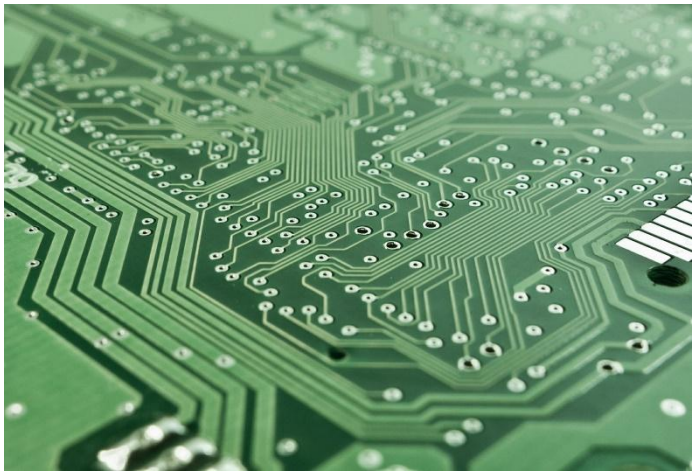
DESARROLLO DE CÓDIGO PARA EL PROCESADOR ARM

Proyecto Hardware: Práctica 1

Octubre 2019



Autores: Irene Fumanal Lacoma 758325
Sergio García Esteban 755844

Contenido

1. Resumen ejecutivo

El objetivo principal de esta práctica era mejorar el rendimiento del juego Reversi, optimizando las funciones más costosas.

Antes de empezar a diseñar el código optimizado tuvimos que familiarizarnos con el material de trabajo, es decir, con la placa Embest S3CEV40 y el entorno de desarrollo Eclipse. Dicha tarea es lo que más tiempo nos ha llevado en esta primera práctica. Además, tuvimos que estudiar y comprender correctamente la arquitectura ARM y el todo el código proporcionado.

Comenzamos la optimización realizando la función **patron_volteo_arm_c()** equivalente a **patron_volteo()**, manteniendo la llamada a la función proporcionada en C **ficha_valida()**. Mejoramos el rendimiento de la original utilizando más registros y cargando los valores más utilizados, evitando de esta forma, operaciones de lectura y escritura en memoria.

La siguiente tarea fue realizar la función **patron_volteo_arm_arm()**, pero ahora eliminando la llamada a la subrutina **ficha_valida()**, realizando el código de ella en ensamblador para incluirla directamente junto con patron_volteo(), ya que solo es llamada por esta.

Para verificar el correcto funcionamiento de las dos funciones anteriores, realizamos otra llamada **patron_volteo_test()** cuyo objetivo era de forma automática verificar que los resultados que devuelven, coinciden con el de la función original **patron_volteo().**

Para poder medir los tiempos, desarrollamos una biblioteca con 4 funciones principales para trabajar con el timer2. Los tiempos obtenidos y el tamaño del código de cada una de las funciones nos han servido como medidas de rendimiento y comparación.

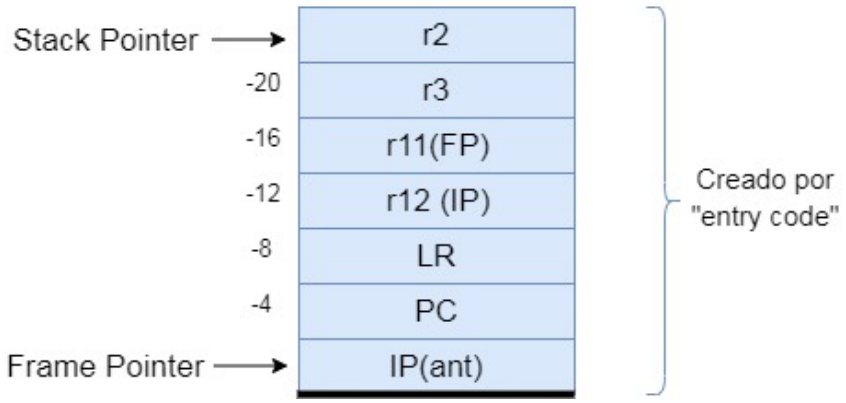
Finalmente, estudiamos las diferentes optimizaciones del compilador (-O0, -O1, -O2, -O3, -Os) y el impacto en el rendimiento de nuestro código.

2. Marcos de pila

En este apartado, mostramos los marcos de pila correspondientes a las funciones y su descripción.

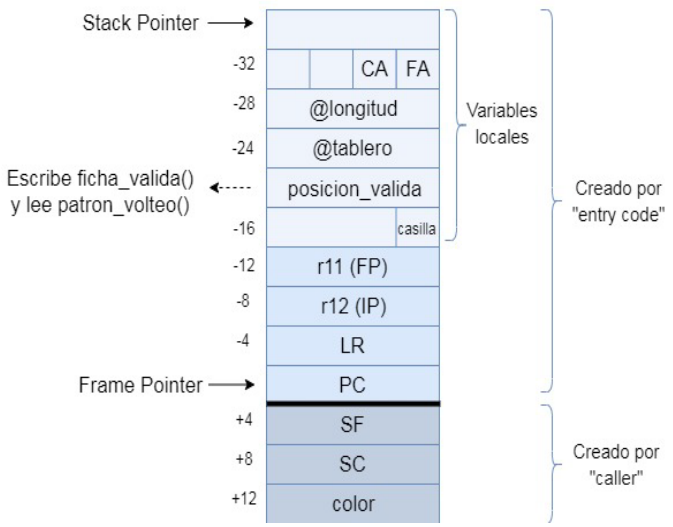
Timer0

En el prólogo de la rutina de excepción, se apila r11-r14 como siempre, pero antes se apila el IP anterior para luego recuperarlo, y se apilan los registros utilizados dentro de la rutina, r2-r3 en este caso.



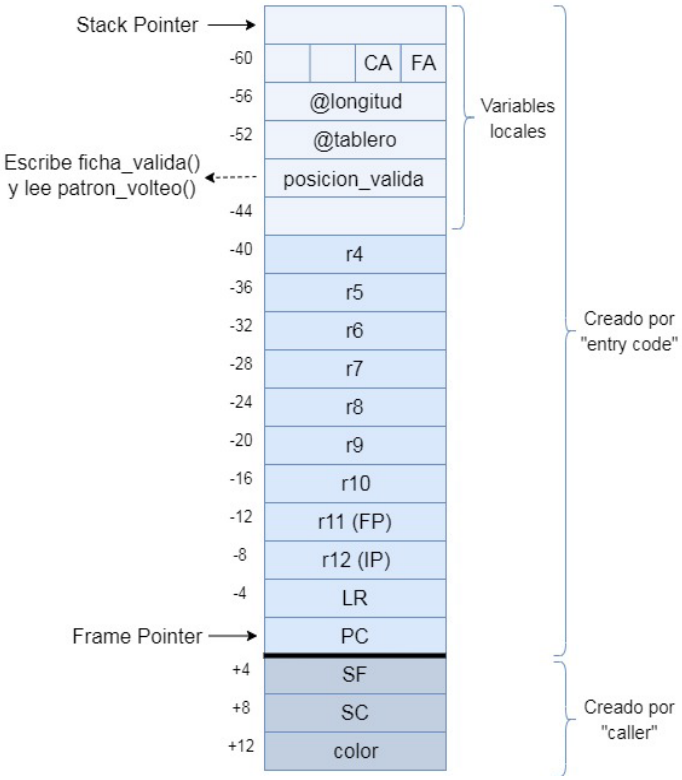
Patron_volteo

El siguiente dibujo muestra el marco de pila de la función patron_volteo() original. Observamos que no hay apilado ningún registro r4-r10, ya que no han sido apilados en el prólogo de la subrutina porque no los utiliza. Todos los valores que calcula los almacena en la pila y cada vez que quiere conocer el valor de uno, lee directamente de memoria.



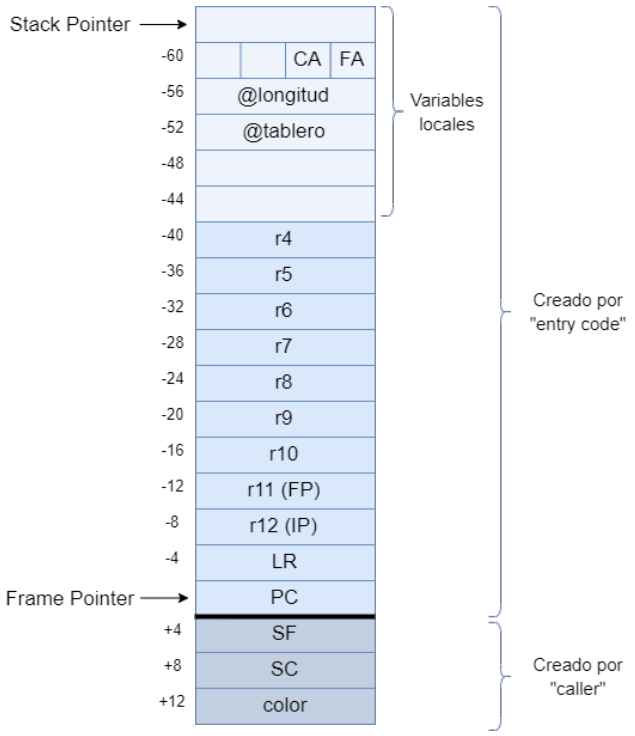
Patron_volteo_arm_c

En el diseño de nuestra función, a diferencia de la original, utilizamos los registros r4-r10 para trabajar y no realizar tantos accesos a memoria. Por este motivo, dichos registros se apilan en el prólogo de la subrutina. Además, como nos evitamos varios accesos a memoria, el resultado casilla de ficha_valida() no lo cargamos en pila.



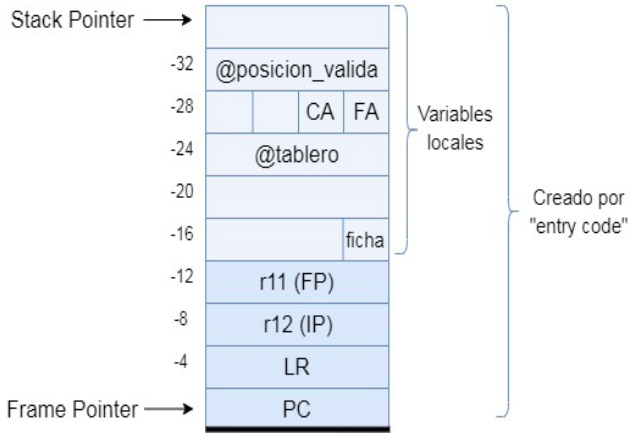
Patron_volteo_arm_arm

El marco de pila de esta función es muy parecido al anterior. La única diferencia es que en la anterior, pasábamos a ficha_valida() una dirección de la pila para que almacenase ahí el resultado de posicion_valida. En esta función, utilizamos los registros r4-r10 para trabajar.



Ficha_valida

En el marco de pila de ficha_valida() observamos que no apila los registros r4-r10 porque no los utiliza. Reserva espacio para las variables globales.



3. Código fuente

```
/****************************************
* r0 = @tablero || longitud     = parametro 0 (r0)
* r1 = FA                       = parametro 2 (r2)
* r2 = CA                       = parametro 3 (r3)
* r3 = @posicion_valida
* r4 = SF                       = parametro 4 (fp+4)
* r5 = SC                       = parametro 5 (fp+8)
* r6 = @longitud || longitud    = parametro 1 (r1)
* r7 = @tablero
* r8 = FA
* r9 = CA
* r10 = color                   = parametro 6 (fp+12)
****************************************/

patron_volteo_arm_c:

        mov r12, sp //prologo
        push {r4-r10,r11, r12, lr, pc}
        sub r11, r12, #4
        sub sp, sp, #24     //24 bytes variables locales

        str r0, [r11, #-52]     //parametro 0 = @tablero
        str r1, [r11, #-56]     //parametro 1 = @longitud
        strb r2, [r11, #-57]    //parametro 2 = FA
        strb r3, [r11, #-58]    //parametro 3 = CA


        mov r6,r1           //r6=@longitud
        ldrb r4, [r11, #4]  //parametro 4 = SF
        add r1,r2,r4        //FA=FA+SF
        bic r1,r1,#256      //convertir a char la suma
        ldrb r5, [r11, #8]  //parametro 5 = SC
        add r2,r3,r5        //CA=CA+SC
        bic r2,r2,#256

        //casilla = ficha_valida(tablero, FA, CA, &posicion_valida);
        mov r8,r1       // r8=FA
        mov r9,r2       // r9=CA
        sub r3,r11,#48  // r3=@posicion_valida
        mov r7,r0       // r7=@tablero
        bl ficha_valida //llamada a c
```

```
        ldr r10, [r11, #12] //parametro 6 = color
        b cond //while ((posicion_valida == 1) && (casilla != color))

while_: add r1,r8,r4    //FA=FA+SF
        bic r1,r1,#256
        add r2,r9,r5    //CA=CA+SC
        bic r2,r2,#256
        ldr r0,[r6]     //*longitud = *longitud + 1;
        add r0,r0,#1
        str r0,[r6]

        //casilla = ficha_valida(tablero, FA, CA, &posicion_valida);
        mov r8,r1       // r8=FA
        mov r9,r2       // r9=CA
        sub r3,r11,#48  // r3=@posicion_valida
        mov r0,r7       // r7=@tablero
        bl ficha_valida //llamada a c

cond:   ldr r3, [r11, #-48] //r3=posicion_valida
        cmp r3,#1           //while ((posicion_valida == 1) &&
        bne if_
        cmp r0,r10          //(casilla != color))
        bne while_


if_:    cmp r3,#1   //if ((posicion_valida == 1) &&
        bne else_
        cmp r0,r10  //(casilla == color) &&
        bne else_
        ldr r6,[r6]
        cmp r6,#0   //(*longitud >0))
        ble else_

        mov r0,#1   //return PATRON_ENCONTRADO;
        b fin

else_:  mov r0,#0   //return NO_HAY_PATRON;

fin:    sub sp, r11, #40//epilogo
        ldm sp, {r4-r10,r11, sp, lr}
        bx lr
```

```
/****************************************
* r0 = @tablero              = parametro 0 (r0)
* r1 = @longitud             = parametro 1 (r1)
* r2 = FA                    = parametro 2 (r2)
* r3 = CA                    = parametro 3 (r3)
* r4 = posicion_valida
* r5 = SF                    = parametro 4 (fp+4)
* r6 = SC                    = parametro 5 (fp+8)
* r7 = casilla
* r8 = color                 = parametro 6 (fp+12)
* r9 = tablero[FA][CA]
* r10 = FA*8 || longitud
****************************************/

patron_volteo_arm_arm:

        mov r12, sp //prologo
        push {r4-r10,r11, r12, lr, pc}
        sub r11, r12, #4
        sub sp, sp, #24     //24 bytes variables locales

        str r0, [r11, #-52]     //parametro 0 = @tablero
        str r1, [r11, #-56]     //parametro 1 = @longitud
        strb r2, [r11, #-57]    //parametro 2 = FA
        strb r3, [r11, #-58]    //parametro 3 = CA


        ldrb r5, [r11, #4]  //parametro 4 = SF
        add r2,r2,r5        //FA=FA+SF
        bic r2,r2,#256      //convertir a char la suma
        ldrb r6, [r11, #8]  //parametro 5 = SC
        add r3,r3,r6        //CA=CA+SC
        bic r3,r3,#256

        ldr r8, [r11, #12]  //parametro 6 = color

        //ficha_valida
        cmp r2,#7       //if((0<=FA<DIM) &&
        bhi fv1_e
        cmp r3,#7       //(0<=CA<DIM) &&
        bhi fv1_e
        lsl r10,r2,#3
        add r9,r0,r10
        ldrb r7,[r9,r3]
        cmp r7,#0       //(tablero[FA][CA]!=0=casilla))
        beq fv1_e
        mov r4,#1       //pos_v=1
        b fv1_f
fv1_e:  mov r4,#0       //pos_v=0
```

```
fv1_f:  b cond  //while ((posicion_valida == 1) && (casilla != color))

while_: add r2,r2,r5    //FA=FA+SF
        bic r2,r2,#256
        add r3,r3,r6    //CA=CA+SC
        bic r3,r3,#256
        ldr r10,[r1]    //*longitud = *longitud + 1;
        add r10,r10,#1
        str r10,[r1]

        //ficha_valida
        cmp r2,#7       //if((0<=FA<DIM) &&
        bhi fv2_e
        cmp r3,#7       //(0<=CA<DIM) &&
        bhi fv2_e
        lsl r10,r2,#3
        add r9,r0,r10
        ldrb r7,[r9,r3]
        cmp r7,#0       //(tablero[FA][CA]!=0=casilla))
        beq fv2_e
        mov r4,#1       //pos_v=1
        b cond
fv2_e:  mov r4,#0       //pos_v=0

cond:   cmp r4,#1   //while((posicion_valida == 1) &&
        bne if_
        cmp r7,r8   //(casilla != color))
        bne while_


if_:    cmp r4,#1   //if ((posicion_valida == 1) &&
        bne else_
        cmp r7,r8   //(casilla == color) &&
        bne else_
        ldr r1,[r1]
        cmp r1,#0   //(*longitud >0))
        ble else_

        mov r0,#1   //return PATRON_ENCONTRADO;
        b fin

else_:  mov r0,#0   //return NO_HAY_PATRON;

fin:    sub sp, r11, #40 //epilogo
        ldm sp, {r4-r10,r11, sp, lr}
        bx lr
```

4. Descripción de las optimizaciones

A la hora de implementar el código en ensamblador hemos utilizado diferentes técnicas o directrices para optimizarlo.

Para reducir las operaciones de lectura y escritura en memoria, guardamos los antiguos valores de r4-r10 en la pila y utilizamos estos registros para gestionar y almacenar nuestras variables sin tener que guardarlas en memoria, al final de la subrutina restauraremos los antiguos valores de los registros.

```
push {r4-r10,r11, r12, lr, pc}
```

Para mover más de un dato entre memoria y registros utilizamos instrucciones de transferencia múltiple como LDM, STM, PUSH o POP.

```
ldm sp, {r4-r10,r11, sp, lr}
```

Una forma de multiplicar en ARM es utilizar el *barrel shifter* y lo utilizamos para calcular la dirección de *tablero[FA][CA]* .

```
lsl r10,r2,#3   //r10=FA*8
add r9,r0,r10   //r9=@tablero[FA][0]
ldrb r7,[r9,r3] //r7=tablero[FA][CA]
```

5. Resultados de la comparación entre versiones

En este apartado reflejaremos los resultados obtenidos al realizar la medición de tiempos de las tres funciones de patron_volteo, con los distintos niveles de optimización del compilador. Además, incluimos el tamaño del código en bytes de cada una de las funciones mencionadas.

Tiempos

Para evaluar el tiempo de ejecución de patron_volteo, patron_volteo_arm_c y patron_volteo_arm_arm al realizar el cálculo de ficha negra en fila 2 y columna 3, hemos realizado dos mediciones distintas.

En la primera, mediante un bucle, obtenemos el tiempo total que tarda en realizar el cálculo 8 veces, es decir, en todas las direcciones posibles reflejadas en los vectores SF y SC. Para ello, inicializamos el timer antes del bucle y lo leemos después de este. En la siguiente tabla vemos reflejados los resultados obtenidos con cada optimización del compilador. El tiempo está en **microsegundos.**

	patron_volteo	**patron_volteo_arm_c**	**patron_volteo_arm_arm**
-O0	232	215	138
-O1	117	119	102
-O2	117	120	102
-O3	59	115	102
-Os	119	126	109

En la segunda, medimos las ocho direcciones mencionadas, pero individualmente. Efectuamos un bucle igualmente, pero hacemos una medición cada iteración. En las siguientes tablas mostramos las mediciones de cada iteración del bucle para cada función y en cada optimización del compilador.

-O0

	0	1	2	3	4	5	6	7
patron_volteo	26	26	26	30	48	26	26	26
patron_volteo_arm_c	25	25	25	27	40	25	25	25
patron_volteo_arm_arm	17	17	17	18	22	17	17	17

-O1

	0	1	2	3	4	5	6	7
patron_volteo	14	14	14	14	20	14	14	14
patron_volteo_arm_c	14	14	14	15	21	14	14	14
patron_volteo_arm_arm	12	12	12	13	17	12	12	12

-O2

	0	1	2	3	4	5	6	7
patron_volteo	14	13	14	20	14	14	14	14
patron_volteo_arm_c	14	14	14	15	21	14	14	14
patron_volteo_arm_arm	12	12	12	13	17	12	12	12

-O3

	0	1	2	3	4	5	6	7
patron_volteo	6	7	6	6	10	6	7	6
patron_volteo_arm_c	13	14	13	15	20	13	13	14
patron_volteo_arm_arm	12	12	12	13	17	12	12	12

-Os

	0	1	2	3	4	5	6	7
patron_volteo	14	14	14	15	20	14	14	14
patron_volteo_arm_c	15	15	15	15	21	15	15	15
patron_volteo_arm_arm	13	13	13	14	17	13	13	13

Tamaño del código en Bytes.

La siguiente tabla muestra el tamaño en bytes de cada una de las funciones según el tipo de optimización del ensamblador. Para patron_volteo y patron_volteo_c, hemos obtenido las medidas sin sumar el tamaño de ficha_valida, que también mostramos en la tabla.

	-O0	-O1	-O2	-O3	-Os
ficha_valida	180	56	56	56	52
patron_volteo	276	240	200	256	196
patron_volteo_arm_c	208	208	208	256	208
patron_volteo_arm_arm	256	256	256	200	256

6. Conclusiones

Al realizar esta práctica hemos conseguido en primer lugar, familiarizarnos con el entorno de trabajo y la placa *Embest S3CEV40.* Hemos aprendido a ejecutar y depurar sobre la placa y también sin la placa.

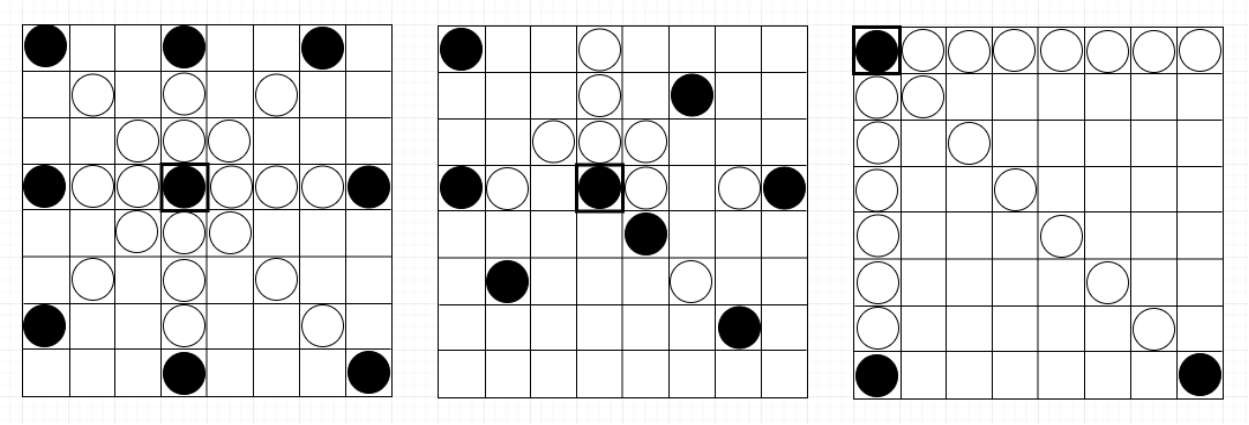
Hemos desarrollado código en ensamblador y utilizado diferentes técnicas para optimizarlo. Para ello hemos profundizado en la interacción C / Ensamblador y en las opciones de optimización del compilador. También hemos estudiado el *ARM Application Procedure Call Standard*.

	patron_volteo	**patron_volteo_arm_c**	**patron_volteo_arm_arm**
-O0	232	215	138
-O1	117	119	102
-O2	117	120	102
-O3	59	115	102
-Os	119	126	109

Estos son los resultados obtenidos del rendimiento de cada función en microsegundos. La función *patron_volteo_arm_arm()* está mejor optimizada que las generadas por el compilador, a excepción de la generada con -O3, que utiliza mejores técnicas.

Por último, para medir los resultados hemos aprendido a gestionar la entrada/salida con un periférico, en concreto con los temporizadores internos de la placa, escribiendo en sus registros de configuración y desarrollando en C las rutinas de tratamiento de interrupción.

Anexo I. Banco de pruebas



Para automatizar el proceso de verificación de las funciones desarrolladas, hemos implementado una función con los mismos parámetros que patron_volteo(), llamada patron_volteo_test(), que se encarga de invocar a las tres versiones y comprobar que tanto el resultado como el valor de *longitud* coincide, y de no ser así, detiene la ejecución.

Hemos diseñado tres tableros para asegurarnos de comprobar todos los casos posibles, ejecutando la función en las 8 direcciones posibles para cada tablero.

```
for (i = 0; i < DIM; i++){
    lon=0;
    SF = vSF[i];
    SC = vSC[i];
    patron_volteo_test(t_1,&lon,f,c,SF,SC
}
for (i = 0; i < DIM; i++){
    lon=0;
    SF = vSF[i];
    SC = vSC[i];
    patron_volteo_test(t_2,&lon,f,c,SF,SC
}
f=0;
c=0;
for (i = 0; i < DIM; i++){
    lon=0;
    SF = vSF[i];
    SC = vSC[i];
    patron_volteo_test(t_3,&lon,f,c,SF,SC
}
```

```
int patron_volteo_test(char tablero[][DIM],
{
    int a1,a2,a3;
    int c1=*longitud;
    int c2=*longitud;
    int c3=*longitud;
    int *b1=&c1;
    int *b2=&c2;
    int *b3=&c3;
    a1=patron_volteo(tablero,b1,FA,CA,SF,SC
    a2=patron_volteo_arm_c(tablero,b2,FA,CA
    a3=patron_volteo_arm_arm(tablero,b3,FA,(
    if(a1!=a2||a1!=a3||*b1!=*b2||*b1!=*b2){
        while(1);
    }
    return a3;
}
```

Anexo II. Timer2

En esta práctica hemos tenido que configurar el timer2 de la placa, con el objetivo de medir tiempos.

El objetivo era que el timer tuviese la máxima precisión posible.

$$timer2\_freq\ (MHz) = CPU\_freq\ (MHz)\ /\ preescaler + 1\ /\ divider$$

Para ello configuramos *preescaler* y *divider* con sus mínimos valores respectivamente (preescaler=0 & divider=2).

Y para conseguir que provoque el menor número de interrupciones configuramos el valor inicial de la cuenta (rTCNTB2) al máximo valor (0xFFFF) y el valor de comparación de la cuenta (rTCMPB2) al mínimo valor (0).

Para convertir los ticks registrados en el timer a tiempo utilizamos la siguiente fórmula:

$$tiempo\ (us) = ticks * timer\_freq\ (MHz)$$

Para calcular el numero de ticks nos ayudamos de una variable global.

$$ticks = 0xFFFF * timer2\_num\_int + 0xFFFF - rTCNTO2$$

Configuramos el timer con *auto-reload* activado para que se reinicie la cuenta automáticamente cuando llega al valor de comparación, y cada vez que iniciamos el timer activamos y desactivamos *manual-update* para que se reinicie la cuenta.

```
rTCON &=  ~(0xF<<12);// 4 bits timer2 (12-15) a 0
rTCON |=   (0x2<<12);// manual-update a 1
rTCON &=  ~(0x2<<12);// manual-update a 0
rTCON |=   (0x8<<12);// auto-reload a 1
rTCON |=   (0x1<<12);// start a 1
```

Anexo III. Mejoras post-entrega

En este apartado comentamos los cambios que hemos realizado, aunque fuera de tiempo, en nuestro proyecto después de la evaluación presencial en el laboratorio.

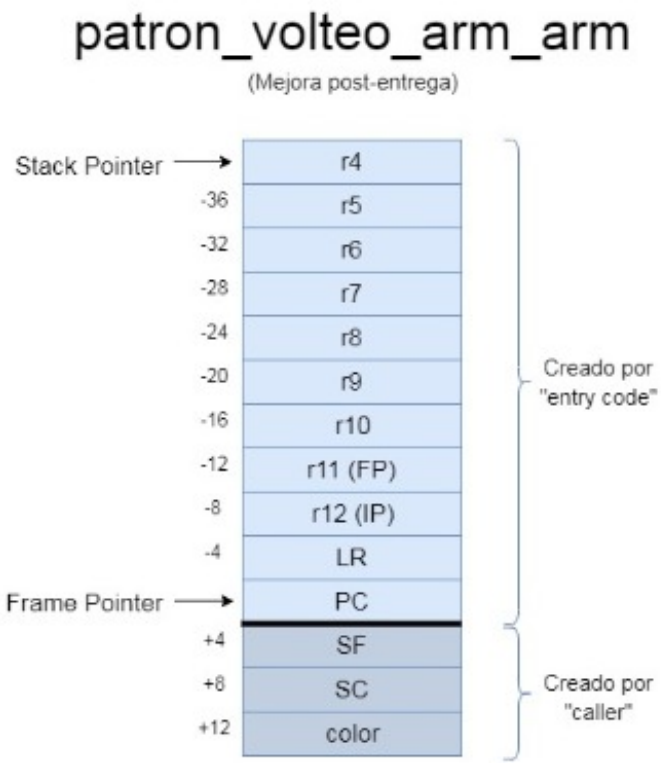
Dejamos de utilizar variables locales en patron_volteo_arm_arm().

Antes:

```
mov r12, sp      //prologo
push {r4-r10,r11, r12, lr, pc}
sub r11, r12, #4
sub sp, sp, #24
str r0, [r11, #-52]
str r1, [r11, #-56]
strb r2, [r11, #-57]
strb r3, [r11, #-58]
```

Después:

```
mov r12, sp      //prologo
push {r4-r10,r11, r12, lr, pc}
sub r11, r12, #4
```



Empleamos las instrucciones condicionales para optimizar código.

Antes:

```
        cmp r1,#0    //(*longitud >0))
        ble else_
        mov r0,#1    //return PATRON_ENCONTRADO;
        b fin
else_:  mov r0,#0    //return NO_HAY_PATRON;
```

Después:

```
        mov r0,#0    //return NO_HAY_PATRON;
        cmp r1,#0    //(*longitud >0))
        movgt r0,#1  //return PATRON_ENCONTRADO;
```

Antes:

```
        cmp r7,#0    //tablero[FA][CA]!=0
        beq fv1_e
        mov r4,#1    //pos_v=1
        b fv1_f
fv1_e:  mov r4,#0    //pos_v=0
```

Después:

```
        mov r4,#0    //pos_v=1
        cmp r7,#0    //tablero[FA][CA]!=0
        movne r4,#1  //pos_v=1
```