Proyecto Hardware

Práctica 1: Desarrollo de código ARM para el juego Conecta 4

Dorian Boleslaw Wozniak (817570@unizar.es)

Pablo Latre Villacampa (778043@unizar.es)

Zaragoza, a 24 de octubre de 2022

Índice

1. Resumen

Se ha desarrollado la implementación del juego «conecta 4» para un sistema ARM. Esta implementación consiste en un tablero representado en memoria y entradas asíncronas de movimientos a realizar mediante las herramientas de depuración del entorno de desarrollo utilizado.

A partir de un código de partida dado por el enunciado del problema, se ha analizado el problema y comprendido su funcionamiento, añadiendo también funcionalidades faltantes.

Se ha analizado el comportamiento del compilador. Se ha obtenido un mapa de memoria de la posición de las funciones. El código se encuentra al inicio del espacio en sus primeros 128KiB, mientras que los datos estáticos y pila se encuentran el un bloque de 32KiB tras la dirección 0x40000000, coincidiendo con lo indicado en la documentación. También se ha observado la construcción de marcos de activación por el compilador. Difiere en el uso de direcciones precalculadas al compilar en vez de almacenar SP y PC antiguos en la pila.

También se ha analizado las diferentes opciones de rendimiento ofrecidas por el compilador, mediante -On, con *n* de 0 a 3. Se puede apreciar la reducción de tiempo de ejecución a medida que aplica optimizaciones sobre código redundante y optimizaciones de pila. En opciones más altas de -On, y utilizando la opción -Otime, se aprecia métodos más avanzados como la incrustación automática y el desenrollado de bucles, aunque sea a costa del tamaño.

Se ha implementado una serie de funciones críticas, buscar_alineamiento y hay_linea, en ensamblador ARM y en varias combinaciones junto a código a C siguiendo estándares de llamada (ATPCS), y comparado respecto a las funciones en C optimizadas. También se ha tratado de optimizar dichas funciones para igualar o batir al compilador, eliminando recursividad y aprovechando oportunidades para optimizar el código usando propiedades del problema. Aunque cerca, no se ha logrado vencer a -O3, pero se han encontrado y descrito oportunidades adicionales para recortar los tiempos de ejecución.

Finalmente, se ha implementado un banco de pruebas automatizado para comprobar la validez y correcto funcionamiento de las funciones descritas en múltiples casos, así como para poder obtener las medidas anteriormente descritas siguiendo una partida predecible.

2. Introducción

Este proyecto forma parte de una serie de tres prácticas que forman parte de la asignatura de Proyecto Hardware. El objetivo de estas prácticas es introducir al desarrollo de software en sistemas empotrados, es decir, sistemas de uso específico y tamaño y consumo limitados.

En concreto, esta práctica sirve como introducción a la programación para sistemas ARM, una arquitectura RISC (con un conjunto de instrucciones reducidas) popular actualmente (por ejemplo, en dispositivos móviles). Para ello, se utiliza el entorno de desarrollo Keil µVision, que contiene un compilador C para ARM y un enlazador de binarios propio, un simulador y herramientas de depuración potentes.

El problema concreto a desarrollar es un juego: conecta 4. El juego consiste en un tablero donde dos jugadores introducen fichas de colores opuestos. Una ficha, al introducirla en una columna del tablero, cae hasta el fondo o hasta la primera ficha que se encuentre por el medio. Los jugadores se alternan turnos. Si un jugador consigue formar una línea recta de 4 fichas de su color, gana. Si el tablero se llena sin haber un ganador, hay empate.

La implementación de este juego se visualiza en memoria, y las entradas para jugar se realizan de forma asíncrona modificando manualmente una dirección de memoria a través del depurador. Además el tablero queda invertido para facilitar la programación. En prácticas siguientes se desarrollará una implementación para poder jugar a través de periféricos.

El objetivo de la práctica no es desarrollarlo de cero: se comienza con una versión del código parcialmente funcional del juego que se debe analizar. Tampoco se empieza con un desconocimiento total del funcionamiento del IDE: en AOC 1 ya se ha realizado programación en ensamblador, y en AOC 2 se ha explorado el funcionamiento interno de un procesador (en este caso MIPS, también una arquitectura RISC), junto a técnicas de optimización.

Junto a los objetivos descritos a continuación, la práctica sirve como refresco y profundización en los conocimientos obtenidos anteriormente, así como un ejercicio de aprendizaje para aprender a consultar y utilizar documentación oficial y externa.

3. Objetivos

Los objetivos principales de este proyecto son:

- Analizar el problema, la estructura del código, su funcionamiento y añadir funcionalidad faltante.
- Analizar el funcionamiento del compilador y observar las diferencias según las opciones suministradas y respecto al código fuente que podría escribir un programador humano y las oportunidades de optimización que suministra.
- Implementar en ensamblador de ARM una serie de funciones que involucran una parte del funcionamiento crítico del juego en ensamblador: comprobar si hay 4 en línea. Además, se desarrollarán diferentes versiones que demuestran la interoperabilidad entre ARM y lenguajes de programación de alto nivel como C siguiendo estándares definidos para ello como el ATPCS.
- Implementar un sistema de pruebas para facilitar la depuración y verificación del correcto funcionamiento del código, así como permitir medir el rendimiento del sistema en un caso de uso típico como es una partida normal entre dos personas.
- Analizar el rendimiento del programa en C según las opciones de optimización definidas al compilar, así como comparar los resultados con las implementaciones anteriormente descritas del juego.
- A partir de estos resultados, analizar en profundidad el problema y aplicar técnicas de optimización conocidas anteriormente para optimizar el código en ensamblador con el objetivo de vencer en rendimiento las opciones más agresivas de un compilador.

4. Metodología

4.1. Análisis inicial del problema

La primera aproximación al problema, tras la lectura del guión adjunto, fue primero identificar los ficheros que forman el proyecto y las funciones que ofrecen, para conocer en primer lugar la estructura del proyecto y cómo funciona; y segundo, probar el programa paso a paso usando la herramienta de depuración y verificando el funcionamiento del programa con el tablero inicial proporcionado.

4.1.1. Análisis del código

En la parte de análisis del código fuente, se identificaron:

- **entrada.h/c**: Implementa la forma de introducir movimientos en el juego. En la dirección 0x40000000, se define un vector de 8 bytes (para no desalinear la cuadrícula, más adelante). En dicho vector:
    - entrada[0]: Un valor distinto de 0 indica que debe realizar una jugada
    - entrada[1]: Indica la columna a colocar la ficha
- **celda.h**: Define un TAD «CELDA» del juego donde se colocan las fichas, y las operaciones para obtener su valor y manipularlas. Representada por 1 byte, la información relevante se codifica en los 3 bytes menos significativos. Por orden de importancia:
    - $CELDA_2$: 1 indica posición ocupada
    - $CELDA_1$: 1 indica ficha negra presente
    - $CELDA_0$: 1 indica ficha blanca presente
- **tableros.h**: Contiene los tableros a utilizar. El tablero se define inmediatamente después del vector de entradas. Contiene etiquetas para cada fila y columna tal que sea más fácil de visualizar en memoria. Cabe destacar que incluye un tablero a medio empezar, con la siguiente posición:

	C1	C2	C3	C4	C5	C6	C7
F1	●	○	●	○	●	●	
F2	○	●	○	○		○	
F3	●			●		○	
F4	○			●		○	
F5	●			○			
F6				●			

Fig. 1: Cuadrícula inicial dada con el enunciado

- **conecta4_2022.h**: Contiene las definiciones de las funciones contenidas por conecta4.c, y además definiciones de constantes mediante *enum*, las más importantes las del tamaño del tablero. También implementa una serie de funciones para comprobar la validez de una entrada y para alternar el color. Estas funciones contienen la palabra clave *inline*, cuya función se verá más adelante.
- **conecta4_2022.c**: La implementación principal del juego.

conecta4.c contiene la mayor parte de la implementación del juego. Las funciones más importantes son:

- C4_calcular_fila: A partir de una columna, obtiene la fila donde colocar una pieza. Devuelve la fila, o error si la columna está fuera de la cuadrícula o la fila desborda esta.
- C4_actualizar_tablero: Llama a la función celda_poner_valor en la fila y columna indicadas, y con el color indicado. El tablero ahora registrará el movimiento.
- C4_comprobar_empate: Comprueba si se ha llenado la tabla con fichas sin haber un ganador. Esta función no se encuentra aún implementada, y se comentará más adelante.
- conecta4_buscar_alineamiento: Función recursiva que, dada la cuadrícula, la fila y columna actual y el color de la ficha, determina la longitud de la línea de fichas del

mismo color más larga, en dirección indicada por los parámetros delta_fila y delta_columna. En caso de encontrarse la ficha dentro del tablero y ser del color indicado, se llama recursivamente sumándole a fila y columna el valor de sus respectivas deltas.

- conecta4_hay_linea: Dada una cuadrícula y la fila y columna a comprobar, explora si hay 4 en línea o más para el color indicado. Si hay 4 fichas del mismo color o más en una línea recta o no, devuelve éxito y fracaso respectivamente. Para recorrer el tablero, utiliza una pareja de vectores de 4 bytes, deltas_fila y deltas_columna, que indican la dirección a explorar. Para cada iteración, se obtiene la suma de las llamadas a buscar_alineamiento con la posición dada y deltas[i], y con -deltas[i] y la posición dada menos las deltas. Las direcciones exploradas son:
    - Izquierda - derecha
    - Abajo-arriba
    - Diagonal izquierda-arriba a derecha-abajo
    - Diagonal a izquierda-abajo a derecha-arriba
- conecta4_verificar_4_en_linea: Llama a hay_linea. Se usa más adelante para seleccionar las versiones de esta función y buscar_alineamiento a usar.
- conecta4_jugar: El programa principal. Tras incluir el vector de entrada y la cuadrícula, y definir variables, realiza el siguiente búcle:
    - Espera a que se introduzca una jugada. Esto se realiza de forma asíncrona.
    - Obtiene la columna de la entrada y calcula la fila donde colocar la pieza
    - Si la posición es correcta, introduce la ficha en el tablero. En caso contrario, limpia la entrada y obliga a reintroducir un movimiento al jugador del mismo color.
    - Verifica si hay 4 en línea. En caso afirmativo, se queda en un bucle (al no haber SO no hay gestión de la salida del programa, y por ende no para). En caso contrario, comprueba si hay empate, con comportamiento similar en caso afirmativo.
    - Si ninguno de los dos supuestos se cumple, cambia de color, vacía el vector de entrada y pasa turno al rival.

4.1.2. Prueba inicial de funcionamiento

A continuación, se procedió a compilar y ejecutar el programa dado utilizando el depurador de Keil. El aspecto más importante a destacar es el uso de la ventana de memoria para visualizar el tablero y la entrada. Como se mencionó anteriormente, ambas se pueden encontrar a partir de la dirección 0x40000000.

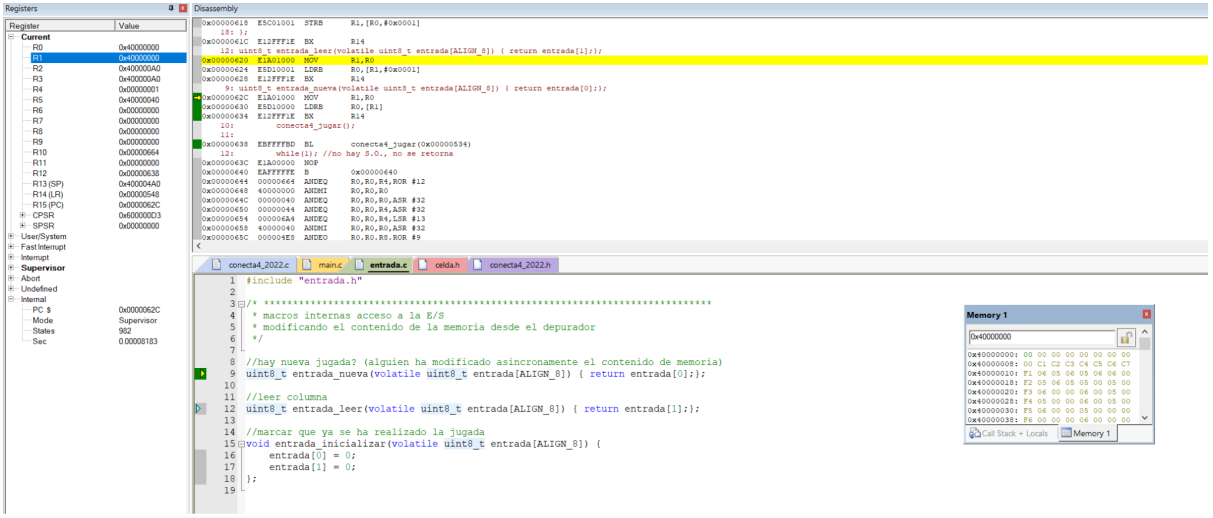

Fig. 2: Entorno de depuración de Keil µVision

Ejecutando paso a paso, se exploró el funcionamiento interno del programa y las funciones. Utilizado *breakpoints* en puntos clave como *while (entrada_nueva(entrada) == 0)* y los bucles infinitos en las pruebas de victoria y empate, y el tablero a medio empezar dado, se probó cómo funcionaba el juego. Se determinó que, salvo la falta de implementación de C4_comprobar_empate, el programa funcionaba como se esperaba.

4.1.3. Implementación de C4_comprobar_empate

Como primera tarea, se implementó el código de C4_comprobar_empate, ausente hasta ahora. Lo más importante a destacar es que sólo revisa la última fila, y que si detecta una celda vacía, devuelve falso automáticamente.

La implementación está incluida en el Anexo 1.1

4.1.4. Mapa de memoria del programa

Para obtener el mapa de la memoria del programa, se ha utilizado el fichero conecta4_2022.map, un fichero generado al compilar y enlazar que contiene, entre otros, las referencias de un objeto a otro, secciones eliminadas del ejecutable final, *veneers* añadidas (que permiten la interoperabilidad entre los modos normal y *Thumb* del procesador[1]), una tabla de símbolos, y lo que concierne a este apartado, el mapa de memoria del programa. A partir de las direcciones de ejecución dadas, se ha obtenido una representación visual de dicho mapa.

4.1.5. Marco de activación de las funciones

Para obtener el marco de activación de las funciones, se ha analizado los momentos anteriores a producirse un salto, comprobando qué variables se guardan y los registros utilizados como parámetros durante el salto, y posteriormente el regreso a la función anterior mediante el desensamblador.

En los Resultados se incluyen los marcos de activación de hay_linea y buscar_alineamiento.

4.2. Análisis inicial del rendimiento del programa

Para analizar el rendimiento del programa, se ha utilizado una prueba automatizada para simular una partida normal entre dos jugadores. Los valores de tiempo de cada función corresponden a un punto en mitad de la partida, representativo de la partida. También se ha analizado el tiempo total que toma la partida hasta que se obtiene una victoria. Los detalles de implementación y tablero utilizado se comentarán más adelante.

Por otro lado, para obtener el rendimiento, se ha utilizado el mismo fichero *map* utilizado anteriormente para obtener el mapa de memoria, el cual también contiene el lugar y tamaño de cada instrucción. Se ha analizado el tamaño de las funciones principales, hay_linea y buscar_alineamiento.

Por último, se han medido los tiempos y tamaño con diferentes opciones de compilación: -O0, -O1, -O2, -O3 y -O3 -Otime. Las diferencias entre todas estas opciones son las siguientes[2][3]:

- **-O0: Optimización mínima**, diseñada para crear una correspondencia directa entre el código en alto nivel y el ensamblador. También incluye «código muerto», es decir, código redundante pero necesario para mantener dicha correspondencia.
- **-O1: Optimización restringida**, aplica mejoras sin disminuir gravemente la capacidad de depuración del programa. Habilita incrustación del código (normalmente del que se especifica con *inline*) y *tail calls* (reutilización de marcos de activación para llamadas al final de una subrutina), realizando llamadas fuera de orden si no tienen efectos laterales. También elimina código muerto. Puede afectar a la visibilidad de variables y trazas de llamadas.
- **-O2: Optimización alta**, cambia las heurísticas del compilador, permite uso de instrucciones vectoriales, y mejor planificación del código.

- **-O3: Optimización máxima**, similar a -O2 pero incluso más agresivo, a costa de la legibilidad del desensamblado. Destacan la incrustación de código automática y el desenrollado de bucles.
- -**Otime** indica al compilador que optimice para mejorar tiempo, a costa de memoria utilizada.

4.3. Implementación inicial de funciones en ARM

La implementación de las funciones de ARM, en un principio, se ha realizado traduciendo directamente las estructuras de hay_linea y buscar_alineamiento en ensamblador.

Para las llamadas a las subrutinas, se ha seguido estrictamente el estándar ATPCS:

- En registros
    - r0 a r3 reservado para parámetros, r0 siendo además valor de salida (si hay)
    - r12 utilizado para guardar SP antiguo en pila
- En pila, por orden
    - Parámetros que no cabían en registros
    - PC
    - LR
    - SP (almacenado en r12)
    - FP
    - Registros variables (r4-r10)
    - Variables locales que no cabrán en registro

Las constantes necesarias se añadieron mediante directivas EQU, de forma equivalente a los *enum* de conecta4_2022.h. Además, para hay_linea, se ha reservado dos secciones de 4 bytes cada una para los vectores de deltas (que en el código de C se encuentran seguidas de la función hay_linea).

La implementación de las funciones se encuentran en los anexos:

- buscar_alineamiento_arm: Anexo 1.2
- hay_linea_arm_c: Anexo 1.3
- hay_linea_c_arm: Anexo 1.4
- hay_linea_arm_arm: Anexo 1.5

También se ha comparado las métricas de tiempo y tamaño con el código de C. Estos se pueden encontrar en su sección de Resultados.

4.4. Implementación del sistema de verificación automática

Para poder agilizar la corrección del código escrito y la prueba del rendimiento del programa, se ha implementado un nuevo fichero de cabecera: jugadas.h.

Dicho fichero contiene una serie de vectores que contienen movimientos a introducir de forma automática. Estos vectores quedan inicializados detrás de la cuadrícula, en memoria estática. Para seleccionar el vector de movimientos a utilizar, se ha utilizado estructuras #ifdef, que compilan el vector definido por un símbolo #define.

Para complementar estos tests, se ha modificado tableros.h para definir tableros para los tests de forma parecida.

Además de una sección de código que se compila condicionalmente si se incluye jugadas.h en conecta4_2022.c que introduce la jugada en entrada[] automaticamente, algunas pruebas incluyen un símbolo UNDO_IF_WIN. Si es así, en vez de detenerse al obtener 4 en línea, se invoca una función celda_borrar_valor en celda.h que limpia la ficha introducida antes de pasar turno, permitiendo comprobar múltiples casos de victoria en una misma prueba.

Si se acaban las jugadas a realizar, el sistema devuelve el control y espera a una jugada manual. Se considera éxito de los casos de prueba (menos el de rendimiento) si se le devuelve el control al usuario en *while(entrada_nueva(entrada) == 0)*.

Los tests disponibles se encuentran en el Anexo 2. Estos son:

- **TEST_1:** Comprueba que no se malinterpreten las celdas vacías, posiciones fuera de la cuadrícula y celdas de su propio color. Esto se hace introduciendo bloques 3x3 de fichas blancas y fichas negras siguiendo varios patrones para llenarlos. Se utiliza el tablero **TABLA_TEST_1**.
- TEST_2: Comprueba que no malinterprete celdas de color contrario. Introduce en un «hoyo» de fichas un color una del color contrario. Se utiliza el tablero **TABLA_TEST_2**.
- TEST_3: Comprueba los casos de victoria al colocar una ficha en los extremos de cada diagonal. Para ello, se define UNDO_IF_WIN para deshacer los movimientos en caso de victoria. Se utiliza con el tablero **TABLA_TEST_3**.
- TEST_4: Comprueba los casos de victoria al colocar una ficha en medio de una diagonal. Se utiliza con el tablero **TABLA_TEST_4**.
- TEST_5: Comprueba los casos de victoria al colocar una ficha al colocarla en una línea horizontal y vertical. Se utiliza con el tablero **TABLA_TEST_5**.
- TEST_PERF: Una partida normal, con la que se evalúa el rendimiento del programa.

Es una partida larga, que no se resuelve hasta casi llenar el tablero. Esta partida se ha generado automaticamente en la página http://connect4.ist.tugraz.at/, con ambos colores marcados como jugador IA y con el nivel «Just Win». Se utiliza con el tablero **TABLA_VACIA.**

4.5. Optimizaciones a las funciones ARM

Para mejorar el rendimiento del programa en comparación al código obtenido por el compilador de C en -O3 -Otime, se ha aplicado una serie de optimizaciones sobre las funciones ya implementadas en ARM, una vez verificado su correcto funcionamiento. Se han aplicado una serie de 4 optimizaciones, cuyo código se puede encontrar en el Anexo 1:

- **Optimización 1**: Consiste en la modificación de buscar_alineamiento_arm en una función iterativa en vez de una función recursiva. En vez de realizar una llamada recursiva, suma en r0 el número de celdas del mismo color, y al encontrarse con una celda no del mismo color regresa a la función hay_linea.
- **Optimización 2**: Mejora ligeramente el uso de los registros en buscar_alineamiento, evitando movimientos redundantes al no llamar a ninguna función, y por tanto, no teniendo efectos de los que preocuparse por utilizar r1 a r3 como registros variables.
- **Optimización 3**: Introduce una optimización en hay_linea donde se asume que la ficha en la posición dada ya es del mismo color, por lo que al realizar la primera llamada a buscar_alineamiento se suman directamente a los índices las deltas obtenidas. Eso implica una iteración de buscar_alineamiento menos.
- **Optimización 4**: Mejora ligera en hay_alineamiento el uso del espacio reservado para las deltas: ahora se encuentran todas en el mismo espacio, entrelazando filas y columnas. Ahora, en vez de cargar la dirección de cada delta en memoria, se mantiene durante toda la subrutina un puntero a dicho espacio, minimizando el número de veces que se carga en un registro la dirección de donde cargar las deltas a una vez.

5. Resultados

5.1: Mapa de memoria

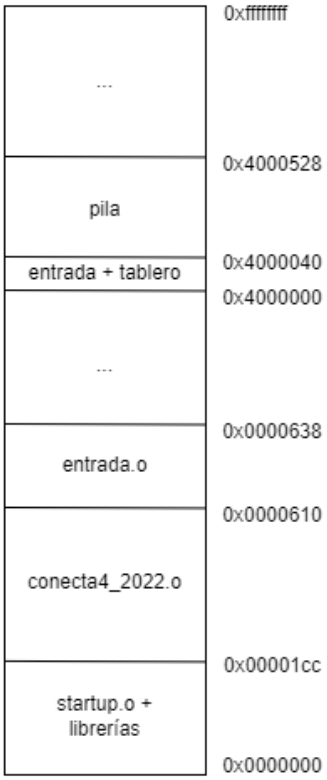

Fig. 3: Mapa de memoria del programa

Comparando con el mapa de memoria general del LPC2105, el código de inicialización del programa y variables se encuentran al inicio de los primeros 128KiB de la memoria del sistema, mientras que los datos asignados como *static* y la pila se encuentran en el bloque de 32KiB de memoria estática localizados en la dirección 0x40000000

5.2: Marco de activación

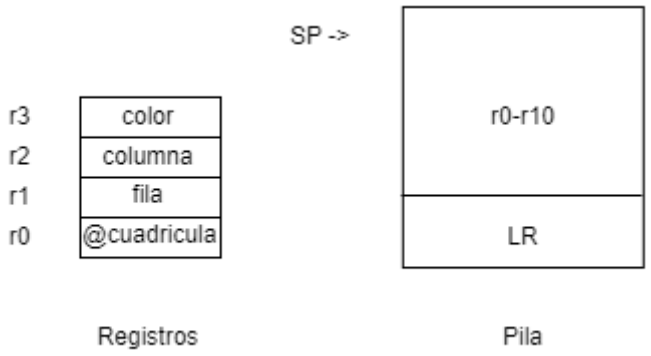

Fig. 4: Marco de activación de conecta4_hay_linea

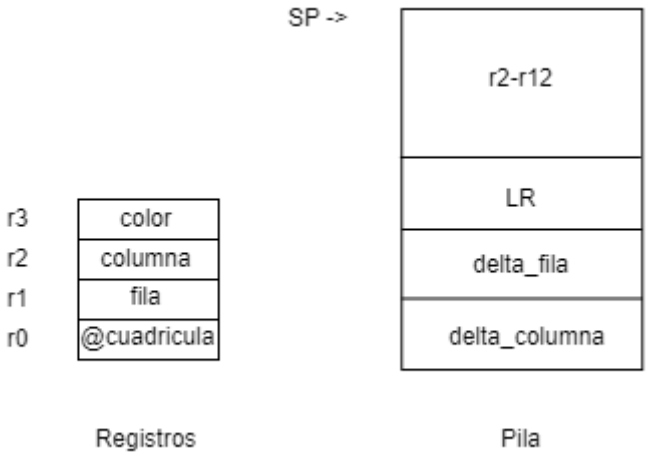

Fig. 5: Marco de activación de conecta4_buscar_alineamiento

Lo más importante a destacar del bloque de activación generado por el compilador es el no uso de esta para almacenar ni el PC ni el SP ni el FP. El primero no se guarda debido a que el compilador precalcula los saltos a realizar, actuando directamente sobre el PC. También precalcula el valor del SP al regresar de una función pues sabe de antemano el tamaño del bloque generado. Por la misma razón no trata el FP de forma especial: obtiene parámetros de la pila utilizando el SP actual como base, y r11 lo usa como registro variable.

5.3: Métricas de rendimiento del programa con distintos niveles de optimización

Tiempo / llamada	-O0	-O1	-O2	-O3	-O3 -Otime
C4_calcular_fila()	3,583 µs	2,25 µs	1,5 µs	1,5 µs	—
C4_fila_valida()	0,75 µs	—	—	—	—
C4_columna_valida()	0,75 µs	—	—	—	—
C4_actualizar_tablero()	1,917 µs	1 µs	0,833 µs	0,833 µs	—
C4_comprobar_empate()	2,75 µs	1,083 µs	0,917 µs	0,833 µs	—
conecta4_hay_linea_c_c()	31,583 µs	26 µs	24,917 µs	23,167 µs	36 µs
conecta4_buscar_alineamiento_c()	5,667 µs	4,667 µs	3,333 µs	3,333 µs	2,917 µs
conecta4_jugar()	0,250 µs	9 µs	7,667 µs	7,667 µs	8,083 µs
celda_vacia()	0,5 µs	—	—	—	—
entrada_nueva()	0,583 µs	0,5 µs	0,5 µs	0,5 µs	0,5 µs
entrada_leer()	0,583 µs	0,5 µs	0,5 µs	0,5 µs	0,5 µs
entrada_inicializar()	0,667 µs	0,667 µs	0,667 µs	0,667 µs	0,667 µs

Fig. 6: Tabla con los tiempos por función para distintos niveles de optimización. La falta de valores significa que la función ha quedado incrustada

Num. llamadas / función	-O0	-O1	-O2	-O3	-O3 -Otime
C4_calcular_fila()	41	41	41	41	—
C4_fila_valida()	596	—	—	—	—
C4_columna_valida()	550	—	—	—	—
C4_actualizar_tablero()	41	41	41	41	—
C4_comprobar_empate()	40	40	40	40	—

conecta4_hay_linea_c_c()	41	41	41	41	41
conecta4_buscar_alineamiento_c()	555	555	555	555	230
conecta4_jugar()	1	1	1	1	1
celda_vacia()	702	—	—	—	—
entrada_nueva()	41	41	41	41	41
entrada_leer()	41	41	41	41	41
entrada_inicializar()	40	40	40	40	40
Tiempo total	7,162 ms	4,696 ms	3,711 ms	3,621 ms	2,823 ms

Fig. 7: Tabla con el número de iteraciones por función y tiempo total de ejecución para diferentes niveles de optimización

En los resultados se puede apreciar la progresiva mejora de los resultados a medida que se aumenta el nivel de optimización del tiempo y del uso del código. Mientras que en -O0 no se incrusta el código y tarda hasta 7 ms, a medida que se aumenta el nivel de optimización se aplican dichas incrustaciones y se reduce el tiempo que toma cada función en ejecutarse. Con -Otime se aprecian las mayores diferencias, con gran parte del código eliminado como subrutinas separadas, y con reducciones en llamadas.

Tamaño (B)	-O0	-O1	-O2	-O3	-O3 -Otime
buscar_alineamento	172	172	112	112	124
hay_linea	320	268	244	232	432

Fig. 8: Tamaño de las funciones a modificar para distintos niveles de optimización

En cuanto al tamaño, se puede apreciar las mejoras al tamaño del código conforme se reduce el número de instrucciones utilizadas. Esto se revierte al utilizar -Otime: en hay_linea, el bucle queda completamente desenrollado, lo cual aumenta el código sustancialmente.

5.4: Métricas de rendimiento de funciones ARM

Todas las métricas se han obtenido utilizando -O3 -Otime

	hay_linea (t)	hay_linea (llamadas)	buscar_alineamiento (t)	buscar_alineamiento (llamadas)	Tiempo total
C_C	36 µs	41	2,917 µs	230	2,823 ms
C_ARM	22.5 µs	inline (41)	4,333 µs	555	4,266 ms
ARM_C	21 µs	41	3,083 µs	555	3,330 ms
ARM_ARM	21 µs	41	4,333 µs	555	4,201 ms

Fig. 9: Tiempo de ejecución de las funciones a analizar según la implementación

Se debe destacar que las funciones implementadas en ARM no están optimizadas. Se observa que la función buscar_alineamiento es la de mayor peso al obtener el tiempo de ejecución: no solo es más lenta la implementación en ARM sino que se llama más veces. hay_linea, por otro lado, si que vence en ejecución a la versión de C, pero no se beneficia del desenrollado de bucles. También cabe destacar que la versión C_ARM incrusta la función hay_linea. El compilador tiene un criterio para decidir si incrustar o no una función, al parecer en este caso supera dicho criterio y lo considera beneficioso.

Tamaño (B)	C_C	C_ARM	ARM_C	ARM_ARM
buscar_alineamento	124	136	124	136
hay_linea	432	244	184	184

Fig. 10: Tamaño de las funciones a analizar según la implementación

En cuanto al tamaño de cada función, se aprecia que la implementación en C de buscar_alineamiento utiliza menos espacio que la de ARM, y que hay_linea en ARM es mucho más pequeña que la de C por las mismas razones expuestas

5.5: Métricas de rendimiento de funciones ARM optimizadas

	hay_linea (t)	hay_linea (llamadas)	buscar_alineamiento (t)	buscar_alineamiento (llamadas)	Tiempo total
C_C	36 µs	41	2,917 µs	230	2,823 ms
ARM_ARM	21 µs	41	4,333 µs	555	4,201 ms
OPT_1	21 µs	41	6,583 µs	325	3,337 ms
OPT_2	21 µs	41	6,167 µs	325	3,25 ms
OPT_3	21,25 µs	41	4,25 µs	325	2,954 ms
OPT_4	19,5 µs	41	4,25 µs	325	2,882 ms

Fig. 11: Tiempos de ejecución de las funciones a analizar según las optimizaciones aplicadas

La optimización más importante es la primera: al evitar llamadas, las cuales son costosas al tener que crear un marco de activación nuevo cada vez que llaman, y utilizar la instrucción *bl*, teniendo cada salto un coste significativo. La otra optimización importante, la tercera, se logra reducir el tiempo de ejecución de buscar_alineamiento al utilizar el conocimiento de que la ficha en la posición dada ya es del color deseado, por lo que se puede empezar visitando directamente celdas vecinas.

Tamaño (B)	ARM_ARM	OPT_1	OPT_2	OPT_3	OPT_4
buscar_alineamento	136	124	108	108	108
hay_linea	184	184	184	176	168

Fig. 12: Tamaños de las funciones a analizar según las optimizaciones aplicadas

El peso, conforme se optimiza, se reduce conforme se usan menos instrucciones (que ocupan 32 bits cada una).

6. Conclusiones

Las conclusiones obtenidas del desarrollo del proyecto son las siguientes:

Sobre el compilador/enlazador:

- El compilador es capaz de precalcular las direcciones del PC y SP durante la compilación, evitando usar registros como el FP para obtener valores en pila, pudiendo utilizar r11 como registro variable; y el IP, pues en vez de almacenar en él un SP antiguo simplemente calcula un desplazamiento fijo según el tamaño del marco de activación.
- El compilador utiliza una serie de técnicas y heurísticas para obtener código optimizado. Inicialmente se utilizan básicas como detectar instrucciones redundantes, optimizar el uso de la pila reutilizando marcos de activación, reordenando las llamadas, no creando marcos para llamadas básicas y la incrustación de código a criterio del programador. En las opciones más avanzadas utiliza principalmente la incrustación automática y el desenrollado de bucles para optimizar el código, con un criterio menos estricto conforme se va aumentando la agresividad y se le indica que priorice tiempo sobre tamaño.

Sobre la programación en ensamblador y optimización:

- Aunque no se ha logrado vencer a -O3 -Otime, el resultado es cercano a igualarlo.
- Por un lado, se considera crucial reducir el número de llamadas a buscar_alineamiento, el principal causante de reducción de rendimiento. La solución original no solo llama 8 veces a dicha función desde hay_linea, sino que la solución recursiva hace una llamada para cada caso complejo (es del mismo color). La eliminación de la recursividad reduce sustancialmente el tiempo de ejecución.
- Otra mejora para reducir el número de llamadas se encuentra en analizar el propio problema. En el caso de comprobar la línea vertical tiene una llamada sobrante: revisar en dirección contraria a la base del tablero es innecesario. Además, para solucionar este problema, se podría aplicar también prácticas del compilador como el desenrollado de bucles.
- Aunque ya se aplican bastantes optimizaciones menores como instrucciones condicionales y desplazamientos, aún existe márgen de mejora para mejorar estos aspectos y otros como la reducción de cargas de memoria (especialmente con los vectores de deltas).

- En conclusión, se considera que aún existen oportunidades de optimización en el propio problema, como la anteriormente descrita llamada a buscar_alineamiento sobrante. Un análisis más profundo podría revelar soluciones que probablemente ayudarían a vencer a -O3.

Anexo 1: Códigos fuente

Anexo 1.1: C4_comprobar_empate

```
// Devuelve verdad sólo si el tablero está lleno sin que ningún jugador
// haya ganado
int C4_comprobar_empate(CELDA cuadricula[TAM_FILS][TAM_COLS]) {

   for (uint8_t i = 1; i <= NUM_COLUMNAS; i++)
       if (celda_vacia(cuadricula[NUM_FILAS][i]))
           return FALSE;
   return TRUE;
}
```

Anexo 1.2: conecta4_buscar_alineamiento_arm

AREA datos, DATA

```
        ; Constantes
NUM_FILAS       EQU 6
NUM_COLUMNAS    EQU 7
PADDING_FIL     EQU 1
PADDING_COL     EQU 1
TAM_FILS        EQU NUM_FILAS + PADDING_FIL
TAM_COLS        EQU NUM_COLUMNAS + PADDING_COL

CELDA_VACIA     EQU 0x04
CELDA_COLOR     EQU 0x03

FALSE           EQU 0
TRUE            EQU 1

                AREA codigo, CODE

                EXPORT conecta4_buscar_alineamiento_arm

;   Entrada:
;       r0 -> @cuadricula
;       r1 -> fila
;       r2 -> columna
;       r3 -> color
;       (en pila) @SP + 4 -> *deltas_fila[i]
;       (en pila) @SP + 8 -> *deltas_columna[i]
;
;   Salida:
;       r0 <- longuitud linea
;
;   Descripción:
;       Devuelve el número de celdas del mismo color consecutivas en
;       la línea recta dada por delta_fila y delta_columna a partir de
;       cuadricula[fila][columna]
conecta4_buscar_alineamiento_arm
                ; Prologo
                mov     r12, r13
                stmdb   r13!, { r4 - r10, r11, r12, r14, r15 }
                sub     r11, r12, #4

                ;   Bloque de activación:
                ;
                ;   SP  -> | r4 - r10          | buscar_alineamiento
                ;          | FP anterior    |
                ;          | SP anterior       |
                ;          | LR  anterior       |
                ;   FP   -> | PC anterior    |
                ;          | ----------------- |
                ;          | delta_fila[i]     | hay_linea
                ;          | delta_columna[i]  |

                ; Mueve parametros a registros variables
                mov     r4, r0          ; r4 = @cuadricula
                mov     r5, r1          ; r5 = fila
```

```
                mov     r6, r2          ; r6 = columna
                mov     r7, r3          ; r7 = color
                add     r9, r11, #0x04  ; Calcula @ base de parametros
                ldmia   r9, { r8 - r9 } ; r8, r9 = delta_columna, delta_fila

                mov     r0, #0          ; Devuelve 0 si no supera los condicionales

                ; if
                ;   !C4_fila_valida(fila) (! 1 <= fila <= NUM_FILAS)
                cmp     r5, #1
                blt     ba_return_zero
                cmp     r5, #NUM_FILAS
                bgt     ba_return_zero

                ;   !C4_columna_valida(columna)) (! 1 <= columna <= NUM_COLUMNAS)
                cmp     r6, #1
                blt     ba_return_zero
                cmp     r6, #NUM_COLUMNAS
                bgt     ba_return_zero

                ;   Obtiene valor de celda en cuadricula[fila][columna]
                add     r10, r4, r5, lsl #3     ; r10 = @cuadricula[fila][0] (@cuad + 8 * filas)
                ldrb    r10, [r10, r6]          ; r10 = *cuadricula[fila][columna]

                ;   celda_vacia(cuadricula[fila][columna]) ( celda & 0x04 == 0)
                and     r3, r10, #CELDA_VACIA
                cmp     r3, #0
                beq     ba_return_zero

                ;   celda_color(cuadricula[fila][columna] != ) ( celda & 0x03 != color)
                and     r3, r10, #CELDA_COLOR
                cmp     r3, r7
                bne     ba_return_zero

                ; Prepara llamada recursiva
                mov     r0, r4
                add     r1, r5, r8  ; Avanza índices
                add     r2, r6, r9
                mov     r3, r7
                stmdb   r13!, { r8 - r9 }

                bl conecta4_buscar_alineamiento_arm

                ; Obtiene resultado y regresa
                add     r0, r0, #1

ba_return_zero  ldmdb   r11, { r4 - r10, r11, r13, r15 }

                                END
```

Anexo 1.3: conecta4_hay_linea_arm_c

```
                AREA datos, DATA

NUM_FILAS       EQU 6
NUM_COLUMNAS    EQU 7
PADDING_FIL     EQU 1
PADDING_COL     EQU 1
TAM_FILS        EQU NUM_FILAS + PADDING_FIL
TAM_COLS        EQU NUM_COLUMNAS + PADDING_COL

CELDA_VACIA     EQU 0x04
CELDA_COLOR     EQU 0x03

FALSE           EQU 0
TRUE            EQU 1

deltas_fila     DCB 0, -1, -1, 1
deltas_columna  DCB -1, 0, -1, -1

                AREA codigo, CODE

                IMPORT conecta4_buscar_alineamiento_c
                PRESERVE8 {TRUE}

                EXPORT conecta4_hay_linea_arm_c

;   Entrada:
;       r0 -> @cuadricula
;       r1 -> fila
;       r2 -> columna
;       r3 -> color
;
;   Salida
;       r0 <- >= 1 si hay linea, 0 si no hay linea
;
;   Descripción:
;               Devuelve verdad si ha encontrado una línea de celdas del mismo color al
;               dado de 4 celdas o más. En caso contrario, devueve falso.
conecta4_hay_linea_arm_c
                ; Prologo
                mov     r12, r13
                stmdb   r13!, { r4 - r10, r11, r12, r14, r15 }
                sub     r11, r12, #4

                ;   Bloque de activación:
                ;
                ;   SP  -> | r4 - r10          | hay_linea
                ;          | FP anterior    |
                ;          | SP anterior       |
                ;          | LR  anterior       |
                ;   FP   -> | PC anterior    |
                ;          | ----------------- |
                ;          |                       | C4_verificar_4_en_linea

                ; Mueve parametros a registros variables
                mov     r4, r0          ; r4 = @cuadricula
                mov     r5, r1          ; r5 = fila
                mov     r6, r2          ; r6 = columna
                mov     r7, r3          ; r7 = color
```

```
                ; Inicializa variables
                mov     r8, #0          ; i
                mov     r9, #0          ; linea
                mov     r10, #0         ; long_linea

                ; for()
                ;   i < 4
hl_for          cmp     r8, #4
                bge     hl_return
                ;   linea == FALSE
                cmp     r9, #FALSE
                bne     hl_return

                ;   Obtiene deltas[i]

                LDR     r0, =deltas_fila
                ldrsb   r0, [r0, r8]
                LDR     r1, =deltas_columna
                ldrsb   r1, [r1, r8]

                ;   Prepara primera llamada a buscar_alineamiento

                stmdb   r13!, {r0, r1}
                mov     r0, r4
                mov     r1, r5
                mov     r2, r6
                mov     r3, r7

                bl      conecta4_buscar_alineamiento_c

                ;   linea = long_linea >= 4
                mov     r10, r0
                cmp     r10, #4
                movge   r9, #TRUE
                bge     hl_for

                ;   Prepara segunda llamada a buscar_alineamiento

                ;   Recupera y niega deltas
                ldmia   r13!, { r0, r3 }
                rsb     r0, r0, #0
                rsb     r3, r3, #0

                ;   fila - deltas_filas[i], columnas - deltas_columna[i]
                add     r1, r5, r0
                add     r2, r6, r3

                ;   Apila -deltas
                stmdb   r13!, { r0, r3 }

                ; cuadricula y color
                mov     r0, r4
                mov     r3, r7

                bl      conecta4_buscar_alineamiento_c

                ;   linea = long_linea >= 4
                add     r10, r10, r0
                cmp     r10, #4
                movge   r9, #TRUE

                ;   i++
                add     r8, r8, #1
```

```
                b       hl_for

hl_return       mov     r0, r9
                ldmdb   r11, { r4 - r10, r11, r13, r15 }

                                END
```

Anexo 1.4: conecta4_hay_linea_c_arm

```
/*
        Comprueba que con la ficha introducida hay una línea de 4 fichas del mismo
        color o más.

        Devuelve TRUE solo si ha encontrado dicha línea,
        en caso contrario devuelve FALSE

        Se definen dos versiones de la función en este fichero:

        - conecta4_hay_linea_c_c: Versíon en C con llamadas a versión de
                                                                buscar_alineamiento en C
        - conecta4_hay_linea_c_arm: Versíon en C con llamadas a versión de
                                                                buscar_alineamiento en ensamblador

        Se comprueba línea en las siguientes líneas, en este orden:

                - Izquierda + derecha
                - Arriba + abajo
                - Diag. sup. izda. + Diag. inf. dcha.
                - Diag. inf. izda. + Diag. sup. dcha
*/

// C-ARM
uint8_t conecta4_hay_linea_c_arm(CELDA cuadricula[TAM_FILS][TAM_COLS],
                                                                uint8_t fila, uint8_t columna,
uint8_t color) {

        // Deltas
        int8_t deltas_fila[4] = {0, -1, -1, 1};
        int8_t deltas_columna[4] = {-1, 0, -1, -1};

        // Variables
        unsigned int i = 0;                // Índice
        uint8_t linea = FALSE;   // Se ha encontrado línea
        uint8_t long_linea = 0;  // Tamaño de línea

        for (i = 0; (i < 4) && (linea == FALSE); ++i) {
                // Comprueba en dirección de las deltas
                long_linea = conecta4_buscar_alineamiento_arm
                                                (cuadricula,
                                                fila, columna, color,
                                                deltas_fila[i], deltas_columna[i]);

                linea = long_linea >= 4;
                if (linea) continue;

                // Comprueba en dirección contraria, suma a la línea encontrada antes
                long_linea +=
                        conecta4_buscar_alineamiento_arm
                                (cuadricula,
                                fila - deltas_fila[i], columna - deltas_columna[i], color,
                                -deltas_fila[i], -deltas_columna[i]);

                linea = long_linea >= 4;
        }

        return linea;
}
```

Anexo 1.5: conecta4_hay_linea_arm_arm

```
                AREA datos, DATA

NUM_FILAS       EQU 6
NUM_COLUMNAS    EQU 7
PADDING_FIL     EQU 1
PADDING_COL     EQU 1
TAM_FILS        EQU NUM_FILAS + PADDING_FIL
TAM_COLS        EQU NUM_COLUMNAS + PADDING_COL

CELDA_VACIA     EQU 0x04
CELDA_COLOR     EQU 0x03

FALSE           EQU 0
TRUE            EQU 1

deltas_fila     DCB 0, -1, -1, 1
deltas_columna  DCB -1, 0, -1, -1


                AREA codigo, CODE

                IMPORT conecta4_buscar_alineamiento_arm
                PRESERVE8 {TRUE}

                EXPORT conecta4_hay_linea_arm_arm

;   Entrada:
;       r0 -> @cuadricula
;       r1 -> fila
;       r2 -> columna
;       r3 -> color
;
;   Salida
;       r0 <- >= 1 si hay linea, 0 si no hay linea
;
;   Descripción:
;               Devuelve verdad si ha encontrado una línea de celdas del mismo color al
;               dado de 4 celdas o más. En caso contrario, devueve falso.
conecta4_hay_linea_arm_arm
                ; Prologo
                mov     r12, r13
                stmdb   r13!, { r4 - r10, r11, r12, r14, r15 }
                sub     r11, r12, #4

                ;   Bloque de activación:
                ;
                ;   SP  -> | r4 - r10          | hay_linea
                ;          | FP anterior    |
                ;          | SP anterior       |
                ;          | LR  anterior       |
                ;   FP   -> | PC anterior    |
                ;          | ----------------- |
                ;          |                        | C4_verificar_4_en_linea

                ; Mueve parametros a registros variables
                mov     r4, r0          ; r4 = @cuadricula
```

```
                mov     r5, r1          ; r5 = fila
                mov     r6, r2          ; r6 = columna
                mov     r7, r3          ; r7 = color

                ; Inicializa variables del bucle
                mov     r8, #0           ; i
                mov     r9, #0           ; linea
                mov     r10, #0          ; long_linea

                ; for()
                ;   i < 4
hl_for          cmp     r8, #4
                bge     hl_return
                ;   linea == FALSE
                cmp     r9, #FALSE
                bne     hl_return

                ;   Obtiene deltas[i]
                LDR     r0, =deltas_fila
                ldrsb   r0, [r0, r8]
                LDR     r1, =deltas_columna
                ldrsb   r1, [r1, r8]

                ;   Prepara primera llamada a buscar_alineamiento
                stmdb   r13!, {r0, r1}
                mov     r0, r4
                mov     r1, r5
                mov     r2, r6
                mov     r3, r7

                        bl      conecta4_buscar_alineamiento_arm

                ;   linea = long_linea >= 4
                mov     r10, r0
                cmp     r10, #4
                movge   r9, #TRUE
                bge     hl_for

                ;   Recupera y niega deltas
                ldmia   r13!, { r0, r3 }
                rsb     r0, r0, #0
                rsb     r3, r3, #0

                        ;   Prepara segunda llamada a buscar_alineamiento
                add     r1, r5, r0
                add     r2, r6, r3
                stmdb   r13!, { r0, r3 }
                mov     r0, r4
                mov     r3, r7

                bl      conecta4_buscar_alineamiento_arm

                ;   linea = long_linea >= 4
                add     r10, r10, r0
                cmp     r10, #4
                movge   r9, #TRUE

                ;   i++
                add     r8, r8, #1

                b       hl_for

hl_return       mov     r0, r9
```

```
                ldmdb   r11, { r4 - r10, r11, r13, r15 }

                        END
```

Anexo 1.6: Optimización 1

Anexo 1.6,1: conecta4_buscar_alineamiento_arm_opt_1

```
        AREA datos, DATA

NUM_FILAS       EQU 6
NUM_COLUMNAS    EQU 7
PADDING_FIL     EQU 1
PADDING_COL     EQU 1
TAM_FILS        EQU NUM_FILAS + PADDING_FIL
TAM_COLS        EQU NUM_COLUMNAS + PADDING_COL

CELDA_VACIA     EQU 0x04
CELDA_COLOR     EQU 0x03

FALSE           EQU 0
TRUE            EQU 1

        AREA codigo, CODE

        EXPORT conecta4_buscar_alineamiento_arm_opt1

;   Entrada:
;       r0 -> @cuadricula
;       r1 -> fila
;       r2 -> columna
;       r3 -> color
;       r11 + 4 -> *deltas_fila[i]
;       r11 + 8 -> *deltas_columna[i]
;
;   Salida:
;       r0 <- longuitud linea
;
;   Descripción:
;       Devuelve el número de celdas del mismo color consecutivas en
;       la línea recta dada por delta_fila y delta_columna a partir de
;       cuadricula[fila][columna]

conecta4_buscar_alineamiento_arm_opt1
                ; Prologo
                mov     r12, r13
                stmdb   r13!, { r4 - r10, r11, r12, r14, r15 }
                sub     r11, r12, #4

                ;   Bloque de activación:
                ;
                ;   SP  -> | r4 - r10          | buscar_alineamiento
                ;          | FP anterior       |
                ;          | SP anterior       |
                ;          | LR anterior       |
                ;   FP  -> | PC anterior       |
                ;          | ----------------- |
                ;          | delta_fila[i]     | hay_linea
                ;          | delta_columna[i]  |

                ; Mueve parametros a registros variables
                mov     r4, r0          ; r4 = @cuadricula
                mov     r5, r1          ; r5 = fila
                mov     r6, r2          ; r6 = columna
```

```
                mov     r7, r3          ; r7 = color
                add     r9, r11, #0x04  ; Calcula @ base de parametros
                ldmia   r9, { r8 - r9 } ; r8, r9 = delta_columna, delta_fila
                mov     r0 , #0         ; i = num lineas encontradas

                ; while
                ;   !C4_fila_valida(fila) (! 1 <= fila <= NUM_FILAS)
ba_while        cmp     r5, #1
                blt     ba_return
                cmp     r5, #NUM_FILAS
                bgt     ba_return

                ;   !C4_columna_valida(columna)) (! 1 <= columna <= NUM_COLUMNAS)
                cmp     r6, #1
                blt     ba_return
                cmp     r6, #NUM_COLUMNAS
                bgt     ba_return

                ;   Obtiene valor de celda en cuadricula[fila][columna]
                add     r10, r4, r5, lsl #3     ; r10 = @cuadricula[fila][0] (@cuad + 8 * filas)
                ldrb    r10, [r10, r6]          ; r10 = *cuadricula[fila][columna]

                ;   celda_vacia(cuadricula[fila][columna]) ( celda & 0x04 == 0)
                and     r3, r10, #CELDA_VACIA
                cmp     r3, #0
                beq     ba_return

                ;   celda_color(cuadricula[fila][columna] != ) ( celda & 0x03 != color)
                and     r3, r10, #CELDA_COLOR
                cmp     r3, r7
                bne     ba_return

                ;   Actualiza índices
                add     r5, r5, r8
                add     r6, r6, r9

                ;   Suma 1 al resultado
                add     r0, r0 , #1

                b ba_while

ba_return       ldmdb   r11, { r4 - r10, r11, r13, r15 }

                END
```

Anexo 1.6.2: conecta4_hay_linea_arm_arm_opt_1

```
        AREA datos, DATA

NUM_FILAS       EQU 6
NUM_COLUMNAS    EQU 7
PADDING_FIL     EQU 1
PADDING_COL     EQU 1
TAM_FILS        EQU NUM_FILAS + PADDING_FIL
TAM_COLS        EQU NUM_COLUMNAS + PADDING_COL

CELDA_VACIA     EQU 0x04
CELDA_COLOR     EQU 0x03
```

```
FALSE           EQU 0
TRUE            EQU 1

deltas_fila     DCB 0, -1, -1, 1
deltas_columna  DCB -1, 0, -1, -1


        AREA codigo, CODE

        IMPORT conecta4_buscar_alineamiento_arm_opt1
        PRESERVE8 {TRUE}

        EXPORT conecta4_hay_linea_arm_arm_opt1

;   Entrada:
;       r0 -> @cuadricula
;       r1 -> fila
;       r2 -> columna
;       r3 -> color
;
;   Salida
;       r0 <- >= 1 si hay linea, 0 si no hay linea
;
;   Descripci'n:
;       Devuelve verdad si ha encontrado una l'nea de celdas del mismo color al
;       dado de 4 celdas o m's. En caso contrario, devueve falso.
conecta4_hay_linea_arm_arm_opt1
                ; Prologo
                mov     r12, r13
                stmdb   r13!, { r4 - r10, r11, r12, r14, r15 }
                sub     r11, r12, #4

                ;   Bloque de activaci'n:
                ;
                ;   SP  -> | r4 - r10          | hay_linea
                ;          | FP anterior       |
                ;          | SP anterior       |
                ;          | LR anterior       |
                ;   FP  -> | PC anterior       |
                ;          | ----------------- |
                ;          |                   | C4_verificar_4_en_linea

                ; Mueve parametros a registros variables
                mov     r4, r0          ; r4 = @cuadricula
                mov     r5, r1          ; r5 = fila
                mov     r6, r2          ; r6 = columna
                mov     r7, r3          ; r7 = color

                ; Inicializa variables del bucle
                mov     r8, #0           ; i
                mov     r9, #0           ; linea
                mov     r10, #0          ; long_linea

                ; for()
                ;   i < 4
hl_for          cmp     r8, #4
                bge     hl_return
                ;   linea == FALSE
                cmp     r9, #FALSE
                bne     hl_return

                ;   Obtiene deltas[i]
```

```
                LDR     r0, =deltas_fila
                ldrsb   r0, [r0, r8]
                LDR     r1, =deltas_columna
                ldrsb   r1, [r1, r8]

                ;   Prepara primera llamada a buscar_alineamiento
                stmdb   r13!, {r0, r1}
                mov     r0, r4
                mov     r1, r5
                mov     r2, r6
                mov     r3, r7

                bl      conecta4_buscar_alineamiento_arm_opt1

                ;   linea = long_linea >= 4
                mov     r10, r0
                cmp     r10, #4
                movge   r9, #TRUE
                bge     hl_for

                ;   Recupera y niega deltas
                ldmia   r13!, { r0, r3 }
                rsb     r0, r0, #0
                rsb     r3, r3, #0

                ;   Prepara segunda llamada a buscar_alineamiento
                add     r1, r5, r0
                add     r2, r6, r3
                stmdb   r13!, { r0, r3 }
                mov     r0, r4
                mov     r3, r7

                bl      conecta4_buscar_alineamiento_arm_opt1

                ;   linea = long_linea >= 4
                add     r10, r10, r0
                cmp     r10, #4
                movge   r9, #TRUE

                ;   i++
                add     r8, r8, #1

                b       hl_for

hl_return       mov     r0, r9
                ldmdb   r11, { r4 - r10, r11, r13, r15 }

                END
```

Anexo 1.7: Optimización 2

Anexo 1.7.1: conecta4_buscar_alineamiento_arm_opt_2

```
        AREA datos, DATA

NUM_FILAS       EQU 6
NUM_COLUMNAS    EQU 7
PADDING_FIL     EQU 1
PADDING_COL     EQU 1
TAM_FILS        EQU NUM_FILAS + PADDING_FIL
TAM_COLS        EQU NUM_COLUMNAS + PADDING_COL

CELDA_VACIA     EQU 0x04
CELDA_COLOR     EQU 0x03

FALSE           EQU 0
TRUE            EQU 1

        AREA codigo, CODE

        EXPORT conecta4_buscar_alineamiento_arm_opt2

;   Entrada:
;       r0 -> @cuadricula
;       r1 -> fila
;       r2 -> columna
;       r3 -> color
;       r11 + 4 -> *deltas_fila[i]
;       r11 + 8 -> *deltas_columna[i]
;
;   Salida:
;       r0 <- longuitud linea
;
;   Descripci'n:
;       Devuelve el n'mero de celdas del mismo color consecutivas en
;       la l'nea recta dada por delta_fila y delta_columna a partir de
;       cuadricula[fila][columna]

conecta4_buscar_alineamiento_arm_opt2
                ; Prologo
                mov     r12, r13
                stmdb   r13!, { r4 - r10, r11, r12, r14, r15 }
                sub     r11, r12, #4

                ;   Bloque de activaci'n:
                ;
                ;   SP  -> | r4 - r10          | buscar_alineamiento
                ;          | FP anterior       |
                ;          | SP anterior       |
                ;          | LR anterior       |
                ;   FP  -> | PC anterior       |
                ;          | ----------------- |
                ;          | delta_fila[i]     | hay_linea
                ;          | delta_columna[i]  |

                ; Mueve parametros a registros variables
```

```
                mov     r4, r0          ; r4 = @cuadricula
                add     r7, r11, #0x04  ; Calcula @ base de parametros
                ldmia   r7, { r5 - r6 } ; r5, r6 = delta_columna, delta_fila
                mov     r0 , #0         ; i = num lineas encontradas

                ; while
                ;   !C4_fila_valida(fila) (! 1 <= fila <= NUM_FILAS)
ba_while        cmp     r1, #1
                blt     ba_return
                cmp     r1, #NUM_FILAS
                bgt     ba_return

                ;   !C4_columna_valida(columna)) (! 1 <= columna <= NUM_COLUMNAS)
                cmp     r2, #1
                blt     ba_return
                cmp     r2, #NUM_COLUMNAS
                bgt     ba_return

                ;   Obtiene valor de celda en cuadricula[fila][columna]
                add     r7, r4, r1, lsl #3     ; r7 = @cuadricula[fila][0] (@cuad + 8 * filas)
                ldrb    r7, [r7, r2]           ; r7 = *cuadricula[fila][columna]

                ;   celda_vacia(cuadricula[fila][columna]) ( celda & 0x04 == 0)
                tst     r7, #CELDA_VACIA
                beq     ba_return

                ;   celda_color(cuadricula[fila][columna] != ) ( celda & 0x03 != color)
                and     r8, r7, #CELDA_COLOR
                cmp     r8, r3
                bne     ba_return

                ;   Actualiza 'ndices
                add     r1, r1, r5
                add     r2, r2, r6

                ;   Suma 1 al resultado
                add     r0, r0 , #1

                b ba_while

ba_return       ldmdb   r11, { r4 - r10, r11, r13, r15 }

                END
```

Anexo 1.7.2: conecta4_hay_linea_arm_arm_opt_2

```
        AREA datos, DATA

NUM_FILAS       EQU 6
NUM_COLUMNAS    EQU 7
PADDING_FIL     EQU 1
PADDING_COL     EQU 1
TAM_FILS        EQU NUM_FILAS + PADDING_FIL
TAM_COLS        EQU NUM_COLUMNAS + PADDING_COL

CELDA_VACIA     EQU 0x04
CELDA_COLOR     EQU 0x03

FALSE           EQU 0
TRUE            EQU 1
```

```
deltas_fila     DCB 0, -1, -1, 1
deltas_columna  DCB -1, 0, -1, -1


        AREA codigo, CODE

        IMPORT conecta4_buscar_alineamiento_arm_opt2
        PRESERVE8 {TRUE}

        EXPORT conecta4_hay_linea_arm_arm_opt2

;   Entrada:
;       r0 -> @cuadricula
;       r1 -> fila
;       r2 -> columna
;       r3 -> color
;
;   Salida
;       r0 <- >= 1 si hay linea, 0 si no hay linea
;
;   Descripci'n:
;       Devuelve verdad si ha encontrado una l'nea de celdas del mismo color al
;       dado de 4 celdas o m's. En caso contrario, devueve falso.
conecta4_hay_linea_arm_arm_opt2
                ; Prologo
                mov     r12, r13
                stmdb   r13!, { r4 - r10, r11, r12, r14, r15 }
                sub     r11, r12, #4

                ;   Bloque de activaci'n:
                ;
                ;   SP  -> | r4 - r10          | hay_linea
                ;          | FP anterior       |
                ;          | SP anterior       |
                ;          | LR anterior       |
                ;   FP  -> | PC anterior       |
                ;          | ----------------- |
                ;          |                   | C4_verificar_4_en_linea

                ; Mueve parametros a registros variables
                mov     r4, r0          ; r4 = @cuadricula
                mov     r5, r1          ; r5 = fila
                mov     r6, r2          ; r6 = columna
                mov     r7, r3          ; r7 = color

                ; Inicializa variables del bucle
                mov     r8, #0          ; i
                mov     r9, #0          ; linea
                mov     r10, #0         ; long_linea

                ; for()
                ;   i < 4
hl_for          cmp     r8, #4
                bge     hl_return
                ;   linea == FALSE
                cmp     r9, #FALSE
                bne     hl_return

                ;   Obtiene deltas[i]
                LDR     r0, =deltas_fila
                ldrsb   r0, [r0, r8]
                LDR     r1, =deltas_columna
```

```
                ldrsb   r1, [r1, r8]

                ;   Prepara primera llamada a buscar_alineamiento
                stmdb   r13!, {r0, r1}
                mov     r0, r4
                mov     r1, r5
                mov     r2, r6
                mov     r3, r7

                bl      conecta4_buscar_alineamiento_arm_opt2

                ;   linea = long_linea >= 4
                mov     r10, r0
                cmp     r10, #4
                movge   r9, #TRUE
                bge     hl_for

                ;   Recupera y niega deltas
                ldmia   r13!, { r0, r3 }
                rsb     r0, r0, #0
                rsb     r3, r3, #0

                ;   Prepara segunda llamada a buscar_alineamiento
                add     r1, r5, r0
                add     r2, r6, r3
                stmdb   r13!, { r0, r3 }
                mov     r0, r4
                mov     r3, r7

                bl      conecta4_buscar_alineamiento_arm_opt2

                ;   linea = long_linea >= 4
                add     r10, r10, r0
                cmp     r10, #4
                movge   r9, #TRUE

                ;   i++
                add     r8, r8, #1

                b       hl_for

hl_return       mov     r0, r9
                ldmdb   r11, { r4 - r10, r11, r13, r15 }

                END
```

Anexo 1.8: Optimización 3 (conecta4_hay_linea_arm_opt3)

```
        AREA datos, DATA

NUM_FILAS       EQU 6
NUM_COLUMNAS    EQU 7
PADDING_FIL     EQU 1
PADDING_COL     EQU 1
TAM_FILS        EQU NUM_FILAS + PADDING_FIL
TAM_COLS        EQU NUM_COLUMNAS + PADDING_COL

CELDA_VACIA     EQU 0x04
CELDA_COLOR     EQU 0x03

FALSE           EQU 0
TRUE            EQU 1

deltas_fila     DCB 0, -1, -1, 1
deltas_columna  DCB -1, 0, -1, -1


        AREA codigo, CODE

        IMPORT conecta4_buscar_alineamiento_arm_opt2
        PRESERVE8 {TRUE}

        EXPORT conecta4_hay_linea_arm_arm_opt3

;   Entrada:
;       r0 -> @cuadricula
;       r1 -> fila
;       r2 -> columna
;       r3 -> color
;
;   Salida
;       r0 <- >= 1 si hay linea, 0 si no hay linea
;
;   Descripci'n:
;       Devuelve verdad si ha encontrado una l'nea de celdas del mismo color al
;       dado de 4 celdas o m's. En caso contrario, devueve falso.
conecta4_hay_linea_arm_arm_opt3
                ; Prologo
                mov     r12, r13
                stmdb   r13!, { r4 - r10, r11, r12, r14, r15 }
                sub     r11, r12, #4

                ;   Bloque de activaci'n:
                ;
                ;   SP  -> | r4 - r10          | hay_linea
                ;          | FP anterior       |
                ;          | SP anterior       |
                ;          | LR anterior       |
                ;   FP  -> | PC anterior       |
                ;          | ----------------- |
                ;          |                   | C4_verificar_4_en_linea

                ; Mueve parametros a registros variables
                mov     r4, r0          ; r4 = @cuadricula
                mov     r5, r1          ; r5 = fila
```

```
                mov     r6, r2          ; r6 = columna
                mov     r7, r3          ; r7 = color

                ; Inicializa variables del bucle
                mov     r8, #0          ; i

                ; for()
hl_for          mov     r9, #1          ; long_linea

                ;   i < 4
                cmp     r8, #4
                movge   r0, #FALSE
                bge     hl_return

                ;   Obtiene deltas[i]
                LDR     r0, =deltas_fila
                ldrsb   r0, [r0, r8]
                LDR     r3, =deltas_columna
                ldrsb   r3, [r3, r8]

                ;   Prepara primera llamada a buscar_alineamiento
                add     r1, r5, r0
                add     r2, r6, r3
                stmdb   r13!, {r0, r3}
                mov     r0, r4
                mov     r3, r7

                bl      conecta4_buscar_alineamiento_arm_opt2

                ;   long_linea >= 4
                add     r9, r0, r9
                cmp     r9, #4
                movge   r0, #TRUE
                bge     hl_return

                ;   Recupera y niega deltas
                ldmia   r13!, { r0, r3 }
                rsb     r0, r0, #0
                rsb     r3, r3, #0

                ;   Prepara segunda llamada a buscar_alineamiento
                add     r1, r5, r0
                add     r2, r6, r3
                stmdb   r13!, {r0, r3}
                mov     r0, r4
                mov     r3, r7

                bl      conecta4_buscar_alineamiento_arm_opt2

                ;   long_linea >= 4
                add     r9, r9, r0
                cmp     r9, #4
                movge   r0, #TRUE
                bge     hl_return

                ;   i++
                add     r8, r8, #1
                b       hl_for

hl_return       ldmdb   r11, { r4 - r10, r11, r13, r15 }

                END
```

Anexo 1.9: Optimización 4 (conecta4_hay_linea_arm_opt4)

```
        AREA datos, DATA

NUM_FILAS       EQU 6
NUM_COLUMNAS    EQU 7
PADDING_FIL     EQU 1
PADDING_COL     EQU 1
TAM_FILS        EQU NUM_FILAS + PADDING_FIL
TAM_COLS        EQU NUM_COLUMNAS + PADDING_COL

CELDA_VACIA     EQU 0x04
CELDA_COLOR     EQU 0x03

FALSE           EQU 0
TRUE            EQU 1

deltas          DCB 0, -1, -1, 0, -1, -1, 1, -1


        AREA codigo, CODE

        IMPORT conecta4_buscar_alineamiento_arm_opt2
        PRESERVE8 {TRUE}

        EXPORT conecta4_hay_linea_arm_arm_opt4

;   Entrada:
;       r0 -> @cuadricula
;       r1 -> fila
;       r2 -> columna
;       r3 -> color
;
;   Salida
;       r0 <- >= 1 si hay linea, 0 si no hay linea
;
;   Descripci'n:
;       Devuelve verdad si ha encontrado una l'nea de celdas del mismo color al
;       dado de 4 celdas o m's. En caso contrario, devueve falso.
conecta4_hay_linea_arm_arm_opt4
                ; Prologo
                mov     r12, r13
                stmdb   r13!, { r4 - r10, r11, r12, r14, r15 }
                sub     r11, r12, #4

                ;   Bloque de activaci'n:
                ;
                ;   SP  -> | r4 - r10          | hay_linea
                ;          | FP anterior       |
                ;          | SP anterior       |
                ;          | LR anterior       |
                ;   FP  -> | PC anterior       |
                ;          | ----------------- |
                ;          |                   | C4_verificar_4_en_linea

                ; Mueve parametros a registros variables
                mov     r4, r0          ; r4 = @cuadricula
                mov     r5, r1          ; r5 = fila
                mov     r6, r2          ; r6 = columna
```

```
                mov     r7, r3          ; r7 = color

                ; Inicializa variables del bucle
                mov     r8, #0          ; i
                LDR     r10, =deltas    ; @deltas
                ; for()
hl_for          mov     r9, #1          ; long_linea

                ;   i < 4
                cmp     r8, #4
                movge   r0, #FALSE
                bge     hl_return

                ;   Obtiene deltas[i]
                ldrsb   r0, [r10], #1
                ldrsb   r3, [r10], #1

                ;   Prepara primera llamada a buscar_alineamiento
                add     r1, r5, r0
                add     r2, r6, r3
                stmdb   r13!, {r0, r3}
                mov     r0, r4
                mov     r3, r7

                bl      conecta4_buscar_alineamiento_arm_opt2

                ;   long_linea >= 4
                add     r9, r0, r9
                cmp     r9, #4
                movge   r0, #TRUE
                bge     hl_return

                ;   Recupera y niega deltas
                ldmia   r13!, { r0, r3 }
                rsb     r0, r0, #0
                rsb     r3, r3, #0

                ;   Prepara segunda llamada a buscar_alineamiento
                add     r1, r5, r0
                add     r2, r6, r3
                stmdb   r13!, {r0, r3}
                mov     r0, r4
                mov     r3, r7

                bl      conecta4_buscar_alineamiento_arm_opt2

                ;   long_linea >= 4
                add     r9, r9, r0
                cmp     r9, #4
                movge   r0, #TRUE
                bge     hl_return

                ;   i++
                add     r8, r8, #1
                b       hl_for

hl_return       ldmdb   r11, { r4 - r10, r11, r13, r15 }

                END
```

Anexo 2: Banco de pruebas

Anexo 2.1: Test 1

TABLERO INICIAL:

	C1	C2	C3	C4	C5	C6	C7
F1							
F2							
F3							
F4							
F5							
F6							

MOVIMIENTOS:

Banca 1, Negra 6,
Blanca 1, Negra 7,
Blanca 2, Negra 7,
Blanca 2, Negra 6,
Blanca 3, Negra 7,
Blanca 3, Negra 6,
Blanca 3, Negra 5,
Blanca 1, Negra 5,
Blanca 2, Negra 5,
Blanca 7, Negra 2,
Blanca 6, Negra 2,
Blanca 7, Negra 1,
Blanca 6, Negra 1,
Blanca 7, Negra 3,
Blanca 5, Negra 3,

Blanca 5, Negra 1,
Blanca 5, Negra 3,
Blanca 6, Negra 2

TABLERO FINAL:

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○		●	●	●
F2	○	○	○		●	●	●
F3	○	○	○		●	●	●
F4	●	●	●		○	○	○
F5	●	●	●		○	○	○
F6	●	●	●		○	○	○

Anexo 2.2: Test 2

TABLERO INICIAL:

	C1	C2	C3	C4	C5	C6	C7
F1	●	●	●		○	○	○
F2	●		●		○		○
F3	●		●		○		○
F4							
F5							
F6							

MOVIMIENTOS:

Blanca 2, Negra 6

TABLERO FINAL:

	C1	C2	C3	C4	C5	C6	C7
F1	●	●	●		○	○	○
F2	●	○	●		○	●	○
F3	●		●		○		○
F4							
F5							
F6							

Anexo 2.3: Test 3

TABLERO INICIAL:

	C1	C2	C3	C4	C5	C6	C7
F1		○	●	○	●	○	
F2			●	○	●		
F3			○	●	○		
F4			●	○	●		
F5			○		○		
F6							

UNDO_IF_WIN definido

MOVIMIENTOS:

Blanca 2 (Victoria), Negra 2,
Blanca 6 (Victoria), Negra 6,
Blanca 2, Negra 6,
Blanca 6, Negra 2,
Blanca 1, Negra 2 (Victoria),
Blanca 7, Negra 6 (Victoria)

	C1	C2	C3	C4	C5	C6	C7
F1		○	●	○	●	○	
F2		○	●	○	●		
F3			○	●	○		
F4			●	○	●		
F5			○		○		
F6							

Detecta victoria de blancas y deshace el movimiento.

	C1	C2	C3	C4	C5	C6	C7
F1		○	●	○	●	○	
F2		●	●	○	●		
F3			○	●	○		
F4			●	○	●		
F5			○		○		
F6							

	C1	C2	C3	C4	C5	C6	C7
F1		○	●	○	●	○	
F2		●	●	○	●	○	
F3			○	●	○		
F4			●	○	●		
F5			○		○		
F6							

Detecta victoria de blancas y deshace el movimiento.

	C1	C2	C3	C4	C5	C6	C7
F1		○	●	○	●	○	
F2		●	●	○	●	●	
F3			○	●	○		
F4			●	○	●		
F5			○		○		
F6							

	C1	C2	C3	C4	C5	C6	C7
F1		○	●	○	●	○	
F2		●	●	○	●	●	
F3		○	○	●	○		
F4			●	○	●		
F5			○		○		
F6							

	C1	C2	C3	C4	C5	C6	C7
F1		○	●	○	●	○	
F2		●	●	○	●	●	
F3		○	○	●	○	●	
F4			●	○	●		
F5			○		○		
F6							

	C1	C2	C3	C4	C5	C6	C7
F1		○	●	○	●	○	
F2		●	●	○	●	●	
F3		○	○	●	○	●	
F4			●	○	●	○	
F5			○		○		
F6							

	C1	C2	C3	C4	C5	C6	C7
F1		○	●	○	●	○	
F2		●	●	○	●	●	
F3		○	○	●	○	●	
F4		●	●	○	●	○	
F5			○		○		
F6							

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	●	○	●	○	
F2		●	●	○	●	●	
F3		○	○	●	○	●	
F4		●	●	○	●	○	
F5			○		○		
F6							

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	●	○	●	○	
F2		●	●	○	●	●	
F3		○	○	●	○	●	
F4		●	●	○	●	○	
F5		●	○		○		
F6							

Detecta victoria de negras y deshace el movimiento.

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	●	○	●	○	○
F2		●	●	○	●	●	
F3		○	○	●	○	●	
F4		●	●	○	●	○	
F5			○		○		
F6							

TABLERO FINAL:

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	●	○	●	○	○
F2		●	●	○	●	●	
F3		○	○	●	○	●	
F4		●	●	○	●	○	
F5			○		○	●	
F6							

Detecta victoria de negras y deshace el movimiento.
Movimientos para probar los casos de victoria diagonales.

Anexo 2.4: Test 4

TABLERO INICIAL:

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○	●	○	●	
F2	●		●	○		●	
F3	○		○	●		○	
F4	●		●	○		○	
F5							
F6							

UNDO_IF_WIN definido

MOVIMIENTOS:

Blanca 2 (Victoria) , Negra 2,
Blanca 7, Negra 2 (Victoria),
Blanca 7, Negra 5 (Victoria),
Blanca 5, Negra 7,
Blanca 5 (Victoria)

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○	●	○	●	
F2	●	○	●	○		●	
F3	○		○	●		○	
F4	●		●	○		○	
F5							
F6							

Detecta victoria de blancas y deshace el movimiento.

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○	●	○	●	
F2	●	●	●	○		●	
F3	○		○	●		○	
F4	●		●	○		○	
F5							
F6							

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○	●	○	●	○
F2	●	●	●	○		●	
F3	○		○	●		○	
F4	●		●	○		○	
F5							
F6							

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○	●	○	●	○
F2	●	●	●	○		●	
F3	○	●	○	●		○	
F4	●		●	○		○	
F5							
F6							

Detecta victoria de negras y deshace el movimiento.

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○	●	○	●	○
F2	●	●	●	○		●	○
F3	○		○	●		○	
F4	●		●	○		○	
F5							
F6							

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○	●	○	●	○
F2	●	●	●	○	●	●	○
F3	○		○	●		○	
F4	●		●	○		○	
F5							
F6							

Detecta victoria de negras y deshace el movimiento.

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○	●	○	●	○
F2	●	●	●	○	○	●	○
F3	○		○	●		○	
F4	●		●	○		○	
F5							
F6							

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○	●	○	●	○
F2	●	●	●	○	○	●	○
F3	○		○	●		○	●
F4	●		●	○		○	
F5							
F6							

TABLERO FINAL:

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○	●	○	●	○
F2	●	●	●	○	○	●	○
F3	○		○	●	○	○	●
F4	●		●	○		○	
F5							
F6							

Detecta victoria de blancas y deshace el movimiento.
Movimientos para probar los casos de victoria diagonales.

Anexo 2.5: Test 5

TABLERO INICIAL:

	C1	C2	C3	C4	C5	C6	C7
F1	○		○	○		○	
F2			○	○			
F3			○				
F4							
F5							
F6							

UNDO_IF_WIN definido

MOVIMIENTOS:

Blanca 2 (Victoria), Negra 2,
Blanca 5 (Victoria), Negra 5,
Blanca 2, Negra 4,
Blanca 1 (Victoria), Negra 1,
Blanca 5 (Victoria), Negra 5,
Blanca 3 (Victoria)

	C1	C2	C3	C4	C5	C6	C7
F1	○	○	○	○		○	
F2			○	○			
F3			○				
F4							
F5							
F6							

Detecta victoria de blancas y deshace el movimiento.

	C1	C2	C3	C4	C5	C6	C7
F1	○	●	○	○		○	
F2			○	○			
F3			○				
F4							
F5							
F6							

	C1	C2	C3	C4	C5	C6	C7
F1	○	●	○	○	○	○	
F2			○	○			
F3			○				
F4							
F5							
F6							

Detecta victoria de blancas y deshace el movimiento.

	C1	C2	C3	C4	C5	C6	C7
F1	○	●	○	○	●	○	
F2			○	○			
F3			○				
F4							
F5							
F6							

	C1	C2	C3	C4	C5	C6	C7
F1	○	●	○	○	●	○	
F2		○	○	○			
F3			○				
F4							
F5							
F6							

	C1	C2	C3	C4	C5	C6	C7
F1	○	●	○	○	●	○	
F2		○	○	○			
F3			○	●			
F4							
F5							
F6							

	C1	C2	C3	C4	C5	C6	C7
F1	○	●	○	○	●	○	
F2	○	○	○	○			
F3			○	●			
F4							
F5							
F6							

Detecta victoria de blancas y deshace el movimiento.

	C1	C2	C3	C4	C5	C6	C7
F1	○	●	○	○	●	○	
F2	●	○	○	○			
F3			○	●			
F4							
F5							
F6							

	C1	C2	C3	C4	C5	C6	C7
F1	○	●	○	○	●	○	
F2	●	○	○	○	○		
F3			○	●			
F4							
F5							
F6							

Detecta victoria de blancas y deshace el movimiento.

	C1	C2	C3	C4	C5	C6	C7
F1	○	●	○	○	●	○	
F2	●	○	○	○	●		
F3			○	●			
F4							
F5							
F6							

TABLERO FINAL:

	C1	C2	C3	C4	C5	C6	C7
F1	○	●	○	○	●	○	
F2	●	○	○	○	●		
F3			○	●			
F4			○				
F5							
F6							

Detecta victoria de blancas y deshace el movimiento.
Movimientos para probar los casos de victoria horizontales y verticales.

Anexo 2.6: Test de rendimiento

TABLERO INICIAL:

	C1	C2	C3	C4	C5	C6	C7
F1							
F2							
F3							
F4							
F5							
F6							

MOVIMIENTOS:

Blanca 4, Negra 4,
Blanca 4, Negra 4,
Blanca 4, Negra 1,
Blanca 5, Negra 6,
Blanca 6, Negra 6,
Blanca 6, Negra 1,
Blanca 4, Negra 2,
Blanca 2, Negra 2,
Blanca 2, Negra 1,
Blanca 1, Negra 1,
Blanca 2, Negra 1,
Blanca 2, Negra 3,
Blanca 6, Negra 3,
Blanca 6, Negra 3,
Blanca 3, Negra 5,
Blanca 5, Negra 7,
Blanca 3, Negra 3,
Blanca 7, Negra 7,
Blanca 7, Negra 7,

Blanca 5, Negra 5,
Blanca 7 (Victoria)

TABLERO FINAL:

	C1	C2	C3	C4	C5	C6	C7
F1	●	●	●	○	○	●	●
F2	●	○	●	●	●	○	○
F3	●	●	●	○	○	●	●
F4	○	○	○	●	○	○	○
F5	●	○	○	○	●	○	●
F6	●	○	●	○			○

Ejemplo de partida completa con victoria de las blancas.

Anexo 3: Trabajo dedicado

horas/tarea/miembro	Dorian Wozniak	Pablo Latre	Total
Análisis inicial	6	6	12
Implementación ARM	2	2	4
Análisis rendimiento C	1.5	1	2.5
Análisis rendimiento ARM	3	1.5	4.5
Implementación banco de pruebas	5	3	8
Optimización ARM	6	4	10
Análisis rendimiento ARM optimizado	2.5	1.5	4
Memoria	12	6	18
Total	38	25	**63 horas totales**

Anexo 4: Ficheros adjuntos

Este documento (Memoria_p1_778043-Latre_Villacampa_817570-Wozniak.pdf) es parte de una entrega de múltiples ficheros.

Junto a la memoria, se deberán haber entregado un .zip con una serie de códigos fuente de las funciones en ensamblador ARM implementadas en formato .txt; y otro zip con el proyecto de Keil µVision enviado anteriormente con todos los ficheros requeridos para compilar y ejecutar el proyecto. Todos los ficheros siguen el siguiente formato de nombre:

fichero_p1_778043-Latre_Villacampa_817570-Wozniak.ext

La entrega entera tiene la siguiente estructura:

```
p1_778043-Latre_Villacampa_817570-Wozniak.zip
|
|- Memoria_p1_778043-Latre_Villacampa_817570-Wozniak.pdf
|
|- conecta4_busar_alineamiento_arm_p1_778043-Latre_Villacampa_817570-Wozniak.txt
|- conecta4_busar_alineamiento_arm_opt1_p1_778043-Latre_Villacampa_817570-Wozniak.txt
|- conecta4_busar_alineamiento_arm_opt2_p1_778043-Latre_Villacampa_817570-Wozniak.txt
|
|- conecta4_hay_linea_arm_c_p1_778043-Latre_Villacampa_817570-Wozniak.txt
|- conecta4_hay_linea_arm_arm_p1_778043-Latre_Villacampa_817570-Wozniak.txt
|- conecta4_hay_linea_arm_arm_opt1_p1_778043-Latre_Villacampa_817570-Wozniak.txt
|- conecta4_hay_linea_arm_arm_opt2_p1_778043-Latre_Villacampa_817570-Wozniak.txt
|- conecta4_hay_linea_arm_arm_opt3_p1_778043-Latre_Villacampa_817570-Wozniak.txt
|- conecta4_hay_linea_arm_arm_opt4_p1_778043-Latre_Villacampa_817570-Wozniak.txt
|
|- proyecto_p1_778043-Latre_Villacampa_817570-Wozniak.zip
```

Anexo 5: Bibliografía

[1]

https://developer.arm.com/documentation/dui0056/d/interworking-arm-and-thumb/c-and-c---interworking-and-veneers/compiling-code-for-interworking

[2]

https://developer.arm.com/documentation/dui0375/g/Compiler-Command-line-Options/-Onum

[3] http://www.spec.org/omp2012/flags/arm_compiler.html