBR ordena no devuelve ningún resultado.
3. **Implementar la subrutina recursiva qksort:** Esta SBR recursiva es la que hace efectiva la ordenación de T. Tiene dos parámetros y no devuelve ningún resultado. Los parámetros pueden verse como las direcciones del primer y último elemento del vector a ordenar (iz y de). **Recordar que los parámetros (iz y de) se deben leer del bloque de activación.** Comprobar si ordena correctamente la tabla `T[10]={4,8,5,9,1,6,2,7,3,0}`

PARTE 2: C + ENSAMBLADOR

Una vez implementado correctamente el algoritmo de ordenación recursivo quicksort, esta segunda parte se va a dedicar a realizar un programa con parte en un lenguaje de alto nivel (C) y otra parte en ensamblador (ARM). La idea es hacer un programa en C que ordene varias tablas de enteros de 32 bits por el método quicksort. La interconexión entre C y ensamblador se hará por medio de la SBR **ordena** anterior (es la única que hay que modificar), manteniendo igual la SBR **qksort**. El esquema del programa en C (archivo <nombre>.c) es el siguiente:

```
#include <stdlib.h>      /* fichero con declaracion de funciones srand,rand */
#define M 5              /* numero de tablas a ordenar */
#define N 10             /* tamaño tabla a ordenar */

int T[N];                /* tabla de datos a ordenar */

extern
void ordena( int *tabla, int num_elem );    /* declara funcion externa de ordenacion */

int main() {
    int i,j,k;

    k=1<<30;              /* k=2^30 */
    srand(23456);         /* semilla numeros pseudo-aleatorios */

    for (i=0;i<M;i=i+1) { /* M ordenaciones a realizar*/
        for (j=0;j<N;j=j+1) {
            T[j]=rand()-k; /* numeros enteros aleatorio en [-2^30,2^30-1]*/
        }
        ordena(T,N);      /* llamada a funcion de ordenacion */
    }
    while(1);             /* bucle infinito de finalizacion */
}
```

Para entender el programa, se describen las principales instrucciones:

```
#include <stdlib.h> Incluir descripción de funciones de biblioteca srand,rand
#define M 3          Declara una constante N con valor 3
int                 Tipo de dato entero de 32 bits.
int *               Tipo de dato puntero (dirección) a un int.
/* .... */          Comentario
extern             Directiva que indica al compilador que la función ordena
                    es externa, que su código no está en el fichero fuente.
srand(23456)         semilla para generador de números aleatorios (poniendo
                    otro número sale una secuencia distinta).
rand()              devuelve un número aleatorio en el rango  $[0, 2^{31}-1]$ .
                    Restándole  $k=2^{30}$  conseguimos un número entero aleatorio en
                    el rango  $[-2^{30}, 2^{30}-1]$ .
```

Todas las arquitecturas definen junto con el repertorio de instrucciones un ABI (Application Binary Interface) para permitir la compatibilidad entre aplicaciones. La definición del ABI incluye tipos de datos, alineamiento y tamaño de los mismos, ficheros objeto generados desde distintos lenguajes (como es nuestro caso), llamadas al sistema y la convención de llamadas a funciones que es el aspecto donde se centra esta práctica. El ABI de ARM define en el ARM Procedure Call Standard (AAPCS) cómo se comunican las SBR entre sí.

Para que el compilador de C pueda traducir la segunda línea remarcada en negrita (la llamada a la función **ordena**), es necesario que el compilador sepa qué parámetros y resultados tiene la función. Para ello está el prototipo o declaración de la misma (primera línea remarcada en negrita).

```
extern void ordena( int *tabla, int num_elem );
```

Esta declaración indica al compilador que la función **ordena** tiene dos parámetros; El primero (**int *tabla**) es un puntero a un entero de 32 bits (es la forma en C de pasar una tabla por referencia) y el segundo es un entero de 32 bits (**num_elem**) que indica el número de enteros que queremos ordenar. La función no devuelve ningún resultado (**void**). Notar que los parámetros de la función **ordena** coinciden exactamente con los de la SBR **ordena** de la parte 1, por lo que se podrá reutilizar. Pero ahora, en vez de tener los parámetros en la pila, hay que adaptarlos a la forma de traducir del compilador de C. Dada la declaración anterior, ahora se tiene lo siguiente:

r0 = Dirección de comienzo de la tabla de enteros a ordenar (tabla por referencia)

r1 = Número de elementos de la tabla (**num_elem** por valor).

Por lo tanto, se debe modificar la SBR **ordena** de la parte 1 para que tome los parámetros de los registros **r0** y **r1**, no de la pila. Para no perder estos parámetros, no se debe apilar/dsapilar los registros **{r0-r1}**, pero si se tiene que apilar/dsapilar cualquier otro registro que se utilice. El esquema del fuente en ensamblador es:

```

AREA codigo, CODE
EXPORT ordena
ordena    ; SBR que ordena una tabla de enteros de 32 bits. Parametros:
          ; r0 = @ de comienzo de la tabla a ordenar (tabla por referencia)
          ; r1 = Numero de elementos de la tabla (num_elem por valor)

          PUSH {lr}
          <apilar registros utilizados, excepto r0,r1>


          <llamar a qksort recursivo>

          <desapilar registros utilizados>
          POP {pc}

qksort    ; SBR qksort realizada en la parte 1 (no es necesario modificarla)
END

```

El fuente en ensamblador es sólo parte de un programa, no un programa entero. Ahora no hay área de datos (porque los datos se declaran en el fuente C) ni hay directiva ENTRY (porque el programa empieza en el función main de C). La directiva marcada en negrita (EXPORT ordena) hace visible al resto de ficheros objeto el símbolo `ordena`, en tiempo de enlazado. Esto es necesario para que el linker identifique la función C ordena con la SBR ensamblador ordena y realice correctamente los saltos. Seguid los siguientes pasos:

1. Crear un proyecto nuevo para el microcontrolador LPC2105 de NXP. Ahora no serán necesarios los ficheros *.sct y *.ini utilizados en prácticas anteriores. En este caso, al crear el proyecto y definir el dispositivo de destino (Target), cuando la herramienta nos pregunte si queremos añadir al proyecto el **código de inicialización** para el microcontrolador, **responder SI**. Esto copiará en la carpeta del proyecto un fichero "Startup.s" que contiene el código de inicialización del dispositivo (no hace falta entenderlo). Luego pulsar el botón de Target Options () de la barra de herramientas (o, a través del menú, Project -> Options for Target 'Target 1'...) y comprobar en la pestaña "Debug" que la opción **"Run to main"** está activada (debería estarlo por defecto). Con esta opción nos aseguramos que la depuración de nuestros programas comienza en la primera instrucción de nuestro programa en C.
2. Copiar el fuente C del primer esquema para realizar una serie de llamadas a la función externa de ordenación.
3. Modificar el fuente en ensamblador de la parte 1 (de acuerdo con el segundo esquema) con las SBR **ordena** (modificada) y **qksort** que encapsule el algoritmo de ordenación y lo conecte con el fuente C anterior.
4. Añadir al proyecto los ficheros fuente C y ensamblador, construir el proyecto y depurarlo hasta que funcione correctamente. Para evitar problemas al compilar, poned nombres distintos a los fuentes C y ensamblador.

EVALUACIÓN:

La evaluación de esta práctica se realizará el día del examen, de forma individual o en parejas. Deberá mostrarse el correcto funcionamiento de la práctica y responder a las preguntas que os hagan los profesores al respecto.