Proyecto Hardware

# Práctica 1: Desarrollo y optimización de código para el procesador ARM

# Proyecto Hardware

# 2017/18

# 

# Jorge Aznar López - 721556

# Alex Oarga Hategan - 718123

Índice

* 1 Introducción Página
* 2 Resumen Página
* 3 Objetivos Página
* 4 Metodología Página
* 5 Resultados Página
* 6 Conclusiones Página

# 1. Resumen

# 2. Introducción

Se van a desarrollar en ARM y THUMB las funciones ficha\_valida y patron\_volteo del codigo C original del juego reversi8.

# 3. Objetivos

El primer objetivo del proyecto es la traduccion del codigo C a ARM y THUMB. Como la funcion patron\_volteo invoca a ficha\_valida, se pretende desarrollar varias combinaciones de las funciones. Estas convinaciones seran (patron\_volteo - ficha\_valida): C-C, C-ARM, C-THUMB, ARM-C, ARM-ARM, ARM-THUMB. Posteriormente, se desarrrollara un repertorio de pruebas para evaluar el correcto comportamiento de cada una de las combinaciones. Finalmente, se evaluacion las prestaciones de dichas combinaciones y se compararan entre ellas ademas de con diferentes versiones de optimizacion del codigo en C (O1, O2, ...).

# 4. Metodologia

4.1. Estructura del proyecto

4.1.1 Ficheros

Los ficheros que contiene el proyecto son los siguientes:

[CAPTURA]

4.1.2 Funciones

Se van a desarrollar en el fichero 'Apartado2.asm' las funciones 'ficha\_valida' y 'patron\_volteo' que se incluyen en el codigo C original del juego reversi8. Las versiones que se van a desarrollar en este fichero son:

la version ARM de la funcion ficha\_valida(), la version thumb de ficha\_valida() y la version ARM de patron\_volteo(). Patron\_volteo() es una funcion que llama a su vez a la funcion ficha\_valida, asi que las 3 versiones que se van a desarrollar y el codigo en c dado, se van a utilizar para las diferentes combinaciones de llamadas.

Como el objetivo es generar las 6 combinaciones: C-C, C-ARM, C-THUMB, ARM-ARM, ARM-C, ARM-THUMB; se va a desarrollar una funcion para cada una de las combinaciones que se organizaran de la siguiente manera:

- en reversi8\_2017.c: las funciones 'patron\_volteo\_c\_c' (las funciones originales sin modificar), 'patron\_volteo\_c\_arm' y 'patron\_volteo\_thumb' (las funciones originales variando la llamada a ficha\_valida() segun cada caso).

- en Apartado2.asm: 'patron\_volteo\_arm\_arm' (version ARM de patron\_volteo que llama a la version ARM de ficha\_valida), 'patron\_volteo\_arm\_c' (version ARM de patron\_volteo que llama a la version C de ficha\_valida) y 'patron\_volteo\_arm\_thumb' (version ARM de patron\_volteo que llama a la version thumb de ficha\_valida), ademas de las propias funciones ficha\_valida\_arm y ficha\_valida\_thumb.

4.1.2 Lineas de codigo

El total de lineas de codigo que se ha obtenido en este proyecto (sin contar comentarios, apertura o cierre con '{' '}') es el siguiente:

reversi8\_2017.c

ficha\_valida (funcion original): 9

patron\_volteo\_c\_c (funcion original): 18

patron\_volteo\_c\_arm: 18 (solo cambia la llamada a ficha\_valida)

patron\_volteo\_c\_thumb: 18 (solo cambia la llamada a ficha\_valida)

Apartado2.asm

ficha\_valida\_arm: 12 instrucciones ARM

ficha\_valida\_thumb: 20 instrucciones thumb

patron\_volteo\_arm\_arm: 39 instrucciones ARM

patron\_volteo\_arm\_c: 39 instrucciones ARM

patron\_volteo\_arm\_thumb: 39 instrucciones ARM

4.2 Desarrollo

4.1 ficha valida arm

La funcion ficha valida recibe 4 argumentos. Se sigue el estandar ATPCS por lo que al ser 4 parametros se pasaran en los registros r0-r3. Los 3 primeros caracteres que se reciben son de tipo char. Segun el estandar estos datos (que ocupan 1 byte) se rellenan con 0s hasta llenar los 32 bits. El primer parametro es un vector de char asi que no se reciben los datos sino la direccion donde comienza el vector. El cuarto parametro es un entero (int) que se pasa como un puntero, por lo que se recibira como parametro la direccion de donde esta contenido.

Lo primero que se ejecuta en la funcion es el conidiconal if. Para compara que f y c (parametros r1 y r2) son mayores o iguales que 0 y menores que 8 (DIM), se va a utilizar la instruccion CMP e instrucciones predicadas. En este caso se ha comparado con 7. De esta forma se puede utilizar los condicionales sin signo HI y LS para comparar a la vez que el valor sea mayor o igual a 0 y menor que 8, ya que si el valor es negativo, si se considera que es un numero sin signo, sera mayor que el propio valor. \*\* REVISAR \*\*

Se realiza la primera comparacion (con el valor de f) y en caso de cumplir las condiciones se evalua la segunda condicion (con el valor de c). En caso de ser mayor sin signo, con el condicional HI se salta con la instruccion B a la seccion correspondiente al codigo else, identificado con la etiqueta 'else\_fvarm'. En caso de cumplir la condicion continua la ejecicion del codigo sin saltar.

ADD r0, r0, r1, lsl#3

Si continua la ejecucion, se pasa a comparar tablero[f][c] con CASILLA\_VACIA (0). El vector tablero contiene los datos (\*\*\*\*\*\*\*\*\*\*\*\*\*\*). Por lo tanto, para acceder al valor [f][c] se tendra que calcular la posicion correspondiente a: @tablero + f\*DIM + c (\*\*\*\*\*\*\*\*\*\*\*\*), siendo DIM en este caso 8. Para multiplicar por 8, se ha utilizado el operador de desplazamiento lsl 3, pues multiplicar por 8 (2^3) un numero binario es equivalente a desplzar el numero 3 bits a la izquierda. Este operador se utiliza en la operacion add que suma a la direccion de tablero el valor f al que se le aplica el desplazamiento, es decir:

ldrb r0, [r0, r2]

Para cargar la posicion [f][c] todavia es necesario sumarle c. Esto se consigue con el operador de offset de la instruccion LDR, para el cual se utiliza el registro de c (r2). Es decir, la operacion carga en r0 la direccion r0+r2. Como el dato que se carga es un tipo char, se utiliza la instruccion ldrb para cargar solo la longitud de un byte de memoria en el registro.

cmp r0, #0

bne .if\_fv\_arm

Por ultimo para concluir la instruccion if, se compara el resultado cargado con CASILLA\_VACIA con la instruccion CMP y en caso de no ser igual (cumple la condicion de if), se jecuta la instruccion de salto B con el condicional NE (si no es igual) a la etiqueta if\_fv\_arm que se corresponde con el codigo del cuerpo de if. En caso contrario, si no se cumple la condicion de if, se entra directamente el cuerpo correspondiente al else por estar el codigo de este seguido.

mov r0, #1 \*\*\*\*\* REVISAR CODIGO IMPORTANTE\*\*\*\*\*

str r0, [r3, #0]

La funcion ficha\_valida solo devuelve un tipo de dato entero (int) por lo que lo devolvera en el registro 0. Como tambien modifica la variable posicion\_valida que se a pasado como parametro por referencia (direccion en memoria), habra que escribir su nuevo valor en memoria.

En el cuerpo de if, se actualiza posicion valida con el valor de ??????. Por eso se mueve a r0 dicho valor y se escribe en la direccion de posicion\_valida, hasta ahora contenida en r3. Se tiene que devolver el valor tablero[f][c], que es el que ya contiene en r0.

mov r0, #0 \*\*\*\*\* REVISAR CODIGO IMPORTANTE\*\*\*\*\*

str r0, [r3, #0]

En el caso de else, debe asignar 0 a la variable posicion\_valida y devolver tambien 0 (CASILLA\_VAICA). Se carga por ello el valor 0 en r0, se escribe en la direccion de posicion\_valida y al estar en r0 es el valor que devolvera la funcion.

BX r14

Por ultimo, tanto en if como en else, para volver a la direccion desde la que se invovo a la funcion (contenida en LR), se ejecuta la anterior instruccion. La instruccion BX cambia tambien el estado entre thumb o ARM, en caso de que la funcion hubiese sido invocada desde thumb.

4.1 ficha valida thumb

En cuanto a planteamiento, la funcion ficha\_valida\_thumb es similar a ficha\_valida\_arm salvo por algunas diferencias que se comentan a continuacion:

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* push \*\*)

thumb no permite la comparacion con inmediatos por lo que hemos tenido que utilizar un registro extra. EL registro r4, al ser para variables privadas, se debe guardar en la pila antes de modificarlo y se debe restaurar al termianr.

MOV r4, #(\*\*\*\*\*\*\*\*\*\*\*DIM-1)

Se carga en r4, DIM-1 para poder usarlo en la instruccion CMP-

BHI .Elsethumb

En caso de ser mayor sin signo (como en ficha\_valida\_arm), se salta a la etiqueta correspondiente con el codigo de else.

lsl r4, r1, #3

Como no se puede aplicar el desplazamiento junto a otra instruccion, como en la funcion anterior, se utiliza la funcion de thumb lsl que realiza dicha operacion.

add r0, r0, r4

ldrb r4, [r0, r2]

cmp r4, #0

beq .Elsethumb

Se carga el valor y se compara igual que en la funcion anterior. En caso de no cumplirse la condicion tablero[f][c] salta al codigo de else con beq. En caso contrario ejecuta el codigo de if por estar seguido.

Las operaciones que se realizan dentro de if y else son las mismas que en la funcion anterior.

(\*\*\*\*\*\*\*\*\*\*\*pop)

BX r14

Al terminar la funcion, se desapila el registro r4 y se vuelve a la direccion invocante(r14/lr) cambiando el estado de thumb a ARM si se invocase desde ARM.

4.1 patron volteo arm

La funcion recibe un total de 7 parametros. Como solo se pueden utilizar los registro de r0 a r3 para parametros, recibe los 3 restantes en la pila. Los parametros que recibira son: r0:@tablero, r1:@longitud, r3:char FA, r4:char FC, pila: char SF, char SC, char color.

( \*\*\*\*\*\*\*\*\*\*\* MARCO PILA \*\*\*\*\*\*\*\*\*\*\*\*)

SUB sp, sp, #4

Se guarda espacio en la pila para la variable local posicion\_valida.

MOV r7, r1

MOV r8, r0

Patron\_volteo va a tener que invocar a ficha\_valida. Los parametros que recibe ficha valida son: r0:@tablero, r1:f, r2:c, r3:@posicion\_valida. Es por eso que tablero y longitud se guardan en r8 y r7 respectivamente para utilizarse luego.

( \*\*\*\*\*\*\*\*\*\*\* AND)

SUB r3, fp, #32

Se almacena en r3 la direccion de la variable local posicion\_valida para la que se reservo espacio al principio. Esta direccion se pasara como parametro a ficha\_valida (posicion\_valida se pasa por referencia). Para acceder a la variable local se resta a fp el numero de elementos apilados al principio\*4.

MOV r6, r1

MOV r9, r2

Las variables FA y FC se podrian utilizar luego en la funcion asi que salvan en los registro r6 y r9.

BL ficha\_valida\_arm

Se llama a la funcion ficha\_valida con la instruccion BL. En el caso de llamar a ficha\_valida\_thumb, como tendria que cambiar de estado, se debe llamar con la instruccion BX.

MOV r3, r9

MOV r2, r6

MOV r1, r7

LDR r6, [fp, #-32]

Se restauran los parametros salvados (tablero no porque todavia no es necesario) y se carga en r6 el valor que ha escrito ficha\_valida en la direccion de posicion\_valida.

CMP r6, #1

BNE else\_pv

Para evaluar el condicional if, primero se comprueba la condicion de si posicion\_valida es igual a (\*\*\*\*\*\*\*\*). En caso de que no lo sea se salta con la iinstruccion BNE a la parte del codigo correspondiente a else.

LDRB r6, [fp, #12] // cargamos color de la pila

LDR r7, [r1] // se carga \*longitud (se utiliza despues)

CMP r6, r0 // IF color==casilla

BEQ elseif\_pv

Una vez comprobado lo anterior, si no se ha relaizado el salto, se carga el ultimo parametro de la pila y se compara con casilla (resultado de ficha\_valida en r0). En caso de ser iguales, entra dentro de la condicion else if (la estructura condicional del codigo en c consta de if, else if y else). En caso contrario, el codigo entra al codigo de if por estar seguido.

ADD r7, r7, #1

STR r7, [r1]

MOV r0, r8 // tablero a r0 para llamar a patron\_volteo

STMFD sp!, {r4-r6} // apilamos sf, sc, color

BL patron\_volteo // patron\_voleto()

ADD sp, sp, #12 // desapilamos los parametros

Dentro del codigo de if, se aumenta el valor de longitud en 1. Como guardamos lingitud mediante su direccion es necesario escribir en memoria el nuevo valor. Despues de esto, se mueve tablero a r0 y se apilan SF, SC y color porque se va a llamar de forma recursiva a patron volteo. Una vez termina la funcion, se desapilan los 3 parametros apilados (3\*4).

La funcion patron\_volteo devuelve un entero (en r0). En el caso de if, devolvera el mismo resultado que devuelve la llamada recursiva a patron\_volteo, valor que ya se encuentra en r0.

CMP r7, #0 // devuelve el resulado segun el valor de longitud

MOVGT r0, #1

MOVLE r0, #0

En el caso de else if, se compara longitud con 0, y en funcion del resultado, patron volteo devuelve en r0 el valor 0 o 1.

MOV r0 #0

En el caso de else, se devuelve 0 en r0.

(\*\*\*\*\*\*\*\*\* DESAPILAR)

4.3 Optimizacion

4.3.1 ficha valida arm

ficha\_valida\_arm

cmp r2, \*\*\*\*\*\*\*\*\*\*\*\*\*

cmpls r1, \*\*\*\*\*\*\*\*\*\*\*\*\*

bhi .else\_fv\_arm

Para evitar la comparacion con 8 y luego con 0, se aprovechan los predicados condicionales sin signo para comprobar a la vez que sea mayor que 0 y menor que 8. Los predicados se utilizan en la segunda y tercera instruccion que solo se evaluan si se cumple la condicion.

add r0, r0, r1, lsl #3 // multiplica por 8 (DIM)

ldrb r0, [r0, r2]

En lugar de utilizar una instrucicon para la multiplicacion, como la dimension del vector es potencia de 2 (8), se utiliza el operador de desplazamiento para la multiplicacion. Ademas mediante el segundo operador (offset) de la instruccion ldr nos ahorramos una instruccion add.

Situando el cuerpo dentro de else seguido de las instrucciones de condicion, nos ahorramos una instruccion de salto, pues si se cumplen las condiciones no saltara y entrara en esta parte del codigo.

Para minimizar el uso de registros, se utilizan los registros r0-r3 para las operaciones, evitando asi tambien tener que guardar los registros de variables privadas en memoria.

(\*\*\*\*\*\*\*\*\*\*\* revisar else)

4.3.2 ficha valida thumb

mov r4, #7

cmp r4, r2 // IF c > DIM

bcc .Elsethumb

cmp r4, r1 // IF f > DIM

bcc .Elsethumb (\*\*\*\*\*\*\*\*\*\* hi)

El planteamiento de la comparacion es igual que en la funcion anterior, pero como la instruccion CMP en thumb no tiene operador condicional, es necesario ejecutar los 2 condicionales y en lugar de 2 instrucciones CMP y una B como en el caso anterior, tenemos 2 CMP y 2 B.

lsl r4, r1, #3 // f \* 8(DIM)

Como en la funcion anterior, se ha utilizado el operador de desplazamiento para la multiplicacion con la diferencia de que en thumb no se ha podido integrar en la instruccion ADD.

Tanto en ficha\_valida\_arm como en la funcion thumb, cargar de memoria el valor de tablero[f][c] se ha dejado para el final de la condicion, de forma que si no cumplen las 4 condiciones anteriores no es ncesario cargar de memoria el dato.

4.3.3 ficha valida arm

CMP r7, #0 // devuelve el resulado segun el valor de longitud

MOVGT r0, #(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

MOVLE r0, #0

Se puede destacar el uso de condicionales para evaluar el valor de longitud.

CMP r6, #1

BNE else\_pv

LDRB r6, [fp, #12] // cargamos color de la pila

LDR r7, [r1] // se carga \*longitud (se utiliza despues)

CMP r6, r0 // IF color==casilla

BEQ elseif\_pv

La evaluacion de la condicion if se divide en dos partes. Primero se comprueba el valor de posicion\_valida y posteriormente se compara casilla con color para determinar si entra en el bloque if o en el bloque else if. Como en los 2 bloques de codigo anteriores se utiliza el valor de longitud, se carga antes de evaluar la condicion para evitar cargar el valor posteriormente en 2 partes del codigo.

4.4 Problemas encontrados

A continuacion se enumeran algunos de los principales problemas que han sirgido durante el desarrollo del proyecto:

- Mala creacion del bloque de pila que resultaba en accesos incorrectos a los parametros.

- Errores que surgen solo en la depuracion del codigo optimizado. Esto obliga a la depuracion paso por paso de todo el codigo.

- Apilar y desapilar un numero diferente de recursos que resulta en errores de ejecucion.

- Errores entre las llamadas de arm al codigo c.

- El planteamiento del parametro por referencia posicion\_valida.

- Incorrecta traduccion del codigo C a ARM o THUMB

- Suma de tipos char

4.5 Reparto del trabajo