Anexos

Apéndice A

LLVM Intermediate Representation

En este anexo se habla del código intermedio LLVM Intermediate Representation (IR) que es lenguaje que se ha analizado a lo largo del proyecto. Poniendo especial énfasis en las instrucciones analizadas para los objetivos del mismo.

Primeramente se comenta una visión general del juego de instrucciones y de los tipos que se definen en el lenguaje. Luego se expone claramente una parte fundamental del control de flujo del lenguaje, la forma Static Single Assignment (SSA) y la función Φ, parte fundamental para calcular el número de vueltas de los bucles y las variables de iteración. A continuación se explican las instrucciones más analizadas en las bibliotecas ya que gracias a ellas se pueden observar los reúsos: las instrucciones load, store y getelementptr. Finalmente se muestra un sencillo ejemplo de código LLVM IR comentado de un bloque que pertenece a un bucle y donde se realizan dos stores, explicando lo que realiza cada instrucción.

A.1. Visión general del juego de instrucciones

El juego de instruciones de LLVM está diseñado como una representación de bajo nivel pero con información de tipos de alto nivel. Provee información de todos los valores del programa y posiciones de memoria. LLVM IR consta de veintiocho instrucciones, numerosos tipos primarios y cuatro tipos derivados.

Aritmética	add, sub, mul, div, rem
Lógica	and, or, xor, shl, shr
Comparación	seteq, setne, setlt, setgt, setle, setge
Control de flujo	ret, br, mbr, invoke, unwind
Memoria	load, store, getelementpt
Otros	cast, call, phi

El sistema de tipos consiste en tipos primarios con tamaño predefinido (ubyte, uint, float, double, etc...) y cuatro tipos derivados (pointer, array, structure, function).

A.2. Tipos primarios y derivados

LLVM es una representación estrictamente tipada, en el que cada valor SSA y cada ubicación de memoria tiene una tipo asociado. Todas las operaciones obedecen a reglas estrictas de tipo.

El sistema de tipos LLVM incluye tipos primitivos (void, booleanos, enteros con y sin signo de 8 a 64 bits, valores de coma flotante de simple y doble precisión) y derivados (punteros, arrays, estructuras y funciones). Estos tipos son independientes del lenguaje de representación de datos que se asignan desde los tipos de lenguaje de alto nivel.

Se va a explicar cada uno de los tipos derivados más en detalle para su mejor comprensión:

Tipo Puntero

Sintaxis:

```
<tipo> *
```

Nota: No están permitidos punteros a void.

Ejemplos:

```
[4 x i32]* Puntero a un array de 4 enteros
i32 (i32*)* Puntero a una función que toma un puntero a un entero
            y devuelve un entero
```

Tipo Array

Sintaxis:

```
[<# elementos> x <tipoelemento>]
```

Ejemplos:

```
[40 x i32] Array de 40 enteros de 32 bits.
[4 x i8] Array de 4 enteros de 8 bits.
[3 x [4 x i32]] Array de 3x4 de enteros de 32 bits.
```

Tipo Estructura

Sintaxis:

```
%T1 = type { <lista de tipos> }
```

Ejemplos:

```
{ i32, i32, i32 } 3 enteros de 32 bits
{ int, [10 x [20 x int]], int} 1 entero, 1 array de 10x20 enteros, 1 entero
```

Tipo Función

Sintaxis:

```
<tipo de retorno> (<lista de parametros>)
```

Ejemplos:

```
i32 (i32) Función que coge un entero y retorna un entero
float(i16,i32*) Función que toma un entero de 16 bits y un puntero a un entero
                y devuelve un float
```

A.3. SSA form (PHINODE)

LLVM usa Static Single Assignment (SSA) form, forma de asignación estática individual. La forma SSA se basa en la premisa de que cada variable es asignada únicamente en una parte del programa, y si se producen otras asignaciones a la misma variable, se crean nuevas versiones de ésta.

A veces no es posible determinar cuál es la última asignación realizada, como resultado de ramificaciones y bucles. Para solventarlo, la forma SSA introduce un nuevo tipo de operación llamada función Φ, que une varias versiones de una variable y genera una variable nueva.

En esencia, lo que permite la forma SSA es unir cada uso de una variable en el programa a su correspondiente y única definición, lo que permite implementaciones muy eficientes de análisis y transformaciones de optimización, porque simplifica enormemente las cadenas de uso-definición de variables, y en consecuencia el flujo de datos.

En LLVM la función Φ está definida en la instrucción phi.

Phinode

Sintaxis:

```
result = phi <type> [<val0>, <label0>], ... , [<valN>, <labelN>]
```

A *result* se le asigna el valor *val0* si el control llega a esta instrucción desde el bloque básico de la etiqueta *label0*, *val1* si estamos aquí desde el bloque básico *label1*, y así sucesivamente.

Ejemplos:

```
Loop:       ; Bucle infinito de 0 hacia arriba
%indvar = phi i32 [ 0, %LoopHeader ], [ %nextindvar, %Loop ]
%nextindvar = add i32 %indvar, 1
br label %Loop
; %indvar valdrá %nextindvar en cada vuelta ya que proviene del bloque %Loop
```

```
bb3:       ; Bucle de 0 a 1000
%indvar46 = phi i32 [ %indvar.next47, %bb3 ], [ 0, %bb ]
%indvar.next47 = add i32 %indvar46, 1
%exitcond48 = icmp eq i32 %indvar.next47, 1000
br i1 %exitcond48, label %bb8.preheader, label %bb3
; %indvar46 valdrá %indvar.next47 en cada vuelta ya que proviene del bloque %bb3
```

A.4. Acceso a direcciones de memoria

Para obtener la dirección de un subelemento de una estructura de datos se usa la instrucción *getelementptr*. Sirve para conocer la dirección de memoria de una posición especifica de arrays y estructuras. Esta instrucción sólo realiza el cálculo de la dirección única pero no accede a la memoria.

Podemos encontrarnos con dos casos:

- Dado un puntero a una estructura y un número de campo, la instrucción getelementptr obtiene un puntero al campo.
- Dado un puntero a una matriz y un número de elemento, la instrucción devuelve un puntero al elemento especificado.

Se pueden especificar varios índices en la misma instrucción, por ejemplo, se puede acceder a la posición A[i][j].

El primer argumento siempre es un puntero, el resto de argumentos son los índices del elemento que queremos acceder.

Getelementptr

Sintaxis:

```
<resultado> = getelementptr <pty>* <ptrval>{, <ty> <idx>}*
```

Ejemplos:

```
%scevgep = getelementptr [100 x [100 x i32]]* @A, i32 0, i32 %i.119, i32 %k.016
; Accedemos a %A[%i][%k]
%scevgep53 = getelementptr [5000 x i32]* %Vector1, i32 0, i32 %varBucle.034
; Accedemos a %Vector1[%varBucle]
%scevgep = getelementptr [5000 x [5000 x i32]]* %A, i32 0, i32 0, i32 %i.17
; Accedemos a %A[0][%i]
%scevgep12 = getelementptr [5000 x %struct..0structura]* %estructuras,
                           i32 0, i32 %i.08, i32 1
; Accedemos a %estructuras[%i].campo1
```

A.5. Lectura y escritura en memoria

Para acceder a la memoria en LLVM se usan las instrucciones load y store. Load para leer de una posición de memoria y store para escribir en una posición.

Load

La instrucción *load* sirve para devolver el valor que reside en una posición de memoria. El primer argumento siempre es una dirección de memoria mediante un puntero a dicha dirección. Ese puntero se obtiene mediante la instrucción *getelementptr*. El parámetro opcional *align* especifica la alineación de la operación, es decir, la alineación de la memoria.

Sintaxis:

```
<resultado> = load <ty>* <pointer>[, align <alignment>]
```

Ejemplos:

```
%tmp5 = load i32* %scevgep, align 4
; Cargamos en %tmp5 el valor que se encuentra en la dirección de puntero %scevgep
%tmp13 = load i32* %tmp12, align 4
; Cargamos en %tmp13 el valor que se encuentra en la dirección de puntero %tmp12
```

Store

La instrucción *store* sirve para escribir un valor en una posición de memoria. La instrucción tiene dos argumentos, el primero es el valor que queremos escribir y el segundo es una dirección de memoria mediante un puntero a dicha dirección. Ese puntero se obtiene mediante la instrucción *getelementptr*. El parametro opcional *align* especifica la alineación de la operación, es decir, la alineación de la memoria.

Sintaxis:

```
store <ty> <value>, <ty>* <pointer>[, align <alignment>]
```

Ejemplos:

```
store i32 %tmp3, i32* %scevgep33, align 4
; Escribimos el valor de %tmp3 en la dirección de memoria del puntero %scevgep33
store i32 0, i32* %scevgep35, align 4
; Escribimos el valor 0 en la dirección de memoria del puntero %scevgep35
```

A.6. Ejemplo completo comentado

```
for (varBucle=0;varBucle<Tam;varBucle++)
{
      Vector1[varBucle]=varBucle*5;
      Vector2[varBucle]=varBucle*5;
}
```

Figura A.1: Código ejemplo en C

```
bb:                                               ; preds = %bb, %bb.nph35
%varBucle.034 = phi i32 [ 0, %bb.nph35 ], [ %tmp, %bb ]
; SSA form phinode %varBucle con variable de iteración interna %tmp
%scevgep53 = getelementptr [5000 x i32]* %Vector1, i32 0, i32 %varBucle.034
; Puntero a la posición de memoria de %Vector1[ %varBucle]
%scevgep54 = getelementptr [5000 x i32]* %Vector2, i32 0, i32 %varBucle.034
; Puntero a la posición de memoria de %Vector2[ %varBucle]
%tmp55 = mul i32 %varBucle.034, 5
; %varBucle*5
store i32 %tmp55, i32* %scevgep53, align 4
; Escritura en la dirección %Vector1[%varBucle] el valor de %varBucle*5
store i32 %tmp55, i32* %scevgep54, align 4
; Escritura en la dirección %Vector2[ %varBucle] el valor de %varBucle*5
%tmp = add nsw i32 %varBucle.034, 1
; Incremento de la variable de iteracion del phinode en 1
%exitcond52 = icmp eq i32 %tmp, 5000
; Comparación si la variable de iteración %tmp es 5000
br i1 %exitcond52, label %bb3, label %bb
; Salto hacia fuera del bloque si se cumple la condición
```

Figura A.2: Código ejemplo comentado en LLVM IR

Apéndice B

Guía de comandos LLVM

A continuación se indican las herramientas de líneas de comandos de LLVM que se han utilizado, explicando su función y un pequeño esquema de opciones.

llvm-gcc:

Es un *frontend* para el compilador LLVM basado en GCC. Compila archivos fuentes en lenguaje C y Objetive C en objetos nativos, LLVM bitcode o LLVM IR, dependiendo de las opciones. Por defecto compila a objetos nativos.

Con las opciones *-emit-llvm -c* genera código LLVM bitcode y con las opciones *-emit-llvm -S* genera código LLVM IR.

Al ser derivado de gcc tiene muchas de sus características y acepta la mayoría de sus opciones, como los niveles de optimización (O0...O3).

Sintaxis:

```
llvm-gcc [opciones] nombrefichero
```

Opciones:

- *-o nombreFichero*: Especifica el archivo de salida.
- *-emit-llvm*: Genera la salida a LLVM bitcode (con -c) o a LLVM IR (con -S).

lli:

Ejecuta un programa compilado en bitcode. Toma el archivo en código bitcode y lo ejecuta en una máquina virtual o lo interpreta.

Sintaxis:

```
lli [opciones][nombreFichero][argumentos del programa]
```

llvm-dis:

Es el desamblador de LLVM. Convierte código LLVM bitcode en LLVM IR.

Sintaxis:

```
llvm-dis [opciones] [nombrefichero]
```

Opciones:

- *-o nombreFichero*: Especifica el archivo de salida. Si se omite el código será mostrado en la salida estandar.

llc:

Compilador estático. Genera código nativo de la máquina a partir de LLVM bitcode o de LLVM IR. Compila el código fuente para una arquitectura específica. Con la opción *-march* podemos elegir dicha arquitectura.

Sintaxis:

```
llc [opciones][nombrefichero]
```

Opciones:

- *-march=arch*: Especifica la arquitectura para generar el código. Por ejemplo -march=arm genera código ARM.
- *-O=numero*: Genera código con determinado nivel de optimización. Corresponde con las opciones (O0...O3) de llvm-gcc o de gcc.

opt:

Es el optimizador y analizador de LLVM. Toma archivos con código LLVM IR o LLVM bitcode y realiza los análisis y las optimizaciones indicadas, generando el archivo optimizado o los resultados de los análisis. Es lo que se conoce como pasadas de optimización. Las optimizaciones o análisis disponibles dependen de las bibliotecas vinculadas a ella, así como cualquier biblioteca cargada mediante la opción *-load*. Esto permite crear pasadas de optimización y análisis propias mediante bibliotecas externas.

Sintaxis:

```
opt [opciones][nombreFichero]
```

Opciones:

- *-o nombreFichero*: Especifica el archivo de salida.
- *-S*: Escribe la salida como LLVM IR.
- *-nombrePase*: Proporciona la capacidad de ejecutar cualquier pasada de optimización o análisis en el orden indicado. Existen numerosas pasadas disponibles como *indvars*, *loops*, *loop-reduce*, etc. También se puede ejecutar cualquier pasada creada, registrada y cargada de una biblioteca mediante la opción *-load*.
- *-load=biblioteca*: Carga la biblioteca seleccionada. Esta biblioteca debe registrar las nuevas pasadas de optimización o análisis implementados en ella. Una vez cargada agrega nuevas opciones de línea a la línea de comandos *opt* para permitir las pasadas registradas en la biblioteca.

Ejemplo:

```
opt -indvars -load ../buclesReusos/libbuclesReusos.so -cuenta ejemplo.ll
```

Esta secuencia realizaría la pasada de optimización *indvars*, cargaría la biblioteca libbuclesReusos y realizaría la pasada *cuenta* que ha sido implementada y registrada en la biblioteca cargada.

Apéndice C

Pruebas

En este anexo se van a exponer algunas de las pruebas que se utilizaron para la verificación del buen funcionamiento de las bibliotecas creadas.

Se creó una gran cantidad de programas, en lenguaje C, con distintas funcionalidades, según el caso a evaluar, para pasárselos a las bibliotecas. Al ser muy extensa la batería de pruebas, se va a mostrar sólo alguno de estos programas, ya que sino se extendería mucho.

En cada ejemplo se ofrece el código en C, parte del código LLVM IR que se crea y el código en lenguaje ARM que se genera al final. Para cada una de las pruebas, se explican los resultados obtenidos para que se pueda comprender lo mejor posible.
Este apéndice, se divide en tres partes:

- Cuenta de iteraciones
- Marcación de loads y stores
- Localización de reúsos temporales y espaciales

C.1. Cuenta de iteraciones

Como lo primero que se creó fue la biblioteca para contar el número de veces que pasaba por un bucle, fue lo primero que se probó. En este caso lo primero que se verificó fueron los casos más típicos de bucles, es decir los *for* y *while* de C.

Uno de los ejemplos que mejor ratifica el correcto comportamiento de la biblioteca ante este tipo de iteraciones, es el fichero bucle.c de la pila de pruebas. El código es el siguiente:

```
#include <stdio.h>

int main()
{
    int i=0, j=0, z=0;

    for (i=0;i<5000;i++)
    {
        for(j=0;j<3000;j++)
        {
            printf("%i", j);
        }
    }

    for (i=0;i<5000;i++)
    {
       printf("%i", i);
    }

    for (j=1;j<5000;j=j*2)
    {
        printf("%i", j);
    }

    for (z=100;z<5000;z+=50)
    {
        printf("%i", z);
    }

    for (z=5000;z>1500;z--)
    {
        printf("%i", z);
    }

    z=5000;

    while (z>0)
    {
        z-=100;
        printf("%i", z);
    }
}
```

Figura C.1: Código del fichero bucle.c

En esta prueba, se contemplan bucles anidados (Figura C.2), con incremento constante de uno en uno y de cincuenta en cincuenta (Figura C.3), con incremento variable (Figura C.4), y con decremento (Figura C.6).

```
for (i=0;i<5000;i++)
{
    for(j=0;j<3000;j++)
    {
        printf("%i", j);
    }
}
```

Figura C.2: Bucle anidado

```
for (i=0;i<5000;i++)
{
    printf("%i", i);
}

for (z=100;z<5000;z+=50)
{
    printf("%i", z);
}
```

Figura C.3: Bucles con incremento constante

```
for (j=1;j<5000;j=j*2)
{
    printf("%i", j);
}
```

Figura C.4: Bucle con incremento variable

```
for (z=5000;z>1500;z--)
{
    printf("%i", z);
}

z=5000;

while (z>0)
{
    z-=100;
    printf("%i", z);
}
```

Figura C.5: Bucles con decremento constante

Al compilarlo y analizarlo pasándole la biblioteca libcuentaBucles.so (primera biblioteca que se creó como se explica en el Capítulo 4), se creaba el fichero con el código LLVM IR siguiente:

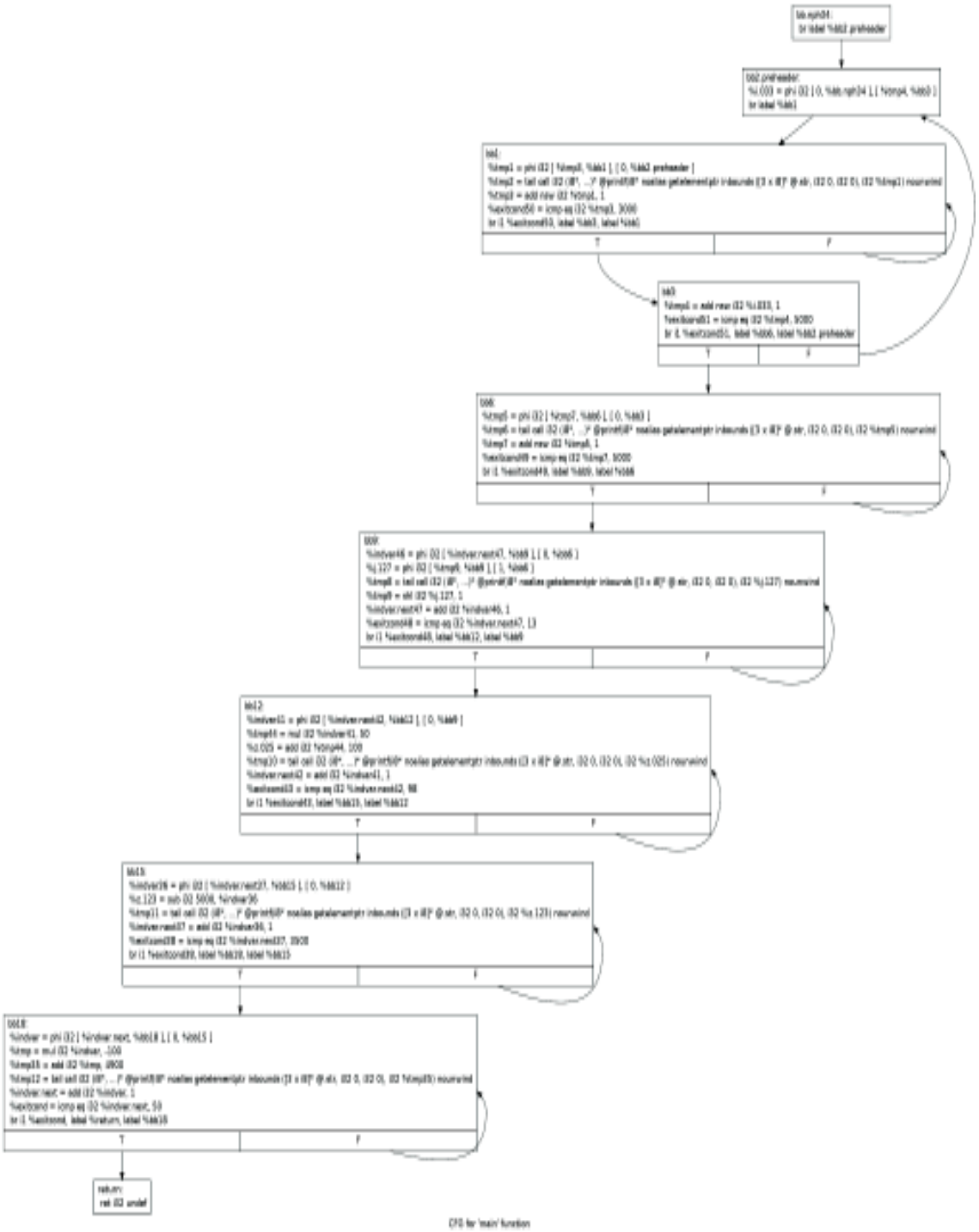

Figura C.6: Fichero bucle.ll

Además de darnos el fichero ARM con el número de iteraciones de cada bucle, daba una salida por pantalla con información de cada bucle, como se puede ver a continuación:

```
printDSpLoc: analizando <main>...
Tiene bucles

--------------------------------------------------------------------------
BUCLE 1

PHI NODE:
  %i.033 = phi i32 [ 0, %bb.nph34 ], [ %3, %bb3 ]

INSTRUCCIONES:
  %3 = add nsw i32 %i.033, 1
  %exitcond11 = icmp eq i32 %3, 5000
  br i1 %exitcond11, label %bb6.preheader, label %bb2.preheader

ITERACIONES: 5000

PHI NODE:
  %0 = phi i32 [ %2, %bb1 ], [ 0, %bb2.preheader ]

INSTRUCCIONES:
  %0 = phi i32 [ %2, %bb1 ], [ 0, %bb2.preheader ]
  %1 = tail call i32 (i8*, ...)* @printf(i8* noalias getelementptr
                  inbounds ([3 x i8]* @.str, i32 0, i32 0), i32 %0) nounwind
  %2 = add nsw i32 %0, 1
  %exitcond10 = icmp eq i32 %2, 3000
  br i1 %exitcond10, label %bb3, label %bb1
ITERACIONES SUBBUCLE NIVEL 2: PARCIALES 3000
ITERACIONES TOTALES BUCLE 15000000
--------------------------------------------------------------------------
BUCLE 3

PHI NODE:
  %4 = phi i32 [ %6, %bb6 ], [ 0, %bb6.preheader ]

INSTRUCCIONES:
  %4 = phi i32 [ %6, %bb6 ], [ 0, %bb6.preheader ]
  %5 = tail call i32 (i8*, ...)* @printf(i8* noalias getelementptr
                  inbounds ([3 x i8]* @.str, i32 0, i32 0), i32 %4) nounwind
  %6 = add nsw i32 %4, 1
  %exitcond9 = icmp eq i32 %6, 5000
  br i1 %exitcond9, label %bb9.preheader, label %bb6

ITERACIONES: 5000
--------------------------------------------------------------------------
```

```
BUCLE 4

PHI NODE:
  %indvar46 = phi i32 [ %indvar.next47, %bb9 ], [ 0, %bb9.preheader ]

INSTRUCCIONES:
  %indvar46 = phi i32 [ %indvar.next47, %bb9 ], [ 0, %bb9.preheader ]
  %j.127 = phi i32 [ %8, %bb9 ], [ 1, %bb9.preheader ]
  %7 = tail call i32 (i8*, ...)* @printf(i8* noalias getelementptr
              inbounds ([3 x i8]* @.str, i32 0, i32 0), i32 %j.127) nounwind
  %8 = shl i32 %j.127, 1
  %indvar.next47 = add i32 %indvar46, 1
  %exitcond8 = icmp eq i32 %indvar.next47, 13
  br i1 %exitcond8, label %bb12.preheader, label %bb9

ITERACIONES: 13
--------------------------------------------------------------------------
BUCLE 5

PHI NODE:
  %indvar41 = phi i32 [ %indvar.next42, %bb12 ], [ 0, %bb12.preheader ]

INSTRUCCIONES:
  %indvar41 = phi i32 [ %indvar.next42, %bb12 ], [ 0, %bb12.preheader ]
  %tmp6 = mul i32 %indvar41, 50
  %z.025 = add i32 %tmp6, 100
  %9 = tail call i32 (i8*, ...)* @printf(i8* noalias getelementptr
               inbounds ([3 x i8]* @.str, i32 0, i32 0), i32 %z.025) nounwind
  %indvar.next42 = add i32 %indvar41, 1
  %exitcond5 = icmp eq i32 %indvar.next42, 98
  br i1 %exitcond5, label %bb15.preheader, label %bb12

ITERACIONES: 98
--------------------------------------------------------------------------
BUCLE 6

PHI NODE:
  %indvar36 = phi i32 [ %indvar.next37, %bb15 ], [ 0, %bb15.preheader ]

INSTRUCCIONES:
  %indvar36 = phi i32 [ %indvar.next37, %bb15 ], [ 0, %bb15.preheader ]
  %tmp = mul i32 %indvar36, -1
  %z.123 = add i32 %tmp, 5000
  %10 = tail call i32 (i8*, ...)* @printf(i8* noalias getelementptr
              inbounds ([3 x i8]* @.str, i32 0, i32 0), i32 %z.123) nounwind
  %indvar.next37 = add i32 %indvar36, 1
  %exitcond = icmp eq i32 %indvar.next37, 3500
  br i1 %exitcond, label %bb18.preheader, label %bb15
```

```
ITERACIONES: 3500
--------------------------------------------------------------------------
BUCLE 7

PHI NODE:
  %indvar = phi i32 [ %indvar.next, %bb18 ], [ 0, %bb18.preheader ]

INSTRUCCIONES:
  %indvar = phi i32 [ %indvar.next, %bb18 ], [ 0, %bb18.preheader ]
  %tmp2 = mul i32 %indvar, -100
  %tmp35 = add i32 %tmp2, 4900
  %11 = tail call i32 (i8*, ...)* @printf(i8* noalias getelementptr
                inbounds ([3 x i8]* @.str, i32 0, i32 0), i32 %tmp35) nounwind
  %indvar.next = add i32 %indvar, 1
  %exitcond1 = icmp eq i32 %indvar.next, 50
  br i1 %exitcond1, label %return, label %bb18

ITERACIONES: 50


-------------------------------RESUMEN-----------------------------------
BUCLE 1 ETIQUETA bb2.preheader VUELTAS 5000
BUCLE 2 ETIQUETA bb1 VUELTAS 3000
BUCLE 3 ETIQUETA bb6 VUELTAS 5000
BUCLE 4 ETIQUETA bb9 VUELTAS 13
BUCLE 5 ETIQUETA bb12 VUELTAS 98
BUCLE 6 ETIQUETA bb15 VUELTAS 3500
BUCLE 7 ETIQUETA bb18 VUELTAS 50
```

Figura C.7: Salida por pantalla al compilar bucle.c

Así podíamos saber :

- El número de veces que pasaba por cada bucle
- Las instrucciones de cada bloque de repetición
- El phinode de cada uno
- La etiqueta en la que está el bucle (tanto en código LLVM como en código ensamblador ARM)

Gracias a la información anterior y al fichero con el código ensamblador ARM se pudo asegurar que la biblioteca cumplía sus funciones, por lo menos, para los casos más usuales de bucles. En la siguientes figuras podemos ver parte del código ensamblador ARM que nos daba.

```
.LBB0_1:                                @ %bb2.preheader
                                        @ Numero de vueltas=5000
                                        @ =>This Loop Header: Depth=1
                                        @     Child Loop BB0_2 Depth 2
mov r7, #0
.LBB0_2:                                @ %bb1
                                        @ Numero de vueltas=3000
                                        @   Parent Loop BB0_1 Depth=1
                                        @ =>  This Inner Loop Header: Depth=2
mov r1, r7
mov r0, r5
add r7, r7, #1
bl printf
cmp r7, r6
bne .LBB0_2
```

Figura C.8: Parte del fichero bucle.arm.opt.s que nos muestra la salida para los bucles anidados

Tanto en la figura anterior como en la siguiente, se puede ver la salida en código ensamblador ARM que resulta de compilar el fichero bucle.c con la biblioteca libcuentaBucles.so. En la anterior, se ve como quedaría comentado en este código, cuando se introduce un bucle anidado. En la posterior, se ve el resultado de otro tipo de bucles para que se pueda ratificar que es correcto.

```
.LBB0_7:                                @ %bb9
                                        @ Numero de vueltas=13
                                        @ =>This Inner Loop Header: Depth=1
mov r1, r6
mov r0, r4
lsl r6, r6, #1
bl printf
subs r8, r8, #1
bne .LBB0_7
@ BB#8:
mov r6, #107, 30
mov r8, #86
ldr r4, .LCPI0_0
orr r6, r6, #3, 22
orr r8, r8, #19, 24
.LBB0_9:                                @ %bb12
                                        @ Numero de vueltas=98
                                        @ =>This Inner Loop Header: Depth=1
mov r0, r4
mov r1, r7
bl printf
add r0, r7, #50
cmp r7, r8
mov r7, r0
bne .LBB0_9
@ BB#10:
mov r7, #201, 30
ldr r4, .LCPI0_0
orr r7, r7, #1, 20
.LBB0_11:                               @ %bb15
                                        @ Numero de vueltas=3500
                                        @ =>This Inner Loop Header: Depth=1
mov r1, r5
mov r0, r4
sub r5, r5, #1
bl printf
subs r6, r6, #1
bne .LBB0_11
```

Figura C.9: Parte del fichero bucle.arm.opt.s que nos muestra la salida para distintos bucles

A partir de ahora, se podía comprobar con bucles más inusuales, pero también posibles, y más complicados. Estas pruebas están en los ficheros bucle2.c y bucle3.c. A continuación se van a poner algunos pequeños ejemplos de distintos tipos de bucles:

```
for (j=0;j<10;j++)
{
    for (m=0;m<20;m++)
    {
        for (k=0;k<20;k++)
        {
            printf("\t%i\n", k);
        }
    }
}
```

Figura C.10: Parte del fichero bucle2.c

A continuación se indica el código resultante para el código C anterior.

```
.LBB0_1:                                @ %bb2
                                        @ Numero de vueltas=20
                                        @   Parent Loop BB0_5 Depth=1
                                        @ =>  This Inner Loop Header: Depth=2
mov r1, r6
mov r0, r5
add r6, r6, #1
bl printf
cmp r6, #20
bne .LBB0_1
@ BB#2:                                 @ %bb4
                                        @   in Loop: Header=BB0_1 Depth=2
subs r7, r7, #1
beq .LBB0_4
@ BB#3:                                 @ %bb3.preheader
                                        @ Numero de vueltas=20
                                        @   in Loop: Header=BB0_1 Depth=2
mov r6, #0
b .LBB0_1
.LBB0_4:                                @ %bb6
                                        @   in Loop: Header=BB0_5 Depth=1
subs r4, r4, #1
moveq r0, #0
ldmiaeq sp!, {r4, r5, r6, r7, pc}    @ Load de pila. Desplazamiento 0
.LBB0_5:                                @ %bb5.preheader
                                        @ Numero de vueltas=10
                                        @ =>This Loop Header: Depth=1
                                        @     Child Loop BB0_1 Depth 2
mov r7, #20
mov r6, #0
b .LBB0_1
```

Figura C.11: Parte del fichero bucle2.arm.opt.s

Otro ejemplo sería el siguiente:

```
int l=0;
while (l<30)
{
    if(l==10) break;
    printf("%i\n", l);
    l++;
}
```

Figura C.12: Parte del fichero bucle3.c

Cuya compilación da:

```
@ BB#0:
stmdb sp!, {r4, r5, lr}
mov r4, #0
ldr r5, .LCPI0_0
.LBB0_1:


mov r1, r4
mov r0, r5
add r4, r4, #1
bl printf
cmp r4, #10
bne .LBB0_1
```

```
@ %bb.nph



@ %bb1
@ Numero de vueltas=10
@ =>This Inner Loop Header: Depth=1
```

Figura C.13: Parte del fichero bucle3.arm.opt.s

Para acabar esta parte, se deja el ejemplo de un bucle con la instrucción *do* en código C.

```
int m=0;

do
{
    printf("%i\n", m);
    m=m+3;
}
while (m<100);
```

Figura C.14: Ejemplo de bucle con instrucción *do*

```
@ BB#0:                                 @ %entry
stmdb sp!, {r4, r5, lr}
mov r4, #0
ldr r5, .LCPI0_0
.LBB0_1:                                @ %bb
                                        @ Numero de vueltas=34
                                        @ =>This Inner Loop Header: Depth=1
mov r1, r4
mov r0, r5
add r4, r4, #3
bl printf
cmp r4, #102
bne .LBB0_1
```

Figura C.15: Tranformación de bucle con instrucción *do* en lenguaje ARM

A parte de estos ejemplos, y de otros muchos más, también se utilizaron otros códigos no diseñados específicamente para probar la biblioteca, sino que existían anteriormente, y todo resultó bien.

C.2. Marcación de loads y stores

Al llegar a esta parte, se empezaron a crear pruebas para la marcación de loads y stores en los ficheros con el código LLVM IR. En este punto también había que mirar si había reúsos temporales y espaciales. Uno de los ejemplos que enseña las primeras pruebas que se hicieron es:

```
#include <stdio.h>
#define Tam 5000
int main()
{
    int i=0;
    int A[Tam];
    int B[Tam];
    for (i=0;i<Tam;i++)
    {
         A[i]=i*5;
         printf("%i", A[i]);
         B[i]=0;
    }

    for (i=0;i<Tam;i++)
    {
       printf("%i", A[i]);
       printf("%i", B[i]);

    }
}
```

Figura C.16: Ejemplo de prueba para marcación de loads y stores

Este programa da como resultado el fichero stores.ll, el cual se puede ver en la Figura C.19.

Además del fichero ARM, al principió se sacaban por pantalla algunos datos para confirmar que iba bien.

```
STORE :  store i32 0, i32* %scevgep20, align 4
OPERANDO 0: i32 0
OPERANDO 1:  %scevgep20 = getelementptr [5000 x i32]* %A, i32 0, i32 %i.013
DIRECCION MEMORIA 0x8b4ac3c

STORE :  store i32 %tmp, i32* %scevgep17, align 4
OPERANDO 0:   %tmp = mul i32 %i.111, 5
OPERANDO 1:  %scevgep17 = getelementptr [5000 x i32]* %A, i32 0, i32 %i.111
DIRECCION MEMORIA 0x8b4aa84

STORE :  store i32 0, i32* %scevgep18, align 4
OPERANDO 0: i32 0
OPERANDO 1:  %scevgep18 = getelementptr [5000 x i32]* %B, i32 0, i32 %i.111
DIRECCION MEMORIA 0x8b4a67c

LOAD :  %4 = load i32* %scevgep, align 4
OPERANDO 0:   %scevgep = getelementptr [5000 x i32]* %A, i32 0, i32 %i.210
DIRECCION MEMORIA 0x8b4a1fc

LOAD :  %6 = load i32* %scevgep15, align 4
OPERANDO 0:   %scevgep15 = getelementptr [5000 x i32]* %B, i32 0, i32 %i.210
DIRECCION MEMORIA 0x8b4a134
```

Figura C.17: Salida por pantalla al compilar el fichero stores.c

La biblioteca libMarcarLoadsStores.so al pasársela en compilación a cualquier fichero de código C, genera un fichero llamado <nombreFichero >Marcado.ll.

Es aquí donde se escriben todos metadatas necesarios. Uno en cada load y store que exista en el fichero. Además se añaden al final del archivo otros metadatas indicando lo que se tiene que escribir en cada uno en el código ensamblador ARM. Para el caso anterior se generó el fichero storesMarcado.ll, de donde se va a sacar una parte:

```
bb:                                               ; preds = %bb, %bb.nph9
  %i.08 = phi i32 [ 0, %bb.nph9 ], [ %tmp2, %bb ]
  %scevgep12 = getelementptr [5000 x i32]* %A, i32 0, i32 %i.08
  %scevgep13 = getelementptr [5000 x i32]* %B, i32 0, i32 %i.08
  %tmp = mul i32 %i.08, 5
  store i32 %tmp, i32* %scevgep12, align 4, !dbg !5
  %tmp1 = call i32 (i8*, ...)* @printf(i8* noalias getelementptr
              inbounds ([3 x i8]* @.str, i32 0, i32 0), i32 %tmp) nounwind
  store i32 0, i32* %scevgep13, align 4, !dbg !6
  %tmp2 = add nsw i32 %i.08, 1
  %exitcond11 = icmp eq i32 %tmp2, 5000
  br i1 %exitcond11, label %bb3, label %bb

bb3:                                              ; preds = %bb3, %bb
  %i.17 = phi i32 [ %tmp7, %bb3 ], [ 0, %bb ]
  %scevgep = getelementptr [5000 x i32]* %A, i32 0, i32 %i.17
  %scevgep10 = getelementptr [5000 x i32]* %B, i32 0, i32 %i.17
  %tmp3 = load i32* %scevgep, align 4, !dbg !7
  %tmp4 = call i32 (i8*, ...)* @printf(i8* noalias getelementptr
              inbounds ([3 x i8]* @.str, i32 0, i32 0), i32 %tmp3) nounwind
  %tmp5 = load i32* %scevgep10, align 4, !dbg !8
  %tmp6 = call i32 (i8*, ...)* @printf(i8* noalias getelementptr
             inbounds ([3 x i8]* @.str, i32 0, i32 0), i32 %tmp5) nounwind
  %tmp7 = add nsw i32 %i.17, 1
  %exitcond = icmp eq i32 %tmp7, 5000
  br i1 %exitcond, label %return, label %bb3

return:                                           ; preds = %bb3
  ret i32 undef
```

Figura C.18: Fichero storesMarcado.ll

En este apartado se muestra un simple ejemplo, ya que todos los resultados son muy parecidos, por lo que no se pueden sacar grandes diferencias, como ocurría en el apartado anterior.

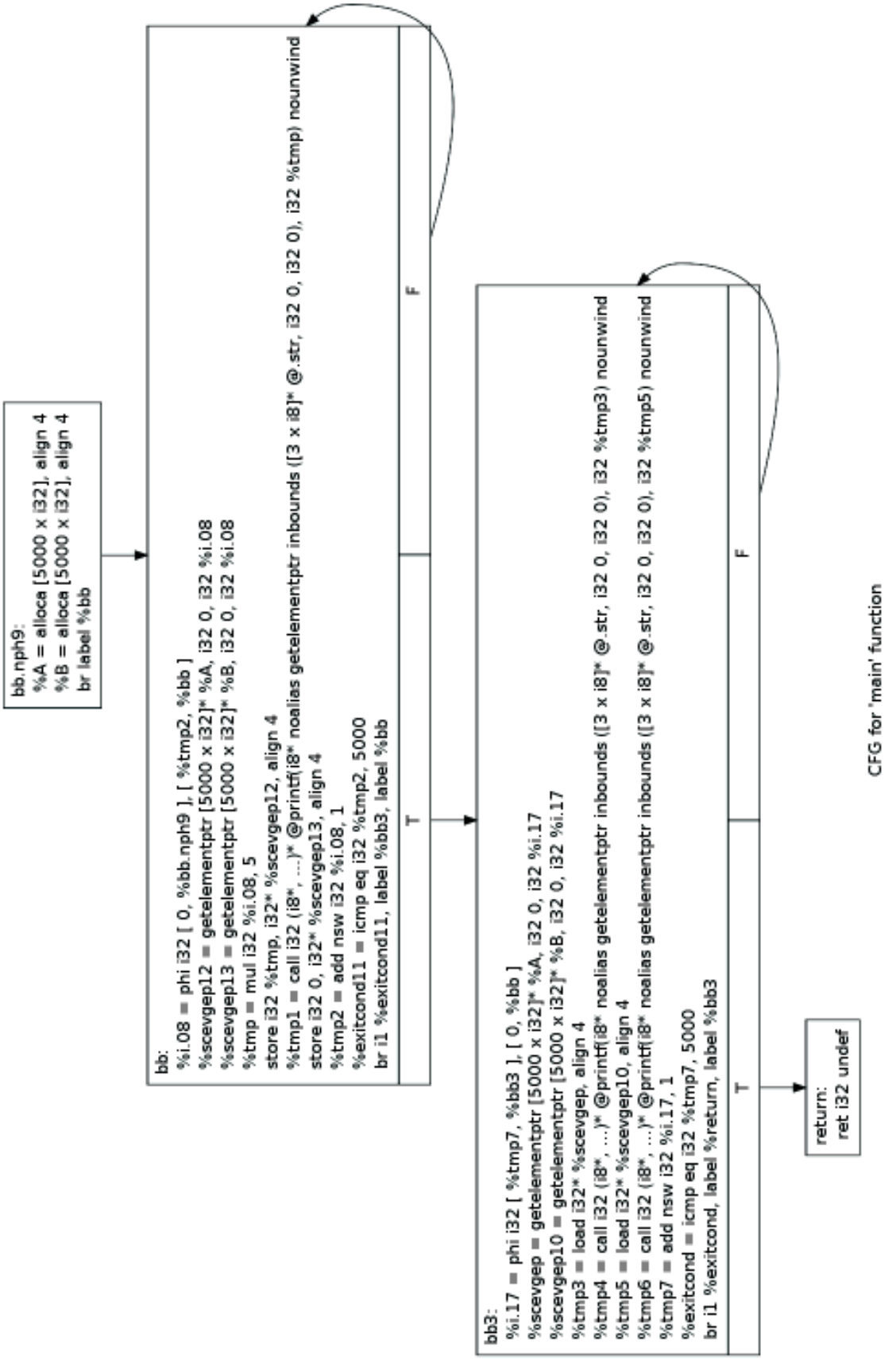

Figura C.19: Fichero stores.ll

C.3. Localización de reúsos temporales y espaciales

Aquí también se hizo una gran cantidad de pruebas. Se van a destacar algunas, aunque es muy difícil enseñar todos ejemplos posibles.

Para estas pruebas, además de crear nuevos programas, se reutilizaron los creados para marcación de loads y stores. En la Figura C.20 podemos apreciar una muestra del resultado del fichero stores.c después de realizar la localización.

```
.LBB0_1:                                @ %bb
                                        @ Numero de vueltas=5000
                                        @ =>This Inner Loop Header: Depth=1
mov r0, r7
mov r1, r9
str r9, [r6], #4            @ Store var "A". Reuso espacial. Var iteracion "i".
                            @ Desplazamiento con "stride" 1
add r9, r9, #5
bl printf
str r4, [r5], #4            @ Store var "B". Reuso espacial. Var iteracion "i".
                            @ Desplazamiento con "stride" 1
cmp r9, r8
bne .LBB0_1
@ BB#2:                                 @ %bb.bb3_crit_edge
add lr, sp, #1, 18
mov r4, #226, 30
orr r4, r4, #1, 20
ldr r7, .LCPI0_0            @ Load Constante .LCPI0_0
mov r5, sp
add r6, lr, #226, 28
.LBB0_3:                                @ %bb3
                                        @ Numero de vueltas=5000
                                        @ =>This Inner Loop Header: Depth=1
ldr r1, [r6], #4            @ Load var "A". Reuso espacial. Var iteracion "i".
                            @ Desplazamiento con "stride" 1
mov r0, r7
bl printf
ldr r1, [r5], #4            @ Load var "B". Reuso espacial. Var iteracion "i".
                            @ Desplazamiento con "stride" 1
mov r0, r7
bl printf
subs r4, r4, #1
bne .LBB0_3
@ BB#4:                                 @ %return
add sp, sp, #113, 26 @ 7232
add sp, sp, #2, 18 @ 32768
ldmia sp!, {r4, r5, r6, r7, r8, r9, r10, pc}    @ Load de pila. Desplazamiento 0
```

Figura C.20: Parte del fichero stores.arm.opt.s

En la figura anterior, se puede comprobar que se indica el reúso espacial que existe en los vectores A y B, y un load de pila (*ldmia sp*). Igualmente, se constata que la variable de iteración es i en los dos bucles, y que el "stride"de ambos es uno.

Un ejemplo más completo se puede ver a continuación. En la Figura C.21 se ve el código del fichero pruebaStores.c.

```
#include <stdio.h>
#define Tam 5000
int main()
{
    int varBucle=0;
    int varBucle2=0;
    int i=0;
    int Vector1[Tam];
    int Vector2[Tam];
    int Vector3[Tam][Tam];
    int mivar=0;
    int mivar2=0;

    for (varBucle=0;varBucle<Tam;varBucle++)
    {
         Vector1[varBucle]=varBucle*5;
         Vector2[varBucle]=varBucle*5;
    }
    for (varBucle2=0;varBucle2<Tam;varBucle2+=5)
    {
         printf("%i", Vector1[varBucle2]);
         printf("%i", Vector2[varBucle2]);
    }
    for (varBucle=0;varBucle<Tam;varBucle++)
    {
         for (varBucle2=0;varBucle2<Tam;varBucle2++)
         {
             Vector3[varBucle][varBucle2]=varBucle*varBucle2;
         }
    }
    for (varBucle=0;varBucle<Tam;varBucle++)
    {
         for (varBucle2=0;varBucle2<Tam;varBucle2++)
         {
             printf("%i", Vector3[varBucle][varBucle2]);
         }
    }
    int j=Tam;
    for (i=1;i<Tam;i=i*5)
    {
         j--;
         printf("%i", Vector1[i]);
         printf("%i", Vector2[j]);
    }
}
```

Figura C.21: Fichero pruebaStores.c

En la Figura C.22 se puede cotejar el funcionamiento de la biblioteca con distintos

tipos de reúso espacial, entre los que están los de "stride" uno, de "stride" cinco, de "stride" negativo y de "stride" no constante. Igualmente, se ven reúsos espaciales sobre múltiples variables y reúsos temporales de constantes.

```
.LBB0_1:                                @ %bb
                                        @ Numero de vueltas=5000
                                        @ =>This Inner Loop Header: Depth=1
str r0, [r2], #4            @ Store var "Vector1". Reuso espacial.
                            @ Var iteracion "varBucle".
                            @ Desplazamiento con "stride" 1
str r0, [r1], #4            @ Store var "Vector2". Reuso espacial.
                            @ Var iteracion "varBucle".
                            @ Desplazamiento con "stride" 1
add r0, r0, #5
cmp r0, r3
bne .LBB0_1
@ BB#2:                                 @ %bb.bb3_crit_edge
add lr, sp, #245, 16 @ 16056320
mov r4, #250, 30 @ 1000
ldr r7, .LCPI0_0            @ Load Constante .LCPI0_0
add lr, lr, #5, 8 @ 83886080
add r5, lr, #225, 24 @ 57600
add lr, sp, #98, 20 @ 401408
add lr, lr, #95, 12 @ 99614720
add r6, lr, #242, 28 @ 3872
.LBB0_3:                                @ %bb3
                                        @ Numero de vueltas=1000
                                        @ =>This Inner Loop Header: Depth=1
ldr r1, [r6], #20           @ Load var "Vector1". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar46".
                            @ Desplazamiento con "stride" 5
mov r0, r7
bl printf
ldr r1, [r5], #20           @ Load var "Vector2". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar46".
                            @ Desplazamiento con "stride" 5
mov r0, r7
bl printf
subs r4, r4, #1
bne .LBB0_3
@ BB#4:                                 @ %bb3.bb8.preheader_crit_edge
mov r0, sp
mov r1, #226, 30 @ 904
orr r1, r1, #1, 20 @ 4096
mov r2, #0
.LBB0_5:                                @ %bb8.preheader
                                        @ Numero de vueltas=5000
                                        @ =>This Loop Header: Depth=1
                                        @     Child Loop BB0_6 Depth 2
mov r3, r0
mov r12, #0
mov lr, r1
```

```
.LBB0_6:                                @ %bb7
                                        @ Numero de vueltas=5000
                                        @   Parent Loop BB0_5 Depth=1
                                        @ =>  This Inner Loop Header: Depth=2
str r12, [r3], #4           @ Store var "Vector3". Reuso espacial.
                            @ Multiples variables "tmp37" "varBucle2"
add r12, r12, r2
subs lr, lr, #1
bne .LBB0_6
@ BB#7:                                 @ %bb9
                                        @   in Loop: Header=BB0_5 Depth=1
add r0, r0, #226, 28 @ 3616
add r2, r2, #1
cmp r2, r1
add r0, r0, #1, 18 @ 16384
bne .LBB0_5
@ BB#8:                                 @ %bb9.bb14.preheader_crit_edge
mov r5, #226, 30 @ 904
orr r5, r5, #1, 20 @ 4096
mov r4, sp
ldr r6, .LCPI0_0            @ Load Constante .LCPI0_0
mov r7, r5
.LBB0_9:                                @ %bb14.preheader
                                        @ Numero de vueltas=5000
                                        @ =>This Loop Header: Depth=1
                                        @     Child Loop BB0_10 Depth 2
mov r8, r4
mov r9, r5
.LBB0_10:                               @ %bb13
                                        @ Numero de vueltas=5000
                                        @   Parent Loop BB0_9 Depth=1
                                        @ =>  This Inner Loop Header: Depth=2
ldr r1, [r8], #4            @ Load var "Vector3". Reuso espacial.
                            @ Multiples variables "varBucle" "varBucle2"
mov r0, r6
bl printf
subs r9, r9, #1
bne .LBB0_10
@ BB#11:                                @ %bb15
                                        @   in Loop: Header=BB0_9 Depth=1
add r4, r4, #226, 28 @ 3616
subs r7, r7, #1
add r4, r4, #1, 18 @ 16384
bne .LBB0_9
@ BB#12:                                @ %bb18.preheader
add lr, sp, #245, 16 @ 16056320
mov r5, #1
mov r6, #0
ldr r8, .LCPI0_0            @ Load Constante .LCPI0_0
add lr, lr, #5, 8 @ 83886080
add r4, lr, #225, 24 @ 57600
add r4, r4, #135, 30 @ 540
add lr, sp, #98, 20 @ 401408
add lr, lr, #95, 12 @ 99614720
add r4, r4, #19, 22 @ 19456
add r7, lr, #242, 28 @ 3872
```

```
.LBB0_13:                               @ %bb18
                                        @ Numero de vueltas=6
                                        @ =>This Inner Loop Header: Depth=1
ldr r1, [r7, r5, lsl #2]    @ Load var "Vector1". Reuso espacial.
                            @ Var iteracion "i".
                            @ Desplazamiento con "stride" no constante
mov r0, r8
add r5, r5, r5, lsl #2
bl printf
ldr r1, [r4, -r6, lsl #2]   @ Load var "Vector2". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" -1
mov r0, r8
add r6, r6, #1
bl printf
cmp r6, #6
bne .LBB0_13
```

Figura C.22: Fichero pruebaStores.arm.opt.s

Para terminar, se va a dejar constancia de una prueba con un fichero ya existente, para demostrar que no son pruebas realizadas vagamente. En la siguiente figura se presenta el código del fichero jfdctint.c.

```
/*************************************************************************/
/*                                                                       */
/*   SNU-RT Benchmark Suite for Worst Case Timing Analysis               */
/*   =====================================================               */
/*                              Collected and Modified by S.-S. Lim       */
/*                                        sslim@archi.snu.ac.kr          */
/*                                     Real-Time Research Group          */
/*                                    Seoul National University          */
/*                                                                       */
/*                                                                       */
/*        < Features > - restrictions for our experimental environment   */
/*                                                                       */
/*          1. Completely structured.                                    */
/*               - There are no unconditional jumps.                     */
/*               - There are no exit from loop bodies.                   */
/*                 (There are no 'break' or 'return' in loop bodies)     */
/*          2. No 'switch' statements.                                   */
/*          3. No 'do..while' statements.                                */
/*          4. Expressions are restricted.                               */
/*               - There are no multiple expressions joined by 'or',     */
/*                 'and' operations.                                     */
/*          5. No library calls.                                         */
/*               - All the functions needed are implemented in the       */
/*                 source file.                                          */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*  FILE: jfdctint.c                                                     */
/*  SOURCE : Thomas G. Lane,  Public domain JPEG source code.            */
/*           Modified by Steven Li at Princeton University.              */
/*                                                                       */
/*  DESCRIPTION :                                                        */
/*                                                                       */
/*     JPEG slow-but-accurate integer implementation of the forward      */
/*     DCT (Discrete Cosine Transform).                                  */
/*                                                                       */
/*  REMARK :                                                             */
/*                                                                       */
/*  EXECUTION TIME :                                                     */
/*                                                                       */
/*                                                                       */
/*************************************************************************/


/*************************************************************************
    Functions to be timed
*************************************************************************/

/* This definitions are added by Steven Li so as to bypass the header
   files.
   */
#define DCT_ISLOW_SUPPORTED
#define DCTSIZE 8
#define BITS_IN_JSAMPLE 8
```

```
#define MULTIPLY16C16(var,const)  ((var) * (const))
#define DCTELEM int
#define INT32   int
#define GLOBAL
#define RIGHT_SHIFT(x,shft) ((x) >> (shft))
#define ONE ((INT32) 1)
#define DESCALE(x,n)  RIGHT_SHIFT((x) + (ONE << ((n)-1)), n)
#define SHIFT_TEMPS


/*
 * jfdctint.c
 *
 * Copyright (C) 1991-1994, Thomas G. Lane.
 * This file is part of the Independent JPEG Group's software.
 * For conditions of distribution and use, see the accompanying README file.
 *
 * This file contains a slow-but-accurate integer implementation of the
 * forward DCT (Discrete Cosine Transform).
 *
 * A 2-D DCT can be done by 1-D DCT on each row followed by 1-D DCT
 * on each column.  Direct algorithms are also available, but they are
 * much more complex and seem not to be any faster when reduced to code.
 *
 * This implementation is based on an algorithm described in
 *   C. Loeffler, A. Ligtenberg and G. Moschytz, "Practical Fast 1-D DCT
 *   Algorithms with 11 Multiplications", Proc. Int'l. Conf. on Acoustics,
 *   Speech, and Signal Processing 1989 (ICASSP '89), pp. 988-991.
 * The primary algorithm described there uses 11 multiplies and 29 adds.
 * We use their alternate method with 12 multiplies and 32 adds.
 * The advantage of this method is that no data path contains more than one
 * multiplication; this allows a very simple and accurate implementation in
 * scaled fixed-point arithmetic, with a minimal number of shifts.
 */

#define JPEG_INTERNALS

#if DCTSIZE != 8
  Sorry, this code only copes with 8x8 DCTs. /* deliberate syntax err */
#endif

#if BITS_IN_JSAMPLE == 8
#define CONST_BITS  13
#define PASS1_BITS  2
#else
#define CONST_BITS  13
#define PASS1_BITS  1 /* lose a little precision to avoid overflow */
#endif
```

```
#if CONST_BITS == 13
#define FIX_0_298631336  ((INT32)  2446) /* FIX(0.298631336) */
#define FIX_0_390180644  ((INT32)  3196) /* FIX(0.390180644) */
#define FIX_0_541196100  ((INT32)  4433) /* FIX(0.541196100) */
#define FIX_0_765366865  ((INT32)  6270) /* FIX(0.765366865) */
#define FIX_0_899976223  ((INT32)  7373) /* FIX(0.899976223) */
#define FIX_1_175875602  ((INT32)  9633) /* FIX(1.175875602) */
#define FIX_1_501321110  ((INT32)  12299) /* FIX(1.501321110) */
#define FIX_1_847759065  ((INT32)  15137) /* FIX(1.847759065) */
#define FIX_1_961570560  ((INT32)  16069) /* FIX(1.961570560) */
#define FIX_2_053119869  ((INT32)  16819) /* FIX(2.053119869) */
#define FIX_2_562915447  ((INT32)  20995) /* FIX(2.562915447) */
#define FIX_3_072711026  ((INT32)  25172) /* FIX(3.072711026) */
#else
#define FIX_0_298631336  FIX(0.298631336)
#define FIX_0_390180644  FIX(0.390180644)
#define FIX_0_541196100  FIX(0.541196100)
#define FIX_0_765366865  FIX(0.765366865)
#define FIX_0_899976223  FIX(0.899976223)
#define FIX_1_175875602  FIX(1.175875602)
#define FIX_1_501321110  FIX(1.501321110)
#define FIX_1_847759065  FIX(1.847759065)
#define FIX_1_961570560  FIX(1.961570560)
#define FIX_2_053119869  FIX(2.053119869)
#define FIX_2_562915447  FIX(2.562915447)
#define FIX_3_072711026  FIX(3.072711026)
#endif

#if BITS_IN_JSAMPLE == 8
#define MULTIPLY(var,const)  MULTIPLY16C16(var,const)
#else
#define MULTIPLY(var,const)  ((var) * (const))
#endif

DCTELEM data[64] = {81, 10854, 1893, 55245, 7746, 47274, 61698, 14040,
 32421, 52299, 9138, 35805, 43626, 35259, 36543, 10710,
 48276, 63894, 43968, 15210, 56961, 39369, 58893, 34185,
 24771, 17874, 18063, 43200, 44136, 37554, 14103, 40800,
 52611, 50634, 49833, 8835, 61041, 57729, 10443, 12765,
 59451, 42864, 64983, 57735, 11241, 53364, 19713, 510,
 2376, 53949, 31983, 59580, 60021, 53139, 55323, 18120,
 50781, 3849, 53253, 4950, 3081, 16644, 51078, 43350};

GLOBAL void
jpeg_fdct_islow ();

int main(int argc, char *argv[])
{
  jpeg_fdct_islow();
  return ;
}
```

```
GLOBAL void
jpeg_fdct_islow ()
{
  INT32 tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
  INT32 tmp10, tmp11, tmp12, tmp13;
  INT32 z1, z2, z3, z4, z5;
  DCTELEM *dataptr;
  int ctr;
  SHIFT_TEMPS

  dataptr = data;
  for (ctr = DCTSIZE-1; ctr >= 0; ctr--)
  {
    tmp0 = dataptr[0] + dataptr[7];
    tmp7 = dataptr[0] - dataptr[7];
    tmp1 = dataptr[1] + dataptr[6];
    tmp6 = dataptr[1] - dataptr[6];
    tmp2 = dataptr[2] + dataptr[5];
    tmp5 = dataptr[2] - dataptr[5];
    tmp3 = dataptr[3] + dataptr[4];
    tmp4 = dataptr[3] - dataptr[4];

    tmp10 = tmp0 + tmp3;
    tmp13 = tmp0 - tmp3;
    tmp11 = tmp1 + tmp2;
    tmp12 = tmp1 - tmp2;

    dataptr[0] = (DCTELEM) ((tmp10 + tmp11) << PASS1_BITS);
    dataptr[4] = (DCTELEM) ((tmp10 - tmp11) << PASS1_BITS);

    z1 = MULTIPLY(tmp12 + tmp13, FIX_0_541196100);
    dataptr[2] = (DCTELEM) DESCALE(z1 + MULTIPLY(tmp13, FIX_0_765366865),
   CONST_BITS-PASS1_BITS);
    dataptr[6] = (DCTELEM) DESCALE(z1 + MULTIPLY(tmp12, - FIX_1_847759065),
   CONST_BITS-PASS1_BITS);

    z1 = tmp4 + tmp7;
    z2 = tmp5 + tmp6;
    z3 = tmp4 + tmp6;
    z4 = tmp5 + tmp7;
    z5 = MULTIPLY(z3 + z4, FIX_1_175875602);  /* sqrt(2) * c3 */

    tmp4 = MULTIPLY(tmp4, FIX_0_298631336);  /* sqrt(2) * (-c1+c3+c5-c7) */
    tmp5 = MULTIPLY(tmp5, FIX_2_053119869);  /* sqrt(2) * ( c1+c3-c5+c7) */
    tmp6 = MULTIPLY(tmp6, FIX_3_072711026);  /* sqrt(2) * ( c1+c3+c5-c7) */
    tmp7 = MULTIPLY(tmp7, FIX_1_501321110);  /* sqrt(2) * ( c1+c3-c5-c7) */
    z1 = MULTIPLY(z1, - FIX_0_899976223);  /* sqrt(2) * (c7-c3) */
    z2 = MULTIPLY(z2, - FIX_2_562915447);  /* sqrt(2) * (-c1-c3) */
    z3 = MULTIPLY(z3, - FIX_1_961570560);  /* sqrt(2) * (-c3-c5) */
    z4 = MULTIPLY(z4, - FIX_0_390180644);  /* sqrt(2) * (c5-c3) */

    z3 += z5;
    z4 += z5;
```

```
  dataptr[7] = (DCTELEM) DESCALE(tmp4 + z1 + z3, CONST_BITS-PASS1_BITS);
  dataptr[5] = (DCTELEM) DESCALE(tmp5 + z2 + z4, CONST_BITS-PASS1_BITS);
  dataptr[3] = (DCTELEM) DESCALE(tmp6 + z2 + z3, CONST_BITS-PASS1_BITS);
  dataptr[1] = (DCTELEM) DESCALE(tmp7 + z1 + z4, CONST_BITS-PASS1_BITS);

  dataptr += DCTSIZE; /* advance pointer to next row */
}

dataptr = data;
for (ctr = DCTSIZE-1; ctr >= 0; ctr--)
{
  tmp0 = dataptr[DCTSIZE*0] + dataptr[DCTSIZE*7];
  tmp7 = dataptr[DCTSIZE*0] - dataptr[DCTSIZE*7];
  tmp1 = dataptr[DCTSIZE*1] + dataptr[DCTSIZE*6];
  tmp6 = dataptr[DCTSIZE*1] - dataptr[DCTSIZE*6];
  tmp2 = dataptr[DCTSIZE*2] + dataptr[DCTSIZE*5];
  tmp5 = dataptr[DCTSIZE*2] - dataptr[DCTSIZE*5];
  tmp3 = dataptr[DCTSIZE*3] + dataptr[DCTSIZE*4];
  tmp4 = dataptr[DCTSIZE*3] - dataptr[DCTSIZE*4];

  tmp10 = tmp0 + tmp3;
  tmp13 = tmp0 - tmp3;
  tmp11 = tmp1 + tmp2;
  tmp12 = tmp1 - tmp2;

  dataptr[DCTSIZE*0] = (DCTELEM) DESCALE(tmp10 + tmp11, PASS1_BITS);
  dataptr[DCTSIZE*4] = (DCTELEM) DESCALE(tmp10 - tmp11, PASS1_BITS);

  z1 = MULTIPLY(tmp12 + tmp13, FIX_0_541196100);
  dataptr[DCTSIZE*2] = (DCTELEM) DESCALE(z1 + MULTIPLY(tmp13, FIX_0_765366865),
 CONST_BITS+PASS1_BITS);
  dataptr[DCTSIZE*6] = (DCTELEM) DESCALE(z1 + MULTIPLY(tmp12, - FIX_1_847759065),
 CONST_BITS+PASS1_BITS);

  z1 = tmp4 + tmp7;
  z2 = tmp5 + tmp6;
  z3 = tmp4 + tmp6;
  z4 = tmp5 + tmp7;
  z5 = MULTIPLY(z3 + z4, FIX_1_175875602);  /* sqrt(2) * c3 */

  tmp4 = MULTIPLY(tmp4, FIX_0_298631336);  /* sqrt(2) * (-c1+c3+c5-c7) */
  tmp5 = MULTIPLY(tmp5, FIX_2_053119869);  /* sqrt(2) * ( c1+c3-c5+c7) */
  tmp6 = MULTIPLY(tmp6, FIX_3_072711026);  /* sqrt(2) * ( c1+c3+c5-c7) */
  tmp7 = MULTIPLY(tmp7, FIX_1_501321110);  /* sqrt(2) * ( c1+c3-c5-c7) */
  z1 = MULTIPLY(z1, - FIX_0_899976223);  /* sqrt(2) * (c7-c3) */
  z2 = MULTIPLY(z2, - FIX_2_562915447);  /* sqrt(2) * (-c1-c3) */
  z3 = MULTIPLY(z3, - FIX_1_961570560);  /* sqrt(2) * (-c3-c5) */
  z4 = MULTIPLY(z4, - FIX_0_390180644);  /* sqrt(2) * (c5-c3) */

  z3 += z5;
  z4 += z5;
```

```
    dataptr[DCTSIZE*7] = (DCTELEM) DESCALE(tmp4 + z1 + z3,
   CONST_BITS+PASS1_BITS);
    dataptr[DCTSIZE*5] = (DCTELEM) DESCALE(tmp5 + z2 + z4,
   CONST_BITS+PASS1_BITS);
    dataptr[DCTSIZE*3] = (DCTELEM) DESCALE(tmp6 + z2 + z3,
   CONST_BITS+PASS1_BITS);
    dataptr[DCTSIZE*1] = (DCTELEM) DESCALE(tmp7 + z1 + z4,
   CONST_BITS+PASS1_BITS);

    dataptr++; /* advance pointer to next column */
  }
}
```

Figura C.23: Fichero jfdctint.c

Y en la Figura C.24 está el fichero ARM que genera. Se muestra entero para tener una visión en conjunto de como queda. Espero que con esto quede suficientemente claro como funcionan las bibliotecas y se entiendan los resultados obtenidos.

```
.syntax unified
.eabi_attribute 6, 2
.eabi_attribute 8, 1
.eabi_attribute 9, 1
.eabi_attribute 20, 1
.eabi_attribute 21, 1
.eabi_attribute 23, 3
.eabi_attribute 24, 1
.eabi_attribute 25, 1
.file "jfdctintMarcado.ll"
.text
.globl main
.align 2
.type main,%function
main:                                   @ @main
@ BB#0:                                 @ %entry
push {r4, r5, r6, r7, r8, r9, r10, r11, lr}
sub sp, sp, #12
mov r0, #0
.LBB0_1:                                @ %bb.i
                                        @ Numero de vueltas=8
                                        @ =>This Inner Loop Header: Depth=1
ldr r1, .LCPI0_0            @ Load Constante .LCPI0_0
ldr r2, [r1, r0]!           @ Load var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8
add r7, r1, #16
```

```
ldr r3, [r1, #4]            @ Load var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8
ldr lr, [r1, #12]           @ Load var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8
str r2, [sp]                @ Store de pila. Desplazamiento 0
str r3, [sp, #8]            @ Store de pila. Desplazamiento 8 bytes

ldr r12, [r1, #8]           @ Load var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8
str lr, [sp, #4]            @ Store de pila. Desplazamiento 4 bytes
add r0, r0, #32
cmp r0, #1, 24              @ 256
ldmia r7, {r4, r5, r6, r7}    @ Load var "data". Reuso espacial.
                                @ Var calculada basada en var iteracion "indvar24".
                                @ Desplazamiento con "stride" 8
add r9, r5, r12
add r8, r6, r3
add r10, r9, r8
add r11, r7, r2
add r2, r4, lr
add r3, r2, r11
add lr, r3, r10
sub r3, r3, r10
lsl lr, lr, #2
str lr, [r1]                @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8
sub r2, r11, r2
lsl r3, r3, #2
str r3, [r1, #16]           @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8
sub r3, r8, r9
mov r9, #81
orr r8, r9, #17, 24         @ 4352
add lr, r2, r3
mul r9, lr, r8
mov r8, #126
orr lr, r8, #6, 22          @ 6144
mla r8, r2, lr, r9
add r2, r8, #1, 22          @ 1024
asr r2, r2, #11

str r2, [r1, #8]            @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8

ldr r2, .LCPI0_1            @ Load Constante .LCPI0_1
mla lr, r3, r2, r9
```

```
sub r3, r12, r5
ldr r5, [sp, #4]            @ Load de pila. Desplazamiento 4 bytes
sub r4, r5, r4
add r2, lr, #1, 22          @ 1024
ldr lr, [sp, #8]            @ Load de pila. Desplazamiento 8 bytes
asr r2, r2, #11
str r2, [r1, #24]           @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8
sub lr, lr, r6
add r5, r4, lr
ldr r2, [sp]                @ Load de pila. Desplazamiento 0
sub r2, r2, r7
ldr r7, .LCPI0_2            @ Load Constante .LCPI0_2
mul r8, r5, r7
add r12, r3, r2
add r6, r5, r12
ldr r9, .LCPI0_3            @ Load Constante .LCPI0_3
add r11, r4, r2
mul r10, r11, r9
mov r7, #161
orr r5, r7, #37, 24         @ 9472
mla r7, r6, r5, r8
mov r8, #142
orr r8, r8, #9, 24          @ 2304
mla r9, r4, r8, r10
add r4, r9, r7
add r4, r4, #1, 22          @ 1024
asr r4, r4, #11
str r4, [r1, #28]           @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8
ldr r4, .LCPI0_4            @ Load Constante .LCPI0_4
mul r8, r12, r4
mla r12, r6, r5, r8
ldr r5, .LCPI0_5            @ Load Constante .LCPI0_5
add r4, r3, lr
mul r6, r4, r5
mov r4, #179
orr r4, r4, #65, 24         @ 16640
mla r5, r3, r4, r6
add r3, r5, r12
add r3, r3, #1, 22          @ 1024
asr r3, r3, #11
str r3, [r1, #20]           @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8
mov r3, #149, 30            @ 596
orr r3, r3, #6, 20          @ 24576
mla r4, lr, r3, r6
add r3, r4, r7
add r3, r3, #1, 22          @ 1024
asr r3, r3, #11
str r3, [r1, #12]           @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8
```

```
mov r3, #11
orr r3, r3, #3, 20          @ 12288
mla lr, r2, r3, r10
add r2, lr, r12
add r2, r2, #1, 22          @ 1024
asr r2, r2, #11
str r2, [r1, #4]            @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar24".
                            @ Desplazamiento con "stride" 8
mvn r1, #31
bne .LBB0_1
.LBB0_2:                                @ %bb3.i
                                        @ Numero de vueltas=8
                                        @ =>This Inner Loop Header: Depth=1
ldr r0, .LCPI0_0            @ Load Constante .LCPI0_0
add r0, r0, r1
ldr r2, [r0, #32]           @ Load var "data". Reuso espacial. Var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
ldr r3, [r0, #64]           @ Load var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
ldr lr, [r0, #128]          @ Load var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
ldr r4, [r0, #256]          @ Load var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
ldr r9, [r0, #224]          @ Load var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
str r2, [sp]                @ Store de pila. Desplazamiento 0
str r3, [sp, #8]            @ Store de pila. Desplazamiento 8 bytes
ldr r12, [r0, #96]          @ Load var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
add r5, r4, r2
ldr r11, [r0, #192]         @ Load var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
add r10, r9, r3
ldr r6, [r0, #160]          @ Load var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
add r7, r6, lr
str lr, [sp, #4]            @ Store de pila. Desplazamiento 4 bytes
add r8, r7, r5
adds r1, r1, #4
add r2, r11, r12
add r3, r2, r10
add lr, r3, r8
rsb r3, r3, #2
add lr, lr, #2
sub r2, r10, r2
add r3, r3, r8
asr lr, lr, #2
```

```
str lr, [r0, #32]           @ Store var "data". Reuso espacial.
                            @ Var iteracion "indvar". Desplazamiento con "stride" 1
asr r3, r3, #2
str r3, [r0, #160]          @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
sub r3, r5, r7
mov r7, #81
orr r5, r7, #17, 24         @ 4352
add lr, r3, r2
mul r7, lr, r5
mov r5, #126
orr lr, r5, #6, 22          @ 6144
mla r5, r3, lr, r7
add r3, r5, #1, 18          @ 16384
asr r3, r3, #15
str r3, [r0, #96]           @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
ldr r3, .LCPI0_1            @ Load Constante .LCPI0_1
mla lr, r2, r3, r7
add r2, lr, #1, 18          @ 16384
ldr r5, [sp, #4]            @ Load de pila. Desplazamiento 4 bytes
asr r2, r2, #15
ldr lr, [sp, #8]            @ Load de pila. Desplazamiento 8 bytes
str r2, [r0, #224]          @ Store var "data". Reuso espacial. V
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
sub r3, r12, r11
ldr r2, [sp]                @ Load de pila. Desplazamiento 0
sub lr, lr, r9
ldr r7, .LCPI0_2            @ Load Constante .LCPI0_2
ldr r9, .LCPI0_3            @ Load Constante .LCPI0_3
sub r2, r2, r4
sub r4, r5, r6
add r5, r4, lr
add r12, r3, r2
add r6, r5, r12
mul r8, r5, r7
add r11, r4, r2
mul r10, r11, r9
mov r5, #161
orr r5, r5, #37, 24         @ 9472
mla r7, r6, r5, r8
mov r8, #142
orr r8, r8, #9, 24          @ 2304
mla r9, r4, r8, r10
add r4, r9, r7
add r4, r4, #1, 18          @ 16384
asr r4, r4, #15
str r4, [r0, #256]          @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
ldr r4, .LCPI0_4            @ Load Constante .LCPI0_4
mul r8, r12, r4
mla r12, r6, r5, r8
```

```
ldr r5, .LCPI0_5            @ Load Constante .LCPI0_5
add r4, r3, lr
mul r6, r4, r5
mov r4, #179
orr r4, r4, #65, 24         @ 16640
mla r5, r3, r4, r6
add r3, r5, r12
add r3, r3, #1, 18          @ 16384
asr r3, r3, #15
str r3, [r0, #192]          @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
mov r3, #149, 30            @ 596
orr r3, r3, #6, 20          @ 24576
mla r4, lr, r3, r6
add r3, r4, r7
add r3, r3, #1, 18          @ 16384
asr r3, r3, #15
str r3, [r0, #128]          @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
mov r3, #11
orr r3, r3, #3, 20          @ 12288
mla lr, r2, r3, r10
add r2, lr, r12
add r2, r2, #1, 18          @ 16384
asr r2, r2, #15
str r2, [r0, #64]           @ Store var "data". Reuso espacial.
                            @ Var calculada basada en var iteracion "indvar".
                            @ Desplazamiento con "stride" 1
bne .LBB0_2
@ BB#3:                                 @ %jpeg_fdct_islow.exit
add sp, sp, #12
pop {r4, r5, r6, r7, r8, r9, r10, r11, lr}
bx lr
@ BB#4:
.align 2
.LCPI0_0:
.long data
.align 2
.LCPI0_1:
.long 4294952159              @ 0xffffc4df
.align 2
.LCPI0_2:
.long 4294951227              @ 0xffffc13b
.align 2
.LCPI0_3:
.long 4294959923              @ 0xffffe333
.align 2
.LCPI0_4:
.long 4294964100              @ 0xfffff384
.align 2
.LCPI0_5:
.long 4294946301              @ 0xffffadfd
.Ltmp0:
.size main, .Ltmp0-main
```

```
.type data,%object          @ @data
.data
.align 5
data:
.long 81                    @ 0x51
.long 10854                 @ 0x2a66
.long 1893                  @ 0x765
.long 55245                 @ 0xd7cd
.long 7746                  @ 0x1e42
.long 47274                 @ 0xb8aa
.long 61698                 @ 0xf102
.long 14040                 @ 0x36d8
.long 32421                 @ 0x7ea5
.long 52299                 @ 0xcc4b
.long 9138                  @ 0x23b2
.long 35805                 @ 0x8bdd
.long 43626                 @ 0xaa6a
.long 35259                 @ 0x89bb
.long 36543                 @ 0x8ebf
.long 10710                 @ 0x29d6
.long 48276                 @ 0xbc94
.long 63894                 @ 0xf996
.long 43968                 @ 0xabc0
.long 15210                 @ 0x3b6a
.long 56961                 @ 0xde81
.long 39369                 @ 0x99c9
.long 58893                 @ 0xe60d
.long 34185                 @ 0x8589
.long 24771                 @ 0x60c3
.long 17874                 @ 0x45d2
.long 18063                 @ 0x468f
.long 43200                 @ 0xa8c0
.long 44136                 @ 0xac68
.long 37554                 @ 0x92b2
.long 14103                 @ 0x3717
.long 40800                 @ 0x9f60
.long 52611                 @ 0xcd83
.long 50634                 @ 0xc5ca
.long 49833                 @ 0xc2a9
.long 8835                  @ 0x2283
.long 61041                 @ 0xee71
.long 57729                 @ 0xe181
.long 10443                 @ 0x28cb
.long 12765                 @ 0x31dd
.long 59451                 @ 0xe83b
.long 42864                 @ 0xa770
.long 64983                 @ 0xfdd7
.long 57735                 @ 0xe187
.long 11241                 @ 0x2be9
.long 53364                 @ 0xd074
.long 19713                 @ 0x4d01
.long 510                   @ 0x1fe
.long 2376                  @ 0x948
.long 53949                 @ 0xd2bd
.long 31983                 @ 0x7cef
.long 59580                 @ 0xe8bc
```

```
.long 60021                    @ 0xea75
.long 53139                    @ 0xcf93
.long 55323                    @ 0xd81b
.long 18120                    @ 0x46c8
.long 50781                    @ 0xc65d
.long 3849                     @ 0xf09
.long 53253                    @ 0xd005
.long 4950                     @ 0x1356
.long 3081                     @ 0xc09
.long 16644                    @ 0x4104
.long 51078                    @ 0xc786
.long 43350                    @ 0xa956
.size data, 256
```

Figura C.24: Fichero jfdctint.arm.opt.s

Apéndice D

Manual de Uso

D.1. Introducción

En muchas ocasiones, cuando se decide realizar una nueva versión de otro proyecto desarrollado hace un tiempo, se encuentran problemas a la hora de probarlo y compilarlo.

Esto es así porque lo que para el desarrollador inicial es algo obvio, dado que ha estado trabajando en ello durante varios meses, para la persona que continúa no es tan fácil. Puede haber problemas de incompatibilidad con el software, problemas de configuración y otros muchos, por los que se pierde bastante tiempo hasta que se llegan a resolver.

Es por ello que se ha escrito este manual, para hacer de guía y facilitar el trabajo a futuros desarrolladores que decidan continuar con el trabajo descrito en este proyecto.

En este manual se describirán los requisitos de software que se tienen que instalar, los pasos que hay que seguir para compilar las bibliotecas y para compilar los códigos fuente que se quieran analizar. Además, se enumerarán los archivos que da como resultado.

D.2. Requerimientos Software

Para poder usar las bibliotecas administradas, es necesario tener instaladas en su ordenador las siguientes herramientas:

Sistema operativo: Cualquier sistema operativo basado en UNIX o similar, como GNU/Linux o Mac OS.

GCC 4.5 o superior: Se ha probado en esta versión y el alguna superior, por lo cual no se puede asegurar su correcto funcionamiento en versiones anteriores.

LLVM 2.7 o superior: Es a partir de esta versión de LLVM la que se añade LLVM metadata, necesario para la correcta compilación.

D.3. Compilación de las bibliotecas

Las dos bibliotecas, como se ha mencionado anteriormente, han sido creadas en GNU/Linux. El código fuente de estas bibliotecas y un script llamado "Makefile" que compila los dos ficheros fuente y crea las bibliotecas, se encuentran dentro del directorio "buclesReusos". Si desea ejecutar el script, siga los siguientes pasos:

1. Abra un terminal.
2. Muévase a través de los directorios hasta donde se encuentre la carpeta llamada "buclesReusos".
3. En la línea de comandos del terminal, escriba la palabra *make* y pulse *intro*.

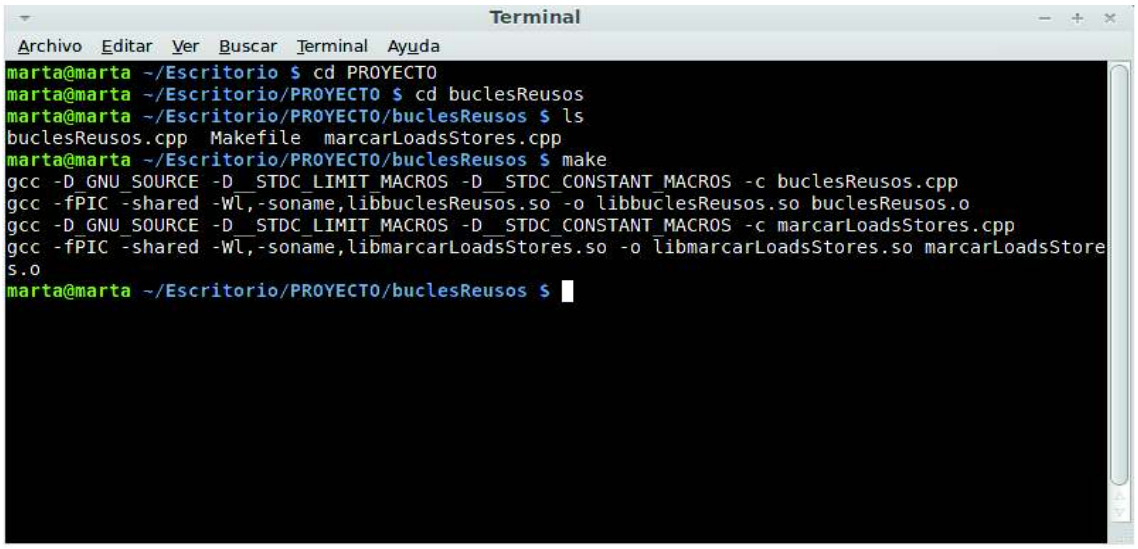

Figura D.1: Ejemplo de compilación de bibliotecas

Si, por el contrario, prefiere crearlas manualmente:

1. Abra un terminal.
2. Muévase a través de los directorios hasta donde se encuentre la carpeta llamada "buclesReusos"
3. Escriba los siguientes comandos dentro de este directorio:

```
gcc -D_GNU_SOURCE -D__STDC_LIMIT_MACROS -D__STDC_CONSTANT_MACROS
        -c buclesReusos.cpp
gcc -fPIC -shared -Wl,-soname,libbuclesReusos.so
        -o libbuclesReusos.so buclesReusos.o
gcc -D_GNU_SOURCE -D__STDC_LIMIT_MACROS -D__STDC_CONSTANT_MACROS
        -c marcarLoadsStores.cpp
gcc -fPIC -shared -Wl,-soname,libmarcarLoadsStores.so
        -o libmarcarLoadsStores.so marcarLoadsStores.o
```

Figura D.2: Comandos para compilar las bibliotecas

D.4. Compilación de los ficheros a analizar

Como se ha indicado a lo largo de la memoria del proyecto, el código fuente de los ficheros a analizar es C. Cada uno de los ejemplos, que se han incluido en el proyecto, está en una carpeta con el mismo nombre del fichero. En cada carpeta hay (como muestra la Figura D.3), además del código fuente, un script con todos pasos necesarios , el cual se puede ejecutar siguiendo unos pasos parecidos a la compilación de bibliotecas.

1. Abra un terminal.
2. Muévase por los directorios hasta que se situe dentro de la carpeta del ejemplo que desea analizar.
3. En la línea de comandos del terminal, puede escribir distintos comandos, según sea su intención:
   - Compilar el código fuente: *make*
   - Pasar las bibliotecas libmarcarLoadsStores y libbuclesReusos: *make opt*
   - Compilar y pasar las bibliotecas: *make all opt*

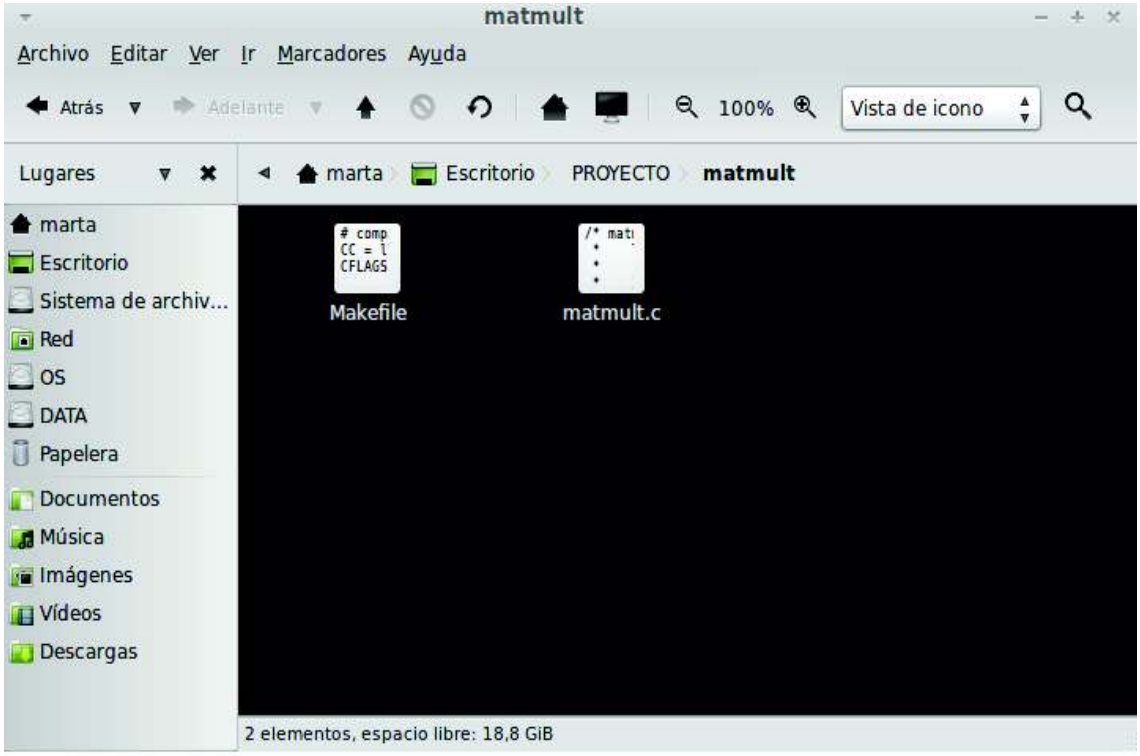

Figura D.3: Contenido de la carpeta

Si lo que desea es probar algún código que no se encuentra entre los administrados, la manera más simple para hacerlo, sería la siguiente:

1. Cree una carpeta con el nombre que usted elija.
2. Copie el fichero con el código que desea analizar dentro de la carpeta creada anteriormente.
3. Cree un fichero nuevo dentro de la carpeta con el nombre "Makefile".
4. Copie el código escrito en la siguiente página, el mostrado en la Figura D.4 .
5. Cambie <nombreFich>por el nombre del fichero que contiene su código (sin extensión) en todos sitios en los que aparece.
6. Cambie <rutaHastaLibreria>por la ruta que necesite hasta llegar al directorio donde se encuentran las dos bibliotecas.
7. Guarde el fichero.
8. Abra un terminal.
9. Muévase por los directorios hasta que se sitúe dentro de la carpeta del ejemplo que desea analizar.
10. En la línea de comandos del terminal, puede escribir distintos comandos, según sea su intención:
    - Compilar el código fuente: *make*
    - Pasar las bibliotecas libmarcarLoadsStores y libbuclesReusos: *make opt*
    - Compilar y pasar las bibliotecas: *make all opt*

```
# compiler
CC = llvm-gcc
CFLAGS = -static -O3 -emit-llvm

# linker
LD = llvm-ld
LDFLAGS =

# native machine code generator
NATIVE= llc
ARCH = arm
NATIVEFLAGS = -march=$(ARCH)

# disassembler .bc -> .ll
DIS = llvm-dis
DISFLAGS =

all: arch bytecode
arch: <nombreFich>.$(ARCH).s
bytecode: <nombreFich>.ll

<nombreFich>.o:
<nombreFich>.bc: <nombreFich>.o
        $(LD) $(LDFLAGS) <nombreFich>.o -o <nombreFich>
%.$(ARCH).s: %.bc
        $(NATIVE) $(NATIVEFLAGS) -o $@ $<
%.ll: %.bc
        $(DIS) $(DISFLAGS) -o $@ $<
opt:
opt -instnamer <nombreFich>.ll -S -o <nombreFich>.ll
opt -load <rutaHastaLibreria>/libmarcarLoadsStores.so
        -marcar -ll-input <nombreFich>.ll -ll-output
        <nombreFich>Marcado.ll <nombreFich>.ll > /dev/null
llc $(NATIVEFLAGS) <nombreFich>Marcado.ll
        -o <nombreFich>.$(ARCH).s
opt -loops -loop-rotate -mem2reg -instcombine -simplifycfg
        -indvars -instnamer -load
        <rutaHastaLibreria>/libbuclesReusos.so -cuenta
        -arm-input <nombreFich>.$(ARCH).s -arm-output
        <nombreFich>.$(ARCH).opt.s < <nombreFich>Marcado.ll
        > /dev/null
clean:
        rm -f *.o *.bc *.s *.ll <nombreFich>
```

Figura D.4: Código ejemplo de Makefile

En la siguiente imagen se muestra cómo quedaría el código del fichero "Makefile" después de hacer los cambios para un fichero llamado "prueba1".

```
# compiler
CC = llvm-gcc
CFLAGS = -static -O3 -emit-llvm

# linker
LD = llvm-ld
LDFLAGS =

# native machine code generator
NATIVE= llc
ARCH = arm
NATIVEFLAGS = -march=$(ARCH)

# disassembler .bc -> .ll
DIS = llvm-dis
DISFLAGS =

all: arch bytecode
arch: prueba1.$(ARCH).s
bytecode: prueba1.ll

prueba1.o:
prueba1.bc: prueba1.o
        $(LD) $(LDFLAGS) prueba1.o -o prueba1
%.$(ARCH).s: %.bc
        $(NATIVE) $(NATIVEFLAGS) -o $@ $<
%.ll: %.bc
        $(DIS) $(DISFLAGS) -o $@ $<
opt: opt -instnamer prueba1.ll -S -o prueba1.ll
opt -load <rutaHastaLibreria>/libmarcarLoadsStores.so
        -marcar -ll-input prueba1.ll -ll-output
        prueba1Marcado.ll prueba1.ll > /dev/null
llc $(NATIVEFLAGS) prueba1Marcado.ll -o prueba1.$(ARCH).s
opt -loops -loop-rotate -mem2reg -instcombine -simplifycfg
        -indvars -instnamer -load <rutaHastaLibreria>/
        libbuclesReusos.so -cuenta -arm-input prueba1.$(ARCH).s
        -arm-output prueba1.$(ARCH).opt.s < prueba1Marcado.ll
        > /dev/null
clean:
        rm -f *.o *.bc *.s *.ll prueba1
```

Figura D.5: Makefile de "Prueba1"

D.5. Ficheros Resultado

Como consecuencia de los pasos dados hasta el momento, se habrán creado en el directorio de código fuente los siguientes ficheros:

<*nombreFichero*>*.ll*: Fichero con el código intermedio de LLVM.

<*nombreFichero*>*marcado.ll*: Fichero que añade al anterior los metadatas de las instrucciones loads y stores.

<*nombreFichero*>*.bc*: Fichero con el código en binario.

<*nombreFichero*>*.o*: Fichero objeto.

<*nombreFichero*>*.arm.s*: Fichero con el código intermedio con las instrucciones de ensamblador ARM.

<*nombreFichero*>*.arm.opt.s*: Fichero que contiene, además del código anterior, la cuenta de iteracciones, y los reusos temporales y espaciales.

<*nombreFichero*>: Ejecutable.

Si desea borrar estos ficheros:

1. Vuelva abrir el terminal.
2. Muévase por los directorios hasta que se sitúe dentro de la carpeta del ejemplo que desea eliminar.
3. En la línea de comandos del terminal escriba: *make clean*

Así ya habrá borrado todos los ficheros nuevos.

Bibliografía

[1] The LLVM Compiler Infrastructure Project. http://www.llvm.org

[2] Gedit text editor. http://projects.gnome.org/gedit/

[3] GCC, the GNU Compiler Collection. http://gcc.gnu.org/

[4] TEX Live. http://www.tug.org/texlive/

[5] Kile - an Integrated LATEX Environment. http://kile.sourceforge.net/

[6] Gantt Project. http://www.ganttproject.biz/

[7] ARM, The Architecture for the Digital World. http://infocenter.arm.com/

[8] LLVM API Documentation. http://llvm.org/doxygen/

[9] A Low-level Virtual Instruction Set Architecture MICRO-36. San Diego: CA. December 2003. http://llvm.org/pubs/2003-10-01-LLVA.pdf

[10] An Infrastructure for Multi-Stage Optimization Masters Thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Dec. 2002. http://llvm.org/pubs/2002-12-LattnerMSThesis.pdf

[11] Edición de documentos en LATEX. Universidad de Zaragoza, 1990.

[12] Aburruzaga, Gerardo. Manual libre de Make en PDF. http://www.uca.es/softwarelibre/publicaciones/make.pdf

[13] Savitch, Walter. Resolución de problemas con C++. Prentice Hall .2006.