

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.

PARTE 1: NO EVALUABLE

La primera parte consiste en hacer un ejercicio relacionado con SBR, trabajando exclusivamente a nivel ensamblador. Para ello crea un nuevo proyecto como en prácticas anteriores e **implementa 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 definida 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 `total` en el área de datos.

PARTE 2: PROBLEMA DE LAS N REINAS

Resolver el problema de las `N` reinas en un tablero de ajedrez. En este problema hay que calcular la posición de `N` reinas en un tablero de tamaño `NxN` de forma que ninguna de las `N` reinas amenace a otra. Recordar que una reina amenaza a todas aquellas piezas que se encuentre en la misma fila, columna o diagonales (ver figura 1).

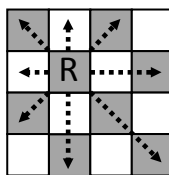


Figura 1. Patrón de amenaza de la figura Reina (R) en un tablero de ajedrez de 4x4.

PLANTEAMIENTO DEL PROBLEMA

Una solución recursiva de este problema consiste en utilizar un algoritmo de backtracking. Debemos tener en cuenta que:

- Cada reina debe ocupar una fila diferente, luego buscaremos para cada fila del tablero una columna que permita una posición válida (sin amenazas a las reinas anteriores).
- Cada reina añadida en el tablero (ocupa una fila distinta) puede causar amenaza si coincide en la diagonal o columna de las reinas anteriores. Si la nueva reina no causa amenaza, se puede asegurar que el estado del tablero es válido hasta ese momento.

Es importante que recordéis que para $N=1$ hay una única solución (trivial) y para $N=2$ y 3 no hay solución, por lo que el problema tiene sentido para $N \geq 4$.

Pasos del algoritmo recursivo:

Para cada columna de la fila i , se comprueba la validez del posicionamiento de la reina i -ésima.

Si la posición escogida para la reina i -ésima amenaza a alguna de las otras reinas del tablero, entonces se elige una columna nueva para la reina i -ésima.

Si la posición escogida para la reina i -ésima no amenaza a ninguna de las reinas anteriores del tablero, entonces se busca una posición válida para la reina $(i+1)$ -ésima (invocación recursiva).

Si la posición escogida para la reina i -ésima no amenaza a ninguna de las otras reinas del tablero y además i es igual al tamaño del tablero, entonces se imprime la solución pantalla.

La codificación del algoritmo a alto nivel sería:

```
// resolverReinas resuelve el problema de n reinas en el vector solucion
void resolverReinas (int n,int solucion[]) {
    int i;
    for (i=0; i<n; i++) solucion[i] = -1;
    if (n>=4) colocarReina(0,solucion,n);
}

// colocarReina calcula recursivamente (backtracking) una solucion
void colocarReina (int fila, int solucion[], int n) {
    if (fila<n) // Quedan reinas por colocar
        for (solucion[fila]=0; solucion[fila]<n; solucion[fila]++)
            if (comprobar(fila,solucion)==TRUE)
                colocarReina (fila+1,solucion,n);
    else // No quedan reinas por colocar (solución)
        mostrarTablero(solucion,n);
}

int comprobar (int fila, int reinas[]) { // comprueba validez de la
    int i;                               // solucion parcial hasta fila
    for (i=0; i<fila; i++)
        if ( (reinas[i]==reinas[fila]) // Misma columna
            || (abs(fila-i)==abs(reinas[fila]-reinas[i])) ) // Misma diagonal
            return FALSE;
    return TRUE;
}

void mostrarTablero (int reinas[], int n) { // dibuja solucion
    int i,j;
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++)
            if (reinas[i]==j) printf("#");
            else printf("-");
        printf("\n");
    }
}
```

PROGRAMACIÓN HÍBRIDA: LENGUAJE C + ENSAMBLADOR ARM

Como parte de los objetivos de la práctica os pedimos que la implementación al problema de las N reinas sea híbrida, es decir, que lo hagáis utilizando tanto lenguaje C como lenguaje ensamblador ARM. La definición de variables globales y la función `main()` se realizará en el fichero fuente `main.c`. Una propuesta de fichero `main.c` es la siguiente.

```
#define N 8      // número de reinas

extern void resolverReinas (int n, int reinas[]);

int main () {
    int solucion[N]; // Vector de posiciones de reinas en cada fila
    int nreinas = N; // Numero de reinas
    int i;           // Contador

    // Inicializar vector (ninguna reina colocada)
    for (i=0; i<nreinas; i++) solucion[i] = -1;
    if (nreinas>=4) resolverReinas (nreinas,solucion);
    while(1);      // bucle infinito finalizacion
}
```

Por otro lado debéis escribir el fuente en ensamblador que implemente el algoritmo de backtracking que encuentre las soluciones al problema de las N reinas. Desde el fichero fuente en C (`main.c`) se invoca a la subrutina `resolverReinas` que es el punto de entrada al algoritmo recursivo `colocarReina`. Un esquema del fichero `practica4.s` es la siguiente.


```
(tab)      AREA codigo, CODE
(tab)      EXPORT resolverReinas
; SBR que resuelve el problema de las N reinas. Parametros:
; r0 = Numero de reinas N (por valor)
; r1 = Vector con las posiciones de las reinas (por referencia)

resolverReinas
(tab)      PUSH {lr}
(tab)      <apilar registros utilizados>
(tab)      <llamar a colocarReina(0,reinas,nreinas)>
(tab)      <desapilar registros utilizados>
(tab)      POP {pc}

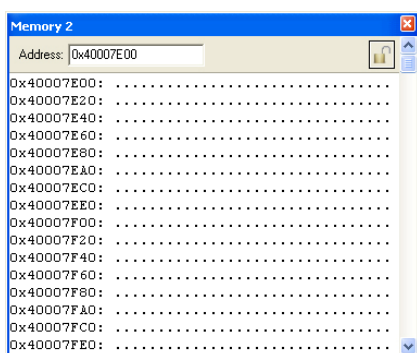
; subrutinas auxiliares que estiméis oportunas (todo en ensamblador)
colocarReina, comprobarReina, mostrarTablero
END
```

El fuente en ensamblador, `practica4.s`, 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, `main.c`) ni hay directiva `ENTRY` (porque el programa empieza en el función `main` de C). La directiva marcada en negrita (`EXPORT resolverReinas`) hace visible al resto de ficheros objeto del proyecto el símbolo `resolverReinas`, en tiempo de enlazado (linker). Esto es necesario para que el linker (enlazador)

identifique la función C `resolverReinas` con la SBR ensamblador `resolverReinas` del fichero `practica4.s` 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 `*.set` 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 la función `main()` de nuestro programa en `C, main.c`.
 2. Crear el fuente C del primer esquema para realizar una serie de llamadas a la función externa de **resolverReinas**.
 3. Crear el fuente en ensamblador con la SBR **resolverReinas** que conecte con el fuente C anterior, `main.c`. Recuerda, para combinar el uso de C y ensamblador, en esta SBR hay que seguir las reglas del compilador de C (paso de parámetros en los registros `r0` a `r3` y resultado en `r0`).
- Añadir al fuente en ensamblador el resto de SBR que resuelven el problema (`colocarReina`, `comprobarReina`, `mostrarTablero`). Estas SBR son internas al módulo en ensamblador y por lo tanto se pueden implementar siguiendo el modelo general de bloque de activación.
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.

VISIONADO



Como parte del problema se os pide mostrar por pantalla las soluciones que encontréis. A modo de pantalla, utilizaremos una ventana que muestre el contenido de una región de memoria, tal y como se muestra en la imagen a la izquierda.

La región de memoria de pantalla de texto será la comprendida entre las direcciones `0x40007E00` y `0x40007FFF`. Para ver el contenido de esa pantalla, conviene añadir (en debug session) una nueva vista de memoria (View->Memory Windows->Memory 2) con la geometría de la figura adjunta (formato

ASCII, 16 filas, 32 columnas a partir de la dirección `0x40007E00`).

Para representar el tablero y las posiciones de las reinas podéis utilizar aquellos caracteres ASCII que creáis oportunos (por ejemplo `'#'` para reina, `'-'` para blanco).

EVALUACIÓN:

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