PROYECTO HARDWARE

Memoria Práctica 1: Desarrollo de código para el procesador ARM

29/10/2015

Daniel Rueda Macías (559207)
Adrián Susinos Moreno (650220)

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

ÍNDICE

RESUMEN

En esta práctica hemos realizado diferentes funciones dedicadas a calcular los candidatos posibles de un tablero de sudoku.

Dado un tablero de 9x9 con algunos números como pista inicial, nuestra labor ha sido crear funciones para recorrer cada posición del tablero y calcular los candidatos de dicha posición, leyendo fila, columna y sub-cuadrícula a la que pertenece, anotando en los últimos 9 bits del número los números no candidatos poniendo el bit correspondiente a 1, quedando así a 0 los candidatos.

Hemos implementado dos tipos de funciones, una que recorre todo el tablero la cual llama en cada iteración a la segunda función, que para cada celda se encarga de calcular sus candidatos (recorriendo fila, columna y sub-cuadrícula por este orden). Estas funciones han sido implementadas en C, ARM y Thumb (esta última tecnología solo ha sido usada en la función de calcular los candidatos).

Una vez que teníamos todas las funciones implementadas, hemos realizado diferentes pruebas combinando ambas versiones entre ellas (C-C, C-ARM, ARM-C, ARM-ARM, C-THUMB y ARM-THUMB).

Una vez comprobado que la ejecución era correcta y que los resultados de la cuadrícula en memoria para cada combinación eran idénticos, iniciamos un estudio del número de instrucciones y tiempo de ejecución para cada una de las combinaciones mencionadas anteriormente.

INTRODUCCIÓN

Nos encontramos ante el entorno de desarrollo de código Eclipse. Tras configurar el compilador y el debugger siguiendo los pasos de los video-tutoriales facilitados por los profesores de la asignatura, creamos un proyecto en C y añadimos los ficheros iniciales. Tras esto hicimos un primer estudio del código que nos proporcionaban, y observamos las diferentes funciones que teníamos que realizar.
Antes de trabajar sobre Eclipse, hicimos un pequeño esquema a papel, donde plasmamos las ideas, algoritmos, alternativas… que más tarde comenzaríamos a programar.

OBJETIVOS

La finalidad de esta práctica es que dado un tablero de sudoku, sepamos calcular los candidatos para cada celda en distintos lenguajes: C, ARM y modo Thumb. Combinándolos como hemos dicho anteriormente.

Además, esto nos sirve para familiarizarnos con el trabajo en un entorno de programación usando estos lenguajes pero además teniendo que intercambiar y llamar entre ellos durante la ejecución del código.

Por último, es muy importante saber depurar las distintas ejecuciones para observar con todo detalle la funcionalidad del código, analizar errores, ver cambios en memoria… y el correcto y continuo desarrollo sobre todo de un programa en ensamblador.

METODOLOGÍA

Esquema

En primer lugar vamos a incluir un esquema del proyecto con los diferentes ficheros y funciones utilizadas:

- **init_b.asm**
    - 6 cuadrículas idénticas en memoria
    - llamada a la función principal de sudoku_2015.c

- **sudoku_2015.c**
    - celda_poner_valor
    - celda_leer_valor
    - sudoku9x9
    - sudoku_candidatos_c
    - sudoku_recalcular_c
    - sudoku_recalcular_c_arm
    - sudoku_recalcular_c_thumb
    - compararCuadriculas

- **sudokuARM.asm**
    - sudoku_recalcular_arm
    - sudoku_candidatos_arm

- **sudokuTHUMB.asm**
    - sudoku_candidatos_thumb

- **sudokuARM_c.asm**
    - sudoku_recalcular_arm_c

- **sudokuARM_THUMB.asm**
    - sudoku_recalcular_arm_thumb

Ficheros del proyecto

A continuación, vamos a explicar brevemente la función de cada fichero en el proyecto:

- **initi_b.asm**: Este fichero crea 6 cuadrículas idénticas, cada una de las cuales será utilizada para las diferentes combinaciones del cálculo de candidatos. Las cuatro primeras cuadrículas se las pasamos a la función sudoku*9x9* por registros, guardando sus direcciones de inicio en los registros de r0-r3. Las cuadrículas restantes las guardamos en la pila y el compilador se encarga de obtenerlas de allí en la función. Además, este fichero es el encargado de saltar a la función principal del fichero *sudoku_2015.c,* la cual es *sudoku9x9.*

- **sudoku_2015.c**:

    - Contiene las declaraciones de las funciones que están contenidas en otros ficheros las cuales serán llamadas desde alguna función de este fichero (*sudoku_recalcular_arm, sudoku_candidatos_arm, sudoku_candidatos_thumb, sudoku_recalcular_arm_c y sudoku_recalcular_arm_thumb*)
    - También, contiene un look-up table el cual sirve para evitar hacer el algoritmo de la división a la hora de calcular los candidatos de la cuadrícula del sudoku. Es decir, cada vez que queramos calcular los candidatos de una cuadrícula, para ponernos al inicio de esta, usaremos este look-up table)
    - Función celda_leer_valor(): Dada una celda que le pasamos por parámetro, calcula el valor que tiene esta, es decir, el valor de los cuatro primeros bits. Cabe destacar que no realizamos la llamada a esta función ni en *sudoku_candidatos_ARM* ni en *sudoku_candidatos_thumb*, ya que el nivel de optimización utilizado (-O0), no tiene en cuenta el *inline*, siendo más sencillo realizar el cálculo del valor dentro de la misma función sin tener que llamar a esta.
    - Función sudoku_recalcular_c(): Recorre el tablero entero y llama a la función *sudoku_candidatos_c* para cada una de las celdas, y además cuenta y devuelve el número de celdas vacías del tablero. Por parámetros le pasamos la dirección de inicio del tablero.
    - Función sudoku_candidatos_c(): Calcula los candidatos en el tablero para una celda, denotada por su fila y su columna y devuelve verdad si la celda está ocupada y falso si la celda está vacía. Por parámetros le pasamos la dirección de inicio del tablero, el número de fila y el número de columna de nuestra celda.

    - Función sudoku_recalcular_c_arm(): Su funcionalidad es la misma que la de la función *sudoku_recalcular_c*, salvo que, en vez de llamar a la función sudoku_candidatos_c() llama a la función *sudoku_candidatos_arm()* que se explicará más adelante ya que se encuentra en otro fichero.
    - Función sudoku_recalcular_c_thumb(): Su funcionalidad es la misma que la función *sudoku_recalcular_c,* salvo que en vez de llamar a sudoku_candidatos_arm() llama a la función sudoku_candidatos_thumb() que se explicará más adelante.
    - Función comparaCuadriculas(): Compara dos cuadrículas recorriéndolas y comparando celda por celda; devuelve verdad si son distintas, falso en cualquier otro caso. Su finalidad es ver si tras ejecutar todas las combinaciones (C, ARM y THUMB), la memoria resultante de cada una son idénticas.

- **sudokuARM.asm**:
    - Función sudoku_recalcular_ARM(): Función en ARM que recorre el tablero y llama a la función *sudoku_candidatos_ARM* para cada celda del tablero, al final devuelve el número de celdas vacías del tablero. Recibe en r0 la dirección de inicio del tablero.
    - Función sudoku_candidatos_ARM: Función en ARM que calcula los candidatos en el tablero para una celda, denotada por su fila y su columna y devuelve verdad si la celda está ocupada y falso si la celda está vacía. Por parámetros le pasamos la dirección de inicio del tablero, el número de fila y el número de columna de nuestra celda.

- **sudokuTHUMB.asm**:
    - Función sudoku_candidatos_thumb: Función en ARM que se encarga de realizar el salto a la función *lanzadera* cambiando a modo Thumb.
    - Función lanzadera: Función en Thumb que calcula los candidatos en el tablero para una celda, denotada por su fila y su columna y devuelve verdad si la celda está ocupada y falso si la celda está vacía. Por parámetros le pasamos la dirección de inicio del tablero, el número de fila y el número de columna de nuestra celda.

- **sudokuARM_c.asm**:
    - Función sudoku_recalcular_arm_c(): Función en ARM que recorre todo el tablero del sudoku llamando a la función *sudoku_candidatos_c* para que calcule los candidatos de cada celda. Por último, cuenta y devuelve el número de celdas vacías del tablero.

- **sudokuARM_THUMB.asm**:
    - Función sudoku_recalcular_arm_thumb(): Función en ARM que recorre todo el tablero del sudoku llamando a la función *sudoku_candidatos_thumb* para que calcule los candidatos de cada celda. Por último, cuenta y devuelve el número de celdas vacías del tablero.

Cabe destacar que tras escribir todo el código y ver su correcto funcionamiento, procedimos a intentar optimizarlo. Para ello, redujimos los accesos a la pila de memoria y utilizamos más los registros del código de ARM.
Además, como primera versión realizábamos distintas instrucciones *mul*, las cuales sustituimos por operaciones de desplazamiento de bits (uso del barrel shifter) ya que vimos que eran múltiplos de dos.

Número de líneas de código de cada fichero

- sudoku_2015.c: 214.
- sudokuARM.asm: 175.
- sudokuTHUMB.asm: 177.
- sudokuARM_c.asm: 34.
- sudokuARM_THUMB.asm: 35.

Horas de dedicación

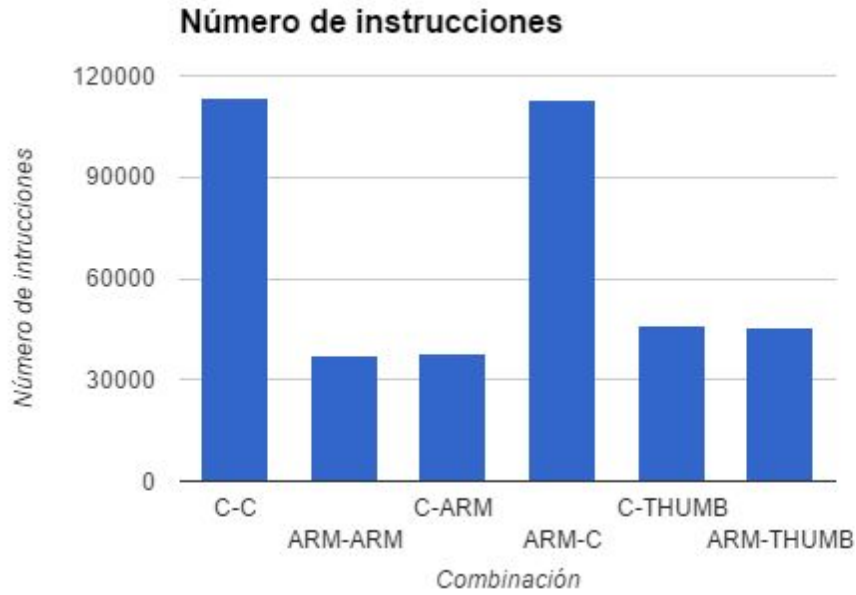
Las horas de dedicación han sido aproximadamente las siguientes:

- 12 horas entre sesiones de laboratorio (días que corresponden)
- 18 horas entre sesiones de laboratorio (días que no correspondían)
- 5 horas de trabajo en grupo fuera del horario de laboratorio
- 8 horas en total entre los dos componentes del grupo (trabajo individual)
- Total horas: 43 horas aproximadas

RESULTADOS

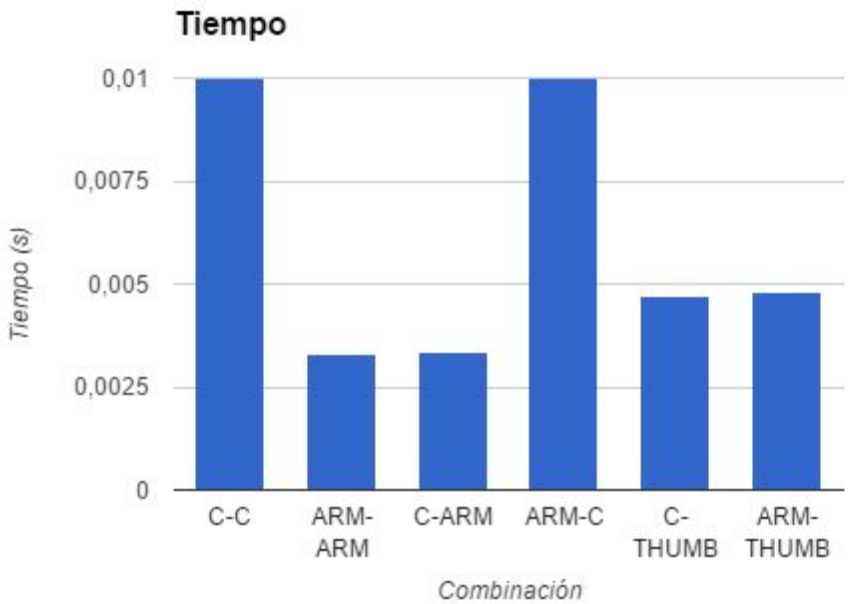
A continuación vamos a mostrar el número de instrucciones para cada ejecución combinada de lenguajes (C, ARM) y modo (THUMB) de las funciones recalcular() y candidatos():

- sudoku_recalcular_c: 113535 //C-C
- sudoku_recalcular_arm: 37244 //ARM-ARM
- sudoku_recalcular_c_arm: 37632 //C-ARM
- sudoku_recalcular_arm_c: 112823 //ARM-C
- sudoku_recalcular_c_thumb: 46200 //C-THUMB
- sudoku_recalcular_arm_thumb: 45812 //ARM-THUMB



Tiempo de ejecución para cada ejecución combinada de lenguajes (C, ARM) y modo (THUMB) de las funciones recalcular() y candidatos():

- sudoku_recalcular_c: 0,01s
- sudoku_recalcular_arm: 0,00329s
- sudoku_recalcular_c_arm: 0,00333s
- sudoku_recalcular_arm_c: 0,01s
- sudoku_recalcular_c_thumb: 0,00473s
- sudoku_recalcular_arm_thumb: 0,0048s



Análisis de los resultados

Dados estos resultados, se puede apreciar que la combinación C-C es la más costosa en número de instrucciones y tiempo, siguiéndole ARM-C.
Tras ellas se sitúan las dos combinaciones en las que interviene THUMB (C-Thumb, ARM-Thumb) y por último, las de ARM (ARM-ARM, C-ARM), siendo la más eficiente ARM-ARM.

CONCLUSIONES

Tras realizar el proyecto podemos concluir que nos ha sido más sencillo programar en C, dado que teníamos más experiencia con este lenguaje, ya que es un lenguaje de alto nivel. En ARM, hemos tenido alguna dificultad más, por ejemplo a la hora de movernos en memoria, o al optimizar e intentar acceder a la pila lo menos posible.
El caso que más nos ha costado ha sido en Thumb, dado que era la primera vez que trabajábamos con él y hemos tenido que tener en cuenta que las instrucciones son distintas, tenemos menos registros… Otro punto que nos trajo dificultad fué el salto al modo Thumb y el retorno del modo thumb a ARM.

La conclusión a la que hemos llegado con esta práctica es que el compilador de C a la hora de realizar su trabajo, introduce muchos accesos a memoria si no se le indica ninguna opción de optimización en la compilación, esto hace que el programa sea menos óptimo.

En ARM hemos intentado, dado que somos personas humanas y no máquinas, optimizar el código al máximo, realizando menos accesos a memoria y utilizando el menor número de instrucciones posibles.

En Thumb hemos comprobado que necesitamos más instrucciones y accesos a memoria para realizar el mismo trabajo, dado que solo existen 8 registros de propósito general, pero dado que las instrucciones de Thumb tienen la mitad de tamaño que las de ARM, hemos llegado a la conclusión de que aunque tengamos más instrucciones que en ARM, como estas son de tamaño menor, se puede puede reducir el tamaño del fichero ejecutable.